Grundlagen für das Steuerungs- und Überwachungssystem der Drei-Spektrometer-Anlage am Elektronenbeschleuniger MAMI

Dissertation

zur Erlangung des Grades

"Doktor

der Naturwissenschaften"

am Fachbereich Physik
der Johannes Gutenberg–Universität
in Mainz

Helmut Kramer
geboren in Mainz

Mainz 1995

Dekan:	Prof. Dr. F.Scheck
1. Berichterstatter:	Prof. Dr. K.Merle
2. Berichterstatter:	Prof. Dr. J.Friedrich

Tag der mündlichen Prüfung:

Inhaltsverzeichnis

Kapitel 1

Einleitung

Seit Mitte der 50er Jahre stellt die Elektronenstreuung [hof56] ein mächtiges Werkzeug zur Untersuchung der Kern- und Nukleonenstruktur dar. Dies liegt vor allem daran, daß die Wechselwirkung zwischen punktförmigen Elektronen und hadronischer Materie im wesentlichen elektromagnetischer Natur und im Rahmen der QED wohl verstanden ist.
Die Elektronenstreuung wird im Rahmen dieser Theorie als Austausch von virtuellen Photonen zwischen den Projektilen und den Bausteinen des hadronischen Targets beschrieben. Bei leichten bis mittelschweren Kernen und bei niedrigen Energien ist die Beschreibung der elektromagnetischen Wechselwirkung in 1.Bornscher Näherung durch den Austausch eines Photons wegen der kleinen Kopplungskonstanten, $\alpha \approx 1/137$, hinreichend genau (Ein-Photon-Austausch). Das ausgetauschte Photon trägt dabei einen raumartigen Viererimpuls.
Sind sowohl die einlaufenden Elektronen als auch das Target unpolarisiert, dann lassen sich die Wirkungsquerschnitte von Reaktionen, bei denen das gestreute Elektron in Koinzidenz mit einem aus dem Target herausgeschlagenen Teilchen nachgewiesen wird, in vier Anteile, die sogenannten Strukturfunktionen, zerlegen [bof92]. Diese enthalten die Information über die Eigenschaften des zu untersuchenden hadronischen Systems. Durch eine geeignete Wahl der Reaktionskinematik, ist es möglich, die Strukturfunktionen aus den gemessenen Wirkungsquerschnitten zu separieren ("Rosenbluth-Separation").

Eine Vorraussetzung für die Durchführbarkeit von solchen Koinzidenzexperimenten ist ein hohes Tastverhältnis des einfallenden Elektronenstrahls. Deshalb wurde am Institut für Kernphysik ein Dauerstrich-Elektronenbeschleuniger, das Mainzer Mikrotron MAMI[her90], entwickelt. MAMI besteht aus einem 3.5 MeV Linearbeschleuniger und drei hintereinandergeschalteten Mikrotrons, die mit normalleitenden Hochfrequenzsektionen arbeiten. Die letzte Stufe liefert einen Strahl mit einer variablen Endenergie von 180 MeV bis maximal 855 MeV.

Zur Durchführung einer Vielzahl unterschiedlichster Streuexperimente, wie z.B. die Anregung der Δ-Resonanz in Kernen, die Pionproduktion an Nukleonen, die π_0-Produktion an der Schwelle, etc., wurde für deren Durchführung von der A1-Kollaboration[a1] eine aus drei Spektrometern bestehende Mehrzweckanlage entwickelt und aufgebaut [neu94]. Sie besteht aus drei um ein gemeinsames Zentrum drehbaren, hochauflösenden, magnetischen Spektrometern, die jeweils mit einem Satz von Bildebenen-Detektoren (Driftkammern, Szintillationszähler, Čerenkov-Detektor) ausgestattet sind. Die Drei-Spektrometer-Anlage wird in Kapitel 2 kurz vorgestellt.

Um eine Anlage dieser Komplexität effizient für Experimente einsetzen zu können, ist es notwendig, Rechner nicht nur zur reinen Auslese der Daten, sondern auch zur Experimentsteuerung und -überwachung einzusetzen. Der Aufbau eines umfangreichen Kontrollsystems hat sich schon beim Aufbau und Betrieb des MAMI-

Beschleunigers als unabdingbar erwiesen. Zielsetzung der vorliegenden Arbeit war es, ein Konzept für die rechnergestützte Steuerung und -überwachung der Drei-Spektrometer-Anlage zu entwickeln und die grundlegenden Werkzeuge dafür zu realisieren.

Bei der Drei-Spektrometer-Anlage ist es sinnvoll, zur Steuerung der Anlage mit mehreren Rechnern zu arbeiten (Abschnitt 3.2), die über ein lokales Netzwerk miteinander verbunden sind. Damit zerfällt allerdings das Steuerungs- und Überwachungssystem der Anlage in eine Vielzahl von Programmen, die auf verschiedene Rechner verteilt sind. Daher sind umfangreiche Synchronisations- und Kommunikationsproble zu lösen. Dafür wurde im Rahmen dieser Arbeit ein Softwarepaket namens MUPIX (Kap.4) erstellt, das eine nicht nur für den Einsatz an der Drei-Spektrometer-Anlage geeignete Lösung dieses Problems darstellt.

Es ist wegen der Komplexität der Anlage und wegen der Größe und Zusammensetzung der Experimentiergruppe davon auszugehen, daß nicht jeder an den Experimenten Beteiligte jedes Detail überblickt. Deshalb ist es notwendig, den Experimentator bei dieser Aufgabe durch geeignete Hilfsmittel zu unterstützen. Dafür sollen zum einen im Rahmen der Experimentsteuerung die Werte aller technischen Parameter der Anlage erfaßt und auf ihre Richtigkeit überprüft werden (Kap.5). Darüberhinaus soll der aktuelle Zustand der Anlage auch auf Grund der während eines Experiments erfaßten (Detektor-)Daten beurteilt werden (Kap.7). Diese beiden Ziele ergänzen sich gegenseitig. Zu ihrer Realisierung waren die entsprechenden Werkzeuge zu entwickeln.

Die Verwendbarkeit der für das Steuerungssystems erstellten allgemeinen Softwarekomponenten wurde speziell am Magnetsystem der Spektrometer überprüft. Dabei wurde zunächst die Einrichtung zur Magnetfeldmessung mit Hilfe der Kernspinresonanzmethode konzipiert und aufgebaut und sodann im konkreten Aufbau der Drei-Spektrometer-Anlage realisiert und getestet. Dieses System wird in Kapitel 6 vorgestellt. Das hierfür entwickelte Softwarepaket ist sehr allgemein und kommt deshalb im Rahmen des Steuerungssystems zum Einsatz.

Kapitel 2

Die Drei-Spektrometer-Anlage

Die Drei-Spektrometer-Anlage (siehe Abb.2.1) besteht aus drei um ein gemeinsames Zentrum, den Targetort, drehbaren magnetischen Spektrometern [ko95, sch95]. Jedes dieser Spektrometer ist mit einem gleichartigen Satz von Teilchendetektoren ausgestattet: Zwei doppelten "vertikalen" Driftkammern [sau95, di95], zwei Ebenen von schnellen Plastik-Szintillationszählern [ri94] und einem Gas-Čerenkov-Detektor[li95]. Die Detektorsysteme der einzelnen Spektrometer unterscheiden sich nur durch ihre geometrischen Ausmaße.
Alle Detektoren sind zur Verminderung des Strahlungsuntergrundes in einer Abschirmeinheit [A1-91] untergebracht, deren Wände aus 40 cm dickem Kiesbeton zur Abschirmung von schnellen Neutronen bestehen. Langsame Neutronen werden durch die Beimischung von Tetraborcarbid unterdrückt. Zur Reduktion des Strahlungsuntergrundes ist die Abschirmeinheit zusätzlich mit Bleiplatten ausgekleidet.
Bei speziellen Experimenten kann dieser Aufbau noch durch andere Detektorsysteme erweitert werden, z.B. einen Neutronendetektor oder einen BGO-Ball.

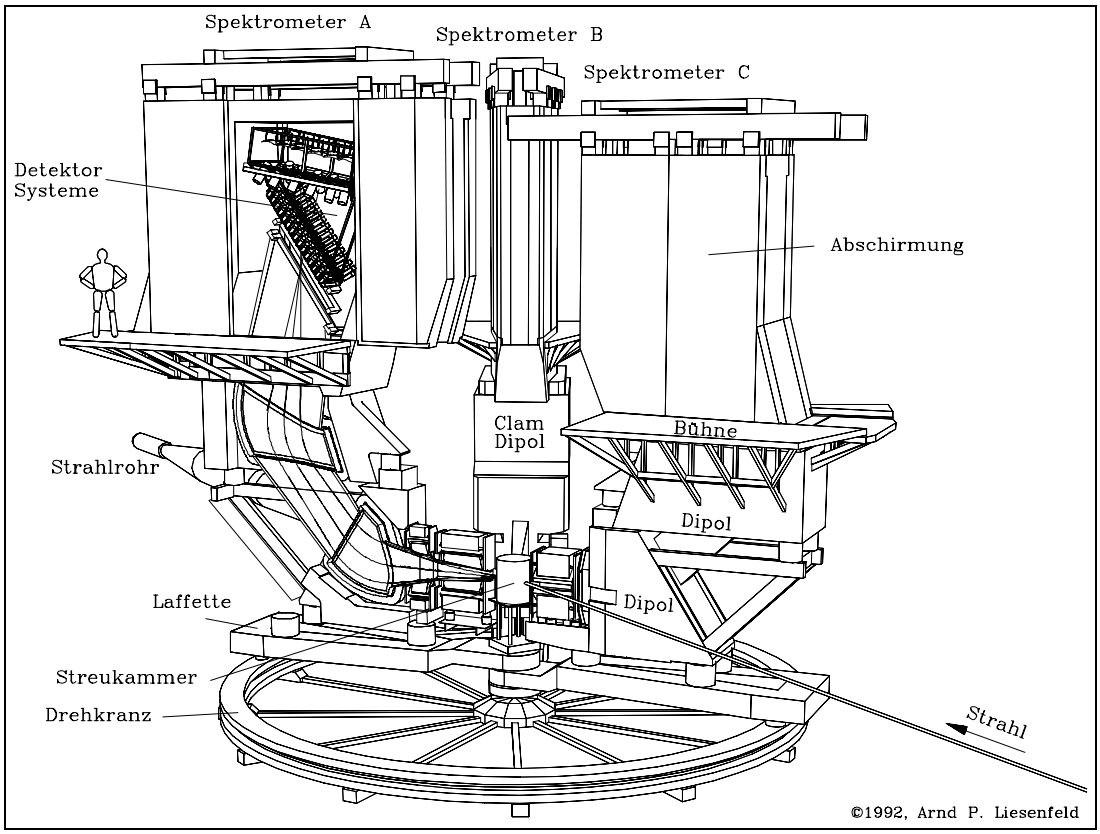


Abb.2.1 stellt die Drei-Spektrometer-Anlage dar.

Neben den durch "konventionelle" Elektronenstreuung zugänglichen Strukturfunktionen (R_L, R_T, R_{LT}) wird der transversal-transversale Interferenzterm (R_{TT})

erst durch die Wahl des Teilchenemissionswinkels außerhalb der Streuebene separierbar. Deshalb wurde Spektrometer B so gebaut, daß es für sogenannte "out-of-plane" Messungen durch einen Hubmechanismus aus der Streuebene herausgekippt werden kann.
Bisher wurden bereits zwei der drei Spektrometer fertiggestellt und in Betrieb genommen. Die Inbetriebnahme des dritten Spektrometers (C) erfolgt während der Niederschrift der vorliegenden Arbeit.

Magnetspektrometer

Die beiden Spektrometer A und C sind als QSDD-Spektrometer konzipiert, d.h. sie bestehen aus einem Quadrupol-, Sextupol- und zwei homogenen Dipolmagneten. Um möglichst hohes Impuls- und Winkelauflösungsvermögen zu erzielen, wurde für beide Spektrometer eine Punkt-zu-Punkt- in der dispersiven und eine Parallel-zu-Punkt-Abbildung in der nicht-dispersiven Ebene gewählt. Spektrometer A (C) kann einfach geladene Teilchen bis zu einem maximalen Impuls von 735 MeV/c (551 MeV/c) nachweisen. Sie haben eine Raumwinkelakzeptanz von 28 msr und eine Impulsakzeptanz $\Delta p/p$ von 20% (25%).
Spektrometer B ist als "Clam-Shell"-Dipolmagnet konzipiert, um ein besseres Ortsauflösungsvermögen der Targetkoordinaten bei der Verwendung von ausgedehnten Targets zu erzielen. Es besitzt eine Punkt-zu-Punkt-Abbildung in beiden Ebenen. Spektrometer B erlaubt den Nachweis einfach geladener Teilchen bis zu einem Impuls von maximal 870 MeV/c. Der Raumwinkel beträgt 5.6 msr bei einer Impulsakzeptanz von 15%. Bei der vollen Raumwinkelakzeptanz wurde mit beiden Spektrometern eine Impulsauflösungsvermögen δp/p von $\leq 10^{-4}$ und eine Winkelauflösung am Target von etwa 3mrad erreicht [ko95].

Driftkammern

Um das geforderte hohe Impuls- und Winkelauflösungsvermögen der Spektrometer zu erzielen, sind positionsempfindliche Detektoren in der Bildebene mit einer hohen Ortsauflösung ($\sigma < 100$ μm) notwendig. Wegen der großen geometrischen Abmessungen der Bildebene und des z.T. großen Winkelbereichs der Teilchenbahnen werden als Spurendetektoren großflächige, vertikale Driftkammern (Zellgröße 5mm horizontal, 24mm vertikal) eingesetzt. Jedes Spektrometer ist mit vier Kammern ausgestattet, mit deren Hilfe zwei Ortskoordinaten und zwei Winkel bestimmt werden. Das jeweils erzielte intrinsische Auflösungsvermögen [sau95, di95] der einzelnen Koordinaten ist in der nachfolgenden Tabelle zusammengefaßt:

	Intrinsisches Auflösungsvermögen				sens. Fläche
	$\sigma_x/\mu m$	$\sigma_y/\mu m$	$\sigma_\phi/mrad$	$\sigma_\theta/mrad$	A/m^2
Kammer A	94	260	1.350	0.216	2.22 × 0.405
Kammer B	114	235	1.223	0.258	2.35 × 0.120
Kammer C ist noch im Aufbau					

Triggerdetektoren

Szintillationszähler

Auf die beiden Driftkammern folgen zwei Ebenen von Plastikszintillationszählern. Sie liefern die für die Driftzeitmessung der Kammern und für die Koinzidenz

benötigten Triggersignale. Beide Ebenen sind in dispersiver Richtung segmentiert. Die Szintillatorsegmente werden an beiden Seiten an einen Photomultiplier gekoppelt[1]. Durch die Segmentierung wird die Laufstrecke des Lichts im Szintillator verringert und damit die Zeitauflösung verbessert. Bei der Inter-Spektrometerkoinzidenz zwischen den Spektrometern A und B wurde bisher eine Koinzidenzzeitauflösung von etwa einer Nanosekunde und besser erzielt. Die Abbildung 2.2 zeigt ein typisches Spektrum.
Die Szintillatoren der ersten Ebene ("dE–Ebene") sind dünner (3mm) als die der zweiten, 10 mm dicken "Time–of–Flight"–Ebene (ToF), um mit der Messung des spezifischen Energieverlusts Pionen und Positronen von Protonen unterscheiden zu können. Eine Trennung von Elektronen und Pionen ist wegen der geringen Differenz im Energieverlust nicht möglich.

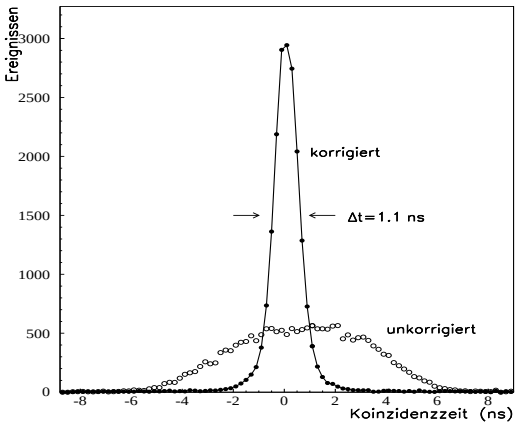


Abb.2.2 zeigt ein Koinzidenzzeitspektren für $^{16}O(e,e'p)$ bei $p_m = 120$ MeV/c. Die Korrekturen des Spektrums enthalten die Flugzeitunterschiede in den Spektrometern und die Timing-Differenzen der einzelnen Szintillatoren. (Bild von E.Offermann [A1-93])

Gas–Čerenkov–Schwellendetektor

Um bei Experimenten mit dem Nachweis von Pionen den unerwünschten Elektronen– bzw. Positronenuntergrund zu reduzieren, wurde jedes Spektrometer mit einem Gas–Čerenkov–Schwellendetektor ausgestattet. Zur Erzeugung des Čerenkov-Lichts wird als Radiator Freon–114 eingesetzt. Die Ansprechwahrscheinlichkeit wurde bei Spektrometer A zu 99.98% ermittelt [li95]. Die Daten für den Čerenkov-Detektor von Spektrometer B sind noch nicht vollständig ausgewertet.

Zum Betrieb der Drei-Spektrometer-Anlage sind im Endausbau etwa 150-200 Geräte notwendig. Sie müssen bei der Durchführung von Experimenten gesteuert und überwacht werden. Weil einige der Geräte zum Betrieb mehrerer Komponenten dienen, ist die Parameterzahl noch um eine Zehnerpotenz größer. Beispielsweise versorgt ein Hochspannungsmodul 32 Photomultiplier. Deshalb ist es zur effizienten Nutzung der Anlage notwendig, Rechner zu deren Steuerung und Überwachung einzusetzen.

[1] Bei Spektrometer B reicht die einseitige Auslese aus.

Kapitel 3

Konzeptionelle Vorüberlegungen

3.1 Motivation

Die große Anzahl an Geräteparametern einerseits und die bei den Streuexperimenten freigesetzte Strahlung andererseits machen eine Handsteuerung der Anlage vor Ort unmöglich. Dies gilt auch für solche Parameter, die zwar nicht eingestellt aber für die Bestimmung der experimentellen Wirkungsquerschnitte oder für Kontrollzwecke benötigt werden. Deshalb muß die Steuerung der Anlage über Rechner vom Kontrollraum aus erfolgen.
Dies ist ohnehin notwendig, weil es dem Experimentator auf Grund der Komplexität der Anlage während einer Messung nicht mehr möglich ist, den Zustand der Anlage vollständig zu überwachen, schnell Fehlersituationen zu erkennen und deren Ursache herauszufinden. Er muß deshalb bei der Erledigung dieser Aufgaben durch geeignete Werkzeuge unterstützt werden, die zusammen das *Überwachungssystem* der Drei-Spektrometer-Anlage bilden.
Schließlich kann man durch die Automatisierung von Einstellvorgängen, das Vereinheitlichen der Bedienoberflächen der Geräte und die Zentralisierung von Statusinformation über den Rechner das Maß an Komplexität bei der Bedienung der Drei-Spektrometer-Anlage reduzieren. Damit verringert sich auch die Anzahl von Fehlersituationen und Fehlbedienungen.

Die Problematik wird anhand einer im Meßbetrieb aufgetretenen Situation deutlich, die in Abb.3.1 dargestellt wird. Hier blieb ein Fehler in der Gasversorgung der Driftkammern über einen längeren Zeitraum unentdeckt. Die Folgen solcher Fehler reichen von einer Verminderung der Datenqualität, die evtl. durch eine aufwendigere Auswertung auszugleichen ist, bis hin zum totalen Datenverlust. Das einfache Beispiel aus Abb.3.1 zeigt bereits die Punkte auf, wo Werkzeuge des Überwachungssystems ansetzen können:

Apparative Überwachung und Steuerung: Unter diesen Aspekt des Überwachungssystems fällt die Aufgabe, die Werte aller technischen Parameter der Anlage zu erfassen und auf ihre Richtigkeit zu überprüfen, d.h. den momentanen Zustand der Apparatur (Ist-Zustand) mit dem Soll-Zustand zu vergleichen (einer dieser Parameter ist z.B. der Gasfluß aus Abb.3.1). Dies ist mit Hilfe der elektronischen Datenverarbeitung möglich. Dazu müssen allerdings die physikalischen Informationen der einzelnen Geräte dem Rechner zugänglich sein. Steuerung und Überwachung der Apparatur sind deshalb eng miteinander verknüpft! Beide Aufgaben sind deshalb Teil eines gemeinsamen Softwaresystems — das Experiment-Kontroll-System (ECS[1]). Man bezeichnet diesen Bereich auch als *Slow Control*, weil Auslese- und Einstellvorgänge im Vergleich zur Erfassung der Detektorsignale langsam erfolgen. Im folgen-

[1]Experiment Control System

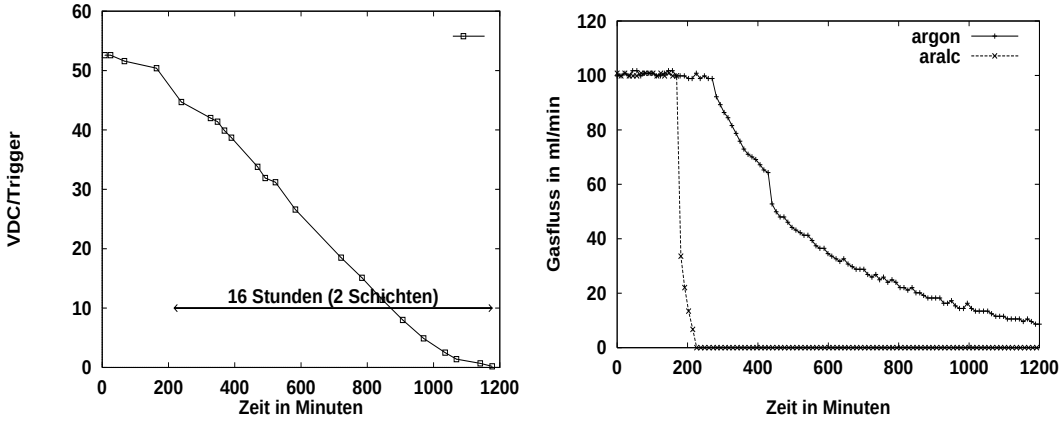


Abb. 3.1: Die linke Abbildung zeigt die zeitliche Entwicklung des Verhältnis aus gültigen Kammertreffen zur Gesamtzahl der Triggerereignisse. Jeder Punkt gibt dabei den Durchschnittswert für einen Run an. Es ist deutlich zu erkennen, daß sich das Verhältnis über einen Zeitraum von etwa 16 Stunden verringerte. Ursache hierfür ist ein Problem in der Versorgung der Kammer mit Argon (rechtes Bild). Das linke Bild allein gibt noch keinen Aufschluß über die Fehlerursache. Der Abfall der Kammereffizienz könnte zum Beispiel auch eine Folge von Driften in der Kammerhochspannung oder in der Schwellenspannung der zur Signalauslese benutzten Vorverstärker sein. Den Diagrammen liegen Daten aus einer Strahlzeit zur Multi-Hadron-Elektroproduktion an ^{12}C [A1-90] zu Grunde.

den werden Einstell-, Auslese- und Regelvorgänge verkürzt unter dem Begriff Steuern zusammengefaßt.

On-line Datenkontrolle: Auf dieser Stufe wird auf der Basis der beobachteten (Koinzidenz-)Ereignisse während der Messung (on-line) ein Vektor abgeleitet, der aus statistischen Größen besteht, die zusammen den aktuellen Zustand der Anlage im Betrieb beschreiben (die "Kammereffizienz" aus Abb.3.1 ist z.B. eine Komponente dieses Vektors). Im Sinne einer noch zu spezifizierenden Norm wird die zeitliche Entwicklung dieses "Zustandsvektors" beurteilt, um ggf. Fehlersituationen festzustellen.

Da in die zu analysierenden Ereignisse das Verhalten der gesamten Apparatur einschließlich der Datenerfassung eingeht, läßt die On-line Datenkontrolle nicht immer den direkten Schluß auf Fehlerursachen zu. Umgekehrt beziehen sich die Aussagen der apparativen Überwachung auf Parameter, die unabhängig von anderen Einflüssen meßbar sind. Allerdings ist die Funktionalität nicht aller Komponenten der Anlage so direkt ablesbar, wie z.B. defekte Auslesekabel. Solche Fehler machen sich erst bei der Datenanalyse bemerkbar, in die auch Änderungen eingehen, die außerhalb der direkt erfaßten Parameter liegen, wie z.B. die Strahlqualität.
Aus diesen Gründen ist es notwendig, daß bei der Realisierung des Überwachungssystems beide Aspekte berücksichtigt werden, um hierauf aufbauend in einer dritten Stufe eine automatische **Diagnose** von Fehlersituationen zu leisten. Während das Diagnose-System auf niedrigster Ebene im Fehlerfall einfach zum Anhalten der laufenden Messung führt, liefert es auf der höchsten Ebene Informationen über die Fehlerursache. Das Zusammenspiel der einzelnen Stufen ist in Abb.3.2 dargestellt.

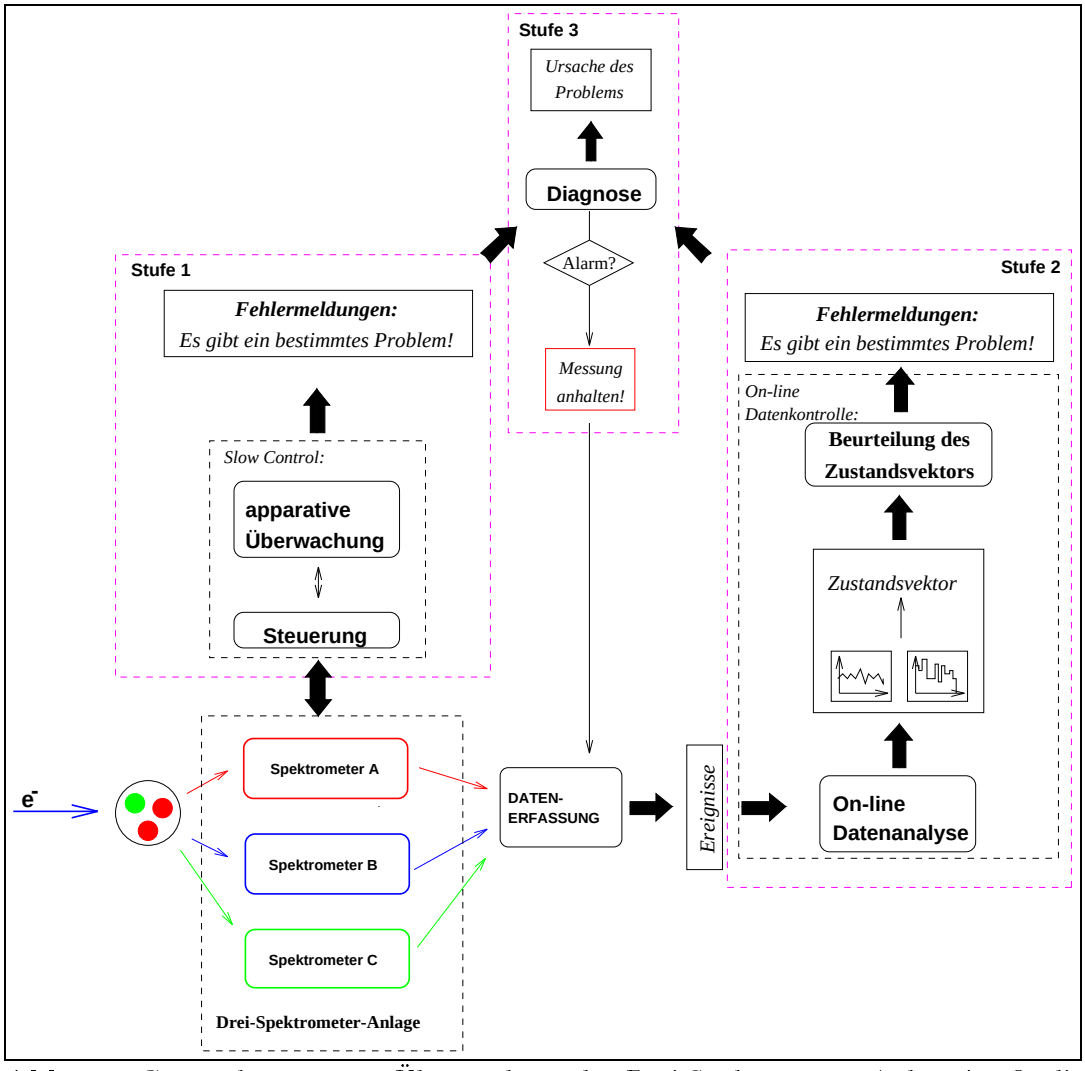


Abb 3.2: Gesamtkonzept zur Überwachung der Drei-Spektrometer-Anlage im On-line System.

Die mit der vorliegenden Arbeit bearbeiteten Projekte liegen im wesentlichen auf der ersten und zweiten Stufe des Gesamtkonzepts. Dabei wurden die grundlegenden Werkzeuge sowohl für die apparative Überwachung und Steuerung (Kap.4 und Kap.5) als auch für die On-line Datenkontrolle (Kap.7) entwickelt und in das On-line System integriert. Mit diesen Werkzeugen soll auch ein automatisch geführtes Protokoll aller überwachten Parameter und ggf. beobachteten Fehler geführt werden, das beim Auftreten von Fehlern zur Diagnose der Fehlersituation dient. Ein solches *elektronisches Protokollbuch* muß alle Parameter enthalten, die man bei der späteren Datenanalyse benötigt (Strahlstrom, Spektrometereinstellungen, ...). Die beobachteten Fehler müssen dem Experimentator während der Messung in geeigneter Weise angezeigt werden, damit er ggfs. die Messung anhält.

Der Schwerpunkt der vorliegenden Arbeit liegt auf Entwicklungen in dem grundlegendem Bereich der rechnergestützen Steuerung und apparativen Überwachung der Drei-Spektrometer-Anlage. Die dazu notwendigen, von speziellen Geräten unabhängigen Softwaremodule sind in gleichem Maße die Grundlage für die Erfüllung der Steuerungs- und Überwachungsaufgaben; sie müssen in Hinblick auf beide

Aufgaben in standardisierter Weise entwickelt werden.

3.2 Eigenschaften des Steuerungssystems

In diesem Abschnitt werden die wesentlichen Eigenschaften des Steuerungs- und Überwachungssystems der Drei-Spektrometer-Anlage diskutiert, die sich weitgehend unabhängig von der speziellen Implementation aus dem apparativen Aufbau ergeben.

3.2.1 Objekte und Klassen

Um die Apparatur mit Hilfe eines Rechners steuern und überwachen zu können, muß sie in Form von Softwaremodulen innerhalb des Rechners widergespiegelt werden. Hierzu werden, wie bei fast allen im Rahmen dieser Arbeit bearbeiteten und vorgestellten Projekten, bei der Analyse, dem Design und der Programmierung objekt-orientierte Methoden eingesetzt, die zur Realisierung komplexer Systeme entwickelt wurden [boo91, coa90].
Der Vorteil dieser Technik liegt u.a. darin, daß die erforderliche Datenkapselung und Modularisierung durch spezielle Sprachelemente einer objekt-orientierten Programmiersprache direkt unterstützt wird, so daß man in der Regel sehr gut les- und wartbare Programme erhält. Diese Sprachelemente erleichtern die Erstellung und Entwicklung von Software-Bausteinen, die sich im Vergleich zur üblichen Programmiermethode durch

- einen einfacheren Entwurf,
- eine verbesserte Test- und Wartbarkeit,
- eine schnellere Anpaßbarkeit an veränderte Anforderungsprofile

auszeichnen [boo91, coa90].

Bei der *Objekt-orientierten Programmierung (OOP)* wird das zu lösende Problem in einzelne Problembereiche zerlegt, die ihrerseits aus miteinander "kommunizierenden Einheiten" bestehen. Diese Einheiten bilden innerhalb des Softwaresystems sogenannte Objekte, die z.B. im Bereich des Steuerungssystems den zu steuernden Komponenten der realen Welt zuzuordenen sind. Im Sinne der OOP besteht ein solches *Objekt* aus:

- Eigenschaften: Daten, die den Zustand eines Objekts beschreiben;
- Diensten: Operationen, die ein Objekt zur Manipulation der Daten anbietet.

Objekte "tauschen untereinander Nachrichten aus", um Aktionen auszulösen. In Abb.3.3 ist die Beschreibung zweier Objekte dargestellt, die die Netzgeräte zum Betrieb der Dipolmagnete AD1 und AD2 repräsentieren. Gleichartige Objekte haben gemeinsame Teile, die sich nur durch die aktuellen Werte ihrer Eigenschaften unterscheiden. Die Extraktion gemeinsamer Teile führt zur Bildung von *Klassen*. Dieser Vorgang wird ebenfalls in Abb.3.3 skizziert.

Während sich *gleichartige* Objekte zu einer Klasse zusammenfassen lassen, können *ähnliche* Objekte Klassen zugeordnet werden, die voneinander abgeleitet werden.

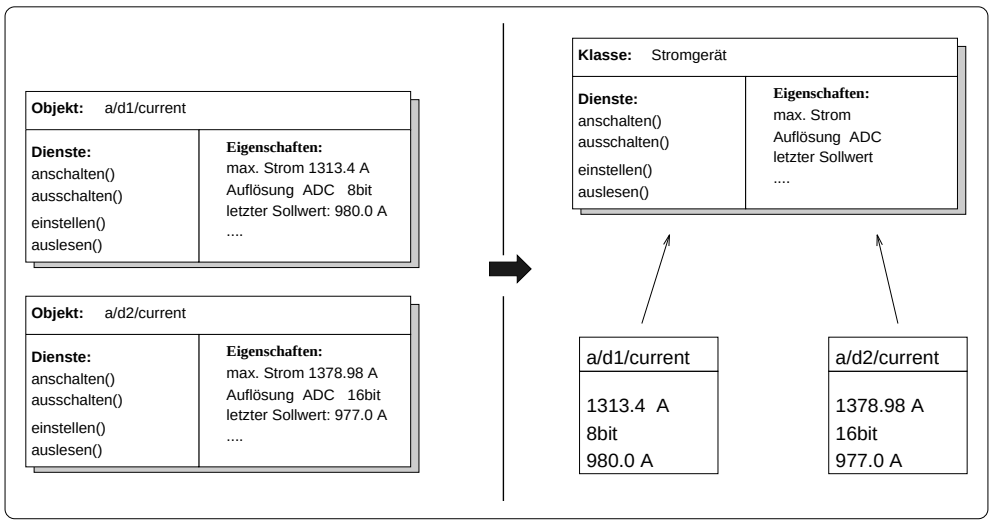


Abb.3.3: Das Verhältnis von Objekten und Klassen am Beispiel der Netzgeräte für die Dipole D1 und D2 von Spektrometer A.

Zum Beispiel unterscheidet sich ein Dipolmagnet im Sinne der Steuerung nur dadurch von einem Quadrupolmagneten, daß er zusätzlich noch mit NMR-Sonden ausgestattet ist, über die die Feineinstellung des Magnetfelds erfolgt. Es ist deshalb zweckmäßig bei der Abbildung dieser beiden Komponenten der Anlage auf Objekte von einer gemeinsamen Basisklasse auszugehen.
Die Realsierung dieser Klassenrelationen in Form der sogenannten *Vererbung* ist ein wesentliches Merkmal der OOP-Sprachen [boo91, str86]. Damit ist es bspw. möglich, eine Basisklasse mit den Diensten und den Datenstrukturen für die Steuerung des "Basismagneten" zu implementieren, aus der dann mit wenigen zusätzlichen Programmzeilen der speziellere Dipolmagnet hervorgeht, ohne daß Änderungen an der Implementation des Basistypen vorgenommen werden müßten (Abb.3.4). Diese Programmiertechnik erlaubt es, in sehr effizienter Weise auf Änderungen der an ein System gestellten Anforderungen zu reagieren, eine Situation, die sich häufig bei einer Vielzweckanlage wie der Drei-Spektrometer-Anlage ergibt.

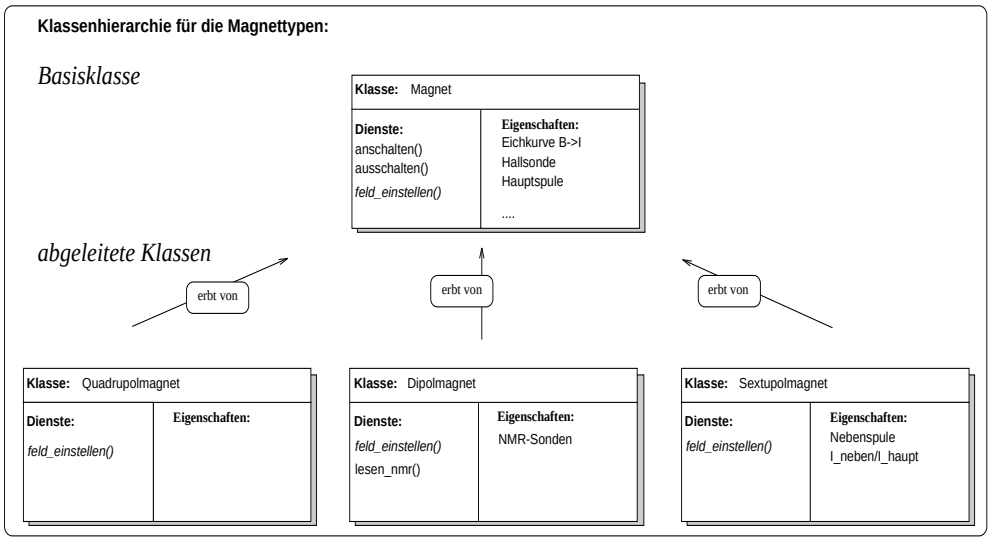


Abb.3.4: Vererbung am Beispiel der Magnettypen.

Da ursprünglich kein leistungsfähiger und zugleich kostengünstiger Compiler einer objekt-orientierten Sprache zur Verfügung stand, wurden zunächst die entsprechenden Techniken im Rahmen der ansonsten im On-line Bereich eingesetzten Sprache ANSI-C [ker88] soweit möglich simuliert. Später[2] wurde C++ [str86] als OOP-Sprache ausgewählt, da sie den Sprachumfang von ANSI-C enthält und da zusätzlich die Koexistenz mit Fortranbibliotheken sichergestellt ist[kun92].

Ein C++ Programm besteht im obigen Sinn aus einer Vielzahl von interagierenden Objekten, von denen jedes eine spezifische Funktionalität anbietet. Im Rahmen von C++ werden die Eigenschaften eines Objekts auch als "Member-Variable" und seine Dienste als "Member-Funktionen" oder "Aktionen" bezeichnet.

3.2.2 Prozesse - Rechner - Kommunikation

Modularisierung - Programme

Zur Modularisierung der Software ist es erforderlich, die Ansteuerung eines Teilgeräts der Apparatur in geräte- und in schnittstellenspezifische Anteile zu zerlegen. Die gerätespezifischen Anteile werden in der dem Gerät zugeordneten Klasse - der Geräte-Klasse - zusammengefaßt. Diese Klasse modelliert die Struktur des speziellen Geräts und stellt alle spezifischen Kontroll- und Steueralgorithmen zur Verfügung. Ein Objekt dieser Klasse tauscht Nachrichten mit den I/O-Objekten (*Schnittstellenumsetzern*) aus, die die Schnittstellen repräsentieren, über die das Gerät an den Rechner angeschlossen ist. Ein Beispiel ist in Abb.3.5 dargestellt.
Um die Flexibilität und Wartbarkeit der Software weiter zu erhöhen, wird darüber hinaus die Modularisierung auf höherer Ebene fortgesetzt, indem man nicht alle Objekte innerhalb eines einzigen Programms realisiert, sondern auf mehrere Programme verteilt. Durch diese Aufteilung wird vermieden, daß bei der Einbindung neuer Geräte in das System Änderungen an bereits getesteten Programmen vorgenommen werden müssen.
Programme, die Geräte- oder I/O-Objekte enthalten, werden im folgenden als *Gerätetreiber* oder kurz als *Treiber* bezeichnet.

Parallelisierung - Prozesse

Da Steuervorgänge z.T. längere Zeit in Anspruch nehmen, ist es notwendig, sie parallel auszuführen, um Wartezeiten zu vermeiden. Das gilt besonders für sehr langwierige Prozeduren, wie z.B. das Einstellen der Magnetfelder, das Fahren der Kollimatoren und Targets, sowie das Hochfahren aller Photomultiplierhochspannungen; das Einstellen der Magnetfelder dauert bspw. bis zu 90 Minuten.

Deshalb ist es erforderlich, das Steuerungssystem in mehrere Programme aufzuspalten, die dann in sogenannten Prozessen vom (Multitasking -) Betriebssystem des Rechners (quasi) parallel abgearbeitet werden. Die Parallelisierung ist in vielen Fällen möglich, z.B. wenn

[2] Seit Ende 1991 steht dem Institut mit dem GNU-C++ Compiler ein stabiler und vor allem kostenfreier Compiler zur Verfügung.

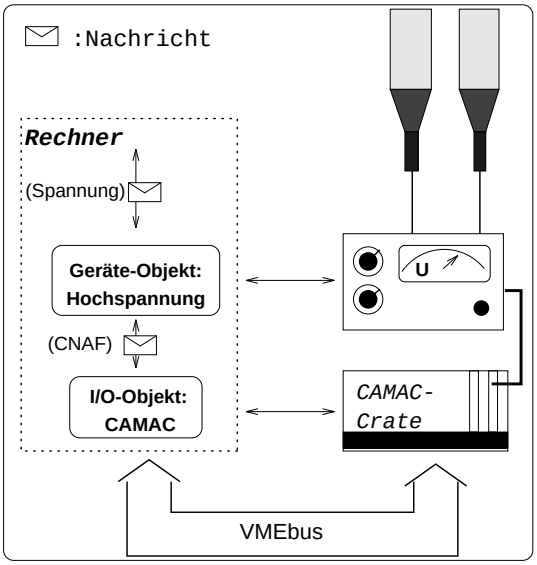


Abb.3.5: Zur Steuerung der Spannungsversorgung für die Photomultiplier der Szintillatoren benötigt man ein Geräte-Objekt und zusätzlich ein I/O-Objekt, welches das CAMAC-Interface[a] bedient, über das die Spannungsversorgung angeschlossen ist. Der Rechner kommuniziert über den VMEbus mit dem CAMAC-Überrahmen, und damit über diesen schließlich mit der Hochspannung.

[a] CAMAC: Computer Automated Measurement and Control, Bus Standard in der Kernphysik zum Anschluß von Modulen an einen Rechner.

- die Aufgaben voneinander unabhängig sind, was insbesondere für die Steuerung und Überwachung von Geräten, die sich auf verschiedenen Spektrometern befinden, gilt, und wenn
- langwierige Einstellvorgänge aus mehreren Einzelschritten bestehen, zwischen denen eine Wartepause eingelegt werden muß. Beim Einstellen des Magnetfelds wird z.B. zwischen jedem Einzelschritt einige Minuten gewartet, damit die Feldverteilung im Magneteisen dem Spulenstrom folgen kann. Diese Wartezeit kann vom Rechner genutzt werden, um andere Aufgaben zu erledigen.

Lokale Verteilung

Damit jedes Spektrometer als eigenständige Einheit betrieben werden kann, wird zum Zugriff auf die Experimentelektronik, die sich bei jedem Spektrometer auf der Arbeitsbühne hinter dem Abschirmhaus befindet, ein eigener *Prozeßrechner* ("Frontend-System") eingesetzt.
Als Prozeßrechner wurden VMEbus-Systeme ausgewählt, weil sie wegen ihrer weiten Verbreitung in der Industrie ein gutes Preis/Leistungs-Verhältnis, eine hohe Zuverlässigkeit und ein großes Angebot von Schnittstellenkarten zur Anbindung der Geräte bieten [ku89]. Bei den im Institut bisher verwendeten VMEbus-Systemen handelt es sich um Rechner der Firma Eltec (Mainz) mit der Typenbezeichnung E6, die auf dem M68030 Prozessor der Firma Motorola basieren.
Jeder Prozeßrechner besitzt ein Netzwerk-Interface, mit dem er an das lokale Netzwerk (Ethernet) des Instituts angeschlossen ist. Um eine höhere Betriebssicherheit zu erreichen, werden die Prozeßrechner ohne jegliche lokale Zusatzperipherie (Platten, Bänder, Graphikbildschirm) betrieben. Um diese Peripherie bereitzustellen und zur Realisierung anderer zentraler Aufgaben, wie z.B. zum Vorhalten von Konfigurationsinformation innerhalb einer Datenbank, wird noch mindestens ein zusätzliches System benötigt.
Für solche Aufgaben steht eine Workstation (Rechner vom Typ DEC5200 der Firma Digital) mit der entsprechenden Peripherie im Kontrollraum der Drei-Spektrometer-Anlage zur Verfügung. Sie ist über das Netzwerk mit den Prozeß-

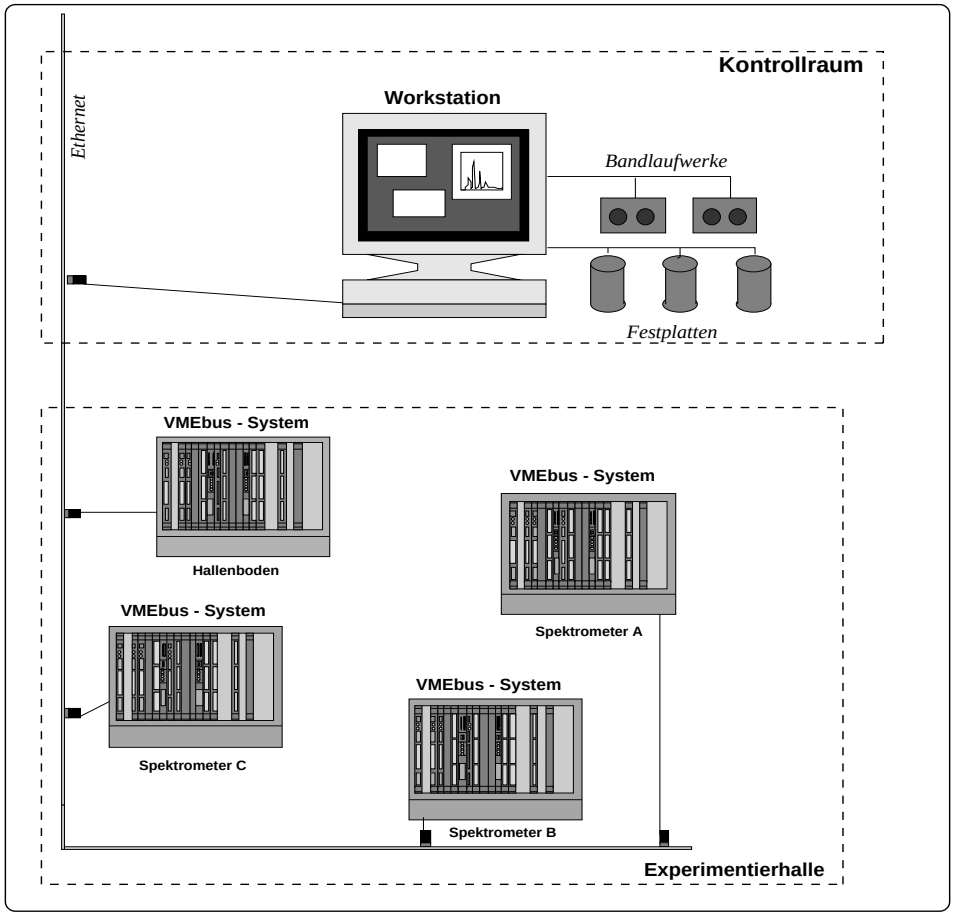


Abb. 3.6 zeigt die an der Drei-Spektrometer-Anlage eingesetzten Rechner. Auf jedem Spektrometer und auf dem Hallenboden befindet sich ein VMEbus System zur direkten Ansteuerung der Experimentelektronik. Diese sind über das Netzwerk mit einer Workstation im Kontrollraum verbunden.

rechnern verbunden. Alle Maschinen laufen mittlerweile unter der Kontrolle eines UNIX-Betriebssystems[3]. Der Netzwerkaufbau ist in Abb.3.6 dargestellt.

Client/Server-Architektur

Aus zwei Gründen ist es allerdings notwendig, daß nur der Teil der Software auf den Frontend-Rechnern läuft, der direkt über den VMEbus auf die Hardware zugreift:

- Die vorhandenen Frontend-Rechner sind in ihrer Leistungsfähigkeit beschränkt und damit nicht in der Lage, alle bei der Steuerung und Datenerfassung anfallenden Aufgaben alleine zu bewältigen[4].
- Es kommt u.a. wegen der direkten Anbindung des Prozeßrechners an die Experimenthardware hin und wieder vor, daß dieser Rechner insgesamt neu

[3]Ursprünglich wurden die VMEbus-Rechner unter OS-9 und die Workstations unter VAX/VMS betrieben.

[4]Wenn auch dieses Argument inzwischen an Gewicht verloren hat, weil mittlerweile leistungsfähigere Rechner erhältlich sind, so war es doch zur Zeit der Entwicklung von erheblicher Bedeutung.

initialisiert ("Reboot") werden muß. In diesem Fall müssen alle auf diesem Rechner arbeitenden Programme neu gestartet werden. Zur Minimierung dieses Aufwandes und der dabei von Zeit zu Zeit auftretenden Fehlern ist die Anzahl der davon betroffenen Programme möglichst klein zu halten.

Insbesondere gehören zu dieser Software die Schnittstellenumsetzer, die von voneinander unabhängigen Objekten genutzt werden. Man spricht in diesem Zusammenhang auch von einer *Client/Server-Architektur*. Zum Beispiel fungiert ein bestimmter CAMAC-Prozeß als Server, da er einen Dienst, nämlich die Ausführung von CAMAC Zugriffen über ein spezielles Interface, anbietet. Dieser Dienst wird von anderen Prozessen, den Klienten, in Anspruch genommen. Einer dieser Klienten[5] ist z.B. ein Prozeß, der das Objekt "Hochspannungsversorgung" aus Abb.3.5 enthält. Dieser Prozeß läuft aus den genannten Gründen nicht auf dem Frontend-Rechner, sondern auf der Workstation. Ein komplexes Zusammenspiel von Objekten ist in Abb.3.7 dargestellt.

Synchronisierung

Es wurde diskutiert, daß es notwendig ist, die Objekte auf verschiedene Programme – und diese auf verschiedene Rechner – aufzuteilen, um die Abarbeitung von Steuerungsaufgaben zu parallelisieren; das Einstellen aller Magnetfelder erfolgt z.B. mit Hilfe mehrerer parallel arbeitender Prozesse, auch das Rampen aller Photomultiplier-Hochspannungen, das Setzen aller Diskriminatorschwellen und die Einstellung der Koinzidenzlogik kann gleichzeitig vorgenommen werden.
Diese Parallelisierung wirft allerdings das Problem auf, den Zugriff auf gemeinsame Ressourcen zu synchronisieren. So benutzt man z.B. bei der Einstellung aller Dipolmagnete aus Kostengründen dieselbe Elektronik bei der Feldmessung mittels der Kernresonanzmethode (siehe Kap. 6), und die Ansteuerung der Module auf einem Spektrometer verläuft über dasselbe CAMAC-Interface, das darüber hinaus vom Programm zur Datennahme genutzt wird. Zwangsläufig ergeben sich Synchronisationsprobleme in allen Fällen, wo die Anzahl der Schnittstellenumsetzer kleiner ist als die Zahl der Geräte, die darüber angesprochen werden, z.B. bei allen physikalischen Schnittstellenumsetzern, die als Bussysteme oder Multiplexer ausgelegt sind.
Deshalb darf beim Zugriff mehrerer Objekte auf eine gemeinsame Ressource die bestimmte Ressource nur einmal als Objekt aktiviert sein. Eine mehrfach genutzte Ressource ist deshalb als Service in einem eigenständigen Prozeß zu implementieren. Über diesen Server-Prozeß laufen alle Zugriffe auf die Ressource von seiten der Klienten. Diese Lösung ist in Abb.3.7 dargestellt.
Die Synchronisation mit anderen Mitteln – zum Beispiel mit einer Semaphore[6] – kommt bei der gegebenen Zielsetzung nicht in Frage, weil dann alle Prozesse, die auf die zu synchronisierende Ressource zugreifen, auf dem gleichen Rechner laufen müßten.

[5] Die Client/Server-Beziehung gilt auch zwischen den Objekten.

[6] Bei Semaphoren [tan87, ste92] handelt es sich um einen vom Betriebssystem zur Verfügung gestellten Zähler, mit dem man Zugriffe auf von mehreren Prozessen genutzten Ressourcen synchronisieren kann. Dieser Zähler kann in einem Schritt gelesen und beschrieben werden.

Inter-Objekt-Kommunikation

Kommunizieren Objekte innerhalb desselben Prozesses miteinander, so beinhaltet der abstrakte Begriff "Nachrichtenaustausch" den direkten Aufruf einer Member-Funktion.
Durch die Aufspaltung der Steuerungsaufgaben auf mehrere Prozesse kommt es jedoch vor, daß Objekte Dienste eines zweiten Objektes, das im Rahmen eines anderen Prozesses realisiert ist, anfordern. Um dies zu ermöglichen, benötigt man einen aufwendigeren Mechanismus zum Austausch von Informationen zwischen den einzelnen Prozessen — ein *Interprozeßkommunikations*-System (IPC). Um das Ziel zu erreichen, nur noch solche Treiberprozesse auf den Frontend-Systemen zu starten, die den direkten Zugriff auf die Schnittstellen-Hardware realisieren, darf sich der IPC-Mechanismus nicht nur auf die lokale Kommunikation beschränken, sondern muß auch den Nachrichtenaustausch über das Netzwerk unterstützen. Damit sind die Schnittstellenumsetzer netzwerkweit erreichbar, also auch von Prozessen auf den Workstations!

Da von den miteinander kommunizierenden Objekten u.U. mehrere im Rahmen eines Prozesses realisiert sind (Abb.3.7), ist es erforderlich, daß die Kommunikationsendpunkte innerhalb der IPC-Software nicht die Prozesse, sondern die einzelnen Objekte sind. In diesem Sinn geht der benötigte IPC-Mechanismus als eine "Inter-Objekt"-Kommunikation über die klassische Prozeß-zu-Prozeß-Kommunikation hinaus.

Symbolische Adressierung

Um die Kommunikation nicht nur überschaubarer zu machen, sondern um auch die Möglichkeit zu gewinnen, Programme von einem Rechner auf einen anderen zu verschieben, ohne dabei das Gesamtsystem zu beeinträchtigen, erfolgt die Adressierung der Kommunikationsendpunkte (Objekte) mit symbolischen Namen. Damit brauchen den Kommunikationspartnern keine Verteilungsdetails mitgeteilt werden. Man spricht dabei von einem *ortstransparenten* System, mit dem größerer Verwaltungsaufwand vermieden wird.
Diese Vorgehensweise ist auch deshalb nützlich, weil die einzelnen Objekte i.a. direkt Teilen der Apparatur zugeordnet sind, die durch einen eindeutigen Namen charakterisiert sind. Durch die Namensvergabe in Anlehnung an die durch den Aufbau der Anlage gegebene "natürliche" Ordnung, läßt sich die Überschaubarkeit der Software erhöhen. Beispielsweise tragen die Objekte zum Betrieb der Dipolmagnete D1 und D2 von Spektrometer A Namen der Form "a/d1" oder "a/d2". Entsprechend werden die Objekte zur Feldmessung mit der Kernresonanz mit "a/d1/nmr" bzw. "a/d2/nmr" bezeichnet.
Prozesse, die mehrere Objekte enthalten, sind unter den Namen aller in ihnen enthaltenen Objekten adressierbar (*Aliasnamen*). Die Aufspaltung des Gesamtsystems ist in Abb.3.7 an einigen Beispielen skizziert.

Organisation der Konfigurationsparameter

Zur Vermeidung von Fehlerquellen muß die Konfigurationsinformation des Steuerungs- und Überwachungssystems sehr kompakt, d.h. insbesondere ohne Redun-

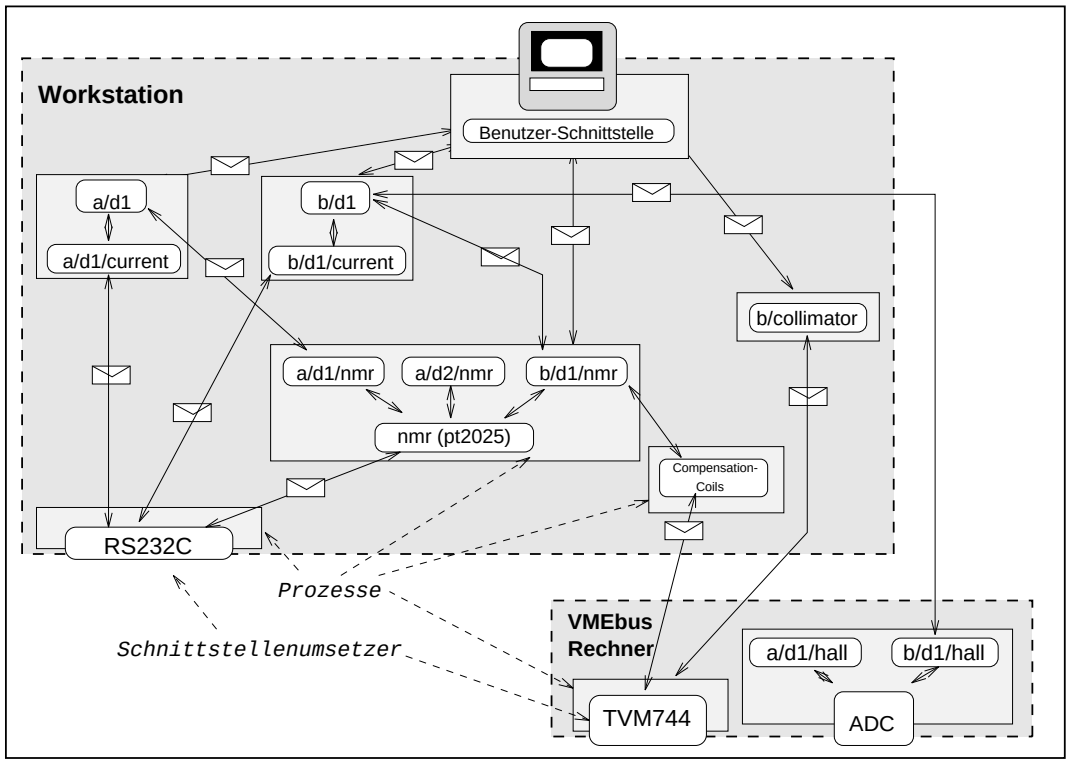


Abb.3.7 skizziert die Aufspaltung des Gesamtsystems am Beispiel einiger ausgewählter Objekte aus dem Bereich der Magnetsteuerung (siehe auch Kap.6). Man erkennt auch die Ausbildung von Client/Server-Paaren, wobei einige der Objekte zum gleichen Zeitpunkt sowohl als Server als auch als Klient fungieren, z.B. das mit "Compensation-Coils" bezeichnete Objekt.

danzen organisiert sein. Deshalb ist es auch an dieser Stelle erforderlich, objektorientiert vorzugehen. Dies sei anhand eines Beispiels verdeutlicht. Da die Steuerung und Überwachung aller Dipolmagnete mit Hilfe von Objekten erfolgt, die alle einer gemeinsamen Klasse angehören, unterscheiden sich die einzelnen Objekte (Dipolmagnete) nur durch spezifische Konfigurationsinformationen, z.B. Erregungskurven, maximale Feldeinstellungen etc., die zur Laufzeit aus einer Datenbank einzulesen sind; dazu wird der zur Adressierung eingeführte symbolische Name eines Objektes als Suchschlüssel verwendet.
Einige Konfigurationsparameter der Klasse "Dipolmagnet" sind jedoch unabhängig vom speziellen Objekt, wie z.B. die maximale Änderung des Feldes pro Einstellschritt oder die Präzision, mit der das Feld einzustellen ist. Bei der Parametersuche wird deshalb nach einem entsprechenden Eintrag in der Tabelle zur Klasse, der das Objekt angehört, gesucht, wenn für das spezielle Objekt kein entsprechender Eintrag existiert. Dabei wird der Klassenname als Primärschlüssel verwendet (im Beispiel ist das die Zeichenkette "Dipolmagnet").
Mit diesem Verfahren erreicht man ein größtes Maß an Flexibilität und man benötigt nur Einträge für die Parameter, die bei den Objekten verschiedenen sind. Auf diese Weise wird die Datenbank, den Anforderungen entsprechend, kleiner und damit leichter zu verwalten.

Hintergrundprozesse

Im allgemeinen reicht es nicht aus, Treiberprogramme nur für die Abarbeitung eines einzigen Steuerungsauftrags zu starten – sie müssen vielmehr während einer ganzen Messung als Hintergrundprozesse präsent sein. Dies geht u.a. aus nachfolgender Überlegung hervor:
Zur Überwachung der Apparatur ist es erforderlich, "ständig" Soll- und Istwerte von Geräteparametern zu überprüfen, um aus Abweichungen auf Fehlersituationen zu schliesen. Entweder sind die Geräte so konstruiert, daß sie Fehlersituationen selbst erkennen und diese eigenständig dem Rechner melden, oder sie müssen regelmäßig vom Rechner abgefragt werden (Polling). In die erste Kategorie fällt z.B. die Überwachung der Kühlwasserzufuhr der Magnete, während die Überwachung der Gasversorgung der Driftkammern der zweiten Kategorie angehört.
In beiden Fällen benötigt man eine Instanz innerhalb des Rechners, die entweder das Auslesen der Geräte anstößt oder auf Meldungen von seiten des Geräts wartet. Damit der Rechner möglichst zeitnah den aktuellen Zustand der Apparatur kennt, müssen die zugehörigen Treiberprozesse während der gesamten Messung aktiv sein.
Wegen der großen Zahl der Geräte ist es jedoch nicht möglich, daß alle Treiberprozesse im "Vordergrund" laufen; also direkt mit einer Tastatur und einem Bildschirm verbunden sind. Dies hat zweierlei Konsequenzen:

- Spezielle Programme, die sogenannten *Benutzerschnittstellen*, nehmen Steuerungskommandos vom Experimentator entgegen und leiten sie an die im Hintergrund laufenden Treiber weiter.
- Informationen (Fehlermeldungen, Statusmeldungen) der verschiedenen und auf mehrere Rechner verteilten Treiberprozesse müssen wieder durch eine zentrale Instanz zusammengefaßt werden, z.B. zur Realisierung eines elektronischen Logbuchs. Dieser Punkt wird in Kapitel 5 weiter ausgeführt.

Auch zur Realisierung dieser beiden Aufgaben benötigt man den bereits angesprochenen Kommunikationsmechanismus. Die dabei ausgetauschten Nachrichten müssen so kodiert sein, daß sie mit allgemeinen, von den speziellen Geräten unabhängigen Programmen, wie z.B. der genannten Benutzerschnittstelle, bearbeitet werden können. Andernfalls müßen solche Programme bei jedem neu hinzugefügten Gerät geändert werden, was den Wartungsaufwand der Software erheblich vergrößern würde. Da der Datenaustausch auch zwischen verschiedenen Rechnern (Prozeßrechner, Workstations) erfolgt, sind die Nachrichten in einer maschinen-unabhängigen Weise zu verpacken; z.B. muß der unterschiedlichen Binärdarstellung Rechnung getragen werden. Die Frage der geeigneten Kodierung der Nachrichten wird in Kapitel 5 behandelt.

Fazit

Zusammenfassend ergibt sich:

> Das Experimentsteuerungs- und Überwachungssystem der Drei-Spektrometer-Anlage besteht aus mehreren Programmen, die auf den verschiedenen am Experiment beteiligten Rechnern, die über ein Netzwerk miteinander verbunden

sind, laufen. Die Aufteilung des Systems wird verursacht durch die Komplexität der Apparatur und die räumliche Verteilung der zu steuernden Geräte. Um die Entwicklung und Wartung der Software so einfach wie möglich zu gestalten, wird die auf einem Rechner zu erledigende Arbeit weiter in kleinere Einheiten (Objekte) aufgespalten, die die Bausteine für die einzelnen Programme bilden. Entsprechend den jeweilligen Aufgaben laufen die Programme für den direkten Hardware-Zugriff auf VMEbus-Rechnern, für alle anderen Aufgaben stehen Workstations zur Verfügung.

Die zentrale Rolle bei der Realisierung des Experimentsteuerungs- und Überwachungssystems spielt die Interprozeßkommunikation. Diese wurde deshalb als erstes im Rahmen dieser Dissertation erarbeitet. Als Ergebnis entstand das Softwarepaket MUPIX[7], das im folgenden Kapitel vorgestellt wird.

Die Realisierung des gesamten Steuerungs- und Überwachungssystems mußte wegen dessen Umfangs in mehrere Teilprojekte aufgespalten werden. Insbesondere wurden im Rahmen von Diplomarbeiten [ma93, ha93] und einer weiteren Dissertation [ku95] die meisten Module zur Ansteuerung der einzelnen Geräte erstellt. Dabei wurde nach dem oben beschriebenen Konzept vorgegangen, dessen Realisierung durch zwei wesentliche Randbedingungen festgelegt ist, nämlich durch die Art und Weise, wie die Prozesse miteinander kommunizieren, und durch die Notwendigkeit, adäquate Überwachungsmechanismen in das verteilte Steuerungssystem einzuführen. Letztere werden in Kapitel 5 diskutiert.
Das Programm zur Verwaltung der Konfigurationsdatenbank wurde im Rahmen der Arbeiten [ku95, ma93, stef93] entwickelt und implementiert; es stellt im wesentlichen einen mit speziellen Suchfunktionen ausgestatteten Datei-Server dar. Eine interaktive, graphische Benutzerschittstelle für das ECS wurde im Rahmen einer Diplomarbeit [stef93] erstellt.
Im Rahmen der vorliegenden Dissertation wurde die Software entwickelt, die die Grundlage für das gesamte Steuerungs- und Überwachungssystem der Drei-Spektrometer-Anlage bildet; dabei wurden wesentliche Beiträge zur Entwicklung des beschriebenen Gesamtkonzepts geleistet.

[7] MUltiProcessor Interprocess Communication System

Kapitel 4

Das Message System MUPIX

4.1 Einleitung

Im vorangegangen Kapitel wurde dargelegt, daß die Experimentsteuerung der Drei-Spektrometer-Anlage auf mehrere Prozesse und Prozessoren verteilt ist. Mit der Aufteilung des Systems in parallel arbeitende Prozesse, die darüber hinaus auf verschiedenen Rechnern ablaufen, stellt sich die Aufgabe, für die Kommunikation unter diesen zu sorgen, um das Gesamtsystem zu synchronisieren und den Informationsfluß zu gewährleisten.
Obwohl dieses Problem auch bei anderen Steuerungs- und Datenerfassungssystemen für größere Experimente in der Mittel- und Hochenergiephysik auftritt, stand zum Beginn dieser Arbeit noch keine allgemein akzeptierte und implementierte Lösung dieses Problems zur Verfügung. Vielmehr existierte eine Vielzahl von Paketen (z.B. [mer84, wat87, hon88, san89]), die auf die speziellen Hard- und Softwarebedingungen der einzelnen Laboratorien abgestimmt und deshalb nicht für den Einsatz an der Drei-Spektrometer-Anlage geeignet waren. Unter diese Kategorie fällt auch das im Rahmen der Steuerung von MAMI eingesetzte "MAMI Messagesystem", das nur beschränkt für nicht-lokale Kommunikation nutzbar und auf Grund seiner starken Bindung an das Betriebssystem VMS wenig portabel [kre89] war.
In der vorliegenden Arbeit wurde ein Konzept für die Interprozeßkommunikation entwickelt, das für den Einsatz in LAN[1]-basierten, verteilten Kontrollsystemen geeignet ist. Die eigentliche Implementierung wurde unter dem Namen MUPIX als abkürzende Schreibweise für MUltiProcessor Interprocess Communication System durchgeführt.

4.2 Das Anforderungsprofil

Dieser Abschnitt beschäftigt sich zunächst mit den allgemeinen Anforderungen, die an das zu entwickelnde System von seiten der Experimentsteuerung gestellt werden. Eine der wesentlichen Eigenschaften wurde bereits im vorangegangenen Kapitel diskutiert, nämlich die Möglichkeit, mehrere verschiedene Empfänger innerhalb eines Prozesses zu adressieren, wobei jeder einzelne durch einen **symbolischen Namen** charakterisiert ist. Darüberhinaus sind folgende Anforderungen zu erfüllen:

Asynchrone Kommunikation: Ein wesentliches Kriterium bei der Auswahl des geeigneten Kommunikationsmechanismus ist die Art der Kopplung zwischen Sender und Empfänger. Bei dem einfachsten Verfahren, der synchronen Kommunikation, führt die rigorose Kopplung zwischen Sender und Empfänger

[1] Local Area Network

dazu, daß der langsamste Prozeß im System die Geschwindigkeit diktiert. Möchte z.B. im Rahmen der Experimentsteuerung ein übergeordnetes Programm eine Aktion bei einem untergeordneten Gerät auslösen, dann blockiert dieses Programm bei vollständig synchroner Kommunikation, falls der Empfänger gerade mit einer anderen Aufgabe beschäftigt ist.
Diese Situation ist im Hinblick auf die angestrebte Parallelisierung von Steuerungsaufgaben zu vermeiden. Dafür ist es erforderlich, die beiden Kommunikationspartner so zu entkoppeln, daß der Sender grundsätzlich nicht blockiert (**no-wait-send** Semantik); man spricht hier von **asynchroner Kommunikation**.
Eine derartige Kommunikation im ECS kann nicht nach dem Prinzip des Remote Procedure Calls [2] (RPC) [nel81] erfolgen, wohl aber in Form eines *Message Passing Systems* (MS). Beim Message Passing tauschen die Prozesse über spezielle Sende- und Empfangsroutinen Nachrichten (Messages) aus. Die Kommunikation ist hierbei ungerichtet, d.h. ein Prozeß kann mit beliebigen Partnern Daten austauschen; sie erfolgt asynchron über einen Zwischenpuffer. Da hierbei die Gefahr besteht, daß dieser Puffer überläuft, bedarf es einer zusätzlichen **Flußkontrolle** .

Zuverlässigkeit: Für die Funktionsfähigkeit des Gesamtsystems spielt die Zuverlässigkeit des Datentransports eine wesentliche Rolle. Im Rahmen der Experimentsteuerung reicht es aus, die sogenannte **At-most-once Zuverlässigkeit** [aut90] zu gewährleisten. Dabei wird eine Nachricht zwar möglicherweise mehrfach gesendet, ihre Duplikate aber ggf. herausgefiltert, so daß jede Nachricht höchstens einmal empfangen wird.

Portabilität: Da das System MUPIX auf allen am Experiment beteiligten Rechnern benötigt wird, muß es unter allen beteiligten Betriebssystemen (UNIX, VMS, OS-9) laufen; es muß entsprechend portierbar sein. Dabei ist es insbesondere erforderlich, daß die Schnittstellen zu MUPIX auf allen Rechnern die gleiche Form haben, um die Programmierung auf den verschiedenen Systemen zu vereinfachen und die **Portierbarkeit** zwischen diesen zu gewährleisten.
Dieser Punkt ist nicht nur für die Wartung der Software von größter Bedeutung, er hat sich auch schon in der Entwicklungsphase bewährt. Zum Beispiel konnten Programmentwicklung und -test auf den dafür besonders geeigneten Rechnern (UNIX-Workstations) betrieben werden, bevor die dann fertigen Programme auf die Zielsysteme portiert wurden. Diese Vorgehensweise erwies sich besonders hilfreich bei den Programmen, die ursprünglich unter OS-9 eingesetzt wurden, weil dieses System aus Geschwindigkeitsgründen ohne Speicherschutz betrieben wird, d.h ein Prozeß kann ungehindert in den Adreßraum eines beliebigen anderen Prozesses oder des Betriebssystemkerns schreiben, was bei eventuell noch vorhandenen Fehlern im Programm ggf. die Funktion des Gesamtsystems stört.

Timeouts: In einem aus mehreren Komponenten bestehenden System ist es erforderlich, Blockierungen (Deadlocks) zu verhindern, die durch den Ausfall einer Komponente entstehen. Deadlocks treten meistens dadurch auf, daß ein

[2] Beim RPC läßt ein Programm eine Prozedur auf einer anderen Maschine stellvertretend durch einen speziellen Prozeß ausführen; dabei kommt es zur Bildung von Prozeßpaaren - die Kommunikation ist gerichtet und vollständig synchron.

Ereignis, auf das gewartet wird, nicht eintritt. Im Rahmen des ECSs handelt es sich bei Ereignissen im allgemeinen um das Eintreffen von Nachrichten. Damit das Ausbleiben eines erwarteten Ereignisses nicht zum Blockieren eines Programms führt, wird die Wartezeit auf einen maximalen Wert, den sogenannten Timeout, begrenzt. Läuft der Timeout ab, so wird der wartende Prozeß aus dem Wartezustand befreit.
Bei asynchroner Kommunikation tritt beim Senden keine Wartezeit auf. Beim Sendevorgang ist allerdings ein Timeout für die Gültigkeitsdauer einer abgesandten Nachricht (time to live) vorzusehen.

I/O-Konformität: Im Rahmen des ECSs werden Programme sowohl durch Nachrichten als auch durch anderweitige Ein- und Ausgabeforderungen (I/O: Input/Output) gesteuert. So müssen beispielsweise die Benutzerschnittstellen-Programme [stef93] sowohl auf Eingaben des Experimentators über eine Tastatur oder eine Maus als auch auf Nachrichten von den Gerätetreibern reagieren (siehe Abb.3.7). Ebenso wird z.B. der Programmablauf von Treibern, die Geräte direkt über eine serielle Schnittstelle ansprechen, einerseits durch Daten von seiten des Geräts und andererseits von Steuerkommandos übergeordneter Treiber bestimmt (siehe Abb.3.7).
Zur Reaktion auf Eingabeforderungen, muß ein Programm zunächst herausfinden, auf welchem der möglichen Kanäle überhaupt Daten (Nachrichten, Tastatureingaben, etc.) anliegen. Dazu könnte es auf jedem Kanal der Reihe nach versuchen, I/O-Operationen auszuführen, wobei es allerdings ununterbrochen laufen müßte.
Umgekehrt stellen moderne Betriebssysteme Systemaufrufe zum sogenannten *I/O-Multiplexing* zur Verfügung [lef89], bei denen ein Prozeß anfragen kann, welcher Kanal bereit ist, I/O-Operationen auszuführen. Falls dies bei keinem der angeforderten Kanäle der Fall ist, kann das System den Prozeß in den Ruhezustand versetzen, bis ein Kanal bereit ist. Hierzu müssen die Datenkanäle allerdings im Prozeß in Form von *(I/O)-Deskriptoren*[3] vorliegen.
Deshalb ist es notwendig, die Ankunft neuer Nachrichten (auf der niedrigsten Programmebene) dem übrigen I/O gleichzustellen und über einen I/O-Deskriptor anzuzeigen. Diese Eigenschaft wird hier als **I/O-Konformität** bezeichnet.

4.3 Das Implementationsprinzip

Um das Anforderungsprofil in möglichst portabler Weise umzusetzen, wurde auf die Realisierung des Systems als Treiber, also als Teil des Betriebssystems, verzichtet. Vielmehr sollten nur Standardbetriebsmittel verwendet werden, die auf allen Betriebssystemen vorhanden sind.
Zur Realisierung des Datentransports innerhalb eines Rechners stellen die verschiedenen Betriebssysteme im wesentlichen die beiden Mechanismen *Named Pipes* und *Shared Memory* zur Verfügung. Eine *Pipe*[4] ist eine IPC-Einrichtung, die den Datentransport zwischen zwei Prozessen in einer Richtung bewerkstelligt

[3] Deskriptoren sind ganze Zahlen, die auf Datenkanäle verweisen.

[4] Named Pipes sind ursprünglich für das UNIX Betriebssystem entwickelt worden, einen äquivalenten Mechanismus gibt es mit der Mailbox unter VMS.

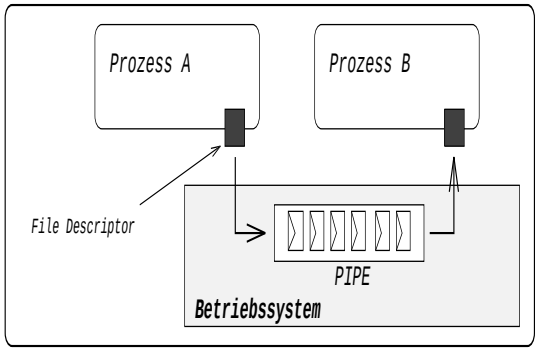


Abb.4.1: Kommunikation über Pipes. Der Zugriff auf eine Pipe gleicht dem Zugriff auf eine normale Datei, da I/O-Deskriptoren verwendet werden.

(Abb.4.1). Dieser Datentransport erfolgt indirekt über das Betriebssystem und ist deshalb im Vergleich zur Verwendung des Shared Memory, bei dem Sender und Empfänger Daten über ein gemeinsam benutztes Speichersegment austauschen, wesentlich langsamer[ste92]. Alternativ kann der Datentransport innerhalb eines Rechners auch durch die Netzwerksoftware geleistet werden, die ohnehin die Kommunikation zwischen zwei Netzwerkknoten realisiert.

Basierend auf diesen Mechanismen wurden zwei unterschiedliche Modelle zur Realisierung von MUPIX entwickelt: das *Mailbox-Modell* und das *Objektbus-Modell*.

4.3.1 Das Mailbox-Modell

Bei dieser Architektur wird zwischen lokaler und nicht-lokaler Kommunikation unterschieden. Dies erlaubt zum einen die Verwendung von Shared Memory, als dem im allgemeinen auf allen Betriebssystemen schnellsten IPC-Mechanismus, für den lokalen Datenaustausch innerhalb eines Rechners. Dies war insbesondere im Hinblick auf die damals schlechte Einbindung der dem Institut für Kernphysik zur Verfügung stehenden Netzwerksoftware ins Betriebsystem VMS von Bedeutung. Zum anderen bietet sich auch eine Implementation des Gesamtsystems in zwei Schritten an: Im ersten Schritt wird das lokale Messagesystem erstellt, im zweiten Schritt werden die Einzelsysteme in sehr transparenter Weise zu einem netzwerkfähigen Gesamtsystem zusammengefügt.

Zur Kommunikation innerhalb des Gesamtsystems bedarf es auf jedem an der Kommunikation beteiligten Rechner eines zentralen Prozesses, des Mailbox-Managers. Dieser ist für die Lokalisierung der Empfänger, die Nachrichtenverschiebung und ggf. für die Pufferung der Nachrichten zuständig. Die Verbindung zwischen den Mailbox-Managern auf entfernten Knoten wird über Netzwerkinterface-Prozesse als Schnittstelle zum Netzwerksystem hergestellt. Der eigentliche Datentransfer läuft beim Mailbox-Modell über einen von den am System teilnehmenden Prozessen und dem Manager gemeinsam benutzten Speicherbereich ab. Zur Synchronisation der Zugriffe auf diesen Speicherbereich kommunizieren die Prozesse mit dem Manager über Pipes[5].
Da der Nachrichtentransport *indirekt* über den vom Mailbox-Manager verwalteten Speicherbereich erfolgt, läßt sich in einfacher Weise das Prinzip der *Mehrfachkommunikation*, der sogenannte (*Multicast*), realisieren, bei dem eine Nachricht

[5]Semaphoren können nicht zur Synchronisation verwendet werden, da sie nicht äquivalent zu anderen I/O-Kanälen, also nicht in Form von Deskriptoren behandelt werden. Damit würde ihre Verwendung nicht die Forderung nach I/O-Konformität erfüllen.

von einem Prozeß gleichzeitig an mehrere Empfänger versandt wird. Dafür dienen bei der symbolischen Adressierung Multicast-Adressen, die sogenannte *Wildcards* enthalten, also Zeichen, die einen oder mehrere Buchstaben repräsentieren. Durch Verwendung von Multicasts läst sich ein erhöhtes Maß an Parallelität erreichen, z.B. kann man bei der Initialisierung des Systems mehreren Gerätetreibern gleichzeitig die erforderlichen Kommandos schicken.
Bei der bisherigen Überlegung wurde davon ausgegangen, daß der jeweilige Adressat bereits existiert. Diese Forderung kann man bei dem sogenannten *Mailbox-Sendemodus* fallen lassen: Hierbei wird das Shared Memory als eine Art "Postfach" benutzt, in dem die Nachricht so lange liegen bleibt, bis sie vom Adressaten abgeholt wird.

Auf dieser Basis wurde ein Prototyp des Messagesystems aufgebaut[6], wozu die erforderlichen Programme (Mailbox-Manager, Netzwerkanbindung) und eine Bibliothek zur Einbindung in die Anwenderprogramme entwickelt und implementiert wurden. Dieser Prototyp fand bereits bei der Steuerung von Laboraufbauten der Drei-Spektrometer-Anlage Verwendung. Dabei traten allerdings eine Reihe von Nachteilen dieser Architektur zu Tage:

- Nachrichten an nicht lokale Empfänger müssen mehrfach bearbeitet werden, nämlich jeweils von den lokalen Mailbox-Managern und den Netzwerkinterface Prozessen. Dadurch verlängern sich bei verteilten Aufgaben wie dem ECS die Antwortzeiten.
- Die Realisierung des Modells ist komplex, weil
  - das System bereits aus mehreren Prozessen besteht,
  - für den Datenabgleich der Mailbox-Manager auf den verschiedenen Prozessoren zu sorgen ist und
  - es beim Datentransport zwischen den beteiligten Knoten zum "Routing" kommt, weil der Transportweg einer Nachricht festgelegt werden muß.
  - Ferner stellte sich heraus, daß sich die Systemschnittstellen für die IPC auf den beteiligten Betriebssystemen teilweise erheblich unterschieden haben, wodurch die Portierbarkeit der Software verschlechtert und der Programmieraufwand erhöht wird.
- Da der Mailbox-Manager auf Anwenderebene läuft, hat er keinerlei direkte Information darüber, ob ein am Messagesystem teilnehmender Prozeß noch arbeitet oder bereits terminiert ist. Nicht ordentlich aus dem System ausgeschiedene Prozesse beanspruchen jedoch weiterhin Ressourcen des Messagesystems, wodurch die Gesamtstabilität des Systems beeinträchtigt wird.

Auf Grund dieser Probleme wurde für die endgültige Implementation von MUPIX eine andere Architektur gewählt, nämlich das Objektbus-Modell.

4.3.2 Das Objektbus-Modell von MUPIX

Um die Verzahnung mit dem Betriebssystem so gering wie möglich zu halten, die Integration des Netzwerks jedoch zu maximieren, bietet es sich an, auch lokal

[6] Diese Voruntersuchungen sind bereits in einem internen Bericht [kraM90] beschrieben.

das Netzwerksubsystem für den Datentransport zu verwenden. Deshalb bedarf es bei diesem Konzept keines zentralen Managerprozesses für die Verteilung der Nachrichten auf die einzelnen Prozesse; die Kommunikation erfolgt vielmehr *direkt* von Prozeß zu Prozeß, womit die oben genannten Nachteile entfallen.

Zum Datentransport kamen dabei nur die beiden Internetprotokolle TCP/IP[7] und UDP/IP[8] aus der Internet-Protokollfamilie ([com88], [ste90]) in Frage, die sich mittlerweile zum Industriestandard entwickelt haben. Neben ihrer Verbreitung haben diese Protokolle den Vorteil, daß sich für ihre Anwendungsschnittstelle eine Standardbibliothek etabliert hat, die sogenannte *4.3BSD-Network Library (Socket-Library)*. Die darin enthaltenen Funktionen, die ursprünglich im Rahmen der Entwicklung des 4.3BSD-UNIX [lef89] entstanden, sind mittlerweile auf nahezu allen Betriebssystemen verfügbar. Dadurch ist eine gute Portabilität gewährleistet.
Wie fast alle fehlergesicherten (zuverlässigen) Protokolle ermöglicht TCP/IP nur eine gerichtete Kommunikation, während das paketorientierte UDP/IP zwar den ungerichteten Transport von Datenpaketen (*Datagramme*) von Prozeß zu Prozeß leistet, aber dem Sender keine Information darüber liefert, ob die Datagramme überhaupt angekommen sind.

Um die allseitige Kommunikation zu gewährleisten, baut MUPIX auf dem ungerichteten UDP/IP-Protokoll auf, über das aber zur Fehlersicherung ein eigenes Protokoll, das MUPIX-Protokoll, gesetzt werden muß.

Ein Prozeß kann nicht zu jedem Zeitpunkt bereit sein, Datagramme auf dem UDP/IP-Kanal (UDP/IP-Socket) entgegenzunehmen. Deshalb ist es erforderlich, den Sender vom Zustand des Empfängers zu entkoppeln; dafür ist die Protokollarbeit asynchron zu verrichten (Abb.4.2). Durch den asynchronen Datenempfang sind die Prozesse quasi immer empfangsbereit: kommt eine Nachricht auf dem unterliegenden UDP-Kanal an, so signalisiert das Betriebssystem dieses Ereignis dem Prozeß, der daraufhin seine "laufende Arbeit" unterbricht, um die Daten entgegenzunehmen; danach setzt der Prozeß seine Arbeit an der Stelle fort, wo er unterbrochen wurde. Damit wird die geforderte No-wait-send Entkoppelung bis auf die auf Seite 29 erläuterten Einschränkungen realisiert.

Jedem UDP/IP-Kanal wird im Rahmen des UDP/IP-Protokolls auf einem Rechner eine Nummer, die sogenannte *Port-Nummer* zugeordnet, die zusammen mit der Rechnernummer eine eindeutige, im UDP/IP-Protokoll verwendete Adresse bildet. Da aber die Adressierung im Rahmen des Message-Systems auf der Basis von symbolischen Namen erfolgen soll, benötigt man einen Mechanismus zur Abbildung von symbolischen Namen auf UDP/IP-Protokolladressen; dieser Mechanismus wird als *Adreßauflösung* bezeichnet. Der einfachste Mechanismus besteht darin, die Adressen zusammen mit den zugeordneten Namen in einer Datei abzulegen und diese auf allen am Experiment beteiligten Rechner verfügbar zu machen. Dieses Verfahren hat allerdings einige wesentliche Nachteile:

- Bei jedem neu hinzukommenden Namen oder bei der Verschiebung von Prozessen auf andere Rechner sind die entsprechenden Dateien zu ändern. Dies

[7] TCP:Transmission Control Protocol, IP:Internet Protocol
[8] UDP:User Datagram Protocol

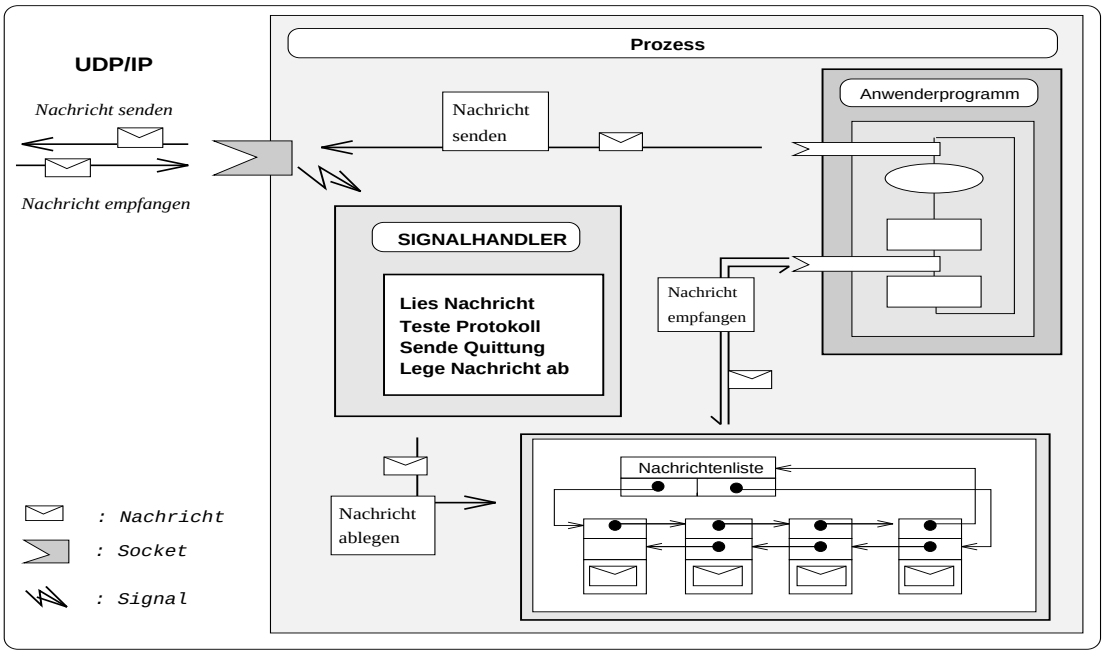


Abbildung 4.2: Während das Senden von Datagrammen durch den Aufruf einer Sende-Routine synchron zum Anwenderprogramm abläuft, erfolgt der Empfang von Datagrammen vollständig asynchron von diesem: Bei der Ankunft eines Datagramms auf dem UDP-Kanal (Socket) wird vom Betriebssystem ein Software-Interupt (Signal) ausgelöst, der im Prozeß eine Routine (Signalhandler) anstößt, die (quasi) parallel zum Anwenderprogramm abgearbeitet wird. Im Signalhandler wird das Datenpaket ausgelesen und die Protokollarbeit verrichtet. Die neue Nachricht wird ans Ende einer linearen, doppelt verketteten *Liste* - der *Nachrichtenliste* - gehängt. Jedes Listenelement enthält eine Nachricht und die Speicheradresse, die auf das vorhergehende bzw. nachfolgende Listenelement verweist[a]. Die Anwendung selbst ist auch in dieser Einbindung ein sequentiell arbeitendes Programm, das über die MUPIX-Empfangsroutine auf die Nachrichtenliste zugreift. Da Zugriffe auf diese Liste sowohl von seiten des Signalhandlers als auch aus dem Anwenderprogramm heraus möglich sind, muß für eine geeignete Synchronisation gesorgt werden (s.S. 31). Die Synchronisations- und Protokollarbeit wird unsichtbar für das Anwenderprogramm durch Routinen der MUPIX-Bibliothek geleistet.

[a] Man bezeichnet Variable, deren Wert eine Speicheradresse ist, als *Zeiger* [wi83] .

erhöht nicht nur den Verwaltungsaufwand, sondern trägt auch zur Fehleranfälligkeit des Systems bei.

- Bei der statischen Zuordnung der Protokolladressen zu den Objektnamen ist es nicht möglich, schon bei der Namensauflösung zu erkennen, ob tatsächlich ein Prozeß existiert, der bereit ist, Nachrichten unter diesem Namen in Empfang zu nehmen.
- Eine technische Komplikation bei der statischen Zuordnung ergibt sich daraus, daß sichergestellt werden muß, daß keine in der Datei eingetragene Portnummer schon anderweitig vom Betriebssystem vergeben wurde.

Aus diesen Gründen ist es wesentlich günstiger, die Adreßverwaltung dynamisch zu gestalten. Dies wurde dadurch erreicht, daß sich die Prozesse, die am System teilnehmen wollen, bei einem für die Adreßauflösung verantwortlichen Prozeß,

dem sogenannten *Nameserver*, mit ihrem Namen und ihrer Adresse *anmelden*. Die symbolischen Namen werden, falls notwendig, beim Senden transparent für das Anwenderprogramm mit Hilfe des Nameservers in Adressen umgesetzt. Die ebenfalls im Rahmen der vorliegenden Arbeit geleistete Realisierung des Nameservers und dessen wesentliche Eigenschaften werden in Abschnitt 4.4.6 diskutiert.

MUPIX stellt mit diesem Konzept den Datenpfad, gewissermaßen einen "Bus", zur Verfügung, über den die auf verschiedene Prozesse und Prozessoren verteilten Objekte des ECSs in standardisierter Weise kommunizieren. Da die den "Bus" nutzenden Objekte keine gemeinsamen Ressourcen teilen, kann eine Fehlfunktion eines dieser Objekte keine Auswirkungen auf die anderen haben. Jedes Objekt kann zu jeder Zeit diesen Software-"Bus" nutzen, bei dem im Gegensatz zu einem Hardware-Bus keine Notwendigkeit zur Synchronisation der Buszugriffe besteht. Der Begriff Bus ist deshalb auch *cum grano salis* zu sehen.

Insgesamt ist das Objektbus-Modell konzeptionell wesentlich einfacher als die im vorherigen Abschnitt dargestellte Mailbox-Architektur, da bis auf den Nameserver keine zusätzlichen Prozesse im Spiel sind. Diese Vereinfachung gewinnt man im wesentlichen dadurch, daß der komplette Datentransport im Gegensatz zur Mailbox-Architektur von Betriebssystemteilen (Netzwerkkomponenten) geleistet wird. Der Übergang auf diese Architektur wurde dadurch ermöglicht, daß sich die Integration der Internet-Netzwerksoftware in das Betriebssystem VMS wesentlich verbesserte (siehe Abb.4.3).

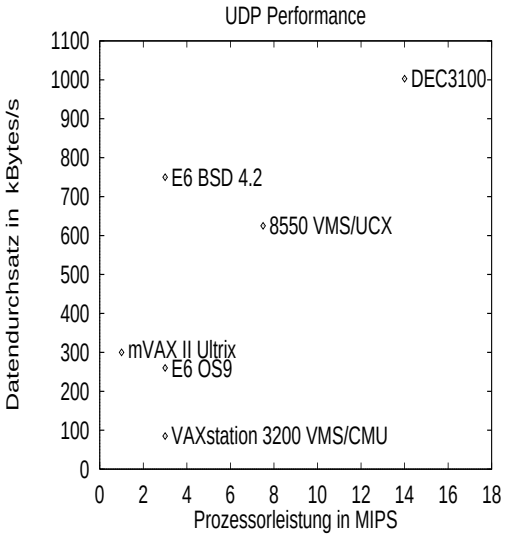


Abbildung 4.3: Datendurchsatz von UDP/IP auf verschiedenen Betriebssystemen und Rechnerarchitekturen. Um den Einfluß des Empfängers auf die Messung des Durchsatzes zu minimieren, diente die zum damaligen Zeitpunkt schnellste Maschine des Instituts, eine DECstation 5000/200 mit 21 MIPS, als Empfangsrechner. Es wird deutlich, daß die ursprünglich unter VMS zur Verfügung stehende Internetsoftware bei gleicher Prozessorleistung einen wesentlich schlechtere Durchsatz aufweist als alle anderen Implementierungen. Diese Situation wird durch die Einführung der UCX-Software wesentlich verbessert.

4.4 Die Realisierung von MUPIX

4.4.1 Die Anwendungsschnittstelle

Bevor auf die Realisierung des Systems im einzelnen eingegangen wird, skizziert dieser Abschnitt die Schnittstelle zum MUPIX-System, ohne jedoch auf alle pro-

grammiertechnischen Details, die sich ich in der gesonderten Softwaredokumentation [kra92a, kra93a] befinden, einzugehen. Alle Schnittstellenfunktionen sind in einer Bibliothek – der MUPIX-Bibliothek – zusammengefaßt. Der Ablauf der Kommunikation zwischen einem Klienten- und einem Serverprozeß unter Verwendung der Bibliotheksfunktionen wird in Abb.4.4 dargestellt, der dazu gehörige Programmtext eines Anwenderprogramms wird in Abb.B.1 gezeigt. Sowohl

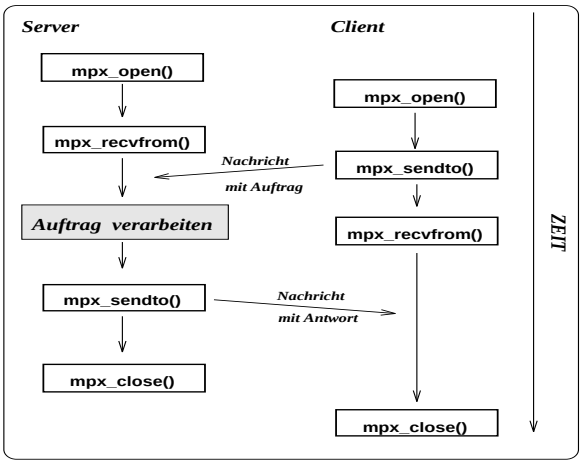


Abb.4.4: Verlauf einer Client/Server-Kommunikation mit Hilfe der MUPIX-Bibliotheksfunktionen. Alle MUPIX-Funktionen sind durch Voranstellung des Präfix `mpx_` gekennzeichnet.

auf Server- als auch auf Klientenseite wird durch das Anwendungsprogramm zunächst die MUPIX-Bibliothek initialisiert (`mpx_open()`). Dabei wird das zur Kommunikation benötigte UDP/IP-Socket (Abb.4.2) geöffnet und für die (asynchrone) Datenauslese bereitgemacht. Beim Initialisieren meldet der Server auch die symbolischen Namen an, unter denen er für diese Kommunikation ansprechbar ist. Der Server wartet dann auf einlaufende Nachrichten (`mpx_recvfrom()`), bearbeitet sie nach ihrem Eintreffen und schickt eine Antwort an den Absender (`mpx_sendto()`) zurück. Der Klient benötigt für die Kommunikation mit dem Server keinen eigenen symbolischen Namen, vielmehr verwendet der Server beim Senden der Antwort die Protokolladresse des Klienten, die als Absenderadresse mit dem Auftrag mitgeliefert wurde. MUPIX unterstützt also sowohl die Adressierung mit symbolischen als auch mit Protokolladressen. Beide Arten werden in dem in Abb.B.1 gezeigten Beispiel verwendet.
Zwei weitere Eigenschaften der Empfangsroutine (`mpx_recvfrom`) werden am Beispiel des Klienten verdeutlicht:

1. Es besteht dem Anforderungsprofil entsprechend die Möglichkeit, nur eine maximale Zeit (Timeout) auf das Eintreffen einer Nachricht zu warten.

2. Da der Klient nach dem Absenden einer Anfrage auf eine Antwort des Servers erwartet, gibt er bei der Empfangsroutine gewissermaßen als Filter dessen Adresse mit an, so daß dem Klient nur Nachrichten des Servers zugestellt werden. Sonstige Nachrichten werden für die spätere Bearbeitung in der Nachrichtenliste abgelegt. Auf diese Weise stellt MUPIX einen einfachen Mechanismus zur Realisierung der RPC-Semantik zur Verfügung.

4.4.2 Das MUPIX-Protokoll

In diesem Abschnitt werden die Mechanismen des MUPIX-Protokolls beschrieben, die zur Realisierung der At-most-once-Zuverlässigkeit führen.
Die Notwendigkeit, zusätzlich für Zuverlässigkeit zu sorgen, ergibt sich aus den Eigenschaften des UDP-Protokolls, die dazu führen können, daß UDP-Datagramme verloren gehen, bzw. verdoppelt oder außer der Reihe angeliefert werden. Die letzte Möglichkeit kommt innerhalb eines LANs nicht vor, da sie auf das Routing der Datagramme zurückzuführen ist. Der Verlust von Datagrammen, hervorgerufen durch technische Störungen oder Überlastung von Netzwerkkapazitäten, muß jedoch in Betracht gezogen werden. Die Möglichkeit, daß Datagramme zwar bei dem Empfänger angeliefert werden, die Daten aber nicht mehr intakt sind, kann durch Ausnutzen einer im UDP definierten Prüfsumme ausgeschlossen werden: Das Betriebssystem testet die Prüfsumme noch bevor das Datagramm das Anwendungsprogramm erreicht; falls dieser Test fehlschlägt, wird das Datagramm verworfen. Auf Grund dieser Eigenschaft kann man bei der Entwicklung des MUPIX-Protokolls davon ausgehen, daß ein Prozeß nur intakte Datagramme vom System angeboten bekommt.

Zuverlässigkeit

Um auf der Basis von UDP/IP einen zuverlässigen Datagrammdienst zu realisieren, antwortet der Empfänger auf eine Nachricht mit einer Quittung. Das MUPIX-Protokoll definiert zwei Quittungstypen - einen positiven (ACK) und einen negativen (CCL):

ACK (acknowledge): Die Nachricht wurde vom Empfängerprozeß empfangen und in dem von ihm bereitgestellten Empfangspuffer abgelegt.

CCL (congestion control): Wegen Pufferplatzmangels beim Empfänger konnte die Nachricht nicht entgegengenommen werden (siehe S.31). Das Einführen von negativen Quittungen verkürzt die Wartezeit des Sendeprozesses.

Um Blockierungen zu vermeiden, wird das Warten auf eine Quittung vom Sender zeitlich überwacht. Falls der Empfänger nach einer bestimmten Zeitspanne (*Timeout*) noch nicht geantwortet hat (keine Quittung), wird die Nachricht erneut versandt (*Retransmission*). Dieser Vorgang wird solange wiederholt, bis eine Quittung eintrifft oder eine vorgegebene maximale Zahl von Wiederholungen überschritten wird. In diesem Fall wird der Sendevorgang mit einer Fehlermeldung abgebrochen. Ist der Timeout ein konstanter Wert, so spricht man von einem *linearen Retransmit-Timer.*
Im allgemeinen kann die Zeit (*round-trip time - RTT*), die ein Paket benötigt, um vom Sender zum Empfänger und wieder zurückzulaufen, zwischen Bruchteilen einer Sekunde und mehreren Sekunden schwanken. Deshalb wurden adaptive Algorithmen zur Bestimmung des Timeouts aus der mittleren, tatsächlich gemessenen RTT bei einer virtuellen Verbindung entwickelt ([com88], [ste90]), mit denen eine automatische Anpassung des Timeouts an die tatsächlichen Verhältnisse vorgenommen werden kann. Dabei muß allerdings die Systemuhr vor und nach Abschluß des Sendevorgangs ausgelesen werden. Diese Operation ist recht aufwendig, da es sich bei diesem Auslesen um einen Systemaufruf handelt. Die

Werte für die RTT weisen innerhalb eines kleineren LANs keine großen Schwankungen auf. Auf UDP/IP-Ebene wurden Zeiten kleiner als 20 ms für den Verkehr zwischen einem E6 und einer DECstation gemessen, die bei manchen Rechnern (z.B. dem E6) auf Anwenderebene gar nicht aufgelöst werden können. Daher ist der Aufwand für einen adaptiven Algorithmus bei der vorliegenden Anwendung nicht gerechtfertigt. Statt dessen wird ein einfacher Algorithmus verwendet[9], bei dem der Timeout nicht gemessen wird, sondern fest vorgegeben ist. Bei jeder Wiederholung (Retransmission) wird dieser Wert um einen Faktor zwei erhöht. Auf diese Weise lassen sich Probleme umgehen, die durch momentane Überbelastung entstehen.
Für die Zeit t_{n_r}, die ein Sendevorgang bei n_r-maligem Fehler benötigt, ergibt sich

$$t_{n_r} = t_0 \sum_{k=0}^{n_r} 2^k = (2^{n_r+1} - 1)t_0;$$

solange blockiert dann eine Anwendung im Fehlerfall. Als Timeout t_0 wurde 1 Sekunde gewählt.
In diesem Sinn wurde die "No-wait-send"-Semantik im Rahmen dieser Arbeit nicht vollständig realisiert. Dies hat sich jedoch in der Praxis als nicht problematisch erwiesen, da Fehler nur sehr selten eintreten. Außerdem erspart man sich beim Senden das sonst notwendige Umkopieren der Daten. Der Code wurde jedoch so weit vorbereitet, daß im Rahmen einer Diplomarbeit[ha93] neben dem synchronen Senden jetzt ein zusätzlicher vollständig asynchroner Sendemodus zur Verfügung gestellt werden konnte.

Zur Ausfilterung von Duplikaten, die z.B. durch verloren gegangene Quittungen entstehen, enthält jede Nachricht eine Sequenznummer, die Empfänger-spezifisch geführt wird. Auf diese Weise unterhält ein Prozeß mehrere virtuelle Verbindungen über einen einzigen UDP/IP-Kanal. Mit Hilfe eines Synchronisationsbits (SYN) im Nachrichtenkopf werden diese Verbindungen mit der ersten ausgetauschten Nachricht geöffnet. Dadurch entsteht kein zusätzlicher Aufwand für den Verbindungsaufbau. Jede Verbindung wird innerhalb der MUPIX-Bibliothek mit Hilfe einer besonderen Struktur, dem Protokoll-Control-Block (PCB), verwaltet.
Für die Zuordnung der von MUPIX aufgebauten virtuellen Verbindungen zwischen den einzelnen Kommunikationspartnern benötigt man einen eindeutigen Schlüssel. Als Schlüssel bieten sich die UDP/IP-Adressen selbst an, da sie eindeutig UDP/IP-Kommunikationsendpunkten zugeordnet sind. Mit der Realisierung der virtuellen Verbindung mit Hilfe von Quittungen und einer Sequenznummer erfüllt MUPIX die Forderung nach einem *at-most-once* zuverlässigen Protokoll.

MUPIX-Nachrichtenlänge

MUPIX-Nachrichten sind von begrenzter, jedoch variabler Länge. Die maximale Länge einer Nachricht beträgt 1024 Bytes, wobei 68 Bytes auf den Nachrichtenkopf entfallen (siehe Abb. 4.5). Diese Einschränkung hat sich aus folgenden Gründen ergeben:

[9] Bei der Implementation der Software wurde jedoch darauf geachtet, daß es jederzeit möglich ist, auf einen adaptiven Algorithmus umzustellen. Tatsächlich wurde das System zunächst mit einem von Karn[ka87] entwickelten adaptiven Algorithmus implementiert. Dieser wurde dann durch den im Aufruf kompatiblen einfachen Mechanismus ersetzt.

- Steuernachrichten, die im Rahmen des ECS zwischen Treiberprogrammen ausgetauscht werden, sind kurz. Deren Länge wurde auf maximal etwa 500 Bytes abgeschätzt [ku95].
- Eine MUPIX-Nachricht dieser Länge paßt in ein einziges UDP-Datagramm. Dadurch wird die notwendige Protokollarbeit stark vereinfacht, da es keinerlei Pufferung für die sonst entstehende Fragmentierung der Nachrichten bedarf. Theoretisch ist die Länge eines UDP-Datagramms auf die maximal erlaubte Datenlänge des IP-Protokolls beschränkt, das für den Knoten-zu-Knoten Transport sorgt, d.h. auf 65.516 Bytes. Praktisch ergeben sich allerdings einige Einschränkungen:
  - Der Pufferbereich (*Socketpuffer*[10]), den das Betriebssystem einzelnen Prozessen zum Zwischenspeichern von UDP-Datagrammen zuteilt, ist begrenzt. Bei den bisher getesteten Systemen wurden Socketpuffer bis herab zu 4096 Bytes festgestellt.
  - Eine weitere Einschränkung ergibt sich aus der maximalen Menge an Daten (*MTU* Maximum Transfer Unit), die man in einem Paket über ein gegebenes physikalisches Netzwerk transferieren kann.
    In einem LAN ist die MTU durch den Netzwerkstandard definiert. Beim Ethernet beträgt sie 1526 Bytes. Überschreitet man diese Größe, so kommt es zur Fragmentierung der IP-Nachrichten und damit zu geringeren Übertragungsgeschwindigkeiten [cab88].
- Pufferung spielt nicht nur auf der Ebene des Betriebssystems eine Rolle, sondern auch bei der Abarbeitung der Nachrichten durch die Anwendung auf der Ebene der MUPIX-Bibliothek. Da das Auslesen der UDP-Datagramme asynchron zum sonstigen Programmablauf erfolgt, müssen die Nachrichten, wie bereits beschrieben, in einer Liste zwischengespeichert werden. Hierzu wird Speicherplatz benötigt. Dieser kann jedoch nicht einfach dynamisch angefordert (allokiert) werden, da die dafür vom Betriebssystem vorgesehenen Routinen nicht für den gleichzeitigen Aufruf aus synchronen und asynchronen Programmteilen vorbereitet sind. Deshalb muß der benötigte Speicher, dessen Größe sich aus dem Produkt von maximaler Nachrichtenlänge und maximaler Anzahl gepufferter Nachrichten ergibt, im vorhinein bereitgestellt werden. Die Nachrichtenlänge sollte zur Minimierung des bereitzustellenden Pufferspeichers möglichst kurz gehalten werden.

Aus diesen Gründen wurde die maximale Länge einer MUPIX-Nachricht auf 1024 Bytes festgelegt, was im Hinblick auf unsere Anwendung ausreicht. Damit können bereits in dem minimalen Socketpuffer vier Nachrichten Platz gespeichert werden und der Speicherplatzverbrauch für eine Pufferlänge von 20 Nachrichten ist mit 20kB sehr moderat. Bild 4.5 zeigt die Kapselung der einzelnen Pakete und den MUPIX-Protokollkopf.

Der Protokollkopf enthält neben den Informationen zur Art der Nachricht und dem Zustand der Verbindung auch die (MUPIX-)Protokolladresse von Sender- und Empfänger. Eine (MUPIX-)Protokolladresse besteht aus der Internetknoten-, Port- und Unitnummer. Die ersten beiden Zahlen bilden die UDP/IP-Adresse

[10] In diesem Bereich werden UDP-Nachrichten abgelegt, die noch nicht vom Empfänger abgeholt wurden. Ist dieser Platz verbraucht, werden alle neu ankommenden Datagramme verworfen.

eines UDP/IP-Kanals. Die Unit-Nummer repräsentiert einen Aliasnamen innerhalb eines Prozesses. Diese Zahl ist nur im Kontext eines Prozesses eindeutig und wird innerhalb der MUPIX-Bibliothek aus Effizienzgründen an Stelle eines symbolischen Namens verwendet.

4.4.3 Signale

Kritische Abschnitte

Durch die asynchrone Verarbeitung von Ereignissen ist der gesamte Programmablauf nicht mehr streng sequentiell: Beim Eintreffen eines Signals wird der normale Programmablauf angehalten und gesichert, bevor mit der Ausführung der Signalroutine begonnen wird; nach deren Ablauf setzt das Programm seine Arbeit dort fort, wo sie unterbrochen wurde. Dieser Aufgabenwechsel, mit dem ein Prozeß quasi gleichzeitig in zwei Programmteilen aktiv ist, wird vom Betriebssystem geleistet. Dabei kann es zu Konflikten kommen, wenn aus beiden Teilen quasi gleichzeitig auf gemeinsam benutzte Ressourcen zugegriffen wird.
Ein Beispiel hierfür ist die Nachrichtenliste (siehe Abb.4.2): Enthält der eine Programmteil Instruktionen (*kritischer Abschnitt, critical section* [tan87]), deren Ergebnis vom Wert von Variablen abhängt, die vom parallel laufenden Programmteil während seines asynchronen Zugriffs geändert werden können, dann wird das Ergebniss der Instruktionen unkontrolliert. Dieses Problem läßt sich dadurch lösen, daß für die Dauer eines kritischen Abschnitts auf solche Variable nur exklusiv zugegriffen wird. Dies wird dadurch erreicht, daß für die Dauer des Zugriffs die Zustellung von Signalen an den Prozeß unterbunden wird[11]. In diesem Zusammenhang spricht man vom Ausmaskieren von Signalen.
Diesen Schutzmechanismus kann man jedoch nur für die MUPIX-internen Ressourcen verwenden, da die komplette Signalbearbeitung vor dem Anwender versteckt bleiben soll. Die wichtigste sich hieraus ergebende Einschränkung ist der Verzicht auf das dynamische Allokieren und Freigeben von Speicher mit Hilfe der Standardfunktionen (`malloc()` und `free()`) in der Signalroutine. Deshalb muß der asynchron benötigte Speicher bereits in der Initialisierungsphase allokiert und entsprechend verwaltet werden.
Aus diesem Grund ist die Anzahl der möglichen virtuellen Verbindungen und speicherbaren Nachrichten beschränkt. Als Konsequenz ergibt sich die Notwendigkeit einer Flußkontrolle im MUPIX-Protokoll.

Alternativ könnte man auch die Bibliotheksroutinen zur Speicherverwaltung unseren Erfordernissen anpassen, indem man sie mit den entsprechenden Systemaufrufen zum Blockieren und Freigeben von Signalen versieht. Diese Vorgehensweise würde bei vielen Programmen jedoch die Laufzeit beträchtlich erhöhen, da die Speicherverwaltung sehr oft aufgerufen wird und Systemaufrufe verhältnismäßig lange dauern (Tab.4.2 auf S.42).

[11] Die Signalroutine selbst kann nicht durch ein weiteres Signal unterbrochen werden Diese Eigenschaft gilt nur für modernere Betriebssysteme mit sogenannten "Sicheren Signalen". Hierzu gehören jedoch mittlerweile alle im Institut eingesetzten Systeme.

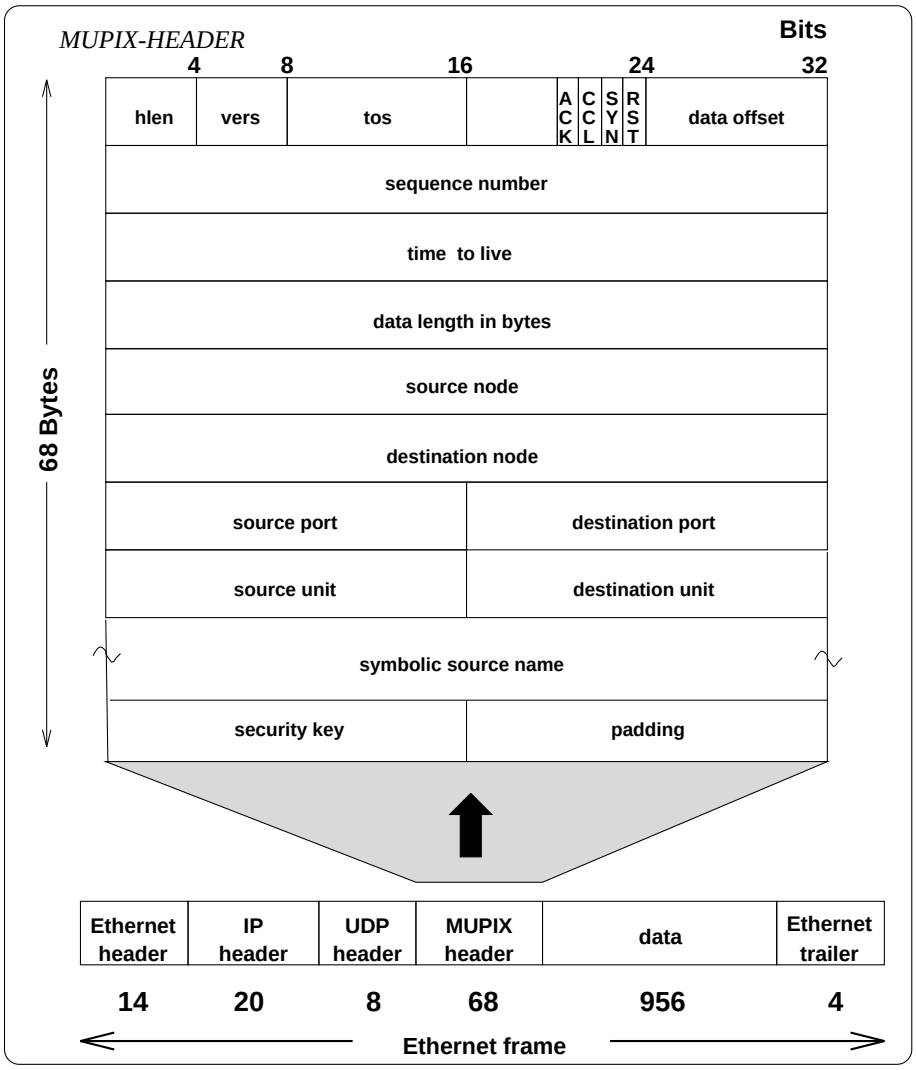


Abb.4.5: Kapselung einer Nachricht innerhalb eines Ethernetpakets und der MUPIX-Protokollkopf. Ein MUPIX-Paket, das aus einem Protokollkopf (Header) von 68 Bytes und der eigentlichen Nachricht (data) von max. 956 Bytes besteht, ist in ein UDP/IP-Datagramm eingebettet. Das gesamte Paket paßt in ein Ethernetpaket (ethernet frame). Die ersten 32 Bits des MUPIX-Protokollkopfs enthalten Informationen über die Länge (hlen) des Protokollkopfs (in Einheiten von 4 Bytes), die Version (vers) der Software, den Typ der Nachricht (tos: type of service), Protokollbits (ack: acknoledge, ccl: congestion control, syn: sync, rst: reset) und eine Adresse die auf den Beginn der Daten (data offset) verweist. Dieser Information folgt die Sequenznummer, die Gültigkeitsdauer einer Nachricht (time to live) und die Länge der Daten (data length). Der Nachrichtenkopf enthält zusätzlich die Knoten-, Port- und Unitnummer von Sender (source) und Empfänger (destination). Dieser Information folgt der symbolische Name des Absenders (symbolic source name) mit maximal 32 Bytes. Darüber hinaus enthält die Nachricht einen Schlüssel zur Absicherung der Kommunikation (siehe Diskussion auf S.44). Um den Protokollkopf auf einer 4 Byte Grenze abzuschließen, wird er mit z.Z. sechszehn ungenutzten Bits aufgefüllt (padding).

Verhindern von Blockierungen

Ein weiteres Problem, das durch die Signalverarbeitung hervorgerufen wird, ist die Gefahr von Blockierungen. Eine solche tritt z.B. im Rahmen der Empfangsroutine auf, wenn das Programm auf das Eintreffen einer neuen Nachricht warten muß: Die Empfangsroutine stellt an Hand der Nachrichtenliste fest, ob bereits eine zu verarbeitende Nachricht vorliegt; falls dies nicht der Fall ist, muß gewartet werden, bis eine Nachricht eintrifft. Hierzu könnte man die Länge der Nachrichtenliste in einer Schleife abfragen, bis sich eine Nachricht in der Nachrichtenliste befindet (*busy-wait*). Dieses Verfahren würde allerdings unnötig Rechenzeit verbrauchen. Deshalb ist es günstiger, den Prozeß in den Ruhezustand (Pause) zu versetzen, aus dem er durch ein Signal befreit wird, das mit der Ankunft einer neuen Nachricht verbunden ist.
Die Instruktionen, die den Test auf die Länge der Nachrichtenliste enthalten und zum Erreichen des Pausezustands führen, bilden einen kritischen Abschnitt. Trifft eine Nachricht kurz nach dem Überprüfen der Liste, aber noch bevor sich der Pausezustand eingestellt hat, ein, so würde der Prozeß erst wieder durch das nächste Signal (u.U. niemals) aus dem Pausezustand befreit. Deshalb müssen auch in dieser Situation die Signale geeignet blockiert werden, der in Abb.4.6 dargestellte Pseudocode zeigt die Lösung des Problems mit Hilfe entsprechender Systemaufrufe.

```
//  Ausmaskieren des I/O-Signals SIGIO
mask = sigblock(sigmask(SIGIO));
FOREVER { // Endlosschleife
  if (messageQueue.len != 0) { // Die Liste ist nicht leer!
    len = getMessageFromQueue(buf); // Nachricht aus Liste holen.
    sigsetmask(mask); // alte Signalmaske wiederherstellen.
    return(len);
  }
  sigpause(0); // Auf das Eintreffen eines Signals warten.
}
```

Abb.4.6: Verriegelung eines kritischen Abschnitts mit Hilfe von Signalfunktionen innerhalb der MUPIX-Empfangsroutine. Das I/O-Signal wird für die Dauer des kritischen Abschnitts ausmaskiert, da auf die Variable `messageQueue.len` sowohl von seiten der Signalhandlers als auch von Anwenderseite aus zugegriffen wird. Das Signal wird erst wieder während des Wartens (`sigpause`) auf ein neues Signal freigegeben. Bei der Rückkehr aus diesem Systemaufruf, ist das Signal erneut ausmaskiert. Diese Vorgehensweise verhindert eine Blockierung, falls ein Signal erst nach dem Lesen der Variablen `messageQueue.len` zugestellt wird. Im Beispiel werden BSD-Systemaufrufe verwendet. Es läßt sich jedoch in analoger Weise mit dem neuen POSIX.1 Standard für die Signalbearbeitung realisieren [ste92].

4.4.4 I/O-Multiplexing und Nachrichten

Da neue Nachrichten über ein UDP/IP-Socket angeliefert werden, das vom Betriebssystem in gleicher Weise wie jeder normale I/O-Deskriptor behandelt wird, ist es möglich, I/O-Multiplexing (siehe S.21) zu betreiben. Damit ist eine der ursprünglichen Anforderungen an die Realisierung von MUPIX erfüllt. Diese Eigenschaft wird insbesondere bei der Implementation von graphischen Benutzerschnittstellen auf der Basis des X-Window-Systems benötigt [stef93]. Bei solchen Programmen werden alle Aktionen durch I/O-Ereignisse angestoßen; Tastatureingaben oder "Mausclicks" liegen bspw. in Form von Daten an verschiedenen Deskriptoren vor. Zum I/O-Multiplexing wird deshalb an einer zentralen Stelle im Programm die vom Betriebssystem bereitgestellte Funktion `select` aufgerufen[12].

Beim I/O-Multiplexing muß jedoch in Verbindung mit dem MUPIX-Kanal eine Besonderheit bedacht werden, nämlich die Möglichkeit, daß sich bereits Nachrichten in der Nachrichtenliste befinden, bevor mit Hilfe des `select` Systemaufrufs auf neue Nachrichten oder anderweitige Eingabeforderungen (I/O) gewartet wird. Deshalb ergibt sich eine ähnliche Kombination aus Testen und Warten wie im Beispiel 4.6, die mit der gleichen Problematik behaftet ist. Hier tritt jedoch die `select` Funktion an die Stelle von `sigpause`. Die Gefahr des Blockierens kann ebenfalls mit Hilfe geeigneter Systemaufrufe unter Berücksichtigung der speziellen Eigenschaften des `select`-Aufrufs beseitigt werden. Für den Anwender wurde deshalb eine Funktion (`mpx_select`) erstellt, die syntaktisch und semantisch kompatibel zum `select`-Aufruf ist, jedoch die komplette Verriegelung enthält. Damit wird die geforderte I/O-Konformität erreicht. Berücksichtigt man ferner, daß man bei MUPIX Signale in Nachrichten umwandeln kann, so wird sogar ein gleichzeitiges Multiplexing auf Signalen und I/O-Deskriptoren möglich. Diese Funktionalität hat sich beim Einsatz im Rahmen des ECS an mehreren Stellen gut bewährt[13] [ha93],[ku95].

Das Problem, auf verschiedene I/O-Kanäle reagieren zu müssen, stellt sich unabhängig von der Benutzung des Messagesystems innerhalb jedes (I/O-)Ereignisgesteuerten Programms. Deshalb wurde zur Verwaltung dieser Ereignisse ein allgemein verwendbares Softwaremodul erstellt. Durch dieses werden die I/O-Kanäle mit Hilfe einer linearen Liste verwaltet; hier ist jedem I/O-Kanal ein Listenelement zugeordnet, das als *Notifier-Element* bezeichnet wird. Es enthält die Funktionen (*Callback-Routinen*), die bei einem bestimmten Ereignis auf dem entsprechenden Kanal aufzurufen sind. Die Übergabeparameter der Callback-Routinen sind ebenfalls Teil des Notifier-Elements. Für den Aufruf der Callback-Routinen und die Verwaltung der Notifier-Elemente sorgt ein übergeordentes Objekt, der sogenannte *Notifier*. Eine Anwendung wird in Abb.4.7 gezeigt.
Der Notifier setzt auf dem Systemaufruf `select` auf, kann allerdings während der

[12] Demgegenüber wäre eine Implementation vom MUPIX auf der Basis von Semaphoren und Shared Memory sehr problematisch. Beide Mechanismen sind selbst in modernen Betriebssystemen nicht den allgemeinen I/O-Kanälen gleichgestellt, damit wäre ein I/O-Multiplexing nicht möglich!

[13] Bei der Implementation der erwähnten graphischen Benutzerschnittstelle hat sich herausgestellt, daß die Verarbeitung von Signalen, wie sie üblicherweise in Zusammenhang mit X-Windows vorgeschlagen wird, im Gegensatz zur MUPIX Lösung[kra93b] falsch ist, weil die Behandlung von kritischen Abschnitten innerhalb der X-Windows-Bibliothek nicht berücksichtigt wird.

Laufzeit auf andere dazu kompatible Funktionen, wie z.B. `mpx_select` umgestellt, werden. Auf der Basis des Notifiers ist es sehr einfach, eingabe-gesteuerte Zustandsmaschinen zu realisieren, indem man bei Zustandsübergängen die Callback-Routinen austauscht. Der Notifier kommt im Rahmen des ECS überall dort zum Einsatz, wo Programme I/O-Multiplexing betreiben.

```
...
/* io1, io2  I/O-descriptoren */
/*  Create aNotify object for descriptor io1 */
aNotify *n1p = aNotify_new(io1, echo, io_error, NULL, NULL);

/*  Create aNotify object for descriptor io2 */
aNotify *n2p = aNotify_new(io1, acceptCon, io_error, NULL, NULL);

if (n1p == NULL || n2p == NULL) {
  aNotify_del(n1p); aNotify_del(n2p);
  return ERROR;
}
/*  Wait for I/O events */
while (aNotify_alert() == OK)
  ;
...
```

Abb.4.7: Verwendung des Notifiers innerhalb einer C-Routine. Falls Daten zur Auslese auf dem I/O-Deskriptor `io1` (`io2`) bereitstehen, wird die Funktion `echo` (`acceptCon`)durch den Notifier aufgerufen. Das Multiplexing erfolgt transparent für die Anwendung in der Routine `aNotify_alert`. Die Funktionen der Notifierbibiotek sind in C implementiert, weil ursprünglich noch nicht auf allen Rechnern ein geeigneter C++-Compiler zur Verfügung stand.

4.4.5 Die Softclock und das Interne Senden

Jeder Prozeß enthält in seinem System-Datensegment eine Art "Weckuhr", die mit Hilfe eines Systemaufrufs gestellt werden kann [lef89]. Wenn der Wecker abgelaufen ist, wird dem Prozeß durch das Betriebssystem ein spezielles Signal, das sogenannte *Alarm-Signal*, zugestellt. Analog zur Abb.4.2 kann man eine Routine (*Alarmhandler*) bereitstellen, die beim Eintreffen dieses Signals aufzurufen ist.

Es besteht nun das Problem, daß die Weckuhr dem Prozeß nur genau einmal zur Verfügung steht, i.a. aber mehrere "Weckaufträge" parallel aktiv sein müssen.
Ein Beispiel: Die meisten der an der Drei-Spektrometer-Anlage beteiligten Geräte sind nicht in der Lage, selbstständig Änderungen von Parametern festzustellen und an den Rechner zu melden. Deshalb müssen solche Geräte von Zeit zu Zeit aktiv abgefragt/überprüft (*Polling*) werden. Da Gerätetreiber i.a. zwischen diesen Abfrageoperation aktiv sind, müssen solche Operationen durch ein Weckersignal angestoßen werden. Bei vielen Treiberprozessen wird nicht nur ein Parameter, sondern es werden gleichzeitig mehrere mit u.U. unterschiedlichen Frequenzen

überwacht. Dafür benötigt man für jeden Parameter eine eigene Weckuhr. Auch innerhalb der MUPIX-Bibliothek wird die Weckuhr benötigt, z.B. zur Realisierung des Timeouts beim Versenden einer Nachricht(s.S. 29).
Aus diesen Gründen ist es notwendig, die vom System bereitgestellte Weckuhr mittels einer geeigneten Verwaltung software-mäßig gewissermaßen zu vervielfältigen. Hierzu wurde im Rahmen der MUPIX Bibliothek ein Satz von Routinen implementiert, die unter dem Begriff *Softclock* zusammengefaßt werden.

Die Softclock-Routinen

In Abb.4.8 ist anhand eines Beispiels die Datenstruktur eines Satzes von Zeitereignissen dargestellt. Es handelt sich dabei um eine doppelt verkettete Liste, die im folgenden als Softclock-Liste bezeichnet wird. In der Liste ist aufgeführt, zu welchen zukünftigen Zeitpunkten welche Funktion im Programm mit welchen Argumenten aufgerufen und abgearbeitet werden soll.
Die Softclock-Liste ist in der zeitlichen Reihenfolge der Ereignisse angelegt; Ereignisse, die am schnellsten eintreten, sind vorne, und die am weitesten entfernt liegenden Ereignisse sind am Ende eingereiht. Die Ablaufzeit für jedes Ereignis wird als Abstand zur Zeit des vorangehenden Ereignisses in der Liste angegeben, deshalb spricht man auch von einer *Delta-Liste*[14]. Auf diese Weise kann man durch Verminderung der Ablaufzeit des ersten Listenelements, die Ablaufzeit aller Ereignisse verringern. Der Alarmhandler entfernt die Ereignisse vom Anfang der Liste her, wenn deren Zeit auf Null verringert ist. Falls noch Ereignisse in der Liste vorhanden sind, wird die Weckuhr von neuem gestartet und ihr Wert auf die Ablaufzeit des ersten Listenelements gestellt.
Damit Ereignisse ggf. auch vor dem Ablauf der Weckzeit effizient aus der Liste entfernt werden können, ist die Softclock-Liste doppelt verkettet. Solche Fälle treten z.B. dann ein, wenn man das Warten auf anderweitige Ereignisse durch einen Timeout zur Vermeidung von Deadlocks (siehe S.33) durch einen Eintrag in der Softclock-Liste abgesichert hat und das Ereignis rechtzeitig vor Ablauf des Timeouts eintrifft, so daß der entsprechende Eintrag gelöscht werden muß.
Genau wie im Fall der Nachrichtenliste stellen Zugriffe auf die Softclock-Liste kritische Abschnitte dar. Sie müssen deshalb mit den im vorhergehenden Abschnitt vorgestellten Mechanismen verriegelt werden.

Internes Senden

Das Anstoßen von Funktionen durch die Softclock erfolgt asynchron. Die dazu erforderlichen Programmabläufe vertragen sich nicht mit dem ansonsten vollständig sequentiellen, durch Nachrichten gesteuerten Programmablauf, z.B. bei einem Gerätetreiber. Dies erfordert komplizierte und, insbesondere bei Anfängern, oft fehlerträchtige Programme.
Deshalb wurde ein eigener Mechanismus entwickelt und innerhalb der MUPIX--Bibliothek implementiert, das sogenannte *Interne Senden*, der das Versenden von

[14]Dieses Schema ist der Zeitverwaltung im UNIX-Betriebssystemkern nachempfunden [lef89].

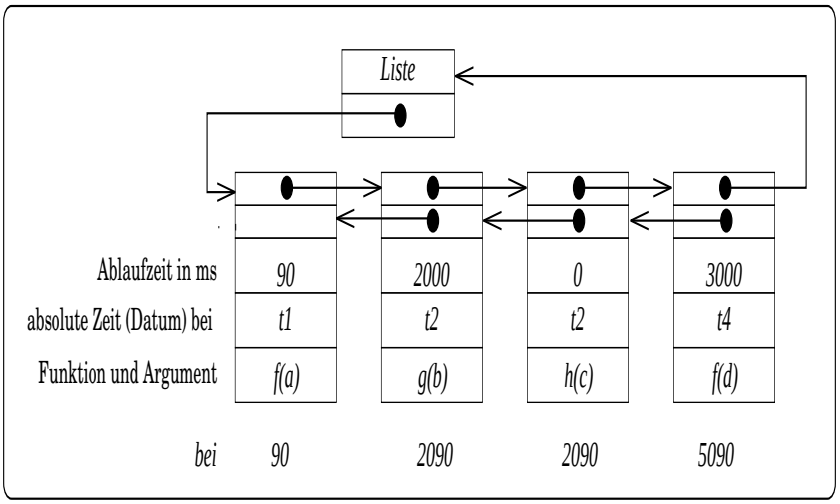


Abb.4.8: Die Softclock-Liste. Jedes Element der Liste zeigt auf seinen Vorgänger und Nachfolger, d.h. enthält deren Speicheradresse. Um einen Weckauftrag für einen zukünftigen Zeitpunkt t_i anzumelden, übergibt das Anwenderprogramm der Softclock-Verwaltung die dann aufzurufende Funktion und deren Argument. Im Gegensatz zur Nachrichtenliste in Abb.4.2 wird kein Zeiger auf das letzte Element der Softclock-Liste benötigt.

Nachrichten auch an prozeßinterne Empfänger unterstützt. Dabei werden Nachrichten entweder sofort oder erst nach einer vorgegebenen Zeitspanne (*Wecknachricht*) in die Nachrichtenliste eingereiht. Damit gehen wieder alle Aktionen im Programm von einer zentralen Stelle, nämlich dem Nachrichtenempfang aus; ihre Abarbeitung über die Nachrichtenliste macht sie damit aus Sicht des Anwenders vollständig sequentiell.
Ein zusätzlicher Vorteil ergibt sich dadurch, daß man mit der Nachricht im Gegensatz zum Signal auch Daten, also Informationen schicken kann, die über das Eintreffen des Ereignisses selbst hinausgehen.
Die Implementierung dieses Mechanismus erfolgt mit Hilfe der Softclock. Das Anwenderprogramm übergibt die interne Nachricht zusammen mit der Ablaufzeit an eine Routine der Softclock-Bibliothek, die sie als ein weiteres Element in die Softclock-Liste einfügt. Nach Ablauf der Weckzeit wird die Nachricht in die Nachrichtenliste umgehängt und von dort abgearbeitet.
Das "Interne Senden" wurde so konzipiert, daß mit seiner Hilfe Aktionssequenzen, die normalerweise längere Zeit in Anspruch nehmen, so aufgeteilt werden können, daß sie durch andere, von außen kommende Nachrichten unterbrechbar sind. Das Einstellen der Magnetfelder oder das Fahren der Kollimatoren [ma93] sind Beispiele, wo dieser Mechanismus erfolgreich eingesetzt wird.

4.4.6 Der MUPIX-Nameserver

In Abschnitt 4.3.2 wurde gezeigt, daß es sinnvoll ist, die Namensauflösung bei MUPIX nach dem Serverprinzip zu realisieren. Alle Objekte, die ihre Dienste netzwerkweit zur Verfügung stellen, melden sich im ersten Schritt bei einem Serverprozeß (*Nameserver*) an (siehe Abb.4.9). Dabei übermitteln sie zusammen mit ihren symbolischen Namen ihre Protokolladresse für die direkte Kommunikation. Falls ein anderes Objekt einen Dienst in Anspruch nehmen will, erfragt

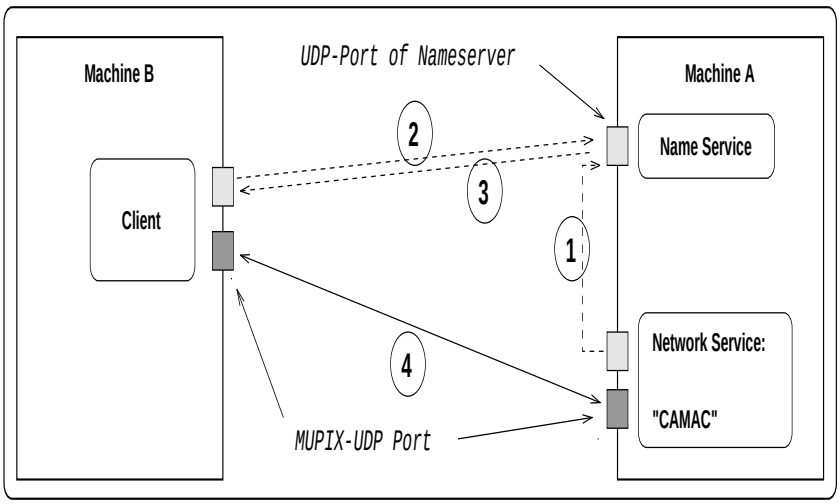


Abbildung 4.9: Prinzip der Namensauflösung.

es zunächst über dessen symbolischen Namen seine Protokolladresse beim Nameserver (Schritte 2 und 3 in Abb.4.9). Danach erfolgt die Kommunikation direkt zwischen den beiden Objekten (Schritt 4). Die UDP/IP Adresse des Nameservers wird den einzelnen Prozessen über eine Umgebungsvariable zugänglich gemacht. Der Namensraum kann in mehrere *Domänen* aufgeteilt werden, indem man jeder Domäne einen Nameserverprozeß zuordnet. Die einzelnen Domänen werden ebenfalls mit Namen bezeichnet. Der Domänenname ist Teil des Namens der Umgebungsvariablen, deren Wert die Nameserveradresse ist.
Mit Hilfe von Domänen lassen sich im selben Rechnernetzwerk mehrere disjunkte Mengen von miteinander kommunizierenden Prozessen bilden; nur innerhalb einer Domäne ist die Zuordnung eines symbolischen Namen zu einem Prozeß eindeutig. An der Drei-Spektrometer-Anlage wird diese Eigenschaft dazu genutzt, um neben dem eigentlichen ECS, weitere Steuerungssoftware an Testaufbauten zu erproben.
Die Möglichkeit, symbolische Adressen dynamisch an Protokolladressen zu binden, ist auch im Bereich anderer Protokollfamilien von Interesse (Abschnitt 5.3.2). Deshalb wurde das Nameservice-System unabhängig von der MUPIX-Bibliothek realisiert. Der Nameserver ist in der Lage, neben MUPIX- auch TCP- und UDP-Protokolladressen zu verwalten. Die notwendige Datenbank wird im dynamischen Speicher des Nameservers aus Baumstrukturen und *Hash-Tabellen* (s.S.57) aufgebaut. Die Funktionen zum An- und Abmelden sowie zum Auflösen von Adressen sind in einer Bibliothek zusammengefaßt, ihre Dokumentation befindet sich in [kra93a].

UDP/IP oder TCP/IP?

Da auf den Nameserver über das Netzwerk zugegriffen wird, stellt sich erneut die Frage, welches Protokoll sich hierfür am Besten eignet - UDP/IP oder TCP/IP? Um diese Frage zu entscheiden, muß man die spezifischen Eigenschaften dieser Kommunikation analysieren.

- Ein Prozeß kommuniziert mehrfach mit dem Nameserver, da er verschiedene Dienste in Anspruch nimmt. Deshalb ist es effizienter, nicht jedesmal von

neuem den Kommunikationskanal zu öffnen und wieder zu schließen (Tab. 4.1), sondern den Kanal nach der ersten Anfrage an den Nameserver offen zu halten. Im Falle des TCP/IP müßte dann allerdings die Verbindung zum Nameserver für die komplette Laufzeit eines Prozesses bestehen bleiben. Da jedoch die Anzahl der Netzwerkkanäle für einen Anwenderprozeß beschränkt ist, müßte auf seiten des Nameservers eine Verwaltung dafür sorgen, daß längere Zeit nicht mehr benötigte TCP-Verbindungen geschlossen werden. Ansonsten könnten mit der Zeit keine neuen Anfragen mehr befriedigt werden. Eine solche Verwaltung kompliziert die Programmierung und den Programmablauf des Nameservers. Schon im Rahmen des ECS übersteigt die Anzahl der am Message-System teilnehmenden Prozesse leicht die Schranke, bei der man noch ohne Verwaltungsaufwand auskommt. Im zeitlichen Mittel müßte für jede Anfrage die Verbindung neu aufgebaut werden.
Im Gegensatz dazu wird beim verbindungslosen UDP/IP-Protokoll, nur ein einziger Kanal benötigt, über den der Nameserver alle Anfragen befriedigen kann.

- Die Kommunikation mit dem Nameserver folgt dem Prinzip des RPC. Nach dem Abschicken einer Anfrage, wartet der Klient auf die Antwort des Nameservers. Verwendet man zur Übermittlung der Antwort ein durch Timeouts gesichertes, zuverlässiges Protokoll wie das TCP/IP, dann kann es in bestimmten Fällen zum Blockieren des Nameservers kommen. Ein solcher Fall tritt z.B. ein, wenn die Verbindung zu dem Knoten unterbrochen ist, auf dem sich das Klientenprogramm befindet. Der Nameserver blockiert dann bis der Sendevorgang für die Antwort beendet ist, d.h. nach Ablauf der TCP/IP-Timeouts. Dieser Fall kann beim verbindungslosen UDP/IP nicht auftreten.

Aus diesem Gründen ist das UDP/IP dem TCP/IP für den Zugriff auf den Nameserver trotz der fehlenden Zuverlässigkeit vorzuziehen. Bei der Diskussion des MUPIX-Protokolls wurde bereits gezeigt, daß man auch auf der Basis von UDP/IP mit Hilfe von Timeouts und Quittungen einen zuverlässiges Datentransport realisieren kann. Für die Kommunikation mit dem Nameserver wird kein zusätzliches Protokoll benötigt, da sich die Kommunikation einseitig durch den Klienten auf Auftragsebene absichern läßt, indem die Antwort des Nameservers selbst als Quittung verwendet wird. Dies ist aus zwei Gründen möglich:

1. Die Nachrichten, die zwischen dem Nameserver und seinen Klienten ausgetauscht werden, sind kurz, sie passen jeweils in ein einziges UDP-Datagramm.
2. Alle Aktionen des Nameservers wurden so gestaltet, daß sie im wesentlichen idempotent sind, d.h. deren mehrmalige Ausführung verändert den Zustand des Servers nur einmal, nochmaliges Senden eines Auftrags/einer Anfrage hat beim Ausbleiben der Rückmeldung des Nameservers keinerlei Konsequenzen.

Da der Nameserver und dessen Klienten sich u.U. auf Rechnern mit verschiedener Binärdarstellung der Daten befinden, ist es notwendig, alle Nachrichten von und zum Nameserver in einem maschinen-unabhängigen Format zu kodieren. Zur Kodierung der Nachrichten wurde deshalb ein Protokoll definiert und die entsprechenden Routinen mit Hilfe des XDR-Verfahrens[15] erstellt. Das komplette

[15]Eine Diskussion des XDR-Verfahrens findet sich auf S.54

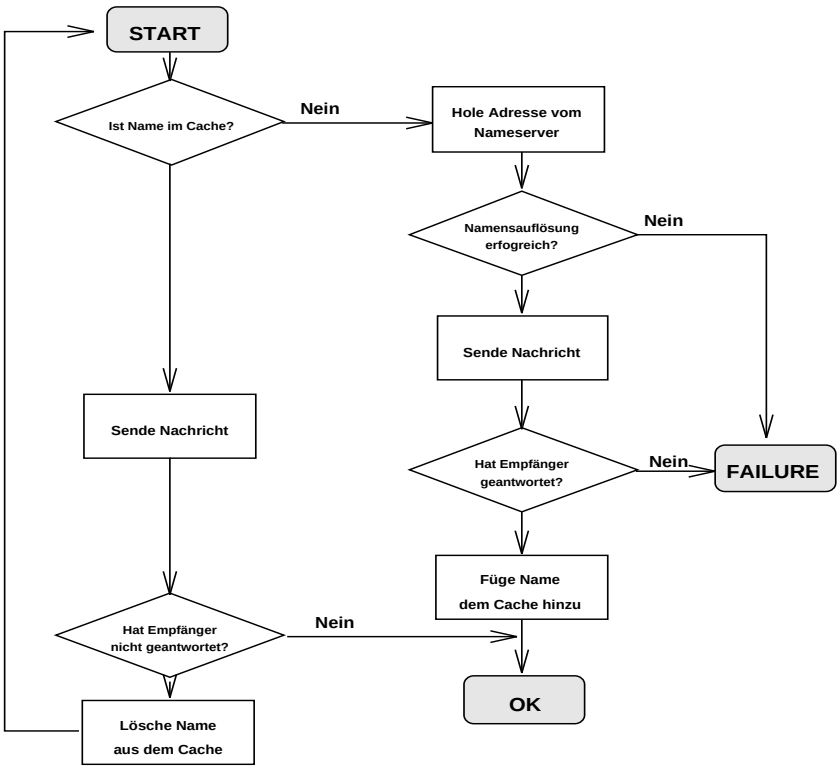


Abb.4.10: Prinzip der Namensauflösung. Der Nameserver wird nur dann direkt abgefragt, wenn sich der Name noch nicht im Namecache des Senders befindet oder das Senden mit einer bereits gepufferten Adresse fehlschlägt. Im zweiten Fall werden Inkonsistenzen zwischen Cache und Nameserver beseitigt. Die Zugriffe auf den Namecache erfolgen versteckt vor dem Anwenderprogramm innerhalb der MUPIX-Senderoutine.

Nameserver-Protokoll befindet sich im Anhang B.1.

Der *Namecache*

Bei rein symbolischer Adressierung würde man pro Nachricht neben den beiden Datagrammen für den eigentlichen Nachrichtentransport, zwei weitere Datagramme für die Namensauflösung benötigen, was zu einem erheblichen Netzwerkverkehr führen würde. Um dies zu vermeiden, werden bereits aufgelöste Namen zusammen mit den zugehörigen Protokolladressen in einem prozeßeigenen Pufferspeicher, den sogenannten *Namecache*, abgelegt (siehe Abb.4.10), wodurch die Anzahl der zu transportierenden UDP-Datagramme halbiert wird.

Weil jede MUPIX-Nachricht auch den Absendernamen im Protokollkopf enthält (Abb. 4.5), wird der Namecache auch beim Nachrichtenempfang aktuallisiert, d.h. mit Name- und Adresse des Absenders erweitert. Hierdurch kommt es zur Verteilung von Adressen ohne direkte Hilfe des Nameservers, was zu einer weiteren Veringerung des Aufwands führt.

	MUPIX				UDP Ping-Pong			TCP-Message	
Sende-Host	t_{sys}^{mup}	t_{user}^{mup}	t_{wall}^{mup}	n_r	t_{sys}^{udp}	t_{wall}^{udp}	t_{sys}^{over}	t_{sys}^{tcp}	t_{wall}^{tcp}
DEC3100	1.76	0.33	3.1	322	1.23	2.1	0.53	4.7	7.0
E6	3.7	0.86	7.4	133	2.38	4.6	1.32	8.6	11.2
	alle Zeiten in ms, n_r in Hz, $t_{sys}^{over} = t_{sys}^{mup} - t_{sys}^{udp}$								

Tabelle 4.1: Zeiten, die für das Versenden einer Nachricht über MUPIX bzw. TCP und für einen UDP-Ping-Pong gemessen wurden. Bei den letzten beiden Fällen kann die User-Zeit vernachlässigt werden. Der Empfangsprozeß befand sich in allen Fällen auf einer DECstation 5000, die wegen ihrer Prozessorleistung in der Lage ist, die Daten so schnell zu verarbeiten, daß der Sender nicht durch den Empfänger behindert wird.

Der Nameserver als *Single Point of Failure*

Wie bereits erwähnt, gibt es nur einen Nameserver pro Domaine. Fällt dieser aus, so kommt die Kommunikation, sieht man vom Namecache ab, zum Erliegen. Im Rahmen des ECS stellt der Nameserver somit einen sogenannten *Single Point of Failure* dar. Allerdings hat sich der Nameserver beim Einsatz[16] als extrem stabiles Programm erwiesen. Ein Ausfall des Nameservers ist praktisch immer nur durch einen Systemabsturz oder durch Eingriffe von außen herbei geführt worden.
Daher wurde von einer weitergehenden Absicherung des Nameservers, z.B. durch parallel laufende Stellvertreter des Servers auf anderen Rechnern, abgesehen, zumal es im Rahmen des ECS sowieso noch andere Serverprozesse gibt, deren Ausfall das System funktionsunfähig machten (z.B. die Datenbank), da sie nur einmal im Gesamtsystem existieren.
Um die Konsequenzen eines Nameserverausfalls, z.B. durch einen Systemabsturz, zu minimieren, wird jeder An- und Abmeldungvorgang durch den Nameserver in einer Datei, dem *Logfile*, mitprotokolliert. Aus diesem Logfile kann der Nameserver beim Neustart seine interne Datenbank rekonstruieren. Damit sind auch nach einem Systemabsturz von der Adressverwaltung her wieder alle Prozesse ansprechbar.

4.4.7 Rechenzeitverbrauch von MUPIX

Sowohl das Senden als auch das Empfangen einer Nachricht ist bei MUPIX mit zusätzlichen Aufwand verbunden, der über den bloßen Aufruf der UDP- Kommunikationsroutinen hinausgeht. Dieser Aufwand resultiert aus

- der Verwaltung der Softclock,
- der internen Speicherverwaltung,
- der Protokollarbeit,
- dem Ver- bzw. Entpacken der Nachricht,
- der Veriegelung von kritischen Abschnitten.

[16] 1993 lief der Nameserver mehr als 2500h, davon 1269h "real time" für A1-Strahlzeit, er ist dabei niemals fehlerbedingt terminiert.

Tabelle 4.2 enthält die gegenüber einem UDP-Ping-Pong zusätzlichen Systemaufrufe. Um sicher zu stellen, daß keine Nachricht verloren geht, muß innerhalb der Signalroutine das UDP-Socket noch mindestens einmal zusätzlich ausgelesen ($i = 1$) werden, weil ein Signal, das bereits ansteht, während es vom Prozeß ausmaskiert ist, nur einmal vom System zugestellt [lef89, ste92] wird, unabhängig davon wie oft es während dieser Zeit generiert wurde. Die Aufrufe $i = 2, 3, 4$ werden für die Signalverriegelung (Abb.4.6), 5 und 6 für die Zeitverwaltung benötigt.

			t_i in ms	
i	Aufruf	n_i	E6	DEC3100
1	`recvfrom`	1	0.22	0.1
2	`sigmask`	1	0.001	0.0001
3	`sigblock`	1	0.09	0.023
4	`sigpause`	1	0.09	0.024
5	`setitimer`	1	0.3	0.23
6	`gettimeofday`	2	0.11	0.04
$t_{sys}^{tot} = \sum n_i t_i$			0.92	0.46
$t_{sys}^{over} = t_{sys}^{mup} - t_{sys}^{udp}$			1.32	0.53
n_i Anzahl der Aufrufe				

Die dabei anfallende zusätzliche Rechenzeit (Overhead) setzt sich aus Systemzeit t_{sys}^{over} und User-Zeit[17] t_{user}^{over} zusammen. Diese Zeiten kann man abschätzen, indem man System- und Userzeit für das Versenden einer Nachricht mittels MUPIX $(t_{sys}^{mup}, t_{user}^{mup})$ bestimmt und mit den entsprechenden Zeiten bei einem sogenannten "UDP-Ping-Pong" $(t_{sys}^{udp}, t_{user}^{udp})$ vergleicht. Beim UDP-Ping-Pong wird eine Nachricht direkt per UDP geschickt und auf eine Antwort gewartet. Die dabei anfallenden Systemaufrufe (`sendto, recvfrom`) bilden die Basis für die MUPIX-Kommunikation. Tabelle 4.1 enthält die gemessenen Werte. Aus der während eines Sendevorgangs verstreichenden Zeit t_{wall}^{mup}, in die zusätzlich die Zeit für den Datentransport über das Netztwerk, die Antwortzeit des Empfängers (Quittung) und die Maschinenlast eingeht, ergibt sich die Anzahl der Nachrichten n_r, die pro Sekunde Sekunde versandt werden können: $n_r = 1/t_{wall}^{mup}$.

Eine Analyse (Tabelle 4.2) der Systemzeit zeigt, daß sich t_{sys}^{over} im wesentlichen durch die anfallenden Systemaufrufe erklären läßt. Die Differenzen von t_{sys}^{tot} und t_{sys}^{over} resultieren aus dem zusätzlichen Aufwand zur Prozeßumschaltung (Scheduling). Da die Anzahl der Systemaufrufe bereits minimiert wurde, läßt sich die Systemzeit nicht weiter verringeren. Lediglich bei der User-Zeit sind noch Einsparungen möglich[18].

Die Ergebnisse in Tabelle 4.1 sind aus zweierlei Gründen eine Bestätigung des gewählten Konzepts:

- Das Versenden einer Nachricht mit MUPIX ist in jedem Falle schneller als mittels einer TCP ("TCP-Message"), obwohl die Protokollarbeit selbst verrichtet werden muß.
- Die erzielte Nachrichtenrate $n_r = 133$ Hz zwischen E6 und DECstation liegt deutlich über den ursprünglich als ausreichend angesehenen 100 Hz.

[17] Die User-Zeit ist die CPU-Zeit, die Instruktionen zugerechnet werden, die auf Anwenderebene abgearbeitet werden. Während der Systemzeit erledigt der Systemkern Aufgaben für einen Prozeß.

[18] Während der Programmierung stand die Wartbarkeit der Software im Vordergrund.

Die zur Zeit im "Frontend"-Bereich eingesetzten VMEbus-Rechner des Typs E6 mußten bereits zu Beginn der Aufbauphase der Drei-Spektrometer-Anlage angeschafft werden, weil man sie bereits zum damaligen Zeitpunkt zur Ansteuerung und Datenerfassung an Laboraufbauten bzw. zur Entwicklung der Software benötigte. Diese Prozessoren sind mittlerweile am unteren Ende der Leistungsskala angesiedelt. Auch im Bereich der Workstation-Hardware hat eine enorme Leistungsentwicklung stattgefunden. Aus diesem Grund wurden Vergleichsmessungen auf neueren Prozessoren durchgeführt, die dem Institut zu Testzwecken zur Verfügung standen. Die Ergebnisse sind in Tab.4.3 zusammengefasst. Erwartungsgemäß sind mit den Rechnern neueren Typs kürzere Zeiten erzielt worden, insbesondere bei den VMEbus-Maschinen. Deshalb sollte in diesem Bereich über eine Ablösung der eingesetzten Systeme nachgedacht werden. Auf Workstationseite wurde die oben als Referenz benutzte Maschine (DEC3100) bereits durch ihr Nachfolgemodell (DEC5200) ersetzt; der damit erzielte Leistungszugewinn ist in Tab.4.3 dokumentiert.
Ähnlich wie der Datendurchsatz in Abb.4.3 sind die gemessenen Zeiten sowohl von der CPU-Leistung als auch vom Rechnerbetriebssystem abhängig. Dies wird durch direkten Vergleich der Messungen 3a und 3b deutlich, die sich nur durch die unterschiedliche Implementierung der Netzwerksoftware der Betriebssysteme unterscheiden: Beispielsweise dauerte der Aufruf der Funktion `recvfrom` (siehe auch Tab.4.2) bei der Messung 3a etwa viermal länger als bei Messung 3b. Weitere Verbesserungen können durch Optimierung der Anpassung der MUPIX-Bibliothek an das jeweilige Betriebssystem erreicht werden. Diese ist eine der zukünftigen Aufgaben.

Vergleich verschiedener Implementationen								
	Maschine	MIPS	t_{sys}^{mup} ms	t_{user}^{mup} ms	$t_{sys}^{mup} + t_{user}^{mup}$ ms	Betriebs-system	CPU	
1	E6	3	3.7	0.86	4.56	BSD 4.3	M68030/20	C,V
2	E7	9	1.16	0.34	1.5	BSD 4.3	M68040/25	C,V
3a	Force 2CE	28.5	1.32	0.82	2.15	Solaris	Sparc	R,V
3b			0.90	0.32	1.22	SunOS 4.1		
4	Force 5CE	112.5	0.91	0.62	1.53	Solaris	microSparc-II	R,V
5	DEC3100	14	1.76	0.33	2.09	Ultrix 4.3	R2000A	R,W
6	DEC5200	21	1.12	0.24	1.36	Ultrix 4.3	R3000A	R,W
7	SUN 10/30	86	0.81	0.5	1.31	Solaris	SuperSparc	R,W
V: VMEbus-Rechner, W: Workstation, R: RISC, C:CISC								

Tabelle 4.3: Gemessene Zeiten für das Versenden einer Nachricht mit Hilfe von MUPIX auf neueren Rechnern im Vergleich mit den derzeit installierten E6-Prozessoren bzw. DEC-Workstations. Zum Vergleich der einzelnen Rechner wird die Prozessorleistung in MIPS angeben.

4.5 Zusammenfassung und Ausblick

MUPIX löst die IPC-Problematik eines LAN-basierten Experimentsteuerungssystems mit seinen besonderen Anforderungen (Abschnitt 4.2) auf portable[19] und

[19]Die aktuelle Version (2.1) der Software wurde bereits unter Ultrix-4.3, BSD 4.3, AIX, SunOS 4.1.1, Solaris, AIX, OSF und FreeBSD eingesetzt, VMS und OS-9 wurden bis zur Version 1.5 aktiv unterstützt.

effiziente Weise. Durch die Möglichkeit verschiedene Empfänger *innerhalb* eines Prozesses mit symbolischen Namen zu adressieren, setzt MUPIX das objektorientierte Programmierkonzept über die engen Grenzen des Programmkontextes hinaus fort.
Insgesamt stellt MUPIX mit den Funktionen zum Senden und Empfangen von Nachrichten, den "Internen Nachrichten" und den Routinen zur Zeit- und Deskriptorverwaltung (Softclock, Notifier) die Basismechanismen zur Verfügung, die zur Programmierung von Nachrichten-gesteuerten Gerätetreibern benötigt werden. MUPIX bildet somit die Grundlage für das gesamte Steuerungs- und Überwachungssystem der Drei-Spektrometer-Anlage .

Das System, das bei allen Test- und Produktionsstrahlzeiten eingesetzt wurde, hat sich als ein stabiles Softwaresystem bewährt, das allen bisherigen Anforderungen gerecht wurde. Dennoch bestehen einige Erweiterungs- bzw. Verbesserungsmöglichkeiten:

- Multicasts und Mailbox-Sendemodus (vgl. S.22) könnten mit Hilfe des Nameservers realisiert werden. Beim Multicast würde der Nameserver eine Nachricht an alle möglichen Empfänger weiterleiten. Beim Mailbox-Sendemodus übernähme er die Rolle des Mailbox-Managers (vgl. S.22).

- Durch Stellvertreter des Nameservers könnte die Ausfallsicherheit des Systems weiter erhöht werden (siehe Abschnitt 4.4.6).

- Der Schutz des ECS vor unbefugter Benutzung sollte durch Authentisierungsmechanismen innerhalb des MUPIX-Protokolls verbessert werden. Die Software ist hierfür vorbereitet, verschiedene Möglichkeiten der Authentisierung wurden bereits erprobt. Der Protokollkopf enthält z.B. ein Feld für eine Sicherheitskennung (Abb. 4.5), die vom Nameserver bei der Adreßauflösung mitgeliefert wird, falls sich der Klient mit der richtigen Benutzerkennung (UIC) ausweist. Das Nameserver Protokoll (Anhang B.1) und der Programmtext des Nameservers enthalten bereits die dazu notwendigen Erweiterungen. Wegen des zusätzlichen Verwaltungsaufwands wurde vom Einsatz im Standardsystem jedoch abgesehen. Um diese Mechanismen endgültig in Betrieb zu nehmen, sollte zuvor noch der Zugriff auf den Nameserver selbst geschützt werden, um die Sicherheit weiter zu erhöhen.

- MUPIX ist auf Applikationen ausgelegt, die mit kurzen Nachrichten bei der Kommunikation auskommen. Ein standardmäßiges Vergrößern der Nachrichtenlänge wäre aus den auf Seite 29 genannten Gründen auf der Basis von UDP/IP nicht sinnvoll, vielmehr sollte ggf. der Transport von größeren Nachrichten über TCP/IP erfolgen. Zur Vermeidung von Pufferungsproblemen läuft das Senden und der Empfang der Daten dabei völlig synchron ab. In diesem Zusammenhang besteht die Aufgabe, MUPIX so zu erweitern, daß TCP-basierte Nachrichten von Anwendersicht in gleicher Weise behandelt werden, wie die bisherigen kurzen UDP/IP-basierten Nachrichten.

- Interessant wäre es, die Protokollarbeit nicht mehr mit Hilfe von Signalmechanismen, sondern mit sogenannten *Threads* zu realisieren, die auf den modernsten Betriebsystemen zur Verfügung stehen [osf91]. Threads (oder auch "leichtgewichtige Prozesse") bewirken eine vom Betriebssystem unterstützte Parallelisierung innerhalb eines Prozesses.

Der Vorteil MUPIX mit Threads an Stelle von Signalen zu implementieren, liegt darin begründet, daß einige der bereitgestellten Bibliotheksfunktionen bereits vom Betriebssystem für die parallele Abarbeitung durch Threads vorgesehen sind. Dadurch entfallen einige der Probleme, die im Abschnitt 4.4.3 im Zusammenhang mit kritischen Abschnitten diskutiert wurden.
Vor der Realisierung dieser Methode müßte allerdings untersucht werden, wie aufwendig die Systemaufrufe zur Threadverwaltung im Gegensatz zur Signalverarbeitung sind. Solche Tests konnten noch nicht durchgeführt werden, weil Rechner mit den entsprechenden Betriebssystemen dem Institut erst seit kurzem zur Verfügung stehen.

Kapitel 5
Apparative Überwachung

Bereits im Kapitel 3 wurde gezeigt, daß es notwendig ist, den Zustand der Drei-Spektrometer-Anlage während einer Messung im Rahmen der Experimentsteuerung zu überwachen. Die Software zur Experimentsteuerung wurde im Rahmen verschiedener Diplom- und Doktorarbeiten parallel zum Aufbau der Drei-Spektrometer-Anlage entwickelt und implementiert. Aufgabe der vorliegenden Dissertation war es, die verschiedenen Teilprojekte in einen Gesamtrahmen zusammenfassen, dem Konzept zur Überwachung der Anlage.
Zur Realisierung eines solchen Monitorsystems sind Software-Werkzeuge zur Verfügung zu stellen, die es erlauben, den aktuellen Zustand der Apparatur zu erfassen, zu protokollieren, und dem Experimentator verfügbar zu machen. Darüber hinaus ist dafür zu sorgen, daß aufgetretene Fehlersituationen dem Experimentator signalisiert werden. Entsprechende Softwaremodule waren zu entwickeln.

5.1 Das Gesamtkonzept

Zur Realisierung von Überwachungsmechanismen in einem verteiltem System kann man zwei Wege einschlagen: Entweder man arbeitet mit einem zentralen Überwachungsprogramm, das aus der Kenntnis des gesamten Anlagenzustands beurteilt, ob eine Fehlersituation vorliegt, oder die Entscheidung wird im wesentlichen von den Geräte-Objekten getroffen.

Bei der zentralen Variante müssen alle - auch die gerätespezifischen - Informationen und Algorithmen an zentraler Stelle verfügbar sein. Dies hat u.a. zur Folge, daß bei jedem dem System hinzugefügten neuartigen Gerät Änderungen am zentralen Überwachungsprogramm vorzunehmen sind. Um die Fehleranfälligkeit bei derartigen Hinzufügen zu minimieren, wird hier im wesentlichen interpretativ gearbeitet, d.h. man arbeitet mit Programmen, die so allgemein formuliert sind, daß die Kriterien zur Überwachung aller Geräte in Form von Eingabetabellen vorliegen.
Die Realisierung dieses Konzepts wurde bereits beim Aufbau des ECS für den ZEUS-Detektor an HERA [be94] untersucht. Dabei hat sich herausgestellt, daß diese Aufgabe nur mit den leistungsfähigsten, kommerziell erhältlichen Programmen aus dem Bereich der Expertensystemtechnik zufriedenstellend gelöst wird. Wegen der hohen Anschaffungskosten solcher Programme und der für die Installation, Programmierung und Pflege notwendigen speziellen Kenntnisse in der Expertensystemtechnik kam diese Lösung als Teil des ECS nicht in Frage.

Daher wurde der Weg eines verteilten Überwachungssystems eingeschlagen, bei dem die Geräte-Objekte selbst auch für die Überwachung des jeweiligen Geräts sorgen. Damit wird vermieden, daß die verschiedenen Werte, die für die Überwachung erforderlich sind und die dem Geräte-Objekt sowieso für Steuerungsaufga-

ben bekannt sind, nochmal zusätzlich an ein zentrales Überwachungsprogramm geschickt werden müssen.

Darauf, daß man allerdings auch bei einem dezentralen Überwachungssystem nicht ganz ohne eine zentrale Instanz auskommt, wurde bereits im Kapitel 3 hingewiesen: Eine solche wird zum Sammeln von Zustandsinformationen aller Geräte-Objekte benötigt z.B. für die Realisierung des bereits diskutierten elektronischen Logbuchs. Hierin sind insbesondere die festgestellten Fehlersituationen einzutragen. Das Gesamtkonzept zur Überwachung der Drei-Spektrometer-Anlage besteht also aus einem System aus *dezentralen und zentralen* Elementen.

5.1.1 Der Statusserver

Das Sammeln von Daten ist eine klassische Aufgabe von Datenbanksystemen (DBMS[1]), insbesondere im Hinblick auf die Mächtigkeit der von solchen Systemen bereitgestellten Werkzeuge zur Verwaltung der gespeicherten Daten. Für den Einsatz im On-line System der Drei-Spektrometer-Anlage muß ein kommerzielles DBMS zwei wesentliche Voraussetzungen erfüllen:

1. Die Schnittstellenbibliotheken zur Kommunikation mit dem Datenbankserver müssen auf allen im On-line Bereich eingesetzten Rechnersystemen, also auch auf den "Frontend-Rechnern", vorhanden sein.

2. Die Transaktionsgeschwindigkeiten müssen so hoch sein, daß die Geräte-Objekte durch das Senden einer Statusmeldung nicht länger blockiert werden.

Dieses Problem wurde auch im Zusammenhang mit der Realisierung der Konfigurationsdatenbank im Steuerungsystem untersucht [ku95]. Dabei stellte sich heraus, daß die beiden aufgeführten Punkte nur bei einfachen DBMSen erfüllt sind, die im wesentlichen nur als Dateiserver arbeiten. Damit entfällt aber einer der wesentlichen Vorteile eines DBMS, nämlich die mächtigen und i.a. effizienten Verwaltungsmechanismen. Deshalb wurde als zentraler Bestandteil des Monitorsystems ein Programm implementiert, das als *Statusserver* nur zum Sammeln und Weiterleiten (s.u.) von Statusmeldungen dient.
Obwohl für die Zukunft der Einsatz eines kommerziellen DBMS für die Verwaltung aller relevanten Daten durchaus in Frage kommt [ku95, gei94], kann auf den Statusserver aus den oben genannten Gründen auch dann nicht verzichtet werden, um die on-line gesammelten Statusmeldungen off-line in das übergeordnete DBMS zu importieren. Das On-line System bleibt auch bei der Einführung eines kommerziellen DBMS unverändert.

Die Geräte-Objekte müssen mindestens immer dann eine Statusnachricht mit den aktuellen Werten der Geräteparameter via MUPIX an den Statusserver schicken, wenn sich ihr Zustand geändert hat. Aus Sicherheitsgründen senden die Geräte-Objekte noch zusätzlich in regelmäßigen, gerätespezifischen Zeitintervallen Zustandsinformation an den Statusserver. Damit läßt sich in Zweifelsfällen auch noch zu einem späteren Zeitpunkt (off-line) anhand der Protokollinformation im elektronischen Logbuch feststellen, ob ein Parameter tatsächlich konstant geblieben ist oder ob die Auslesesoftware nicht funktioniert hat.

[1] Database Management System

Mit den einlaufenden Statusmeldungen baut der Statusserver eine prozeßinterne Datenbank auf, die immer nur die letzten gemeldeten Werte der Geräteparameter enthält. Diese Datenbank kann von Klientenprogrammen (*Abfrage-Klienten*) in einem typischen Datenbankzugriff abgefragt werden. In diesem Sinn fungiert der Statusserver auch als Datenbank im On-line System, wo es auf kurze Transaktionszeiten ankommt und nur die aktuellen Statusmeldungen benötigt werden.

Bei den Statusmeldungen wird zwischen den sogenannten *Zustandsmeldungen* und den sog. *Sondermeldungen* unterschieden. *Zustandsmeldungen* enthalten den aktuellen Wert eines Geräteparameters. *Sondermeldungen* liefern weitergehende Informationen, die nicht immer einem Parameter zugeordnet werden können. Sie können zum Beispiel einen Fehler signalisieren.

Damit differenziert auf verschiedene Situationen reagiert werden kann, wird bei den Sondermeldungen zwischen Warnungen sowie Fehler- und Alarmmeldungen unterschieden. Um bei der Entwicklung von Gerätetreibern die Fehlersuche zu vereinfachen, gibt es eine weitere Form der Sondermeldung, die sogenannte Debugmeldung. Die Art der Statusinformation wird mittels einer Flagge innerhalb der Statusnachricht angezeigt.
Im Abb.3.1 verschickt z.B. das für die Gasversorgung der Kammern verantwortliche Objekt den jeweils aktuellen Wert des Gasfluß als Zustandsmeldung. Das Absinken des Gasfluß unterhalb eine kritische Grenze zeigt der Gerätetreiber mit einer Alarmmeldung an.

5.1.2 Monitorklienten

Damit sich der Experimentator während einer Messung schnell einen Überblick über den Zustand ausgewählter Gerätegruppen verschaffen kann, werden die wesentlichen Parameter der betreffenden Geräte auf speziellen Monitoren im Kontrollraum dargestellt (siehe Abb.5.1). Hierzu muß das darstellende Programm laufend mit den aktuellen Statusmeldungen versorgt werden. Programme mit dieser Anforderung werden im folgenden allgemein als *Monitorprogramme* bezeichnet.
Eine Möglichkeit, für die Aktualisierung der Statusanzeige zu sorgen besteht darin, daß das Monitorprogramm periodisch die aktuelle Statusinformation bei den betreffenden Geräte-Objekten abfragt. Diese Methode hat allerdings den Nachteil, daß dem Monitorprogramm gerätespezifische Information, wie z.B. die Frequenz, mit der ein bestimmtes Gerät auszulesen ist, bekannt sein muß. Zudem ist es in diesem *Polling-Betrieb* praktisch unmöglich, Änderungen eines Parameters stets zeitnah festzustellen; Änderungen zwischen zwei Abfragen bleiben dabei eventuell unentdeckt. Deshalb wird der umgekehrte Weg beschritten: die Statusinformation wird automatisch von seiten der Geräte-Objekte geliefert.

Die Geräte-Objekte schicken die Statusmeldungen nicht direkt an die Monitorprogramme, sondern an einen zentralen Verteiler, der dann die Information weiterreicht (siehe Abb.5.2). Diese Architektur bietet folgende Vorteile:

- Die Fehleranfälligkeit des Gesamtsystems wird verringert, da die Gerätetreiber von den Monitorprogrammen entkoppelt sind.
  Dies wird am Beispiel einer Situation deutlich, die auftritt, wenn ein Monitorprogramm, das Statusinformationen von mehreren Geräte-Objekten angefor-

```
                          Magnet_Spektrometer_A
              speca/magnet                              speca/magnet
d1             0.96890 T     05:21  sext               0.21414 T   05:21
d1/hall          0.965 T     05:20  sext/m/current      279.31 A   05:19
d1/nmr         0.96890 T     05:20  sext/m/water/flow   687.0 l/h 05:19
d1/current       781.9 A     05:19  sext/m/water/temp  37.668 deg 05:20
d1/water/flow  6107.0 l/h    05:19
d1/water/temp  20.861 deg    05:20  sext/a/current       49.33 A   05:19
                                    sext/a/water/flow    193.0 l/h 05:21
              speca/magnet          sext/a/water/temp   28.050 deg 05:21
d2             0.96906 T     05:23
d2/hall          0.979 T     05:18                speca/magnet
d2/nmr         0.96906 T     05:23  quad               0.33311 T   05:21
d2/current      781.94 A     05:19  quad/current        384.87 A   05:19
d2/water/flow  6816.0 l/h    05:20  quad/water/flow     2213.0 l/h 05:21
d2/water/temp  20.469 deg    05:14  quad/water/temp      35.57 deg 05:19
```

Abbildung 5.1: Typische Ausgabe eines Programms, das den Status ausgewählter Geräte auf einem Terminalbildschirm im Messraum anzeigt.

dert hat, vom Anwender so abgebrochen wird, daß es nicht ordnungsgemäß aus dem System ausscheidet. Würden die einzelnen Gerätetreiber die Statusmeldungen direkt an das Monitorprogramm schicken, dann müßte jeder Treiber diese Fehlersituation selbst erkennen. Dies würde z.B. über den Ablauf eines Timeouts beim Versenden der Statusmeldung erfolgen. Dies hätte erhebliche Nachteile auf das Verhalten des Gesamtsystems. Ein zentraler Verteiler kann wesentlich besser auf diese Situation reagieren.

- Ein weiterer Vorteil des zentralen Verteilers besteht darin, daß es aus Sicht der Monitorprogramme nur eine Informationsquelle gibt, unabhängig davon wieviele Gerätetreiber jeweils im Gesamtsystem aktiv sind; ohne zusätzlichen Verwaltungsaufwand erhalten die Monitorprogramme automatisch die Meldungen neu hinzukommender Geräte vom Verteiler.

Da zur Realisierung des elektronischen Logbuchs bereits die Einrichtung des Statusserver erforderlich ist, übernimmt dieser auch die Rolle des zentralen Verteilers. Dafür wurde der Statusserver mit einem besonderen Modus zur "Dauerabfrage" der internen Datenbank[2], dem sogenannten *Monitormodus*, ausgestattet.
Bei diesem Modus werden den Monitorklienten, die Statusmeldungen der Geräte-Objekte automatisch weitergeleitet. Beim Anmelden einer Dauerabfrage spezifiert ein Monitorklient eine *Filterbedingung* zur Auswahl der Statusmeldungen, die Bedingung besteht aus einem sogenannten *regulären Ausdrucks*[3] mit dem der Monitorklient eine Menge von Geräten über ihren Namen auswählt und dem Meldungstyp, dem diese Meldungen angehören sollen. Eine solche Dauerabfrage wird (zusammen mit ihrer Filterbedingung) als *Terminal* bezeichnet. Terminals, die nur Zustandsmeldungen zugestellt bekommen, werden als *Monitorterminals* be-

[2]Bei neueren, kommerziellen Datenbanksystemen sind ähnliche Mechanismen, sog. *Event Alerts* implementiert, die automatisch bei der Änderung von Datenbankeinträgen vom Klienten spezifizierte Aktionen anstoßen.

[3]Reguläre Ausdrücke sind Textmuster, die Sonderzeichen enthalten und ein Menge aus Zeichenketten repräsentieren [hop79]. Das Muster "abc.*" steht z.B. stellvertretend für die Menge aller Zeichenketten, die mit "abc" beginnen.

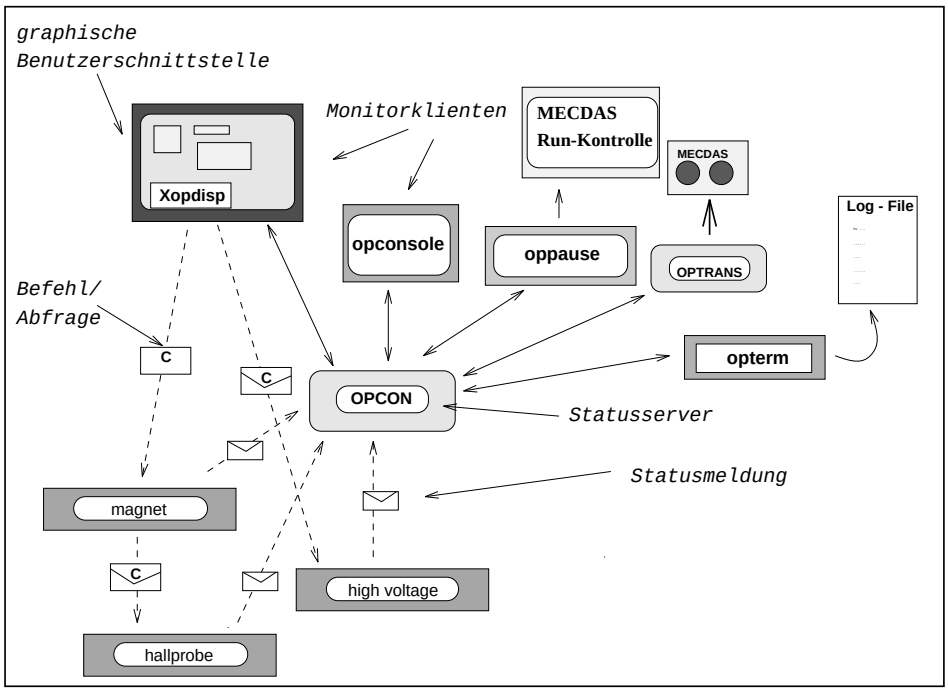


Abb.5.2 zeigt schematisch den Aufbau des Überwachungssystems der Drei-Spektrometer-Anlage . Die Pfeile deuten den Nachrichtenaustausch zwischen den Objekten an. Die Kommunikation erfolgt entweder über MUPIX (gestrichelte Pfeile) oder über TCP/IP (durchgezogene Pfeile).

zeichnet; an *Konsolenterminals* hingegen werden nur Sondermeldungen weitergeleitet. Um zu vermeiden, daß durch Fehlbedienung Alarmmeldungen ausgeblendet werden, ist bei Konsolenterminals keine Geräteauswahl möglich. Programme, die beim Statusserver eine Daueranfrage angemeldet haben, werden als *Monitorklienten* bezeichnet. Sie können mehrere Monitorterminals und Konsolenterminals enthalten. Beispiele für Monitorklienten werden unten (S.50) aufgeführt.

5.2 Die Software-Realisierung

Die skizzierte Architektur aus zentralen und dezentralen Elementen wurde als ein System aus mehreren Programmen und Unterprogrammbibliotheken implementiert, die zusammen ein Software-Werkzeugkasten namens *Opcon-Toolkit*[4] bilden. Sein zentraler Bestandteil, der Statusserver, wird durch das Programm `opcon`[5] realisiert.

5.2.1 Die Klienten des Statusservers

Eine Reihe standardisierter Klienten ist als Teil des Gesamtsystems fest implementiert und wird beim Beginn der Messung automatisch gestartet. Darüberhinaus kann der Experimentator weitere, im Einzelfall benötigte Monitorklien-

[4] Operator Control

[5] Allen Programmen und Funktionen des Opcon-Toolkits beginnen im Namen mit dem Präfix `op`.

ten selbst schreiben. Dafür steht die in [kra92b] dokumentierte Opcon-Toolkit-Bibliothek zur Verfügung, in der alle im Zusammenhang mit dem Monitorsystems verwendbaren Bausteine bereitgestellt sind.

Zu den standardisierter Klientenprogrammen gehören im wesentlichen:

- `opterm` schreibt alle einlaufenden Zustandsmeldungen im ASCII-Format auf ein Terminal oder in eine Datei. Zur Auswahl der Zustandsmeldungen wird diesem Programm ein regulärer Ausdruck als Kommandozeilenparameter übergeben. Die Meldungen werden bei der Ausgabe in standardisierter Weise, im sogenannten Opcon-Datenbank- oder kurz ODB-Format, ausgegeben. Damit ist eine einfache Weiterverarbeitung mit den üblichen, vom Betriebssystem bereitgestellten Programmen wie z.B. `grep, awk, sort` und `uniq` möglich. Zusätzlich enthält das Opcon-Toolkit ein eigenes Werkzeug zur Analyse solcher Dateien:
  - Mit dem Programm `opprint` lassen sich Zeitverläufe ausgewählter Geräteparameter graphisch darstellen. Zum Beispiel sind Abb.3.1 und die Abb.5.7 damit erstellt worden. Das Programm `opprint` ist insbesondere bei der Diagnose von Fehlersituationen sehr hilfreich.
- `opdisp` [ku95] baut auf dem Programm `opterm` auf. Es hat eine insbesondere zur Statusanzeige auf Kontrollmonitoren geeignete Ausgabe. Die Abb.5.1 wurde durch dieses Programm erstellt.
- `opconsole` enthält ein Konsolenterminal und schreibt die Sondermeldungen aller Geräte auf ein Terminal oder in eine Datei. Die Abb.5.3 zeigt einige typische Beispiele. Bei der Ausgabe auf einem Terminal wird bei Alarmmeldungen noch zusätzlich ein akustisches Signal erzeugt ("Klingelton").

```
WARNING  speca/magnet/quad/current  0  15:17:19  08.03.94 :
   error turn current off, current is off !!!

FAILURE  dev/camac/camac2/mod4032a  0  18:07:24  11.03.94 :
   dev/camac/camac2/mod4032a(lam_and_data_processing):
   Parity error in main  frame 61

ALARM  speca/vdc/hv2/cathode/u/get  3  07:01:22  16.03.94 :
   1.74507 below threshold 5800

FAILURE  dev/nmr/nmr1  0  09:29:04  16.03.94 :
   specb/magnet/d1/nmr: can't find resonance

ALARM  dev/current/2  0  15:07:28  17.03.94 :
   interlock occured 440271 (hex): !...!!!..!.......!...!.
```

Abb.5.3: Typische Ausgabe von Sondermeldungen mit dem Programm `opconsole` auf einem Terminal.

- `optrans` ist ein Schnittstellenprozeß, der alle Zustandsmeldungen an den Eventbuilder der Datenerfassung (siehe S.7.3.1) weiterleitet von dem aus diese Information zusammen mit den Ereignisdaten auf Band geschrieben werden.

Mit Hilfe der Funktionen des Opcon-Toolkits kann man diese Information im Rahmen eines Analyseprogramms aus den Daten extrahieren und bei der Datenanalyse berücksichtigen. Zur Extraktion steht bereits das Programm `opextract` zur Verfügung. Es produziert eine Datei im ODB-Format.

- `oppause` ist ein Klientenprogramm, das auf Sondermeldungen reagiert. Falls es sich bei der Meldung um einen Alarm handelt, startet es ein Programm, das weitergehende Maßnahmen einleitet, das z.B. die laufende Messung anhält. Die Auswahl des Programms erfolgt auf Grund des Absendernamens in der Statusmeldung. Die Behandlung von Sondermeldungen in einem Programm außerhalb des Statusservers hat den Vorteil, daß der Statusserver seinen normalen Dienst weiter fortsetzen kann, während das gestartete Programm abgearbeitet wird. Darüber hinaus kann man damit sehr flexibel auf Meldungen reagieren. Die nachfolgende Tabelle zeigt eine Konfigurationsdatei für `oppause`:

```
.*vdc/hv./cathode/u/get  runstop

.*/magnet/.*current      runstop
```

  Im Beispiel wird bei allen Alarmmeldungen einer Hochspannungsversorgung der Kammern oder eines Magnetnetzgeräts die laufende Messung angehalten.

Die beiden Programme `opterm` und `opconsole` realisieren zusammen ein einfaches elektronisches Logbuch. Das Gesamtsystem ist in Abb.5.2 dargestellt.

Das Opcon-Toolkit enthält auch die Funktionen zum Verpacken und Versenden der Statusmeldungen an den Statusserver. Auf die Kodierung der Statusmeldungen wird im nächsten Abschnitt eingegangen.

5.3 Einige Realisierungsdetails

5.3.1 Klassenbeschreibungsdatei und Struktur einer Statusmeldung

Klassenbeschreibungsdatei

Um eine geräteunabhängige Bearbeitung von Statusmeldungen, z.B. deren Darstellung auf dem Bildschirm, zu ermöglichen, müssen Statusnachrichten in einheitlicher Weise versandt werden. Dabei muß die in den Nachrichten enthaltene Statusinformation so kodiert sein, daß sie auch außerhalb eines Treiberprogramms den physikalischen Geräteparametern zugeordnet werden kann. Gleiches gilt für die im elektronischen Logbuch enthaltene Information.
Die Lösung dieser Aufgabe ist eng verbunden mit dem im Abschnitt 3.2.1 beschriebenen Erfordernis allgemeiner, tabellen-gesteuerter Benutzerschnittstellenprogramme, da diese ebenfalls in standardisierter Weise mit den Geräte-Objekten kommunizieren müssen.

Zu diesem Zweck wurde das Konzept der Klassenbeschreibungsdatei (*"Class Description Files"(CDF)*) entwickelt. Die CDF-Datei beschreibt die Struktur eines

Klasse:	*Stromgerät*
Klassenschlüssel:	`_CurrentCl`

Zustandsvariablen:

Name:	Variablenschlüssel:	Datentyp:
Strom (stell)	`_CurrentDac_VK`	FLOAT
Strom (ist)	`_CurrentAdc_VK`	FLOAT
Schalter	`_OnOff_VK`	BOOL

Dienste:	Dienstschlüssel		Parameter:	
	Variablenschlüssel:	Aktionsschlüssel:	Eingang:	Ausgang:
einschalten	`_OnOff_VK`	`_ON_AK`	---	BOOL
ausschalten	`_OnOff_VK`	`_OFF_AK`	---	BOOL
einstellen	`_CurrentDac_VK`	`_SET_AK`	FLOAT	BOOL
auslesen	`_CurrentAdc_VK`	`_GET_AK`	---	FLOAT
auslesenStell	`_CurrentDac_VK`	`_GET_AK`	---	FLOAT

Abb.5.4: Klassenbeschreibungsdatei für das Netzgerät eines Magneten (siehe auch Abb.3.3). Die Schlüssel sind in der Abbildung in symbolischer Notation angeben, d.h. die mit einem Unterstrich _ beginnenden Größen repräsentieren einen Integer. Auch jeder Klasse ist ein Schlüssel zugeordnet.

bestimmten Gerätetyps (Klasse) aus der Perspektive eines Bedieners (Experimentator); sie ist unabhängig von der Implementation der Geräteklasse. Die CDF-Datei definiert ein Protokoll, das die Verteilung und Speicherung von Geräteparametern sowie die Kommunikation mit einem Geräte-Objekt in allgemeiner Weise ermöglicht.

Dazu enthält das CDF die Liste der Variablen, die den Zustand eines Geräts beschreiben; diese Variablen werden als *Zustandsvariablen* bezeichnet. Jede Zustandsvariable ist zusammen mit deren Datentyp und einer ihr als Schlüssel zugeordneten Nummer, dem *Variablenschlüssel*, aufgeführt (siehe Abb.5.4).

Zusätzlich enthält der CDF-File eine Liste aller Dienste (Aktionen), die von einem Objekt dieser Geräteklasse angeboten wird. Jedem Dienst ist ein Schlüssel zugeordnet, der einen Variablenschlüssel und einen *Aktionsschlüssel* (Nachrichtentyp) enthält. Der Variablenschlüssel ist Teil des *Dienstschlüssels*, um zu dokumentieren, daß i.a. Aktionen dazu dienen, den Wert einer Zustandsvariablen *abzufragen* oder zu *verändern*. Jede Aktion ist zusammen mit ihren Ein- und Ausgabeparametern aufgeführt. Mit Hilfe des Operationsschlüssels ist es möglich, ein Protokoll zur Kommunikation mit Gerätetreibern zu definieren. Ein allgemeiner Mechanismus zur Abbildung dieses Protokolls auf die Dienste einer C++-Klasse (*Methode*) wurde im Rahmen weiterer Arbeiten [ku95, ha93, ma93, stef93] entwickelt und aufbauend auf den Routinen des Opcon-Toolkits in Form einer C++-Bibliothek und in Form eines Satzes von Datenbanktabellen implementiert.

Die Anzahl der erlaubten Datentypen für die Ein- und Ausgabeparameter von Aktionion sowie für eine Zustandsvariable wurde eingeschränkt, um in einfacher Weise ein Protokoll zum Datenaustausch mit den Geräteobjekten realisieren zu können. Es stellt sich heraus, daß sich die Variablen aller bisher eingesetzten Geräte mit einer kleinen Anzahl an Datentypen beschreiben lassen. Sie bilden die Menge der Basisdatentypen des ECSs, die im Anhang B.2 dargestellt sind. Zur Definition eines Protokolls ist jedem Datentyp eine Zahl zugeordnet, der

sogenannte *ECS-Datenschlüssel*. Indem der Datenschlüssel als Teil der Daten in einer Statusnachricht mitgeliefert wird, ist es ohne großen Aufwand möglich, die Typinformation zu verteilen; die Daten sind somit selbstbeschreibend.

Für den Austausch zwischen Rechnern (Prozeßrechner, Workstations) mit unterschiedlicher Binärdarstellung müssen die Daten in einem maschinen-unabhängigen Format kodiert werden. Hierzu gibt es mit dem XDR-Format (eXternal Data Representation) einen de facto Standard[6] [cor91]. Mit Hilfe des XDR-Standards wurden die zur machinen-unabhängigen Verpackung der Nachrichten notwendigen Routinen erstellt; sie sind Teil der Opcon-Toolkits-Bibliothek, die auch noch eine Reihe von Routinen zur Verarbeitung der ECS-Basisdatentypen, z.B. Ein- und Ausgabefunktionen, enthält.

Die Abb.5.4 zeigt als Beispiel das CDF eines Netzgeräts.

Statusmeldungen

Mit Hilfe des Objektnamens und des Variablenschlüssels können die Angaben in einer Statusmeldung eindeutig den jeweiligen Geräteparametern zugeordnet werden. Eine Statusmeldung setzt sich aus folgenden Komponenten zusammen:

Meldungstyp: Der Meldungstyp dient zur Unterscheidung von Zustands- und Sondermeldungen.

Zeitstempel: Er gibt den Zeitpunkt an, zu dem die Nachricht abgesandt wurde. Damit läßt sich die zeitliche Entwicklung eines Parameters verfolgen.

Objektnamen: Der (symbolische) Name identifiziert ein Geräte-Objekt und erlaubt die Zuordnung zur betreffenden Geräteklasse.

Klassenschlüssel: Dieser gibt die Klasse an, der das Objekt angehört (optional).

Variablenschlüssel: Dieser Eintrag gibt die physikalische Bedeutung der in der Nachricht enthaltenen Information an.

Kommentar: Eine Sondermeldung enthält (optional) einen String zur näheren Beschreibung (s.Abb.5.3).

Daten: Die Daten enthalten den aktuellen Wert der Zustandsvariablen auf den sich die Meldung bezieht (optional bei Sondermeldungen). Die Daten sind selbstbeschreibend in einem maschinen-unabhängigen Format verpackt.

5.3.2 Statusserver

Dieser Abschnitt beschreibt die wesentlichen Eigenschaften der Implementierung des Programms `opcon`.

[6] Der XDR-Standard hat zwei Aspekte. Er definiert erstens die Darstellung von Daten auf einheitliche Weise und definiert zweitens eine Sprache zur Beschreibung von beliebig komplexen Datenstrukturen. Zum Standard gehört auch eine Bibliothek aus C-Funktionen, die für die Konversion der Daten von der lokalen (maschinen-abhängigen) in die XDR-Darstellung und umgekehrt sorgen. Diese Bibliothek ist mittlerweile auf allen Rechnern implementiert.

Kommunikation zwischen Klienten und Statusserver

Da die Anzahl der Monitorklienten des Statusservers in der Regel so klein ist, daß die maximale Zahl erlaubter Verbindungen pro Prozeß nicht überschritten wird, ist es möglich verbindungsorientierte Protokolle, wie z.B. TCP/IP einzusetzen, die sich insbesondere für die gerichtete und dauerhafte Kommunikation zwischen dem Statusserver und dessen Monitorklienten eignen:

- Bei einem verbindungsorientiertem Protokoll kann jeder der Kommunikationspartner feststellen, ob der jeweils andere Partner noch bereit ist, Daten zu empfangen. Ist dies nicht der Fall, so kann der Sender schließen, daß der Adressat keine Nachrichten mehr benötigt, und die Kommunikation beenden, d.h. die Verbindung schließen. Dies ist für den Statusserver von Bedeutung, weil die Klienten Ressourcen innerhalb des Servers verbrauchen, die der Statusserver freigeben kann, wenn ein Klientenprogramm terminiert. Auf der anderen Seite liegt für ein Klientenprogramm ggf. eine Fehlersituation vor, wenn der Server terminiert. Da das Terminieren des Servers zum Abbruch der Kommunikationsverbindung führt, kann diese Fehlersituation von den Klientenprogrammen erkannt werden.

- Nachdem ein Klient ein Monitorterminal beim Statusserver angemeldet hat, bekommt er umgehend von diesem alle Statusmeldungen geliefert, die das Auswahlkriterium des Terminals erfüllen und die bereits in der prozeßinternen Datenbank des Servers gespeichert sind. Dies erfordert ggf. den Transport einer größeren Datenmenge, für den verbindungsorientierte Protokolle besser geeignet sind als das auf kurze Nachrichten ausgelegte MUPIX-Protokoll.

Aus diesen Gründen erfolgt die Kommunikation zwischen den Monitorklienten und dem Statusserver nicht über MUPIX, sondern über TCP/IP.

Weil der Statusserver sequentiell arbeitet, ist es allerdings notwendig, Vorkehrungen zu treffen, um die auf Seite 39 geschilderten Verklemmungen, die bei der Verwendung eines verbindungsorientierten und synchronen Protokolls entstehen können, zu vermeiden. Deshalb erfolgt das Senden auf seiten des Statusservers nicht blockierend, d.h. eine I/O-Operation, die nicht sofort ausgeführt werden kann, wird abgebrochen. In diesem Fall löst der Statusserver die Kommunikationsverbindung mit dem betreffenden Klienten. Der Klient erkennt diese Situation und gibt eine entsprechende Fehlermeldung aus.

Obwohl die Kommunikation zwischen dem Statusserver und seinen Klienten auf dem TCP/IP-Protokoll beruht, erfolgt auch hier die Adressierung des Statusservers von seiten der Klientenprogramme mit einem symbolischen Namen an Stelle einer Protokolladresse. Dies wurde dadurch ermöglicht, daß der MUPIX-Nameserver, u.a. im Hinblick auf Anwendungen wie diese die dynamische Zuordnung von symbolischen Namen auch zu TCP/IP-Protokolladressen verwalten kann (siehe Abschnitt 4.4.6). Die Adreßauflösung erfolgt nach dem im Abb.4.9 dargestellten Prinzip. Durch die Verwendung des Nameservers wird der Verwaltungsaufwand zur Verteilung der Nameserveradresse minimiert.

Arbeitsabläufe innerhalb des Statusservers

Alle Aktionen des Statusservers werden, wie bei allen Serverprozessen, durch Ereignisse auf dessen I/O-Kanälen eingeleitet. Es handelt sich dabei entweder um die Ankunft einer neuen Statusnachricht, um eine Aufforderung zum Verbindungsaufbau durch einen Monitorklienen oder um eine neue Anfrage auf einer bereits bestehenden Verbindung. Der Kontrollfluß innerhalb des Statusservers zerfällt deshalb im wesentlichen in drei weitgehend voneinander unabhängige Ablauffolgen, die alle an einer zentralen Stelle im Programm angestoßen werden, nämlich dort, wo das Multiplexing auf den I/O-Kanälen vorgenommen wird. Hierzu dient der auf S.34 beschriebene Multiplexer-Mechanismus (Notifier). Da der Statusserver, abgesehen vom asynchronen Nachrichtenempfang, streng sequentiell arbeitet, reagiert er zu jedem Zeitpunkt auf eines dieser Ereignisse.

Ankunft einer Statusnachricht

In den meisten Fällen handelt es sich bei dem I/O-Ereignis um die Ankunft einer neuen Statusnachricht auf dem MUPIX-Kanal des Statusservers. Diese Nachricht wird ausgepackt und im Falle einer Sondermeldung an alle Konsolenterminals weitergeleitet. Die Verwaltung der Konsolenterminals erfolgt mit einer linearen Liste. Handelt es sich bei der Nachricht um eine Zustandsmeldung, so muß zunächst herausgefunden werden, welcher der Monitorklienten an dieser Meldung interessiert ist. Dazu wird für jeden Monitorklienten die zugehörige Liste der angemeldeten Monitorterminals durchlaufen. Dabei wird bei jedem Terminal überprüft, ob der Absendername in der Meldung die Filterbedingung erfüllt[7]. Falls ja, leitet der Statusserver die Meldung zusammen mit einer sogenannten *Terminalkennzahl* an den Klienten weiter. Die Terminalkennzahl wird bei der Anmeldung eines Terminals zusammen mit der Filterbedingung vom Klienten selbst geliefert. Dadurch wird erreicht, daß auf Klientenseite die Meldungen für die verschiedenen Terminals einfach zu unterscheiden sind, ohne erneut aufwendige Vergleiche mit Zeichenketten durchzuführen.

Die Zustandsmeldungen werden zusätzlich in eine prozeßinterne Datenbank des Statusservers eingefügt. Dabei ersetzen sie eine evtl. schon vorhandene Meldung über eine Variable eines Geräte-Objekts. Als Datenschlüssel dient der Absendername einer Meldung. Die interne Datenbank ist mit Hilfe eines sogenannten *Binary Threaded Tree* [knu68] aufgebaut, d.h. in Form eines binären Baumes, den man ohne Rekursion durchlaufen kann. Ein solcher Baum ist in Abb.5.5 dargestellt. Jeder Knoten im Baum ist im Falle des Statusservers darüber hinaus noch der Kopf einer Liste von Elementen, die die Meldungen für die einzelnen Variablen eines Objekts enthalten.

[7]Der Test, ob eine Zeichenkette durch einen regulären Ausdruck beschrieben wird, erfolgt mit Hilfe einer allgemeinen *Zustandsmaschine*, deren Zustandsübergänge aus dem regulären Ausdruck generiert (kompiliert) werden. Um Rechenzeit zu sparen, erfolgt das Kompilieren des regulären Ausdrucks nur einmal direkt nach Anmeldung eines Monitorterminals. Eine Zustandsmaschine (endlicher Automat) [hop79] ist ein System mit diskreten Ein- und Ausgaben, das sich immer in einem aus einer endlichen Anzahl von internen Konfiguration, den sogenannten Zuständen, befindet. Ein Zustand erfaßt alle Informationen, die sich aus den bisherigen Eingaben ergeben haben und die benötigt werden, um weitere Reaktionen des Systems zu bestimmen.

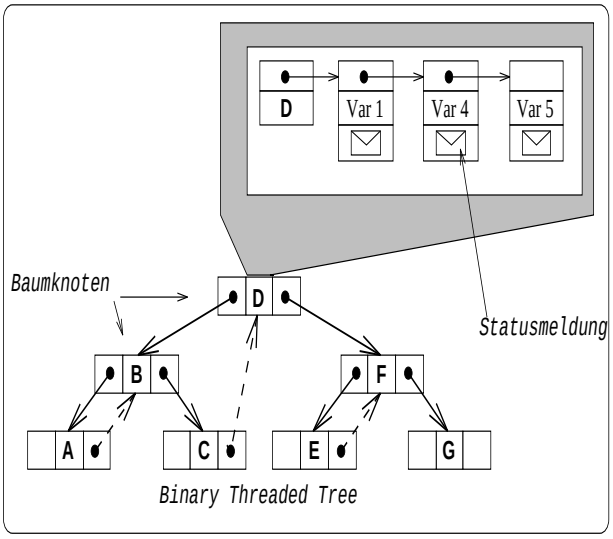


Abb.5.5: Organisation der internen Datenbank des Statusservers in Form eines Binary Threaded Tree. Beim Durchlaufen des Baumes vermeidet man eine rekursives Vorgehen dadurch, daß jeder Knoten des Baumes zusätzlich eine Komponente enthält, die auf den jeweiligen Nachfolgeknoten zeigt. Rechts oben ist angedeutet, daß jeder Knoten wiederum Kopf einer Liste von Statusmeldungen ist, die die Meldungen für die verschiedenen Zustandsvariablen (Var i) enthält. Die Buchstaben A bis G stehen symbolisch für Gerätenamen.

Eine Baumstruktur bietet sich an, weil die Anzahl der Vergleiche zum Wiederauffinden eines Elements im Mittel etwa $log(n) - 1$ beträgt, wobei n die Anzahl der Elemente im Baum ist [wi83]. Bei linearen Listen muß man hingegen im Mittel mit $n/2$ Vergleichen rechnen.

Eine mögliche Alternative zu den Baumstrukturen wären sogenannte *Hashing-* oder auch *Streuspeicher-Algorithmen*[wi83]. Die bei dieser Technik verwendete Datenorganisation ist die Array-Struktur, wobei der Schlüssel über eine Transformation in die Array-Indizes überführt wird. Die Anzahl der Vergleiche beträgt im Mittel nur 1.05 bei einer Auslastung des Arrays von 10% (10.5 bei 95%) unabhängig von der Anzahl der Elemente.
Der Nachteil gegenüber den dynamisch erweiterbaren Baumstrukturen besteht in der festen Größe des Speicherarrays, die man während der Laufzeit nicht dem Bedarf anpassen kann. Um eine schlechte Auslastung des Speichers zu vermeiden, muß die Zahl der Datenelemente vorher gut abgeschätzt werden. Selbst dann muß man die Dimension der Tabellen etwas zu groß wählen, um eine gute Leistung des Algorithmus zu erzielen. Ein weiterer Nachteil besteht darin, daß das Durchlaufen aller Datenelemente in einer Hash-Tabelle sehr aufwendig ist, falls man nicht für eine zusätzliche Verkettung der Elemente untereinander sorgt. Das geht allerdings zu Lasten der Rechenzeit beim Einsortieren. Deshalb wurde für die Anwendung im Statusserver die Baumstruktur einer Hash-Tabelle vorgezogen. Im Gegensatz dazu findet die Hash-Tabelle beim Nameserver von MUPIX Verwendung.

Anforderung zum Verbindungsaufbau

Die Anforderungen, eine Verbindung mit einem Klienten aufzubauen, erreichen den Statusserver auf dem Kanal, den er beim MUPIX-Nameserver angemeldet hat. Der Statusserver reagiert hierauf, indem er zunächst die Anforderung akzeptiert, und dann auf einem neuen Kanal eine Verbindung mit dem Klienten herstellt. Dadurch bleibt der beim Nameserver bekannte Kanal frei für neue Anfragen. Zur Verwaltung dieses Kanals wird ein Objekt der Klientklasse erzeugt.

Es reagiert auf Anfragen von seiten der Klientenprogramme und wird über den Notifier-Mechanismus aktiviert.

Klientenanfragen

Die Art der Anfragen hängt vom Typ des Klienten ab. Monitorklienten können nur zusätzliche Terminals auf einer Verbindung anmelden. Dabei bekommen sie optional bereits die gespeicherten Meldungen zugesandt, die die Filterbedingung des Terminals erfüllen. Abfrageklienten haben die Möglichkeit,

- die bereits gespeicherten Meldungen eines Geräte-Objekts abzufragen,
- eine Liste aller bekannten Geräte-Objekte anzufordern,
- während der Laufzeit des Statusservers, das Mitschreiben von Debuginformation des Servers ein- und auszuschalten. Dabei schreibt der Statusserver programminterne Zustände in eine Datei. Diese Eigenschaft wurde insbesondere in der Entwicklungsphase des Serverprogramms bei der Fehlerdiagnose benötigt.

Das Program `opman` ist ein einfacher Abfrageklient mit der oben beschrieben Funktionalität. Es gehört zum Standardsystem des Opcon-Toolkits.

5.3.3 Die Anwendungsschnittstelle

Auf der Basis der implementierten Software können Anwender in sehr übersichtlicher Weise Programme für spezielle Zwecke erstellen. Beispielhaft zeigt Abb.5.3.3 den Quelltext eines Programms, das die Statusmeldungen aller Geräte, die zum Spektrometer A gehören und beim Statusserver einlaufen, auf einem Bildschirm ausdruckt. Es arbeitet deshalb als Monitorklient. Zur Bedeutung der Programmschritte im einzelnen:

Mit der Deklaration des Objekts `aServer` vom Typ `OPmon`[8] baut das Programm die Verbindung zu einem Statusserver[9] im ECS mit dem Namen "Server" auf; damit stellt das Programm einen Monitorklienten dar.

Mit der Deklaration `OPterm` meldet der Klient beim Server ein Monitorterminal an, das als Filterbedingung den *regulären Ausdruck* " speca.*" enthält. Dadurch werden in Zukunft alle Zustandsmeldungen aller Geräte von Spektrometer A an den Klienten weitergeleitet; beim Eintreffen einer Meldung soll die Funktion `output` aufgerufen werden, die die Meldung auf dem Bildschirm ausgibt.

Auf die gleiche Weise kann der Klient weitere Monitorterminals anmelden.

[8]Das Präfix OP wird verwendet um die Zugegehörigkeit zum OPCON-Toolkit anzuzeigen. Klassennamen werden i.a. groß, Objektnamen klein geschrieben.

[9]Grundsätzlich kann ein Klientenprogramm mit mehreren Statusservern arbeiten. Das Arbeiten mit mehreren Statusservern ist insbesondere dann erforderlich, wenn Statusinformation aus mehreren (MUPIX-)Domänen verarbeitet werden soll, z.B. neben dem Bereich der Drei-Spektrometer-Anlage auch aus dem des Beschleunigers. Es ist daher erforderlich, daß sowohl der Name des Statusservers als auch der MUPIX-Domäne beim Aufbau der Verbindung durch den Klienten mit angegeben wird.

Danach befindet sich das Programm solange in einer Schleife bis die Multiplexerroutine (`Notify_run`) des Notifiers[10] (s.S.34) auf, der die Grundlage der Klientenbibliothek darstellt, mit einem Fehler zurückkehrt. Der Notifier sorgt zusammen mit Funktionen aus der Opcon-Toolkit-Bibliothek für den Aufruf der Ausgaberoutine `output` beim Eintreffen einer Statusmeldung; dafür werden die Nachrichten ausgepackt und mit Hilfe des des mitgelieferten Terminalschlüssels dem entsprechenden Monitorterminal zugeordnet. Dann wird die vom Terminal spezifizierte Routine aufgerufen.

```
int main() {
  OPmon  aServer("Server", "ecs"); // Verbindungsaufbau
  OPterm aTerminal(aServer, " speca.*", output); // Terminal

  while (Notify_alert() == OK)  // Schleife
    ;
  return 0;
}

void output(caddr_t data, aStatusReport * rp, aBasicData * dp)
{
  cout << *rp << " " << " " << *dp << endl; // Ausgabe
}
```

Abbildung 5.6: Quelltext eines Monitorklienten in C++

Das Programm in Abb.5.3.3 enthält keinerlei Anweisungen darüber, wie am Ende des Programms zur Erledigung von "Aufräumarbeiten", wie z.B. das Abbauen der Verbindung zum Statusserver, zu verfahren ist. Auf derartige Anweisungen kann verzichtet werden, weil hierbei die besonderen Eigenschaften einer objektorientierte Sprache wie C++ zum Tragen kommen:

Es wird dabei ausgenutzt, daß alle nur innerhalb eines bestimmten Programmbereichs gültigen Variablen beim Verlassen desselben vernichtet werden. Dies gilt insbesondere für Variablen (Objekte) eines vom Benutzer definierten Datentyps (Klasse); hier wird beim Vernichten eines Objekts eine entsprechende klassenspezifische Funktion, der sogenannte *Destruktor,* aufgerufen, der die "Aufräumarbeiten" erledigt. Da die Destruktoren automatisch durch den Compiler ins Programm eingefügt werden, lassen sich mit Hilfe dieses Mechanismus Software-Bibliotheken erstellen, die sehr robust gegenüber Fehlbenutzung sind, wie z.B. das Vergessen der "Aufräumroutinen", und eine sehr einfache Schnittstelle besitzen.

5.3.4 Das Logbuch-Programm

Das Konzept des Monitorsystems ist so angelegt, daß der Gesamtzustand der Anlage jederzeit auf der Basis einer Hierarchie von abstrakten Zustandsmaschinen, die selbst Teil der Geräte-Objekte sind, gegeben sein sollte. Zum Beispiel

[10] Durch die Verwendung des Notifiers wird es ermöglicht, auch anderweitige I/O-Kanäle zu berücksichtigen, um z.B. auf Benutzereingaben über die Tastatur oder eine Maus zu reagieren. Von dieser Möglichkeit wird z.B. im Benutzerschnittstellenprogramm Xopdisp Gebrauch gemacht [stef93].

kann man damit vor jeder Messung selbst feststellen, ob die Anlage meßbereit ist, indem man den Zustand des Objekts, das am höchsten in der Hierarchie steht, überprüft.
Dies setzt voraus, daß der komplette experimentelle Aufbau innerhalb des Rechners durch eine aus Geräte-Objekten aufgebaute Hierarchie widergespiegelt wird, die nicht nur auf Schnittstellenumsetzer und einzelne kleinere Geräte beschränkt ist, sondern alle Komponenten der Anlage enthält, wie z.B. das Magnetsystem eines Spektrometers.

Für die bisher geschilderte Funktionalität benötigt man mindestens zwei Zustände als Teil der abstrakten Zustandsmaschine: den sogenannten *An-Zustand* und den sogenannten *Bereit-Zustand*. Ein Geräte-Objekt ist im *An-Zustand*, wenn es prinzipiell bereit ist, neue Aufträge entgegenzunehmen, wohingegen es sich in dem *Bereit-Zustand* befindet, wenn die Werte aller Geräteparameter sich in einem Bereich befinden, in dem es sinnvoll ist, eine Messung durchzuführen.

Die Einstellung der Geräte hängt i.a. vom durchzuführenden Experiment ab und wird im folgenden als (*Setup*) bezeichnet. Der Setup sollte in einer Datenbank abgelegt sein und darin mit Hilfe eine symbolischen Namens, dem sog. *Setup-Namen*, charakterisiert werden, um die Einstellung aller Geräte über diesen Namen vornehmen zu können.

Da bei der Entwicklung des bestehenden Steuerungssystems die direkte Ansteuerung der Geräte im Vordergrund stand [ku95], wurde das dargestellte Konzept von Zustandsmaschinen innerhalb der Geräte-Objekte nicht (durchgehend) realisiert. Insbesondere fehlt noch die darin enthaltene Möglichkeit, Einstellungen anhand von symbolischen Namen vorzunehmen. Außerdem fehlen einige der übergeordneten Geräte-Objekte.

Deshalb wurde es notwendig, dem Monitorsystem ein zusätzliches Dienstprogramm namens `logbook` beizufügen. Dieses Programm wird beim Beginn einer Messung (Run) automatisch gestartet[11] und überprüft parallel zur Messung den Zustand verschiedener Geräte mit einer vorgegeben Konfiguration. Der Vergleich erfolgt parallel zur Messung, weil die Auslese einiger Parameter längere Zeit in Anspruch nimmt. Jede Konfiguration wird mit Hilfe des Setup-Namens ausgewählt. Ob die Überprüfung überhaupt stattfindet, wird an Hand des Runtyps (Produktions- oder Testrun) entschieden. Unabhängig davon wird jeder Run zusammen mit dessen Typ und Setup-Namen in eine Tabelle (Datei) eingetragen, somit hat man eine Grundlage, um Informationen mehrerer Runs einer Konfiguration zu verarbeiten.

[11] Da das an der Drei-Spektrometer-Anlage eingesetzte Datenerfassungssystem als reines Auslesesystem konzipiert wurde, kennt es als größte logische Einheit nur einen Run [kry95]. Die Systemkonfiguration erfolgt nahezu vollständig innerhalb von Programmquellen bzw. Kommandoprozeduren. Somit gibt es keine (persistente) Information, die außerhalb der Ausleseprogramme zugreifbar ist. Deshalb mußten die für die Datennahme verantwortlichen Kommando-Prozeduren für die Verwendung des `logbook`-Programms erweitert werden. Insgesamt sollte auch die Run-Kontrolle auf das oben beschriebene tabellengesteuerte (Datenbank-orientierte) Konzept umgestellt werden.

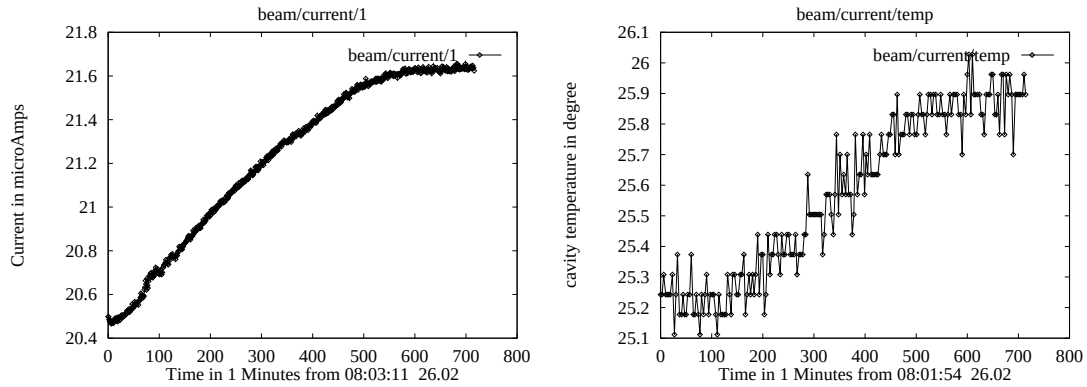


Abb.5.7: Das linke Bild zeigt die zeitliche Entwicklung des mit Hilfe eines HF–Resonators bestimmten Strahlstroms von MAMI. Der Elektronenstrahl facht im Resonator eine HF-Welle an (TM_{010}-Mode), aus der über eine Antenne eine Leistung ausgekoppelt wird, die proportional zum Quadrat des Elektronenstroms ist [mit92]. Der Linearitätsfehler des Meßsignals beträgt $\pm 0.5\%$ ($\pm 3\%$) der Anzeige für Strahlströme größer (kleiner) als 1 μA. Bei den bisherigen Messungen wurden statistische Schwankungen von ± 10 nA beobachtet. Wegen der Temperaturabhängigkeit der Resonanzkurve des Resonators muß dieser mit Hilfe eines Wasserkühlkreises auf konstanter Temperatur gehalten werden. Das dazu ursprünglich installierte System konnte die geforderte Temperaturstabilität von $\pm 1°C$ jedoch nicht gewährleisten; es wurden Schwankungen von $\pm 5°C$ beobachtet. Das rechte Bild zeigt einen Ausschnitt des Temperaturverlaufs über den gleichen Zeitraum wie im linken Bild. Der Verlauf des gemessenen Stroms korreliert zeitlich mit der Kühlwassertemperatur. Daß diese Korrelation dem tatsächlichen physikalischem Zusammenhang entspricht kann man durch die Hinzunahme weiter vorhandener Strommonitore (siehe S.94) belegen. Die Mängel im Kühlkreislauf wurden mit Hilfe des Monitorsystems entdeckt, sie sind inzwischen beseitigt worden [sch95].

5.4 Erfahrungen und Erweiterungsmöglichkeiten

Die Programme und Funktionen, die im Rahmen dieser Arbeit zum Aufbau des Steuerungs- und Überwachungssystems der Drei-Spektrometer-Anlage entwickelt wurden, sind seit Beginn der Experimentierphase im Einsatz und haben sich dabei sehr gut bewährt. Damit ist die ursprüngliche Zielsetzung Werkzeuge zur Verfügung zu stellen, die im Rahmen des Steuerungssystems den Zustand der Anlage erfassen, protokollieren und dem Experimentator verfügbar machen, erfüllt.

Die auf der Basis der entwickelten Konzepte entstandene Softwarebibliothek bildet die Grundlage aller weiteren im Steuerungssystem eingesetzte Programme. Sie enthält zusammen mit der MUPIX-Bibliothek einen Großteil der Software, die zur Implementation von Gerätetreibern notwendig ist.

Das umfangreiche, automatisch geführte Protokoll hat sich auch außerhalb von Meßperioden bei der Auswertung von Daten als nützlich erwiesen und zum Verständnis und damit zur Verbesserung der Anlage beigetragen. Das in Abb.5.7 dargestellte Beispiel illustriert eine dieser Anwendungen.

Ausblick

Aus den beim Einsatz in den Strahlzeiten gemachten Erfahrungen, ergeben sich auch einige Verbesserungs- und Erweiterungsmöglichkeiten, die teilweise das gesamte On-line System betreffen:

a) Die einfache Ausgabe von Fehler- und Alarmmeldungen auf einem im Kontrollraum befindlichen Terminal reicht nicht in allen Fällen zur Information des Experimentators aus. Das ist insbesondere dann der Fall, wenn auf Grund von dauerhaft unzuverlässig arbeitenden Komponenten die Anzahl der Meldungen auf dem Schirm so groß wird, daß wichtigere Meldungen übersehen werden. Hier könnte auf einfache Weise z.B. eine graphische Darstellungsweise Abhilfe schaffen, bei der die Komponenten für die schon eine Fehlermeldung eingetroffen ist, farblich von den anderen unterschieden werden. Hieran wird im Rahmen einer Diplomarbeit gearbeitet[cap95].

b) Die Verpackungsroutinen für die Datentypen und Statusnachrichten wurden bisher mit Hilfe des XDR-Compilers generiert. Indem man in den so erzeugten Routinen die höheren Funktionen der XDR-Bibliothek durch solche aus den niedrigen Schichten ersetzt, kann das das Ver- und Entpacken der Daten beschleunigt werden.

c) In das ECS sollte das auf Seite 60 vorgestellte Konzept der Zustandsmaschine integriert werden. Dazu muß das ECS durch die noch fehlenden übergeordneten Geräte-Objekte, z.B. für die Kammer- und Triggerdetektoren, vervollständigt werden.

Kapitel 6

Das NMR-System der Spektrometer

Damit das Einstellen und Messen der Spektrometerfelder vollständig in das allgemeine Steuerungs- und Überwachungssystem integriert werden kann, müssen alle dazu erforderlichen apparativen Größen voll automatisch vom Rechner erfaßt werden; dies gilt insbesondere für die Magnetfeldmessung mit dem Kernresonanzverfahren (NMR). Im Rahmen dieser Dissertation wurde ein Konzept für die Magnetfeldmessung für die Drei-Spektrometer-Anlage entwickelt und sowohl im apparativen Bereich als auch auf der Software-Seite realisiert.

Einleitung

Bei den Spektrometern der Drei-Spektrometer-Anlage wird das Kernresonanzverfahren verwendet, um die Felder der Dipolmagnete mit hoher Genauigkeit reproduzierbar einzustellen. Die Magnetfeldmessung mit dem Kernresonanzverfahren ist wesentlich genauer als mit einer Hallsonde; sie ermöglicht eine Genauigkeit, die besser ist als die Genauigkeit der Energieeichung des Mikrotrons von $\frac{\Delta E}{E} = \pm 2 * 10^{-4}$ [hert92].

Für die Drei-Spektrometer-Anlage wird eine kommerziell erhältliche Kernresonanzanordnung [1] benutzt. Der Nachteil dieses Systems gegenüber anderen Verfahren, z.B. dem Wilkingverfahren [tat65], besteht darin, daß man mit mehreren Sonden arbeiten muß, um den gesamten Feldbereich abzudecken; Systeme, die auf dem Wilking-Verfahren beruhen, sind aber nicht auf dem Markt erhältlich. So ist jeder Dipolmagnet mit jeweils vier Sonden ausgestattet, die zusammen einen dynamischen Bereich von 0.09 bis 2.1 Tesla und damit den vollen Bereich der Magnete abdecken. Die Sonden werden im nachfolgenden entsprechend ihrer vom Hersteller spezifizierten Typennummer (zwei bis fünf) bezeichnet. Die Meßbereiche sind in der Tabelle 6.1 angegeben.

Der absolute Meßfehler der Kernresonanz-Sonden ist mit $\pm 5 ppm$ und der relative Fehler bei absolut homogenen Feldern mit $0.1 ppm$ vom Hersteller spezifiziert [met89]. Durch mehrmaliges Auslesen der installierten Sonden bei derselben Feldeinstellung[2], ergab sich ein relativer statistischer Fehler von kleiner als $\pm 2 ppm$ [3] Diese Unsicherheit ist klein im Vergleich zu allen sonstigen Fehlerquellen und kann deshalb für alle weiteren Betrachtungen vernachlässigt werden.

[1] PT2025 High Precision NMR, Metrolab Instruments S.A., Genf

[2] Zur Messung waren die Sonden dabei im homogenen Feldbereich von Dipolmagnet AD1 installiert.

[3] Der Unterschied zur Herstellerangabe ist auf Restinhomogenitäten des Feldes zurückzuführen.

Alle Sonden sind über ein zweistufiges Multiplexersystem an eine Zentraleinheit angeschlossen, das sogenannte Teslameter. Es enthält die gesamte weitere Elektronik. Das Teslameter ermöglicht ein automatisches Auffinden der Resonanz; es wird über eine serielle Schnittstelle gesteuert und ausgelesen. Aufbauend auf den allgemeinen Routinen des ECS wurden im Rahmen dieser Arbeit die dazu notwendigen Softwaremodule erstellt.

Während die kommerzielle Anordnung bei den Dipolmagneten der Spektrometer A und C direkt anwendbar ist, mußte wegen der starken lokalen Inhomogenität beim Clam-Shell Dipolmagneten dessen Feldgradient im Bereich der Kernresonanzsonde kompensiert werden, die zur Kompensation erforderliche Anordnung wurde im Rahmen dieser Arbeit entwickelt.

Die Implementation der Feldmessung in den beiden Spektrometertypen wird in den beiden nachfolgenden Abschnitten besprochen.

6.1 Die Feldmessung der Dipolemagnete von Spektrometer A

Für den Betrieb an Spektrometer A mußten die Halterungen für die jeweils vier NMR-Meßsonden in den beiden Dipolmagneten und für eine im selben Vakuum-Port[4] untergebrachte Hallsonde konstruiert und gebaut werden. Da sich die Sonden im Randbereich des Spektrometerfeldes befinden, sind die Feldwerte an den Sondenorten auf die im Innern der Spektrometerdipolmagnete zu eichen. Dafür wurde ein zusätzlicher Satz von NMR-Sonden mittels einer Haltestange mitten im Dipolmagneten installiert. Mit den Messungen mit denselben Sonden wurden auch die Erregungskurven für die Dipolmagnete (Tab.C.1, Tab.C.2 im Anhang) und die Kalibration der Hallsonden ermittelt.
Da das Einstellen aller Meßpunkte in dem zu überprüfenden Feldbereich von 0.1 - 1.5T sehr zeitaufwendig ist, wurde der Meßvorgang weitgehend automatisiert. Dafür wurde ein Computer-Programm namens `bigloop` erstellt, das auch für die Kalibrationsmessungen am Spektrometer C verwendbar[5] ist, da es vollständig über die Namen der einzelnen Geräte-Objekte konfiguriert wird. In diesem Zusammenhang wurden auch Programme zur schnellen Statusüberwachung der Spektrometer (`magstat`) und zur Magnetfeldeinstellung aller Magnete (`setCentMom`) für den Einsatz während einer Strahlzeit entwickelt. Die dabei erstellten Routinen zur Ansteuerung der Stromgeräte, Hallsonden und des Teslameters sind inzwischen in die Treiberobjekte [ku95] der jeweiligen Geräte eingebaut worden.

6.1.1 Messungen am Dipolmagneten AD1

Die Abb.6.1 zeigt den Verlauf der Differenz der Feldstärke, die zwischen der am Polschuhrand fest eingebauten NMR-Sonde (Port-Sonde) und der in der Mitte des Polschuhs installierten Referenzsonde in Abhängigkeit vom Feld B_r der Referenzsonde gemessen wurde. Es zeigt sich, daß die Feldstärke B_p am Ort der

[4] Einlass in die Vakuumkammer

[5] Die Messungen wurden parallel zur Niederschrift dieser Arbeit durchgeführt. Die Ergebnisse befinden sich deshalb noch im Anhang (Tab.C.4 und Tab.C.4).

Tabelle 6.1: Meßbereiche der verschiedenen Sondentypen und dazugehörige Bereiche des zentralen Impulses für die Spektrometer A und B. Die Angaben für Spektrometer B beziehen sich auf den Ort der Kernresonanzsonde (siehe auch S.75).

Typ	Meßbereich in Tesla	Impulsbereich in MeV/c	
		A	B
2	0.09 - 0.26	40 - 114	57 - 164
3	0.17 - 0.52	75 - 229	107 - 328
4	0.35 - 1.05	154 - 462	221 - 663
5	0.7 - 2.1	> 308	> 442

Port-Sonde erwartungsgemäß kleiner ist als in der Mitte des Polschuhs. Unterhalb einer Feldstärke von etwa 1.25 T ergibt sich eine Differenz, deren Effekt $\frac{\Delta B}{B}$ kleiner ist als $2 * 10^{-4}$, also unterhalb der Eichgenauigkeit des Mikrotrons liegt. Deshalb ist in Abb. 6.1 nur der Bereich von 1.05 T bis 1.55 T abgebildet, der mit dem Sondentyp 5 aufgenommen wurde[6]. Oberhalb von 1.25 T setzen Sättigungseffekte des Magnetstahls ein, die man recht gut mit Hilfe einer Modellrechnung [ko95], die ebenfalls in Abb.6.1 gezeigt ist, beschreiben kann.

Für die Einstellung der Spektrometerfelder [7] wird die Abhängigkeit der Differenz vom Feld der Port-Sonde benötigt, um die beiden Dipolmagnete auf das gleiche Innenfeld zu bringen. Dafür werden in der Steuersoftware als Eichkurven Polynome verwendet, deren Koeffizienten an die Daten angepaßt wurden, die in der folgenden Tabelle dargestellt sind.

	Sonde 5		Sonde 4	
$a_0[T]$	$4.8758 * 10^{-2}$	$\pm 1.4324 * 10^{-2}$	$-1.301 * 10^{-3}$	$\pm 5.06 * 10^{-4}$
$a_1[1]$	-0.343487	$\pm$ 8.1194 $*10^{-2}$	$9.1225 * 10^{-3}$	$\pm 3.5297 * 10^{-3}$
$a_2[1/T]$	0.985802	$\pm$ 0.189718	$-2.30869 * 10^{-2}$	$\pm 9.67939 * 10^{-3}$
$a_3[1/T^2]$	-1.47812	$\pm$ 0.233884	$2.93032 * 10^{-2}$	$\pm 1.30619 * 10^{-2}$
$a_4[1/T^3]$	1.22416	$\pm$ 0.16045	$-1.84266 * 10^{-2}$	$\pm 8.67996 * 10^{-3}$
$a_5[1/T^4]$	-0.53202	$\pm 5.8083 * 10^{-2}$	$4.5610 * 10^{-3}$	$\pm 2.2739 * 10^{-2}$
$a_6[1/T^5]$	$9.4944 * 10^{-2}$	$\pm 8.66961 * 10^{-3}$		

Tab.6.2: Koeffizienten der Polynomanpassung $B_r - B_p = \sum_j a_j * B_p^j$, wobei B_r die Feldstärke im Innern und B_p die von den Port-Sonden gemessene Feldstärke darstellen. Bei den beiden übrigen Sonden ist der Effekt kleiner als $7.5 * 10^{-5}$.

6.1.2 Messungen am Dipolmagneten AD2

Beim Dipolmagneten AD2 ist die Feldstärke im Innern über den gesamten Meßbereich kleiner als am Ort der Meßsonden im Vakuum-Port. In Abb.6.2 ist die Differenz der beiden Werte gegen die Feldstärke in der Mitte des Polschuhs aufgetragen. Der lineare Anstieg der Differenz im Bereich von 0.1 T bis 1.2 T hängt vermutlich mit der Verbiegung der Polschuhe auf Grund der zwischen Polschuh und Joch wirkenden magnetischen Kräfte [sch95] zusammen (Abb.C.1 im Anhang C.1). Diese Kräfte führen bei einer Erregung des Magneten von 1.5 Tesla zu einer

[6] Die Meßwerte aller Sonden sind in der Tabelle C.1 im Anhang aufgeführt

[7] Bei der Einstellung der Felder müssen nur solche Effekte berücksichtigt werden, die oberhalb der Energieungenauigkeit des Mikrotrons liegen. Darüber hinaus ist die Einstellgenauigkeit der Felder durch die Regelgenauigkeit der Netzgeräte limitiert; sie beträgt $\pm 5 * 10^{-5}$.

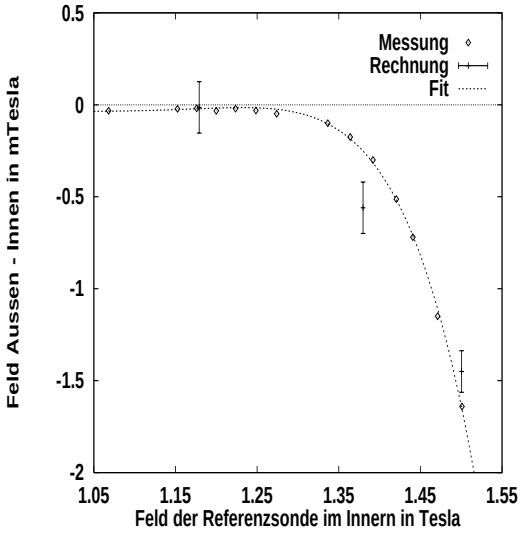


Abb.6.1: Differenz (in Millitesla) zwischen dem mit der Port-Sonde gemessenen Feld und der mit der Referenzsonde gemessenen Feldstärke in Abhängigkeit vom Feld der Referenzsonde des Dipolmagneten AD1. Der für die Modellrechnungen angegebene Fehler ergibt sich aus der Unsicherheit in der Position der Port-Sonde. Die Sonden im Vakuumport befinden sich etwa 60 mm außerhalb der Mittelebene des Dipolmagneten.

Sonde		$a_0[T]$	$a_1[1]$	$a_2[1/T]$	$a_3[1/T^2]$	$a_4[1/T^3]$	$a_5[1/T^4]$
2		$3.41 * 10^{-5}$	$-4.07 * 10^{-4}$				
	$\pm$	$2.1 * 10^{-6}$	$1.3 * 10^{-5}$				
3		$4.24 * 10^{-5}$	$-4.05 * 10^{-4}$				
	$\pm$	$6.8 * 10^{-6}$	$2.1 * 10^{-5}$				
4		$7.63 * 10^{-5}$	$-3.77 * 10^{-4}$				
	$\pm$	$1.10 * 10^{-5}$	$1.6 * 10^{-5}$				
5		-0.181734	0.864132	-1.62021	1.49487	-0.678473	0.121069
	$\pm$	0.020114	0.092249	0.166858	0.14880	0.065440	0.011359

Tab.6.3: Koeffizienten der Polynomanpassung für AD2.

Vergrößerung des Polschuhabstandes[8] um $50 \pm 25\mu$m. Das entspricht bei einem Polschuhabstand von 20 cm einer Reduktion in der Feldstärke B von 0.19 bis 0.56 mTesla, also in etwa dem im Bereich bis 1.2T beobachteten Verhalten (Abb.6.2).

Für das starke Ansteigen der Differenz oberhalb von 1.2 T kann keine schlüssige Erklärung gegeben werden; hier spielen die Auswirkungen von Sättigungseffekten im Bereich der Meßsonden eine nicht näher untersuchte Rolle; auch für diesen Dipolmagneten AD2 wurden Polynome an die gemessenen Differenzen angepaßt (Tab.6.3).

6.2 Feldmessung am Spektrometer B

6.2.1 Aufgabenstellung

Das Spektrometer B besteht aus einem sogenannten Clam-Shell Dipolmagneten, bei dem die Polschuhe zwar eben, jedoch nicht planparallel sind. Die beiden Polschuhe haben entlang einer Bezugslinie, der sogenannten "1.5T-Linie", einen Abstand a_0 von $200 \pm 0.05\ mm$; ihre Flächen öffnen sich in Richtung auf den äußeren Polschuhrand mit einem Winkel θ_0 von $3.480 \pm 0.002°$ symmetrisch zur Mitte-

[8]Neben den magnetischen Kräften führen auch die Vakuumkräfte zu einer Veränderung des Polschuhabstands. Ihr Einfluß auf die NMR-Eichung wurde mit einem Streuexperiment (siehe Anhang C.1) studiert.

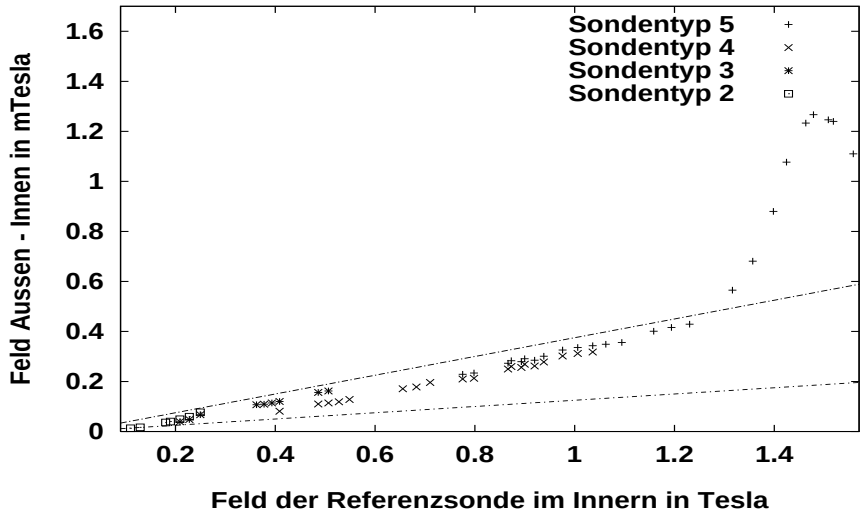


Abb.6.2: Differenz der gemessenen Feldwerte im Innern und am NMR-Port beim Dipolmagneten AD2 von Spektrometer A. Die Werte sind zusammen mit dem jeweils eingestellten Strom in Tab.C.2 im Anhang aufgeführt. Die Inhomogenität des Feldes im Bereich des NMR-Ports ist die Ursache für die Unterschiede der Sonden. Die gestrichelten Kurven zeigen den durch die abgeschätzte Verbiegung des Polschuhs hervorgerufenen minimalen bzw. maximalen Effekt.

lebene (Abb.6.3 und Abb.C.3). Bei der maximal vorgesehenen Erregung beträgt die Soll–Feldstärke 1.505 T entlang der Bezugslinie.

Die Feldstärke B_z ist konstant auf allen Parallelen zur Bezugslinie. In der Mittelebene des Magneten hängt B_z nur vom senkrechten Abstand r zur 1.5T-Linie ab, gemäß

$$B_z(r) = \frac{B_{z0}}{1 + \frac{r}{D_0}} \ , \tag{6.1}$$

wobei B_{z0} den Feldwert auf der 1.5T-Linie bezeichnet [sch95, ko95]. Die Größe D_0 ergibt sich aus dem Polschuhabstand $a_0 = 200$mm und dem Öffnungswinkel

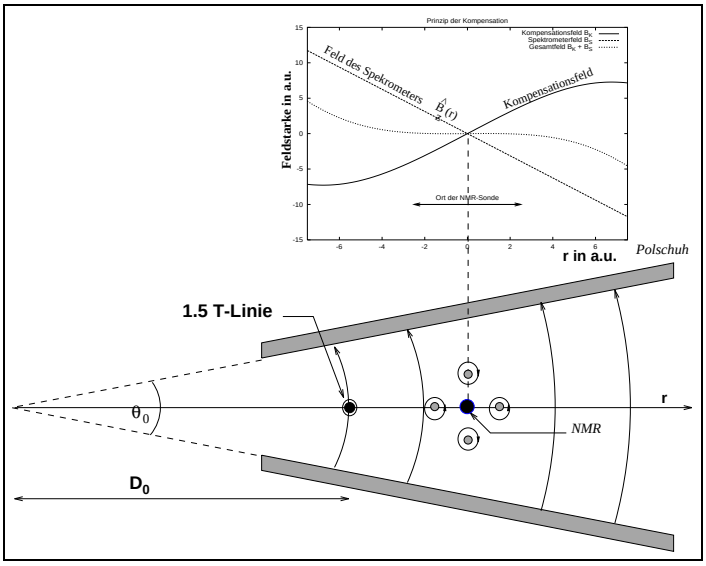


Abb.6.3: Prinzip der Kompensation des Feldgradienten durch ein Quadrupolfeld, das von vier Stromleitern erzeugt wird. Das Bild zeigt einen Schnitt senkrecht zur Mittelebene des Clam-Shell-Magneten (nicht maßstabsgerecht). Rechts oben ist der Verlauf der Feldstärke dargestellt, wobei $\hat{B}_z(r) = B_z(r) - B_z(r_0)$; r_0 Zentrum der Sonde.

$\theta_0 = 3.48°$ gemäß

$$tan(\frac{\theta_0}{2}) = \frac{a_0/2}{D_0} \quad \text{zu} \quad D_0 = 3291.8\,\text{mm}.$$

Aus Glg. 6.1 ergibt sich ein Feldgradient $G_z(r)$ von

$$G_z(r) \equiv \frac{\delta B_z(r)}{\delta r} = -\frac{B_{z0}}{(1+\frac{r}{D_0})^2}\frac{1}{D_0}\ . \tag{6.2}$$

Der relative Feldgradient $G_z(r)/B_z(r)$ beträgt damit auf der 1.5T-Linie:

$$G_z(0)/B_z(0) = \frac{1}{D_0} \approx 3038 \quad \text{ppm/cm}.$$

Bei einer solchen Feldinhomogenität läßt sich die Feldmessung nicht mehr mit den kommerziellen Kernresonanz-Sonden durchführen; sie arbeiten nur bis zu einer maximalen Feldinhomogenität von etwa 200 ppm/cm [met89]. Deshalb ist es notwendig, die Inhomogenität durch die Überlagerung mit einem am Ort der Kernresonanzsonde befindlichem *Kompensationsfeld* auszugleichen.

Da man den Gradienten des Spektrometerfeldes über den Bereich der Sonde als konstant ansehen kann, wird die Kompensation am einfachsten mit einem Feld erreicht, das einen Gradienten aufweist, der gerade dem des Spektrometerfelds entgegengesetzt ist. Ein solches Feld läßt sich durch einen Quadrupolmagneten erzeugen, dessen Symmetrieachse entlang der Äquipotentiallinie des Spektrometerfelds verläuft[9], d.h. parallel zur 1.5T-Linie.

Abb.6.3 zeigt das Prinzip der Kompensation.

6.2.2 Anforderungen an den Aufbau

Bei der Konstruktion des Quadrupolmagneten sind eine Reihe von Anforderungen zu erfüllen:

- Alle vier Sonden müssen samt ihrer jeweiligen Kompensationsvorrichtung und der zusätzlich zur Magnetfeldmessung bereitstehenden Hallsonde in einen recht schmalen Vakuum-Port (Innenmaß: Höhe 50mm, Breite 146mm) des Spektrometers passen. Die Sonden selbst haben ein Außenmaß von jeweils $16.8 * 12.8$ mm^2. Die Abmessungen der Hallsonde betragen $22 * 22$mm^2.
- Die Gradientenrichtung der Kompensation muß aus zwei Gründen variabel sein:
  1. Der Vakuum-Port muß schräg gegen die Horizontale ins Spektrometer (Abb.6.4) eingebaut werden. Der *Neigungswinkel* des Vakuumports hängt davon ab, wie weit man die Sonden ins Feld einbringen muß, um ein gutes Signal zu erzielen. Dies kann man nur mit Hilfe der Sonden selbst feststellen, wozu bereits die Gradientenkompensation benötigt wird. Anhand der

[9] Diese Methode hat sich bereits beim doppelfokussierenden Spektrometer der Elektronenstreuanordnung am Mainzer 400-MeV-Linearbeschleuniger bewährt [neu64]; sie geht auf Denisov [den60] zurück.

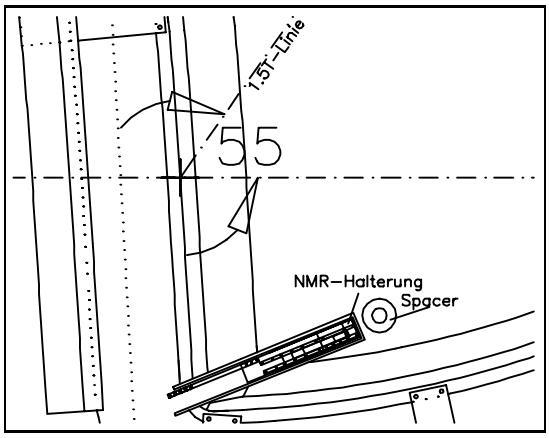


Abb.6.4: Bereich des Eintrittsfensters von Spektrometer B. Die Lage des schräg eingebauten Vakuum-Ports ist eingezeichnet. Er enthält die Halterungen für die NMR-Sonden. Das Bild stellt einen Ausschnitt von Abb.C.3 im Anhang dar.

Feldkarten ist ein Neigungswinkel von etwa 9° zu erwarten. Die Äquipotentiallinie schneidet dann die Sonden unter einem Winkel von etwa 51° [blo91].

2. Im Bereich der Sondenposition, d.h. in der Nähe des Polschuhrandes, könnte die Äquipotentiallinie leicht gegenüber den gerechneten Werten gedreht sein[blo91].

Die Lage des NMR-Vakuumports ist in Abb.6.4 eingezeichnet. Der maximale Gradient beträgt nach Glg.6.2 im Bereich der Sonden etwa 0.35 T/m.

- Die nach der Kompensation verbleibende *Restinhomogenität* δ des Feldes muß kleiner als $\pm 1.0 * 10^{-5}$ sein, da bei einer Feldinhomogenität von mehr als $\pm 2.5 * 10^{-5}$ das automatische Auffinden der Resonanz durch das Teslameter nicht mehr möglich ist.

- Die Wärmeabgabe der Kompensationsspulen muß so gering sein, daß keine zusätzliche Kühlung erforderlich wird (zusätzlicher Platzbedarf!).

- Die Sonden müssen so installiert werden, daß man sie im Bedarfsfall einzeln ausbauen kann. Wegen des Überlapps im Meßbereich der Sonden benötigt man dann keine zusätzlichen Referenzmessungen, um die Meßwerte nach dem Wiedereinbau wieder mit den vorherigen in Beziehung zu setzen. Das ist in diesem Fall besonders wichtig, weil man auch für die Referenzsonden eine Kompensationsvorrichtung benötigte.

6.2.3 Erzeugung des Kompensationsfelds

Zum Bau des Kompensationsmagneten bietet sich wegen des geringen Platzbedarfs und wegen der einfachen Handhabung ein Aufbau aus vier gedruckten Spulen auf zwei parallel zueinander angebrachten Platinen an. Die Verwendbarkeit von auf dem Markt erhältlichen Platinen[10], die in Abb.6.6 (links) skizziert sind, konnte durch Feldrechnungen für verschiedene gedruckte Spulenkonfigurationen nachgewiesen werden.

[10]Die Platinen wurden von der Firma Metrolab aus Genf geliefert.

Feldrechnungen

Zur Abschätzung der zu erwartenden Restinhomogenität wird folgendermaßen vorgegangen:
In einem ersten Schritt wird das Feld B_z^{Sp} der Spulenkonfiguration nach dem Gesetz von Biot-Savart berechnet; dafür wird - vereinfachend - jedes gerade Leitungsstück endlicher Breite jeweils durch einen unendlich dünnen Leiter gleicher Länge ersetzt.
In einem zweiten Schritt wird in dem interessierenden Bereich der NMR-Sonde - in der Abb.6.6 rechts mit "nmr-sample" bezeichnet - an das für die Spulen berechnete Feld auf einem Gitter mit N Stützstellen $\vec{r_i}$ ein reines Quadrupolfeld

$$B_z^Q(\vec{r}) = a * x + b * y \tag{6.3}$$

durch Minimierung des Abstands

$$\bar{\Delta}(a,b) = \frac{1}{N}\sum_{i=0}^{N}\left|B_z^{Sp}(\vec{r_i}) - B_z^Q(\vec{r_i})\right| = \frac{1}{N}\sum_{i=0}^{N}|\Delta(x_i,y_i)| \tag{6.4}$$

angepaßt. Das Feld B_z^Q repräsentiert die Fähigkeit des durch die Spulen erzeugten Feldes B_z^{Sp}, einen Gradienten zu kompensieren: Wäre B_z^{Sp} ein reines Quadrupolfeld, dann könnte es den Gradienten des Spektrometerfeldes im gesamten Bereich der Sonde exakt kompensieren. Die Abweichung des Spulenfeldes B_z^{Sp} von einem Quadrupolfeld, die bei einer Kompensation als Restinhomogenität übrig bleibt, wird durch $\bar{\Delta}$ quantifiziert.
Damit läßt sich die relative Restinhomogenität δ des zu messenden Feldes im Bereich der NMR-Sonde beschreiben durch den Quotienten aus $\bar{\Delta}$ und dem Spektrometerfeld $B_z(r_0)$ am Sondenort ($r_0 \approx 0.55\ m$). Dabei ist als Spektrometerfeld die Feldstärke zu nehmen, die zum gleichen Gradienten führt wie das Feld B_z^{Sp} der Spulen beziehungsweise das sie repräsentierende Quadrupolfeld B_z^Q, d.h. zum Gradienten $\sqrt{a^2+b^2}$. Mit Glg.6.2 ergibt sich damit die Restinhomogenität zu

$$\delta = \frac{\bar{\Delta}}{\sqrt{a^2+b^2}(1+\frac{r_0}{D_0})^2 D_0} \approx \frac{\bar{\Delta}}{\sqrt{a^2+b^2} * 4.47\ m}\ , \tag{6.5}$$

wobei a und b in T/m gegeben sind.

Abbildung 6.5 zeigt die Abweichung $\Delta(x,y)$ in der Mittelebene. Dabei werden die beiden Platinen in einem Abstand von 12.6 mm mit einem Strom von 1 A betrieben. Für diese Konfiguration ergibt sich eine Restinhomogenität von $\delta = 5.3 * 10^{-6}$, die deutlich unterhalb der Grenze für die Verwendbarkeit der NMR-Sonden liegt.

Drehen der Gradientenrichtung

Auf Grund des schmalen NMR-Ports stellt auch die mechanische Konstruktion der Anordnung ein Problem dar, da die Richtung des Kompensationsgradienten einstellbar sein muß. Zunächst wurde die Idee verfolgt, den Quadrupol um die Sonde rotierbar aufzuhängen. Dieses Verfahren stellte sich jedoch als zu platzraubend heraus. Außerdem müßte die Rotation der nach dem Einbau nicht mehr

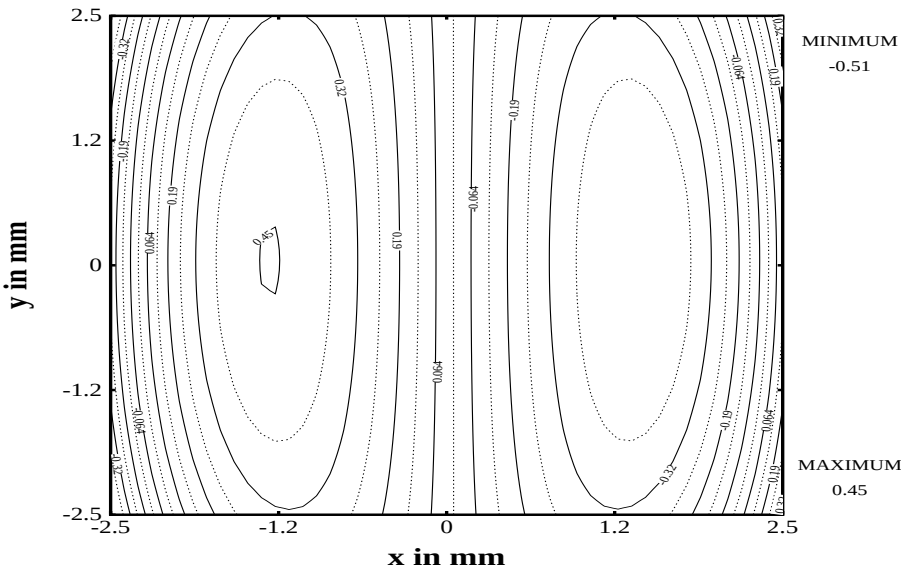


Abb.6.5: Kompensationsfehler in Einheiten von μT als Funktion vom Ort (x, y). Das Zentrum der Kernresonanzsonde befindet sich am Punkt $(0, 0)$. Bei der Modellrechnung befanden sich die beiden Platinen in einem Abstand von 12.6 mm.

direkt zugänglichen Quadrupolspulen über eine aufwendige Mechanik mit einem Motor erfolgen.

Die dazu alternativ entwickelte, konzeptionell wesentlich einfachere Methode besteht darin, die Richtung des Kompensationsgradienten "magnetisch" zu drehen, indem man die Ströme zweier gegeneinander verschert aufgebauter Quadrupolspulen variiert (siehe Abb.6.6 rechts). Die Symmetrieachsen der beiden Quadrupole schließen dabei einen Öffnungswinkel γ ein. Da die Spulen als Platinen vorliegen, ist ein solcher Aufbau sehr kompakt möglich.

Die prinzipielle Tragfähigkeit dieser Methode wurde mit Feldrechnungen nachgewiesen. Die Kompensation des maximal auftretenden Gradienten von 0.35 T/m erfordert einen Strom von etwa 1.35 A pro Quadrupol, falls die Symmetrieachsen der beiden Quadrupole einen Winkel von 20° einschließen und sich die Platinen in einem Abstand von 12.6 mm voneinander befinden. Bei diesem Strom haben sich die Platinen bei einem Test im Dauerbetrieb auf etwa 45° Celsius aufgeheizt. Deshalb ist keine zusätzliche Kühlung erforderlich. Bei dieser Anordnung ist eine Restinhomogenität in der Größenordnung von $8 * 10^{-6}$ zu erwarten.

Die praktische Durchführbarkeit dieser Methode hängt auch davon ab, wie genau die Einstellung des Kompensationsgradienten vorzunehmen ist. Diese Fragestellung wurde durch Modellrechnungen zur Überlagerung zweier Quadrupolfelder untersucht, bei denen die Abhängigkeit der oben definierten Restinhomogenität vom Kompensationsstrom I und vom Winkel α zwischen dem Spektrometerfeldgradienten und dem Kompensationsgradienten bestimmt wurde. Die Rechnungen wurden auf Gitterpunkten (I_i, α_i) im Bereich der Optimaleinstellung (I_0, α_0) von Strom und Winkel durchgeführt. In Abb.6.7 sind Schnitte entlang der Linie mit $I = I_0$ bzw. $\alpha = \alpha_0$ dargestellt. Ebensfalls eingetragen ist die Restinhomogenität, die für die automatische Resonanzsuche nicht überschritten werden darf. Man erkennt, daß man bei der Suche nach der Resonanz Winkel und Strom der

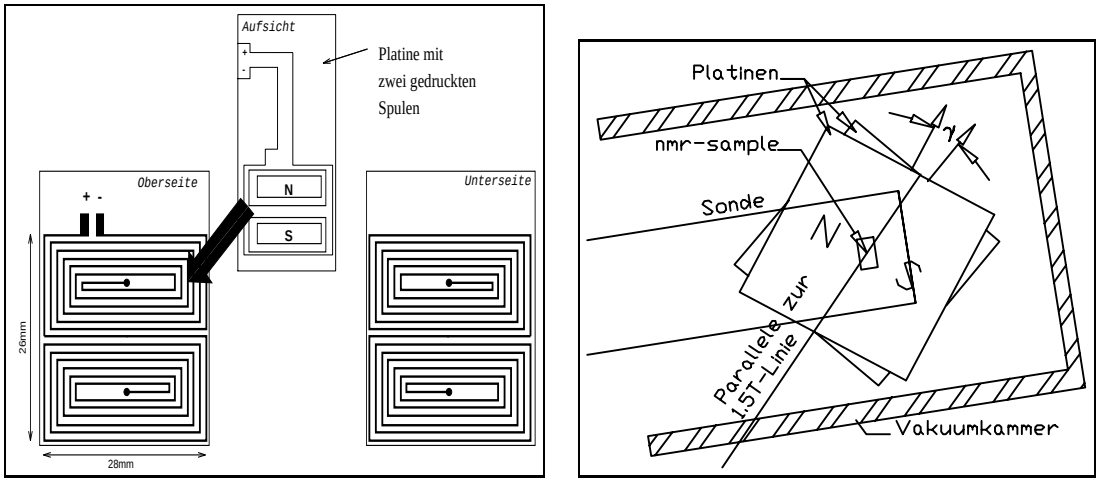


Abb.6.6: Linke Seite: Ober- und Unterseite der Platinen, die zum Bau des Quadrupols verwendet wurden. Ein Quadrupol wird aus zwei solcher Platinen gebildet, in deren Mitte sich die NMR-Sonde befindet. **Rechte Seite:** Ausschnittsvergrößerung der NMR-Halterung aus Abb.6.4. Die Kompensationsanordnung wird aus zwei Quadrupolen gebildet, die gegeneinander um den Öffnungswinkel verdreht sind.

Kompensationsanordnung in groben Schritten von $\Delta\alpha = 2°$ und $\Delta I/I = 0.05$ variieren kann.

Auf der Grundlage dieser Voruntersuchungen wurde eine Testanordnung gebaut, in der gedruckte Spulen zwei um 20° gegeneinander verscherte Quadrupole bilden. Damit ist es möglich, die Richtung der gemeinsamen Symmetrieachse um bis zu $\pm 10°$ zu drehen. Das erzeugte Gesamtfeld wurde mit einer Hallsonde vermessen, die mit Hilfe eines X-Y-Tisches in Schritten von 0.66 ± 0.05 mm durch die Mittelebene der Anordnung bewegt wurde. Die aktive Fläche der Hallsonde hatte einen Durchmesser von 1.1 mm. Ihr Meßfehler kann nach oben mit $5 * 10^{-3}$ abgeschätzt werden. Abb. 6.8 zeigt eine gemessene Feldkarte.

An die gemessenen Feldwerte wurde im Bereich der NMR-Sonde ein Feld der Form von Glg.6.3 angepaßt. Mit einem Spulenstrom von 1.2 ± 0.01 A wurde im Bereich der NMR-Sonde ein Gradient von 0.32 T/m erreicht und eine Restinhomogenität von $\delta = 7.8 * 10^{-6}$ ermittelt. Der Testaufbau erfüllt also alle Erwartungen.

6.2.4 Einsatz am Spektrometer

Auf der Grundlage der mit dem Testaufbau erzielten Ergebnisse wurde eine Kompensationsvorrichtung nach dem beschriebenen Konzept für den Einsatz am Spektrometer gebaut. Dabei wurde der Öffnungswinkel der beiden Quadrupole zu 22° gewählt. Die Symmetrieachse der Gesamtanordnung ist um 41.5° gegen die Sondenebene verdreht, um die Schräge des Vakuumports zu berücksichtigen (siehe Abb.6.4). Die komplette Halterung wurde in der feinmechanischen Werkstatt des Instituts aus Aluminium gefertigt.
Um den Neigungswinkel des Vakuumports zu bestimmen, wurde eine Kompensationsvorrichtung zusammen mit einer Resonanzsonde an einer im Winkel beweglichen Halterung angebracht. Für verschiedene Neigungswinkel wurden dann

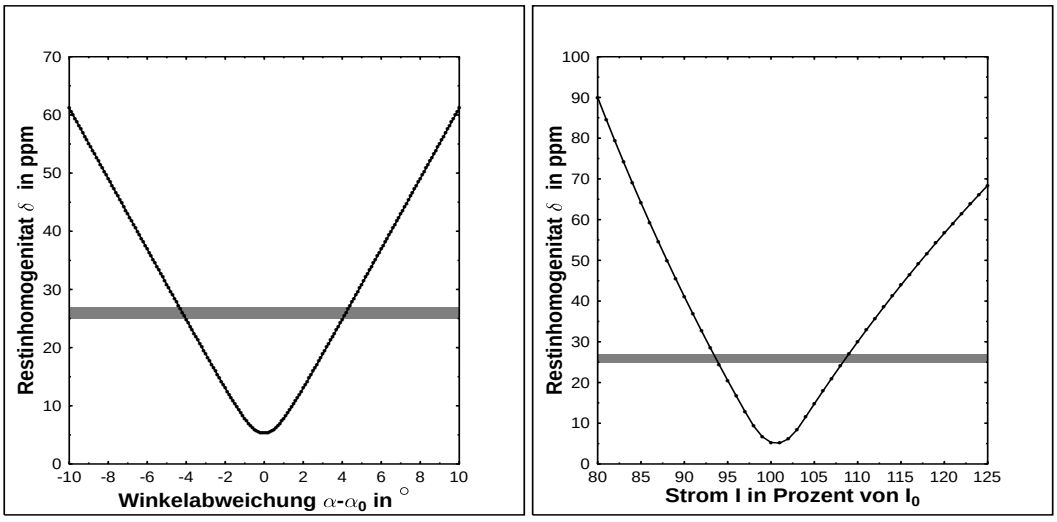


Abb:6.7: Abhängigkeit der Restinhomogenität δ von der Einstellung der Gradientenrichtung α und des Kompensationsstroms I. Das graue Band deutet die Inhomogenität an, oberhalb derer die automatische Resonanzsuche nicht mehr funktioniert.

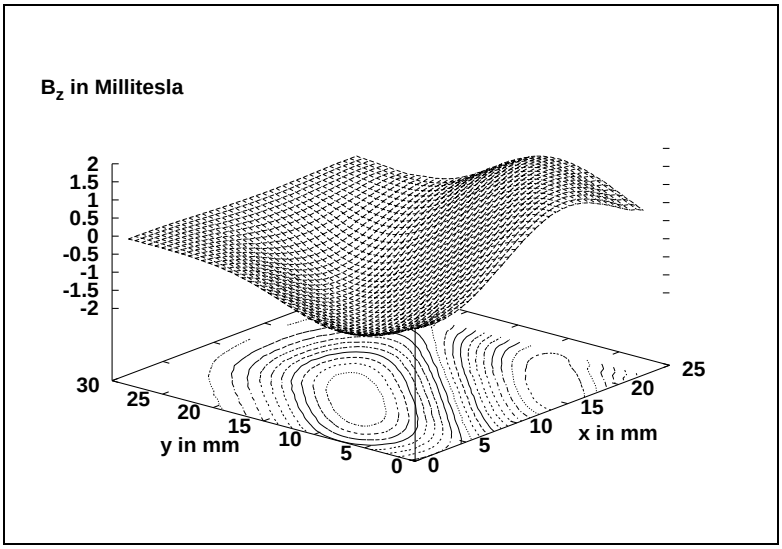


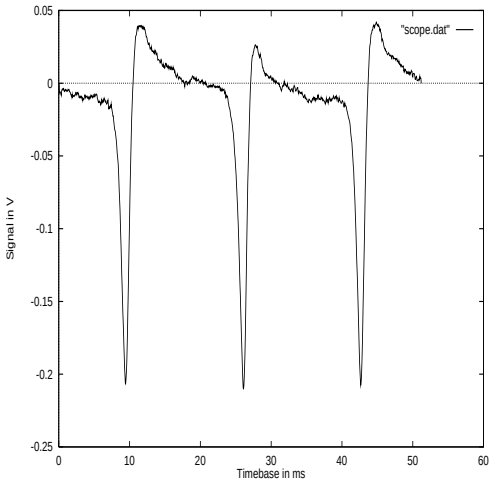
Abb.6.8: Gemessene Feldstärke B_z in der Mittelebene der Kompensationsanordnung.

die Kompensationsströme $I_1(B)$ und $I_2(B)$ des Quadrupolpaares für optimales Resonanzsignal eingestellt. Hierbei stellte sich heraus, daß das NMR-Signal erst ab einem Neigungswinkel von 16° ausreichend stabil für die automatische Resonanzsuche war. Der im Gegensatz zur ursprünglichen Annahme von 9° um 7° vergrößerte Neigungswinkel hat zur Folge, daß die Hauptlast von einem der beiden Kompensationsmagnete getragen wird. Um eine zu hohe Strombelastung der Platinen bei hohen Feldstärken (starken Gradienten) zu vermeiden, wurde deshalb die Sonde fünf mit der doppelten Anzahl an Windungen (Platinen) pro Quadrupol ausgestattet.

Die Stromversorgung der Quadrupolpaare aller Sonden erfolgt durch ein fernsteuerbares Netzgerät, das mit zwei jeweils über einen 16-Bit DAC einstellbaren Konstant-Stromquellen ausgerüstet ist, die einen maximalen Strom von 2 A liefern[11]. Die Stromquellen versorgen immer nur das Quadrupolpaar, das gera-

[11]Die Kalibrationskurven für den DAC und die Bitbelegung zur Ansteuerung der Stromver-

Abb.6.9: Das mit einem Oszilloskop aufgezeichnete Resonanzsignal bei einer Spektrometereinstellung von 760MeV/c. Die Signalamplitude liegt deutlich oberhalb des zum automatischen Auffinden der Resonanz notwendigen Wertes von 0.1 Volt [met89].



de für die Messung benötigt wird. Die Umschaltung erfolgt über ein Relais. Ein zusätzliches Relais erlaubt die Umkehrung der Stromrichtung, um sie der Polarität des Spektrometerfeldes anzupassen.
Die Ansteuerung des Geräts erfolgt über einen 32Bit-Bus, der über ein Digital-I/O Modul (TVM744) in einem VMEbus-System angesprochen wird. Bei diesem Bus[12] handelt es sich um eine Eigenentwicklung des Instituts für Kernphysik [ku95],[ma93]. Die zum Betrieb des Steuerbusses und des Netzteils benötigten Softwaremodule wurden in Form von C++-Klassen erstellt [kra93c].

Der zu kompensierende Feldgradient hängt gemäß Glg.6.2 vom eingestellten Spektrometerfeld ab. Deshalb muß der Gesamtstrom I der Kompensationsspulen für die Feldmessung mit der Kernresonanz an die Spektrometerfeldstärke B mit Hilfe einer Eichkurve $I(B)$ angepaßt werden. Dafür wird die Feldstärke mit der im NMR-Port eingebauten Hallsonde als B_H ausgelesen und damit der Wert des Kompensationsstroms $I(B_H)$ - einschließlich Vorzeichen - mit genügender Genauigkeit bestimmt.

6.2.5 Ergebnisse und Ausblick

Die Kompensation der Feldinhomogenität im Spektrometer B durch die Überlagerung zweier gegeneinander gedrehter Quadrupolfelder hat bei den bisherigen Messungen sehr gut funktioniert, so daß die Einstellung und Messung des Spektrometerfeldes vollständig automatisiert werden konnte. Die Abbildung 6.9 zeigt ein dabei gewonnenes NMR-Signal. Bei den in den bisherigen Strahlzeiten eingesetzten Sonden ist die relative Meßgenauigkeit δ_B besser als $5.0 * 10^{-6}$ (Tab.6.4); dies übertrifft die Anforderungen an die Anordnung.
Aus den Kalibrationsexperimenten mit den Sonden 4 und 5 ergibt sich im Überlappbereich folgender Zusammenhang:

$$\begin{aligned} B_5(B_4) &= a_0 + a_1 * B_4 \\ a_0 &= (-9.4 \pm 0.8) * 10^{-5}\mathrm{T} \\ a_1 &= 0.995028 \pm 9.3 * 10^{-6} \ . \end{aligned}$$

sorgung befindet sich im Anhang (S.118).

[12] Über denselben Bus werden die Motoren der Kollimatoren von Spektrometer A und B angesteuert.

Tab.6.4: Tabelle der relativen Meßgenauigkeit ΔB in Abhängigkeit von der eingestellten Feldstärke B für die Sondentypen 3 bis 5. Zur Bestimmung der relativen Meßgenauigkeit wurde bei derselben Feldeinstellung mehrmals nacheinander eine Feldmessung mit der Kernresonanzmethode durchgeführt. Als Fehler ΔB wird in der Tabelle die Standardabweichung der gemessenen Feldwerte angegeben. Bei der Messung ist darauf zu achten, daß sich die Temperatur der Polschuhe nicht verändert. Der Fehler fällt mit steigendem Feld innerhalb des Meßbereichs einer Sonde ab. Dies gilt auch für die Sonden im homogenen Feld der Dipolmagnete von Spektrometer A, da das Signal/Rauschverhältnis mit steigendem Feld zunimmt.

$\overline{B}$ in T	ΔB in μT	$\delta_B = \frac{\Delta B}{\overline{B}}$ in ppm	Typ
0.4252	0.36	0.85	3
0.4282	2.08	4.86	4
0.6135	2.38	3.88	4
0.9272	1.12	1.21	4
0.7225	3.62	5.01	5
0.9375	1.92	2.09	5
1.109	0.83	0.75	5

Mit den installierten Sonden wurde die Erregungskurve für den Clam-Shell Magneten, d.h. die Abhängigkeit der Feldstärke vom eingestellten Strom, gemessen. Die Meßwerte für die 4 Sonden werden für den jeweiligen Gültigkeitsbereich (Tab.6.1) durch folgende Polynome[13] beschrieben:

$$\begin{aligned} I(B_5) &= -194.844 + 1386.63 * B_5 - 271.066 * B_5^2 - 155.171 * B_5^3 \\ &\quad -39.889 * B_5^4 + 144.174 * B_5^5, \\ I(B_4) &= -11.4633 + 876.71 * B_4, \\ I(B_3) &= 4.70269 + 851.73 * B_3, \\ I(B_2) &= 1.41098 + 862.646 * B_2, \end{aligned} \tag{6.6}$$

die Felder werden dabei in Tesla, die Ströme in Ampère angegeben. Für die Sonde 5 (höchster Feldbereich!) benötigt man auf Grund der Sättigungseffekte ein Polynom höherer Ordnung.

Da der Zusammenhang zwischen Strom und Feldstärke auf der 1.5T-Linie des Clam-Shell Magneten bereits mit Hilfe einer Hallsonde bestimmt wurde [sch95], läßt sich aus den Polynomen der Glg.6.6 zusammen mit Glg.6.1 für $B_z(r)$ der Abstand r der Sonden von der Bezugslinie überprüfen. Für die Sonde 4 ergibt sich z.B. ein Abstand von 0.556 m. Das entspricht auch der tatsächlichen, in Abb.6.4 eingezeichneten Position.

Verbesserungsmöglichkeiten

Das System hat sich inzwischen ohne jede Einschränkung im Einsatz bewährt. Mit den nun bekannten Winkeleinstellungen, ist es möglich, Platinen zu ferti-

[13]Die Eichkurven werden in der Steuerungssoftware zur automatischen Magnetfeldeinstellung benötigt. Dazu sind die Polynomkoeffizienten in die Konfigurationsdatenbank eingetragen. Die Umkehrfunktion zur Bestimmung von B aus I wird aus diesen Polynomen mit Hilfe eines Bisektionsverfahrens ermittelt, um Redundanzen innerhalb der Datenbank zu vermeiden Diese Vorgehensweise wird auch an anderen Stellen des Steuerungssystems angewendet.

gen, bei denen die gedruckten Spulen und deren Zuleitungen optimal für diesen Einsatz angeordnet sind. Man könnte damit die zur Zeit verwendeten Platinen ersetzen, bei denen teilweise die Isolation der Leiterbahnen entfernt werden mußte, um räumlich das Anbringen der Zuleitung zu ermöglichen.
Auf der Softwareseite sollte man die Stromversorgung über einen Mikrokontroller steuern, der über eine serielle Schnittstelle, Ethernet oder einen Feldbus angesprochen wird. Der Mikrokontroller sollte sich direkt in der Stromversorgungseinheit befinden. Dadurch könnte der oben (S.74) erwähnte Steuerbus ersetzt und damit das gesamte System wesentlich kompakter werden.

Kapitel 7

On-line Datenkontrolle

7.1 Problemdefinition

Das in Abschnitt 3.1 vorgestellte Gesamtkonzept zur Überwachung der Experimente an der Drei-Spektrometer-Anlage sieht eine zweite Stufe vor, bei der Informationen über den Zustand der Apparatur aus den erfaßten Ereignissen der jeweils untersuchten physikalischen Reaktion selbst gewonnen werden (siehe Abb.3.2). Dieser Teilaspekt wird in diesem 7. Kapitel der vorliegenden Arbeit behandelt. Es wird untersucht, welche Möglichkeiten der Informationsauswertung bestehen und welcher Ergänzungen um weitere Software-Werkzeuge das On-line System zur Realisierung dieser Stufe bedarf.

Den Messungen, deren Informationsgehalt unter dem genannten Gesichtspunkt bestimmt werden soll, liegen statistische Prozesse zu Grunde. Die Beurteilung des Anlagenzustands soll also auf der Grundlage der n Einzelereignisse e erfolgen, die in einem Zeitintervall $I_T = [T - \Delta t/2, T + \Delta t/2[$ beobachtet wurden. Diese Ereignisse sind nach der Zeit ihres Eintretens geordnet:

$$e_{t_1}, \ldots, e_{t_n} \quad \text{mit} \quad T - \Delta t/2 \leq t_1 < \ldots < t_n < T + \Delta t/2;$$

e_{t_ν} steht dabei für den vollen Satz der zum Zeitpunkt t_ν gemessenen Werte. Die Menge der im Zeitintervall I_T eingetretenen Einzelereignisse stellt eine Stichprobe für eine multidimensionale Zufallsvariable $\mathcal{E}$ aus dem entsprechenden Ereignisraum $\mathbb{E}$ dar; im folgenden wird in diesem Zusammenhang von der Stichprobe zur Zeit T gesprochen: $\mathcal{E}(T) = \{e_{t_1}, e_{t_2}, \ldots, e_{t_n}\}$. Ein Einzelereignis e setzt sich aus allen Informationen der einzelnen Detektoren der Drei-Spektrometer-Anlage zusammen. Es besteht bei einem Koinzidenzexperiment mit den Spektrometern A und B aus etwa 3400 Einträgen. Diese große Zahl von einzelnen Werten muß zur Beschreibung des Zustands der Anlage wesentlich reduziert werden.
Eine erste Reduktion erreicht man durch Zusammenfassung der Stichprobe $\mathcal{E}(T)$ in geeignete Histogramme, d.h. in m Häufigkeitsverteilungen $\mathcal{H}_\mu(T)$ von geeignet gewählten Histogrammvariablen h_μ mit $\mu = 1, \ldots, m$. Der Wert der Variablen h_μ berechnet für ein Einzelereignis e_{t_ν} sei mit $h_{\mu,\nu}$ bezeichnet:

$$h_{\mu,\nu} = h_\mu(e_{t_\nu}) \quad \text{mit} \quad \nu = 1, 2, \ldots, n; \mu = 1, 2, \ldots, m \quad \text{und} \quad h_\mu : \mathbb{E} \to \mathbb{R} \; .$$

Ein typisches Beispiel für eine Histogrammvariable ist der Impuls des gestreuten Elektrons, der sich aus den in der Bildebene des Elektronenspektrometers gemessenen Koordinaten berechnen läßt (Anhang A).
Über die Reduktion auf Histogramme hinaus bedarf es einer weiteren Reduktion der Information zur Zeit T auf einzelne Variable, die im folgenden als Zustandsparameter mit $p_\lambda(T)$, $\lambda = 1, 2, \ldots, l$ bezeichnet werden. Der Wert des Zustandsparameters p_λ zur Zeit T ergibt sich aus den Elementen der Stichprobe:

$$p_\lambda(T) := p_\lambda(\mathcal{E}(T)) \; .$$

Ein typischer Zustandsparameter ist z.B. das in Abb.3.1 als Funktion von T dargestellte Verhältnis der gültigen Kammerspuren zur Gesamtzahl der Triggerereignisse.
Insgesamt wird der Zustand der Anlage zur Zeit T durch den $L = m + l$-komponentigen sogenannten *Zustandsvektor* $\vec{Q}(T)$ aus allen m Histogrammen $\mathcal{H}_\mu(T)$ und allen l Zustandsparametern $p_\lambda(T)$ beschrieben, d.h.

$$\vec{Q}(T) := (\mathcal{H}_1(T), \mathcal{H}_2(T), \ldots, \mathcal{H}_m(T), p_1(T), p_2(T), \ldots, p_l(T)) = (q_1(T), \ldots, q_L(T))$$

Für die Bewertung des Zustands der Anlage während der Messung bedarf es einer noch zu spezifizierenden "Norm" zum Vergleich der Komponenten $q_\lambda(T)$ des Zustandsvektors $\vec{Q}(T)$ mit den entsprechenden Komponenten eines Referenzzustands $\vec{Q}_0$. Bei der Bewertung von Abweichungen muß der statistische Charakter von $\vec{Q}(T)$ berücksichtigt werden, d.h. es geht um die Feststellung von Abweichungen, die nicht statistischer Natur sind.
Im günstigsten - sehr seltenen - Fall steht die Referenz $\vec{Q}_0$ aus einer früheren Messung[1] zur Verfügung. Im allgemeinen ist dies nicht der Fall, so daß die Vergleiche *dynamisch* erfolgen, d.h. der Referenzzustand wird dadurch gebildet, daß die unter festen, gleichbleibenden Bedingungen laufende Messung in k Teilmessungen eingeteilt und jeweils entsprechende Zustandsvektoren $\vec{Q}(T_\kappa)$ berechnet werden. Aus dem Vergleich der $\vec{Q}(T_k)$ Zustandsvektoren untereinander, insbesondere mit den bis zur Zeit T_{k-1} bekannten, läßt sich auf Änderungen und damit ggf. auf auftretende Fehler schließen. Bei vollständiger Realisierung der Analysesoftware (siehe Abschnitt 7.3) kann bei entsprechend aufgearbeiteter Auswertung der Information das Experiment weitgehend selbstkontrollierend ablaufen.

Zur Realisierung einer solchen Selbstkontrolle sind folgende Aufgaben zu erledigen:

1. Es ist ein Maß für die Bewertung der Abweichungen zwischen den Komponenten der Zustandsvektoren festzulegen. Ein wesentlicher Teil dieser Aufgabe besteht in einer zweckmäßigen Definition der L verallgemeinerten Zustandsparameter q_λ. In Abschnitt 7.2 werden die grundlegenden Prinzipien für den Einsatz einer solchen Selbstkontrolle im On-line System der Drei-Spektrometer-Anlage behandelt.

2. Diese Methoden sind in Form von Software-Werkzeugen zu realisieren.

3. Es ist ein Prototyp für den standardmäßigen Einsatz im On-line-System zur Überwachung der Experimente an der Drei-Spektrometer-Anlage zu erstellen.

7.2 Grundlagen

Zur Beschreibung des Zustands der Anlage werden spezifische *Testvariable* mit spezifischen Namen (*kname*) eingeführt. Die Überprüfung des Zustands der Anlage besteht in der Überprüfung des Wahrheitsgehalts der Testvariablen. Der Wert der Testvariablen **kname** zur Zeit T_k berechnet sich aus den Zustandsvektoren

[1] Dies gilt z.B. für Messungen mit Höhenstrahlung, d.h. ohne Elektronenstrahl, die vor jedem Streuexperiment und in Meßpausen, z.B. während des Einstellens der Magnetfelder, zum Test der Apparatur und zum Aufbau einer Langzeitreferenz durchgeführt werden sollten.

zu den Zeiten T_κ mit $\kappa \leq k$ anhand vorgegebener Kriterien, die durch Parameter α_i spezifiziert sind. Somit berechnet sich die Testvariable **kname** gemäß

$$\mathbf{kname}(T_k) = g_{kname}(\vec{Q}(T_1), \ldots, \vec{Q}(T_k); \alpha_\lambda),$$

wobei g_{kname} eine für die jeweilige Testvariable spezifische Funktion der angegebenen Variablen zur Berechnung des Wahrheitsgehalts ist. g_{kname} nimmt im einfachsten Fall die Werte 1 oder 0 für wahr bzw. falsch an, i.a. aber im Sinne einer *fuzzy logic* [kos92] Werte im Intervall $[0, 1]$. Diese allgemeinere Situation liegt insbesondere dann vor, wenn die statistische Natur der Daten bei der Überwachung berücksichtigt wird.
Der Schwerpunkt dieses Abschnitts liegt auf der Methode zur Bewertung statistischer Daten. Die Entwicklung dieser Methode ist unabhängig von der konkreten Realisierung des Überwachungsalgorithmus erforderlich, auch dann, wenn zur Überwachung ein kommerzielles Expertensystem herangezogen wird, da diese Systeme in dieser Hinsicht i.a. keine Lösung anbieten [vi92].

7.2.1 Tests auf der Basis von Statusparametern

Bereichstest

Im einfachsten Fall läßt sich für einige Statusparameter der Wertebereich so einschränken, daß das Vorliegen eines Wertes *außerhalb* dieses Bereichs direkt als Fehlersituation bewertet werden kann - der Wert der Testfunktion ist dann gleich Null. Derartige *Bereichstests* kommen zur Anwendung bei Parametern, bei deren Messung statistische Fluktuationen keine Rolle spielen oder wenn diese nicht berücksichtigt werden sollen.
Seien $q_{\lambda,min}$ und $q_{\lambda,max}$ die Grenzen des zulässigen Bereichs für die Zustandsparameter q_λ dann lautet die für den Bereichstest als **range** eingeführte Testfunktion:

$$\mathbf{range}(q_\lambda; q_{\lambda,min}, q_{\lambda,max}) = \begin{cases} 1 & : \quad q_{\lambda,min} < q_\lambda < q_{\lambda,max} \\ 0 & : \quad \text{sonst} \end{cases} . \tag{7.1}$$

Mittelwerte

Um einen statistisch fluktuierenden Parameter q_λ auf Konstanz zu überprüfen, baut das Überwachungssystem (im einfachsten Verfahren) auf Mittelwert $\bar{q}_\lambda$ und Varianz s^2_λ auf, die zur Zeit T_k aus den vorhergehenden Werten $q_{i,\kappa} = q_\lambda(T_\kappa)$ in der üblichen Weise gemäß

$$\bar{q}_\lambda(T_k) := \sum_{\kappa<=k} q_{\lambda,\kappa}/k \quad \text{und} \quad s^2_\lambda(T_k) := 1/(k-1) \sum_{\kappa<=k} (q_{\lambda,\kappa} - \bar{q}_\lambda(T_k))^2 \tag{7.2}$$

berechnet werden. Um den Umfang der bereitzuhaltenden Information und damit den Speicherplatz klein zu halten, werden diese Größen jeweils rekursiv berechnet gemäß

$$\bar{q}_{\lambda,k} = \frac{q_{\lambda,k} + (k-1)\bar{q}_{\lambda,k-1}}{k} \tag{7.3}$$

für den Mittelwert und wegen

$$\begin{aligned}
(k-1)s_k^2 &= \sum_{\kappa<=k}(q_\kappa-\bar{q}_k)^2 \\
&= \sum_{\kappa<=k-1}(q_\kappa-\bar{q}_k)^2+(q_k-\bar{q}_k)^2 \\
&= \sum_{\kappa<=k-1}(q_\kappa-\bar{q}_{k-1}+\bar{q}_{k-1}-\bar{q}_k)^2+(q_k-\bar{q}_k)^2 \\
\Rightarrow s_k^2 &= \tfrac{k-2}{k-1}s_{k-1}^2+2\tfrac{\bar{q}_{k-1}-\bar{q}_k}{k-1}\underbrace{\sum_{\kappa<=k-1}(q_\kappa-\bar{q}_{k-1})}_{=0}+(\bar{q}_{k-1}-\bar{q}_k)^2+\tfrac{(q_k-\bar{q}_k)^2}{k-1}
\end{aligned}$$

gemäß

$$s_{\lambda,k}^2=\frac{(k-2)s_{\lambda,k-1}^2+(\bar{q}_{\lambda,k}-q_{\lambda,k})^2}{k-1}+(\bar{q}_{\lambda,k-1}-\bar{q}_{\lambda,k})^2 \tag{7.4}$$

für die Varianz. Zur Durchführung der Überwachung des Parameters q_λ wird mit

$$q_{\lambda,k}^{\pm}=\bar{q}_\lambda(T_k)\pm t_{\alpha_\lambda/2}s_\lambda(T_k)/\sqrt{k}$$

das Konfidenzinterval [mil90] zum Signifikanzniveaus α_λ vorgegeben - $t_{\alpha/2}$ ist das $\alpha/2$-Quantil der Standard-Normalverteilung -, in das der für q_λ gemessene Wert fallen soll. Für diesen Test wird die Testfunktion **checkMean** durch

$$\textbf{checkMean}(q_{\lambda,k+1},\bar{q}_\lambda(T_k),s_\lambda(T_k);\alpha_\lambda)=\begin{cases}1 & : \quad q_{\lambda,k+1}\in[q_{\lambda,k}^-,q_{\lambda,k}^+]\\ 0 & : \quad \text{sonst}\end{cases} \tag{7.5}$$

definiert. Als einfachere Form dieses Tests kann auch auf größere Abweichungen des Parameters vom bisherigen Mittelwert, die durch den Parameter α_λ vorgegeben werden, abgeprüft werden:

$$\textbf{spuriousVal}(q_{i,k+1},\bar{q}_\lambda(T_k);\alpha_\lambda)=\begin{cases}1 & : \quad q_{\lambda,k+1}\in[(1-\alpha)\bar{q}_\lambda(T_k),(1+\alpha)\bar{q}_\lambda(T_k)]\\ 0 & : \quad \text{sonst}\end{cases} \tag{7.6}$$

Zeitlich gewichteter Mittelwert

Bei der Überwachung der Apparatur durch Vergleich derzeitiger Werte mit aufgelaufenen Mittelwerten gemäß Gleichung 7.5 kann es günstiger sein, daß man die Beiträge der einzelnen Werte gemäß ihrem zeitlichen Alter wichtet. Hierbei bietet sich folgende (rekursive) Methode an:

$$\bar{q}_{i,k}^{(\tau)}=\bar{q}_\lambda^{(\tau)}(T_k):=(1-\tau)q_{\lambda,k}+\tau\bar{q}_\lambda^{(\tau)}(T_{k-1})\quad 0<\tau<1\,. \tag{7.7}$$

Der Parameter τ spielt dabei die Rolle eines Alterungskoeffizienten, da

$$\begin{aligned}
\bar{q}_{\lambda,k}^{(\tau)} &= (1-\tau)q_{\lambda,k}+\tau\bar{q}_{\lambda,k-1}^{(\tau)} \\
&= (1-\tau)q_{\lambda,k}+\tau(1-\tau)q_{\lambda,k-1}+\ldots+\lambda^{k-1}(1-\tau)q_{\lambda,1}
\end{aligned}$$

und sich so das Gewicht des um j Zeitintervalle zurückliegenden Beitrags ablesen läßt:

$$w_j=\tau^{k-j}(1-\tau). \tag{7.8}$$

Insgesamt führt diese Methode auf eine gegenüber **checkMean** verallgemeinerte Testfunktion:

$$\textbf{checkTmean}(q_{\lambda,k+1},\bar{q}_{\lambda,k}^{(\tau)},s_{\lambda,k}^{(\tau)};,\tau,\alpha_\lambda)\ ,$$

die für $\tau=1$ in **checkMean** übergeht.

Zählraten

Handelt es sich bei dem Parameter q_λ um eine Zählrate $R = N/\Delta t$ (N ist die im Zeitintervall Δt gemessene Zahl der Ereignisse), dann kann die Varianz von R direkt zu $s_R^2 = \sqrt{N}/\Delta t$ abgeschätzt werden, sofern die Bedingungen für das Vorliegen einer Poisson-Verteilung erfüllt werden.
Damit liegen alle Informationen zur Festlegung von Konfidenzintervallen und somit zur Definition einer Testfunktion vor. Bei richtiger Berücksichtigung des jeweils kleineren bzw. größeren Wertes genügt beim Test auf Übereinstimmung zweier poisson-verteilter Raten R_1 und R_2 eine einseitige Abfrage; dafür wird mit dem vorgegebenen Konfidenzniveau α_P die Testfunktion

$$\textbf{ratePoisson}(R_1, R_2; \alpha_P) = \begin{cases} 1 & : \quad \int_0^{\min(R_1,R_2)} P(x, \max(R_1, R_2))dx > \alpha_P \\ 0 & : \quad \text{sonst} \end{cases} \tag{7.9}$$

definiert, wobei $P(x, \mu)$ die Poisson-Verteilung mit dem Mittelwert μ darstellt.
Bei großen Zahlen ist es zweckmäßig mit der dann äquivalenten und leichter handhabbaren Gaußverteilung zu rechnen und als entsprechende Testfunktion zu definieren

$$\textbf{rateGauss}(R_1, R_2; \alpha_G) = \begin{cases} 1 & : \quad \sqrt{\frac{2}{\pi}} \int_{-\infty}^{\frac{-|R_1-R_2|}{\max(\Delta R_1, \Delta R_2)}} e^{-x^2/2} dx > \alpha_G \\ 0 & : \quad \text{sonst .} \end{cases} \tag{7.10}$$

Driften

Ist bei einem statistisch fluktuierendem Parameter q_λ der statistischen Fluktuation eine systematische Drift überlagert, dann kann diese festgestellt werden, falls sie größer ist als das "statistische Rauschen". Insbesondere ist die Drift nicht feststellbar in einem Zeitintervall, in dem die Drift kleiner ist als die zu diesem Zeitintervall gehörende Varianz.
Eine einfache Möglichkeit, die anfallende Information zur Feststellung einer Drift zu nutzen, besteht darin, an die zu den Zeiten T_κ gemessenen Werte $q_{\lambda,\kappa}$ mittels einer linearen Regression [mil90] laufend eine Gerade mit dem Steigungsparameter β_k anzupassen. Geprüft wird, ob β_k auf vorgegebenem Konfidenzniveau α_D mit Null, d.h. keiner Drift, verträglich ist:

$$\textbf{noDrift}(q_{\lambda,1}, \ldots, q_{\lambda,k}, \beta_k; \alpha_D) = \begin{cases} 1 & : \quad \beta \in [\beta_k^-, \beta_k^+] \\ 0 & : \quad sonst \end{cases} \tag{7.11}$$

Hierbei sind $\beta_k^\pm(\alpha_D)$ die Grenzen des Intervalls für den Steigungsparameter β_k zum Konfidenzniveau α_D. Die Berechnung der Regression wird beim Prototypen rekursiv durchgeführt.

7.2.2 Tests auf der Basis von Histogrammen

In ähnlicher Weise wie nach der Übereinstimmung der beobachteten Parameter $q_{\lambda,k}$ mit vorliegenden Werten kann man auch nach der zwischen gemessenen Histogrammen fragen. Zum Vergleich zweier Histogramme wurden drei Verfahren [ead71] in Erwägung gezogen, nämlich der

- χ^2-Test,
- Kolmogorov-Smirnov-Test,
- und der Smirnov-Cramer-Von-Mises-Test.

Der Smirnov-Cramer-Von-Mises-Test wurde bisher wegen seiner numerischen Komplexität für die Realisierung nicht weiter verfolgt. Im folgenden bezeichnen h_1, h_2 zwei zu vergleichende Histogramme und N_1, N_2 die Summe ihrer Kanalinhalte.

χ^2 - Test

Falls die statistischen Schwankungen der Inhalte $h_1(i)$ und $h_2(i)$ der I Histogrammkanäle normalverteilt[2] sind, dann gilt dies ebenfalls für deren Differenz, und zwar mit mit der Varianz $h_1(i) + h_2(i)$. Damit folgt die Größe

$$\chi^2_{obs} = \sum_i^I \frac{(h_1(i) - h_2(i))^2}{h_1(i) + h_2(i)} \tag{7.12}$$

unter der Voraussetzung, daß die beiden Histogramme nur verschiedene Stichproben derselben Grundgesamtheit sind, und daß sie unter der Nebenbedingung $N_1 = N_2$ aufgenommen[3] wurden, einer χ^2-Verteilung mit $I - 1$ Freiheitsgraden. Die Wahrscheinlichkeit $P(\chi^2 \geq \chi^2_{obs}|H_0)$, daß χ^2 unter der Hypothese H_0 - Übereinstimmung der beiden verglichenen Histogramme - *zufällig* einen Wert χ^2_{obs} oder größer annimmt, beträgt

$$P(\chi^2 \geq \chi^2_{obs}|H_0) = 1 - P(\frac{I-1}{2}, \frac{\chi^2_{obs}}{2}) \quad \text{mit} \quad P(a,x) = \frac{\int_0^x e^{-t} t^{a-1} dt}{\Gamma(a)} \ . \tag{7.13}$$

Damit läßt sich auf der Basis des χ^2-Tests zu vorgegebenem Signifikanzniveau α_{χ^2} eine Testfunktion für den χ^2-Test formulieren:

$$\textbf{chi2}(h_1, h_2; \alpha_{\chi^2}) = \begin{cases} 1 & : \quad P(\chi^2 \geq \chi^2_{obs}|H_0) > \alpha_{\chi^2} \\ 0 & : \quad \text{sonst} \end{cases} \tag{7.14}$$

Kolmogorov-Smirnov-Test

Der *Kolmogorov-Smirnov-Test* [ead71, pr88] stellt ein im Vergleich zum χ^2-Test einfacheres Verfahren zum Vergleich von Histogrammen dar. Der Kolmogorov-Smirnov-Test vergleicht die aus zwei unabhängigen Stichproben einer eindimensionalen, stetigen Zufallsvariablen gewonnenen kumulativen Verteilungen. Deshalb ist dieser Test zunächst nicht auf das vorliegende Problem anwendbar, so daß die im folgenden geschilderte "pragmatische" Vorgehensweise, bei der der Kolmogorov-Smirnov-Test einfach auf den Vergleich von Histogrammen übertragen wurde, noch auf ihre Tragfähigkeit untersucht werden muß (s.u.).

[2] Diese Voraussetzung ist nur bei genügend großen Kanalinhalten erfüllt, andernfalls wären diese poissonverteilt; über die Erfüllung dieser Voraussetzung können keine einfachen Aussagen gemacht werden, aber man geht davon aus, daß kein Kanal weniger als 5 und nur wenige Kanäle weniger als 10 Ereignisse enthalten darf[ead71].

[3] Andernfalls sind die Kanalinhalte und die Varianzen entsprechend zu normieren.

Als Maß D für den Abstand der zu vergleichenden Histogramme h_1, h_2, wird der maximale Abstand zwischen den beiden kumulativen Verteilungen S_1 und S_2 herangezogen:

$$D_{obs} = \max|S_1(i) - S_2(i)| \quad \text{mit} \quad S_k(i) = \frac{1}{N_k} \sum_{j<=i} h_k(j) \; . \tag{7.15}$$

Die Wahrscheinlichkeit $P(D \geq D_{obs}|H_0)$ einen Abstand $\geq D_{obs}$ unter der Hypothese H_0 – die beiden Verteilungen entstammen der gleichen Gesamtmenge – zu beobachten, beträgt:

$$\begin{aligned} P(D \geq D_{obs}|H_0) &= Q_{KS}(\hat{D}_{obs} := \sqrt{\tfrac{N_1 N_2}{N_1+N_2}} D_{obs}) \quad \text{mit} \\ Q_{KS}(x) &:= \begin{cases} 1 & : \; x \leq 0 \\ 2\sum_{j=1}^{\infty}(-)^{j-1} e^{-2j^2x^2} & : \; x > 0 \end{cases} \; . \end{aligned} \tag{7.16}$$

Einen Algorithmus zur numerischen Berechnung von Q_{KS} findet sich in [pr88]. Unter Vorgabe des Signifikanzniveaus α_{KS} läßt sich mit 7.15 und 7.16 eine Testfunktion für den Kolmogorov-Smirnov-Test angeben:

$$\mathbf{kst}(h_1, h_2; \alpha_{KS}) = \begin{cases} 1 & : \; P(D \geq D_{obs}|H_0) > \alpha_{KS} \\ 0 & : \; \text{sonst} \end{cases} \; . \tag{7.17}$$

Anmerkung:Kolmogorov-Smirnov-Test und Binierung

Die Anwendbarkeit des Kolmogorov-Smirnov-Test auf Histogramme wurde durch Monte-Carlo-Rechnungen überprüft. Dazu wurden zunächst zwei Stichproben im Umfang von je n Elementen einer im Interval $[0, 1]$ gleichverteilten Zufallsvariablen x gewürfelt und mit dem Kolmogorov-Smirnov-Test auf ihre Gleichheit überprüft. Dies geschah insgesamt $N = 5000$ mal, so daß die Häufigkeitsverteilung $\Delta N/\Delta P$ der diesen Vergleich bewertenden Wahrscheinlichkeit $P(D \geq D_{obs}|H_0)$ (Glg.7.16) gewonnen wurde. Diese Verteilung ist als Teilbild (1) in Abb.7.1 dargestellt: Die Monte-Carlo-Rechnungen reproduzieren die bei Gültigkeit des Tests erwartete gleiche Häufigkeit von ΔP-Bins gleicher Größe.

In weiteren Schritten wurden die gleichen Stichproben der eindimensionalen Variablen x zu Histogrammen zusammengefaßt durch Einsortieren der Einzelereignisse in Bins der Breite 0.01, 0.03 und 0.05. Für alle drei Binierungen wurde wiederum der Vergleich von je zwei Histogrammen durch den Kolmogorov-Smirnov-Test bewertet. Die Häufigkeitsverteilung der dabei ermittelten Wahrscheinlichkeiten P ist in den Teilbildern (2) - (4) der Abb.7.1 dargestellt. Man erkennt eine Verschiebung der Häufigkeit zu höheren P-Werten, d.h. die Abstände $\hat{D}_{obs}$ der binierten Verteilungen sind nicht mehr streng Kolmogorov-Smirnov verteilt, dies umso mehr, je gröber die Binierung.

Dies Ergebnis entspricht in der Tendenz durchaus der Erwartung. Da man nämlich mit dem Binieren der ursprünglichen Variablen die Abstände zwischen zwei Stichproben nur noch an den Randstellen der Bins vergleicht und damit die Abstände gegenüber den ursprünglichen Stichproben im Mittel verkleinert

Diese Verschiebung der Wahrscheinlichkeit bei binierten Daten zu höheren Werten hin kann man bei der Anwendung des Kolmogorov-Smirnov-Tests durch entsprechende Angabe der akzeptierenden Grenze durchaus Rechnung tragen, so

daß dieser Test durchaus auch zur Bewertung von Histogrammen herangezogen werden kann (siehe auch Abb.7.3).

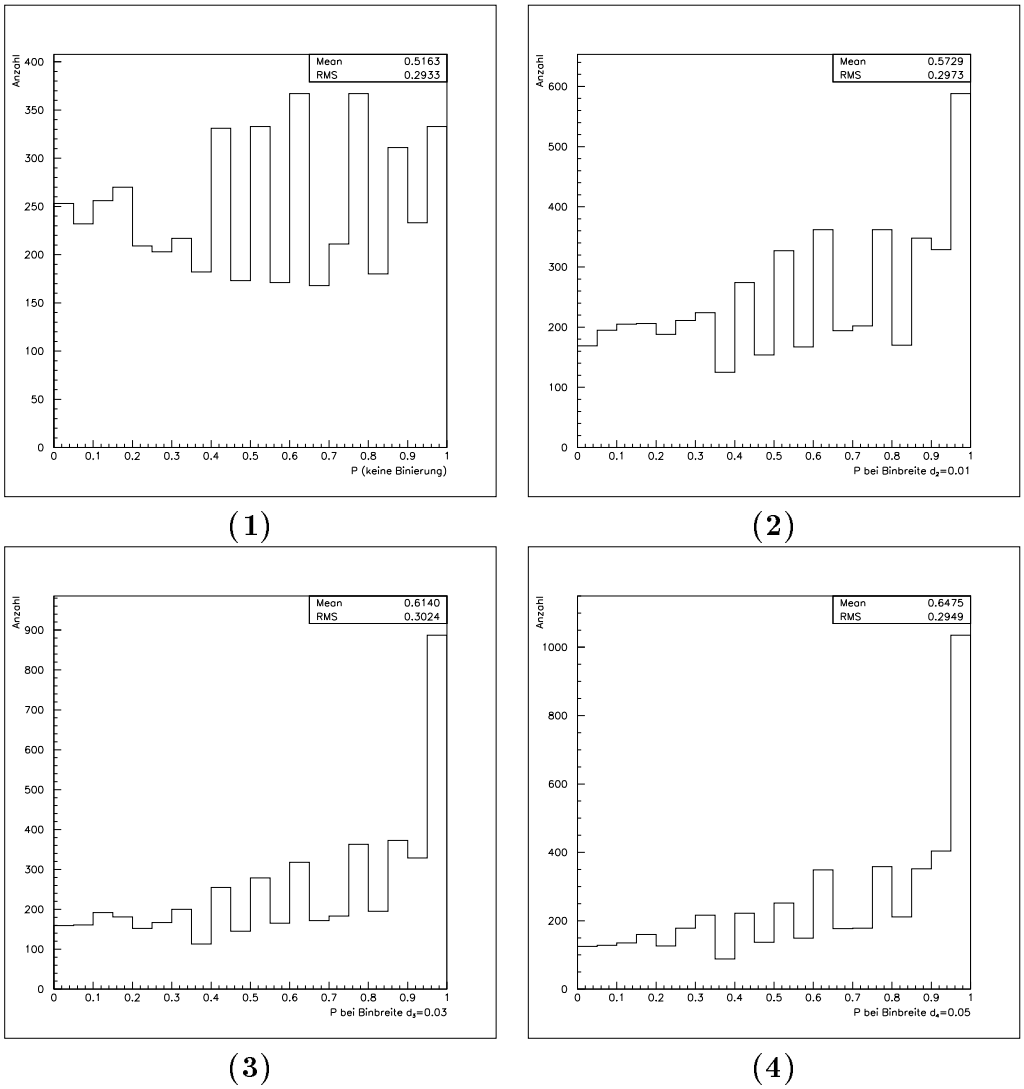


Abb.7.1: Häufigkeitsverteilung der P-Werte bei den Rechnungen zur Überprüfung der Anwendbarkeit des Kolmogorov-Smirnov-Tests auf Histogramme. **(1):** Ursprüngliche Daten, **(2)-(4):** Binierte Daten mit Binbreiten 0.01, 0.03 und 0.05.

7.2.3 Zwei Anwendungsbeispiele

Ausfall einer TDC-Karte

Die Wirksamkeit der verschiedenen Testfunktionen wurde an verschiedenen Beispielen aus Messungen und aus Datensimulationen überprüft. Im folgenden soll als Beispiel der Ausfall einer TDC-Karte im Auslesesystem der X1-Drahtebene der Driftkammer im Spektrometer B diskutiert werden[4]. Dem Beispiel liegt eine konkrete Messung aus einer Strahlzeit im Mai 1995 zu Grunde.

[4]Die für die Tests entwickelten Software-Werkzeuge werden im folgenden Abschnitt 7.3 besprochen.

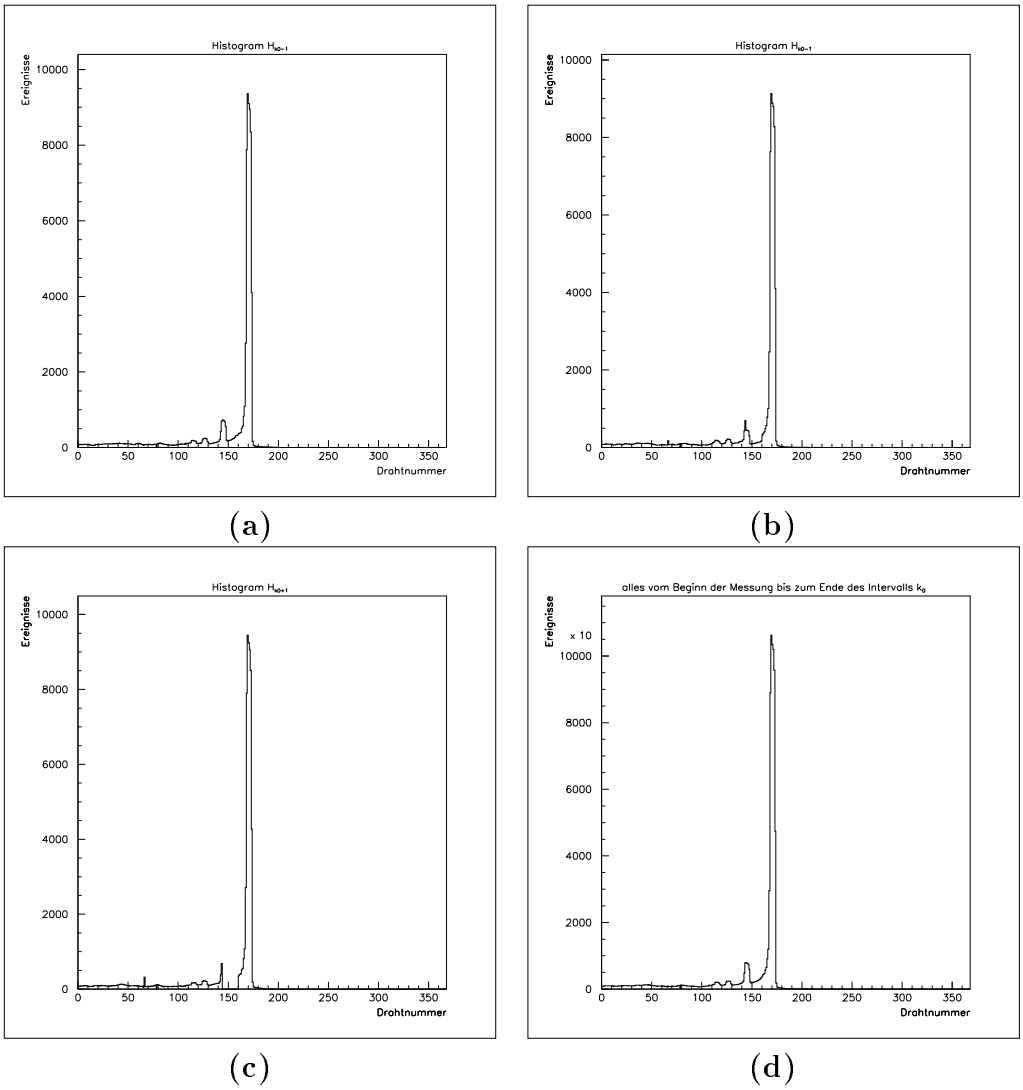


Abb.7.2: Häufigkeitsverteilungen der Drähte, die in der X1-Ebene von Spektrometer B angesprochen haben.

Für die Auswertung der auf Band gespeicherten Daten wird die Meßzeit in k gleiche Zeitintervalle eingeteilt. Aus der Ereignismenge im κ-ten Intervall werden Histogramme über geeigneten Variablen gebildet, in diesem Fall die Häufigkeit mit der Drähte in der X1-Ebene der Driftkammer in Spektrometer B angesprochen haben; diese Histogramme werden mit $H_\kappa = H(T_\kappa)$ bezeichnet.
Auf diesen Daten wird der Ausfall einer TDC-Karte zur Zeit T_0 im Zeitintervall T_{κ_0} simuliert, indem ab diesem Zeitpunkt die Driftzeitinformation der betreffenden Drähte ausgeblendet wird. Abbildung 7.2 zeigt unter (a) mit H_{κ_0-1} das letzte Histogramm aus der Zeit, in der die TDC-Karte noch richtig arbeitete, unter (b) und (c) folgen die Histogramme der beiden nachfolgenden Zeitintervalle, unter (d) wird das seit Beginn der Messung bis zum Ende des Intervalls κ_0 akkumulierte Histogramm gezeigt.
Auf die k Histogramme wurden der χ^2-Test und der Kolmogorv-Smirnov-Test[5]

[5]Bei der Berechnung von $\hat{D}_{obs}$ in Glg.7.16 wird für N_1 bzw. N_2 nicht die Summe über alle Histogrammkanäle, sondern die Anzahl der Ereignisse verwendet, weil bei einem Ereignis mehrere Drähte "feuern" und damit mehrere Kanäle im Historgramm *korreliert* inkrementiert werden.

angewandt. In Abb. 7.3 ist das Ergebnis für den Vergleich der einzelnen Histogramme mit ihrem jeweiligem Vorläufer dargestellt. Der Einbruch in den berechneten Wahrscheinlichkeiten im 10.Intervall ist deutlich zu sehen. Bei diesem Beispiel handelt es sich zwar um eine nachträgliche Manipulation an bereits gemessenen Daten, der Anwendung der Methode zur Feststellung von Fehlern in der laufenden Messung steht aber nichts im Wege.

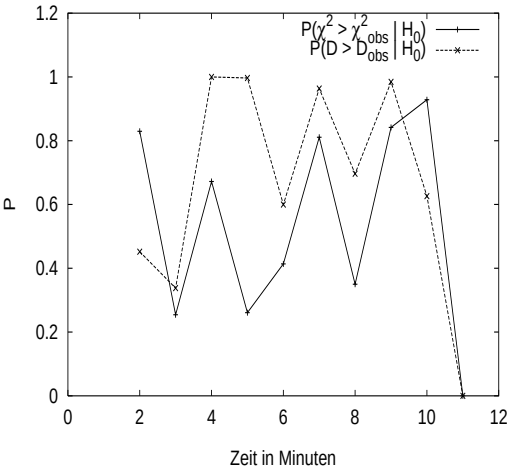


Abb.7.3: Zeitlicher Verlauf der Wahrscheinlichkeiten $P(\chi^2 \geq \chi^2_{obs}|H_0)$ und $P(D \geq D_{obs}|H_0)$. Beide Tests zeigen deutlich den im Zeitintervall $I_{\kappa_0=10}$, d.h. zwischen der zehnten und elften Minute einsetzenden Defekt an. Entsprechend dem oben diskutierten Monte-Carlo-Rechnungen zu binierten Daten liegt der Kolmogorov-Smirnov-Test systematisch höher.

Einbruch in der Ansprechwahrscheinlichkeit

Zur Überwachung der Driftkammern wird im Prototypsystem z.B. die Ansprechwahrscheinlichkeit η_i (*Effizienz*) für jede Kammerebene i im Zeitintervall κ berechnet: $\eta_i(T_\kappa)$. Sie ergibt sich hier aus der Anzahl N_{i-hit} der Ereignisse, bei denen die Ebene i und mindestens zwei weitere der insgesamt vier Ebenen angesprochen haben, und der Anzahl $N_{i-nohit}$ der Ereignisse, bei denen mindestens zwei Ebenen nicht aber die Ebene i angesprochen haben:

$$\eta_i = \frac{N_{i-hit}}{N_{i-nohit} + N_{i-hit}} = \frac{N_{i-hit}}{N_{trials}} \ .$$

Die Entwicklung der Kammereffizienz $\eta_i(T_\kappa)$ in Abhängigkeit der Zeit ist in Abb.7.4 dargestellt. Der deutlich erkennbare "Sprung" in der Effizienz ist auf den Ausfall einer TDC-Karte zurückzuführen.

Die besprochenen Tests können als ein Expertensystem zur Überwachung der Funktionsfähigkeit der Anlage und ggf. zur Diagnose auftretender Fehler genutzt werden.

7.3 Werkzeuge zur On-line Datenkontrolle

Die Software zur Durchführung der angesprochenen Testverfahren wurde als offenes System konzipiert, das aus einem Satz flexibel verwendbarer Bauteile in Form von C++-Klassen besteht. In einem ersten Schritt wurde zur Demonstration der Leistungsfähigkeit des Konzepts ein Prototyp-System erstellt, mit dem erste Erfahrungen gewonnen werden konnten.

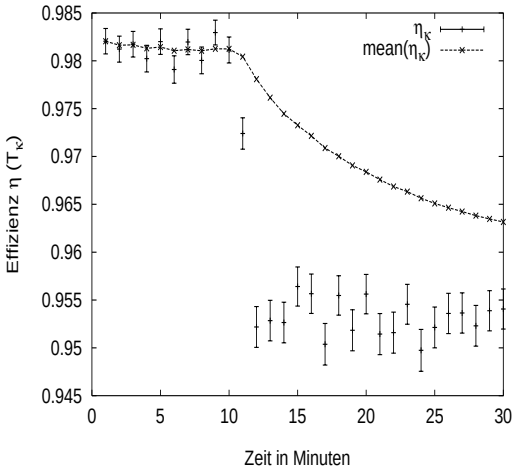


Abb.7.4: Zeitlicher Verlauf der Effizienz η_κ und deren Mittelwert $\bar{\eta}_\kappa$. Der statistische Fehler der gemessenen Effizienz errechnet sich gemäß $\Delta\eta_\kappa = \Delta N_{i-hit}/N_{trials} = \sqrt{\eta_\kappa(1-\eta_\kappa)/N_{trials}(T_\kappa)}$

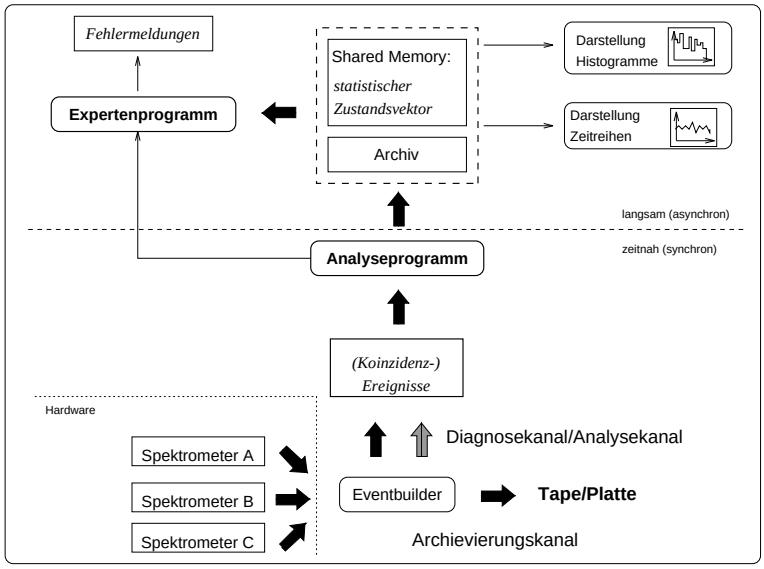


Abb.7.5: Zusammenspiel der verschiedenen Programme zur Aufnahme und Analyse der Daten sowie zur Überwachung der Apparatur im On-line Betrieb.

7.3.1 Organisation der Programme

Die Komplexität der Experimentieranordnung erfordert es, daß parallel zur Datenerfassung ein Programm zur Kontrolle der Anlage abläuft. In einem Koinzidenzexperiment werden die Daten mit Hilfe der Datenerfassungssoftware MECDAS [kry95] zunächst auf jedem der Spektrometer getrennt erfaßt und dann von einem zentralen Programm, dem sogenannten Eventbuilder [kry95], auf einem zentralen Rechner zu einem Gesamtereignis zusammmengefaßt (Abb.7.5), die dabei erforderliche Information wird in Form eines wohldefinierten Triggermusters, das von einer zentralen Koinzidenz-Logik erzeugt, bereitgestellt [ri94].

Die Eventbuilder-Software ist im wesentlichen ein einfaches Puffer-Management-System für die in einem Shared-Memory-Segment gespeicherten Ereignisse. Der Eventbuilder stellt die im Puffer gespeicherten Ereignisse auch Anwenderprogrammen zu einer parallel zur Messung laufenden Weiterverarbeitung zur Verfügung. Der Eventbuilder fungiert dabei als Serverprozeß, der die Daten auf (maximal) drei Kanälen anbietet:

- dem Archivierungskanal,
- dem Analysekanal,

- dem Diagnosekanal.

Über den Archivierungskanal werden die Daten der Sicherung auf Massenspeichern (Band, Platte) zugeleitet. Dieser Kanal ist während der Messung belegt. Die Daten werden dem Analyseprogramm über den Analyse- oder den Diagnosekanal zugeleitet. Hierbei ist insbesondere der Diagnosekanal von Bedeutung, weil nur über diesen Kanal *alle* Ereignisse weitergereicht werden (siehe [kry95]).

Die so zur Verfügung gestellte Information wird benutzt, um den Zustandsvektor $\vec{Q}(T)$ zur Zeit T aufzubauen und zu bewerten. Dabei ist es günstiger, die Aufgaben des Erstellens (Analyse/Reduktion) und des Bewertens (Überwachung, Expertise) des Zustandsvektors auf verschiedene Programme zu verteilen, da die Ereignisse schnell verarbeitet werden müssen, damit der Puffer des Eventbuilders, in dem die Ereignisse auf ihre Verarbeitung warten, nicht überläuft; durch die Aufspaltung in zwei Programme kann die Weiterverarbeitung parallelisiert und damit beschleunigt werden. Diese Aufteilung ist möglich, da die Aufgaben des Analyseprogramms unabhängig von den oft langwierigen Rechnungen des Expertensystems[6] sind, und sie ist sinnvoll, da das Analyseprogramm für sich allein schon ein sehr nützliches Hilfsmittel darstellt.

Das Analyseprogramm kommuniziert über den Diagnosekanal mit dem Eventbuilder. Es wird zu Beginn einer jeden Messung gestartet. Bei jedem Ereignis sortiert es die Information in die entsprechenden Histogramme h_μ ein und berechnet in periodischen Abständen die Zustandsparameter p_λ. Damit berechnet es den Zustandsvektor $\vec{Q}$, der in einem Shared-Memory-Segment abgelegt wird, damit weiterverarbeitende Programme jederzeit (asynchron) auf diese Information zugreifen können, z.B. zur Darstellung der Histogramme. Ein eigenes Programm sorgt für die graphische Darstellung des zeitlichen Verlaufs von Zustandsparametern (siehe z.B. Abb.7.4), wozu auch die Informationen aus vorausgegangen (Teil-)Messungen benötigt werden, die in einem Archiv auf Platte gespeichert sind. Histogramme und Parameter sind durch Klassen repräsentiert, deren Daten aus dem Shared-Memory abgerufen werden können. Der Zugriff auf die Daten im Shared-Memory erfolgt mit Hilfe von symbolischen Namen.

Das Expertenprogramm zur Überwachung wird vom Analyseprogramm immer dann, wenn es einen neuen Zustandsvektor berechnet hat, zur Überprüfung des Zustandsvektors aufgefordert. Stellt das Expertenprogramm mit den oben dargestellten Methoden eine Fehlersituation fest, dann gibt es eine entsprechende Meldung aus. Diese Ausgabe erfolgt mit Hilfe des Statusservers des ECS (s.S.47), über den der Experimentator in einheitlicher Weise auf Fehler aufmerksam gemacht wird.

[6] Der Begriff Expertensystemsystem wird auf das Überwachungsprogramm angewandt, obwohl bei dessen augenblicklicher Realisierung keine Techniken aus dem Bereich der "künstlichen Intelligenz" (AI) verwendet wurden.

7.3.2 Einige Realisierungsdetails des On-line Systems

Histogramme

Zur Akkumulation und Verwaltung von Histogrammen waren bereits einige am Institut entwickelte Softwaremodule vorhanden, die sich allerdings aus folgenden Gründen als nicht ausreichend erwiesen [hei92]:

- Sie erlaubten keine Behandlung von Fließkommazahlen als Grenzen oder als Kanalinhalte und von negativen Zahlen als Grenzen.
- Es fehlte die Möglichkeit zum Abspeichern von Histogrammen in Dateien.
- Der Aufruf dieser Module konnte nicht aus einer Bibliothek heraus erfolgen.

Um diese Mängel zu beseitigen, wurde eine neue Bibliothek, die *Tasty-Bibliothek*[kra92d], erstellt, die wegen der bereits genannten Vorteile in der Sprache C++ implementiert wurde. Dabei werden Histogramme als eine Klasse realisiert, die die zur Verwaltung, zur Analyse und zur Bewertung benötigten Dienste zur Verfügung stellt.
Damit man die Histogramme, den Anforderungen entsprechend, parallel zur quasi kontinuierlichen Akkumulation darstellen oder verarbeiten kann, befindet sich der Datenteil von Histogramm-Objekten im Shared-Memory. Da im Batchbetrieb aber i.a. kein Interesse daran besteht, asynchron auf Histogramme zuzugreifen, kann sich der Datenteil alternativ zum Shared-Memory auch innerhalb des prozeßeigenen Speichers befinden.
Um insbesondere auf Platte gesicherte Histogramme in einfacher Weise von einem Rechner auf einen anderen Rechner kopieren zu können, erfolgt ihre Speicherung im maschinen-unabhängigen XDR-Format (s.S.54).
Die Implementation generischer Funktionen wird dadurch ermöglicht, daß die Histogramme der beiden Speichertypen durch verschiedene Klassen repräsentiert sind, die von einer gemeinsamen Basisklasse abstammen. Ein Beispiel für die Verwendung der neuen Histogrammbibliothek ist in Abb.7.6 dargestellt.
Zur Darstellung der Histogramme können die Programme `xanalyse` [schri93] und `display` [kry95] verwendet werden. Um eine weitergehende graphische Verarbeitung mit den Software-Werkzeugen der CERN-Bibliothek, z.B. `paw`, zu ermöglichen, steht das Programm `tasty2paw` zur Umwandlung dieser Histogramme in das dort zu Grunde liegende HBOOK-Format [bru91] zur Verfügung.

Monitorfelder

Der zweite von der konkreten Anwendung an der Drei-Spektrometer-Anlage unabhängige Baustein des Prototypsystems ist die Software zur Verwaltung der im Shared-Memory gespeicherten Zustandsparameter. Sie sind in einem Feld (Array) aus n-Tupeln, bei dem jede Komponente des Feldes durch einen symbolischen Namen charakterisiert wird, abgelegt; eine solche Komponente wird im folgenden als *Monitorfeld* bezeichnet.
In der Prototypinstallation besteht jedes n-Tupel aus maximal drei reellwertigen Zahlen: dem aktuellen Wert eines Zustandsparameters, dessen zeitlichem Mittelwert und einer Fehlerangabe. Diese Angaben haben sich bisher als ausreichend

```
...
HcsFastHisto a (0, 10, 100); // declare two histograms a,b
HcsFastHisto b (0, 10, 100); // with 100 bins from 0 to 10

for (int i = 0; i < 1000; ++i) { // loop
   a += randomGauss(); // fill a and b with a random number
   b += randomGauss(); //
}
HcsFastHisto c = a + b;// build new histogram c by adding a and b
c.write("data");       // save histogram c on file named data
```

Abb.7.6: C++ Beispiel mit Tasty-Histogrammen. Allen Routinen und Datenklassen der Bibliothek ist das Präfix "Hcs" für **H**istogram-**C**omparing-**S**ystem vorangestellt. `HcsFastHisto` ist eine Histogrammklasse, deren Objekte keinen zweiten internen Array zum Vorhalten einer kanal-spezifischen Varianz mitführen - sie sind damit schneller (fast) zu akkumulieren.

erwiesen. Die weitere Bedeutung dieser Zahlen, z.B. die Art des Fehlers, wird mit dem Typ des Monitorfeldes beschrieben, der zusammen mit dem Namen zusätzlich abgespeichert wird. Damit sind diese Daten voll selbstbeschreibend und können als Basis für daten-gesteuerte Programme dienen.

Um sehr modulare Programme zu erhalten, sind die Monitorfelder ebenfalls in Form von C++-Klassen implementiert, die alle von einer gemeinsamen Basisklasse abstammen. Die Basisklasse enthält alle allgemein notwendigen Routinen, z.B. die Dienste zum Ein- und Auslesen eines Monitorfeldes. Außerdem definiert sie die allgemeine Schnittstelle aller Feldtypen. Die Klassen unterscheiden sich im wesentlichen in der Bedeutung der aktuellen Werte, insbesondere der Fehlerangabe, und durch die Funktionen, die zum Vergleich zweier Objekte der gleichen Klasse herangezogen werden.
Um die Verwaltung der Monitorfelder im Shared-Memory zu vereinfachen und um den Gesamtzustand zu einem Zeitpunkt $\vec{Q}(T_{\kappa_0})$ einfacher festhalten zu können, befinden sich die Daten aller Monitorfelder für diesen Zeitpunkt innerhalb eines einzigen Speicherblocks. Die Verwaltung eines solchen Speicherblocks erfolgt durch das sogenannte `MonitorBoard`. Für den Zugriff auf die Daten im Shared-Memory werden die Daten eines Monitorfeldes über das `MonitorBoard` indirekt adressiert.
Da das Analyseprogramm zur Berechnung des aktuellen Zustands $\vec{Q}(T_{\kappa_0})$ auch den jeweils vorhergehenden Zustand $\vec{Q}(T_{\kappa_0-1})$ benötigt (siehe z.B. Glg.7.3) und darüberhinaus zur Durchführung von längerfristigen Vergleichen auch weiter zurückliegende Zustandsvektoren $\vec{Q}(T_{\kappa_0-\xi})$, arbeitet das `MonitorBoard` mit drei Speicherblöcken, die jeweils den genannten Zuständen zugeordnet sind. Über die indirekte Adressierung kann das Expertenprogramm in einfacher Weise auf Zustände verschiedener Zeitpunkte zugreifen.

Die apparatespezifischen Klassen im Analyseprogramm

Ausgangspunkt für die Datenanalyse ist ein Objekt der `EventDecoder`-Klasse, das die im MECDAS-Format [kry95] kodierten Einzelereignisse entpackt. Nach dem Entpacken stehen die Daten für Anwendungen als C-Struktur zur Verfügung. Im Rahmen der Datenerfassungssoftware wird diese C-Struktur als "logische Konfiguration" bezeichnet. Die Struktur ist äquivalent zu einer Baumstruktur, bei der jedes Blatt einer Strukturkomponente entspricht.
Die Einführung einer Klasse zum Dekodieren der Ereignisse hat den Vorteil, daß alle Dekodierungsroutinen und der Zugriff auf die Daten gekapselt sind. Damit ist es möglich, erforderlichenfalls Dekodiermethoden auf einfache Weise auszuwechseln[7].

Um das Analyseprogramm soweit wie möglich von der Ereignisstruktur unabhängig zu machen, bietet jeder `EventDecoder` den Dienst an, auf Knoten oder Blätter der "logischen Konfiguration", d.h. auf die Speicheradresse jeder Strukturkomponente über deren (in Form eines Strings vorliegenden) Namen direkt zuzugreifen. Dieser Dienst wird z.B. von den sogenannten `DetektorEvents` in Anspruch genommen, die die Detektoren der Drei-Spektrometer-Anlage repräsentieren. Die `DetektorEvents` definieren eine weitgehend von der "logischen Konfiguration" unabhängige Beschreibung eines Detektors. Über diese Klasse wird auf die einem Detektor zugeordneten Daten in einem Ereignis zugegriffen. Darüberhinaus werden Dienste zur Berechnung detektorspezifischer Größen bereitgestellt[8].

Auf der Basis der standardisierten Aufarbeitung der Daten durch `DetektorEvents` ist die standardisierte Softwarebibliothek aufgebaut. Damit wird erreicht, daß ohne Eingriffe in die Programme Änderungen am Aufbau der "logischen Struktur" gemacht werden können. Beispielsweise muß beim Tausch eines 12 Bit Zählers gegen einen 24 Bit Zähler wegen des größeren Datentyps die "logische Konfiguration" geändert werden, die Beschreibung des betreffenden Detektors durch eine entsprechende Klasse als solche bleibt davon jedoch unberührt. Gleiches gilt, wenn man eine Komponente zur Gesamtstruktur hinzufügt.

Die für die Überwachung eines Detektors notwendigen Histogramme werden von einer dem jeweiligen Detektortyp zugeordneten Klasse (`DetektorStatistics`) verwaltet und akkumuliert. Durch das Zusammenfassen aller zu einem Detektor gehörenden Histogramme wird die Programmierung überschaubar, insbesondere deshalb, weil die Verwaltungsarbeit im Falle von C++ durch den Compiler erledigt wird (siehe Diskussion auf S.59).
Schließlich gibt es noch für jeden Detektor eine sogenannte `DetektorMonitor`-Klasse, die auf den oben genannten Klassen aufbaut. Sie enthält die Algorithmen zur periodischen Berechnung der detektorspezifischen Zustandsparameter, die in Monitorfeldern abgelegt sind.

[7] Dies ist im vorliegenden Fall von besonders aktueller Bedeutung, weil alternativ zu den MECDAS-Entpackungsroutinen im Rahmen einer anderen Arbeit [di95] ein neuer Dekodierer entwickelt wurde, der z.B. um mehr als einen Faktor zehn schneller ist.

[8] Zum Beispiel enthält die Klasse zur Beschreibung der Spurendetektoren Funktionen zur Berechnung der Bildebenenkoordinaten aus den gemessenen Driftzeiten.

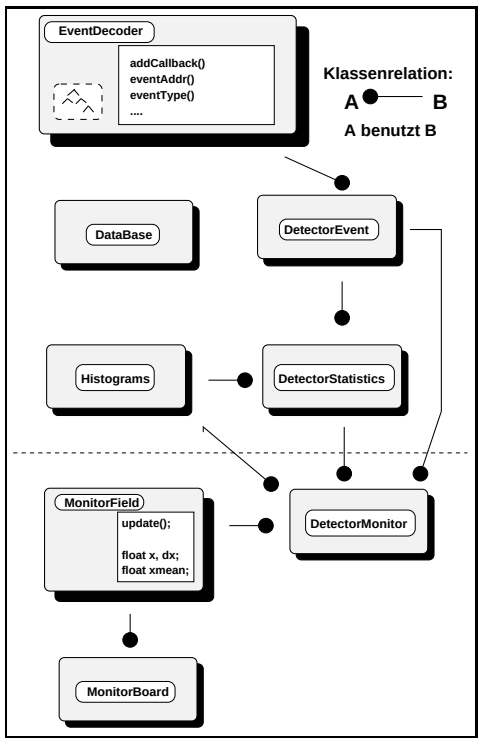


Abbildung 7.7: Bestandteile eines Analyseprogramms und deren Beziehungen untereinander. Die jedem Detektor zugeordenete `DetektorEvent`-Klasse holt sich die Daten mit Hilfe des `Event-Decoders` aus dem einlaufenden Datenstrom. Alle Histogramme, die während der On-line Analyse zur Überwachung eines Detektors benötigt werden, stellt die detektorspezifischen `Statistics`-Klasse zur Verfügung. Weitergehende Statusinformation, die das Expertensystem benötigt, wird aus diesen Bausteinen in der detektor-spezifischen `Monitor`-Klasse errechnet. Diese Information wird dann in den `MonitorFields` abgelegt, die insgesamt vom `MonitorBoard` verwaltet werden. Durch diesen strengen Aufbau ist es möglich, das Programm auch als "abgespecktes" Analyseprogramm zu betreiben, indem mittels eines Konfigurationsparameters der gesamte Anteil unterhalb der gestrichelten Linie ausgeblendet wird. Die Datenbank-Klasse wird von nahezu allen Klassen im Analyseprogramm genutzt, deshalb wurde zur besseren Übersicht deren Relation zu den anderen Klassen nicht eingezeichnet.

Aus diesen Bausteinen wurde im Analyseprogramms eine Klasse zur Beschreibung des Zustands eines Spektrometers im Analyseprogramm aufgebaut.

In vielen Fällen ist es schon während der Messung notwendig, Spektren aus den auf das Target zurückgerechneten Koordinaten aufzubauen, z.B. ein Missing-Energy Spektrum. Die Rückrechnung steht dem Analyseprogramm wiederum in Form einer Klasse, der sogenannten `TraceMachine` [kra92c] zur Verfügung. Für diese sehr rechenintesive Operation wurde ein Algorithmus entwickelt, dessen Aufbau im Anhang A dargestellt wird.
Überall dort, wo das Analyseprogramm[9] Konfigurationsinformation benötigt, wird sie durch ein Datenbankobjekt bereitgestellt, das auf einer Datei arbeitet.
Die wesentlichen Bestandteile eines Analyseprogramms sind in Abb.7.7 dargestellt.
Aus den oben genannten Komponenten wurde das Analyseprogramm `dwatch` (**d**ata**watch**) des Prototypsystems aufgebaut. Da es zum größten Teil über Parameter gesteuert wird, kann es bei nahezu allen Ein-, Zwei- und Dreiarmexperimenten mit der Drei-Spektrometer-Anlage als Standardprogramm zur Analyse während der Datennahme eingesetzt werden.

Das Expertenprogramm

[9] Dies gilt für alle Programme und Bibliotheken des Prototyps.

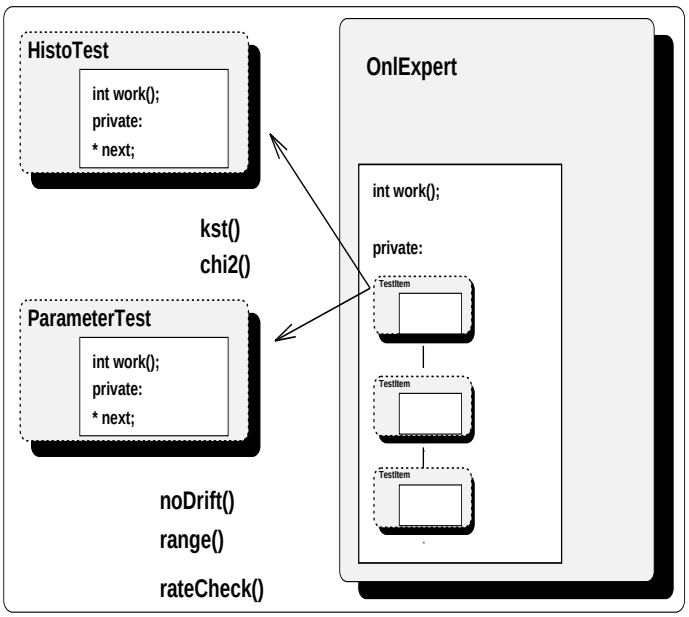


Abb.7.8: Bestandteile des Expertenprogramms. Die Tests, wie z.B. die Histogrammvergleiche oder die Parametertests, sind Teil einer Liste, die von einem Objekt der `OnlExpert`-Klasse verwaltet wird. Diese Liste ist geschützt vor dem Zugriff anderer Objekte im Programm, d.h. sie befindet sich im sogenannten *privaten* Bereich der `OnlExpert`-Klasse.

Im Abschnitt 5.1 wurde dargelegt, daß die Verwendung eines kommerziellen Expertensystems für die Überwachung im Rahmen des On-line Systems der Drei-Spektrometer-Anlage nicht in Frage kommt. Es war ein Ziel der vorliegenden Arbeit, mit dem "Expertenprogramm" ein Gerüst für den zukünftigen Ausbau der On-line Kontrolle zur Verfügung zu stellen.
Das Programm baut auf den bereits vorgestellten Modulen (Histogramme, Felder) auf. Um es einfach erweitern zu können, wurde der Vererbungsmechanismus benutzt: Jede Testmethode (Vergleich) wird in Form einer Klasse implementiert, die von einer allen gemeinsamen Basisklasse (`TestItem`) abstammt. Jedes `TestItem`-Objekt ist Teil einer Liste, die vom sogenannten `OnlExpert` verwaltet wird. Nach jeder Neuberechnung des Statusvektors $\vec{Q}(T)$ wird diese Liste durchlaufen und bei jedem Element die sogenannte `work()`-Funktion aufgerufen, die spezifisch für die jeweilige Klasse ist. In ihr werden die entsprechenden Überprüfungen ausgeführt, z.B. der Vergleich zweier Histogramme, und ggf. eine Fehlermeldung generiert. Dieses Arbeitsprinzip ist in der Abb.7.8 dargestellt.
Eine Erweiterung der Apparaturdiagnose kann leicht durch Implementierung weiterer Elemente in das Expertenprogramm vorgenommen werden: Zusätzliche Tests werden vom Experimentator durch die Deklaration einer Variablen für den entsprechenden Testtyp hinzugefügt, die Einführung neuer Testtypen zur Anpassung des Programms an neue Erfordernisse erfolgt durch die Hinzunahme einer neuen Klasse über den Vererbungsmechanismus.

7.4 Anwendungsbeispiele

Der Parametersatz, der zur Überwachung der Drei-Spektrometer-Anlage im Rahmen des Prototypsystems dient, wurde so gewählt, daß bei Änderungen der Experimentierbedingungen möglichst wenig Änderungen von seiten des Experimentators notwendig sind, womit sich die Prototypimplementation auf die Standardbestandteile der Anlage konzentriert.
Damit die laufende Berechnung des Zustandsvektors und dessen Beurteilung möglichst parallel zur Datennahme erfolgt, sind bisher Zustandsparameter gewählt

worden, die sehr direkt aus den Daten hervorgehen. Auf die Anwendung komplizierter Auswertebedingungen oder die Berechnung von Wirkungsquerschnitten ist zunächst verzichtet worden. Im folgenden werden die Parameter beschrieben, die als minimale Menge im Prototypsystem zur Überwachung herangezogen werden.

7.4.1 Strommonitore

Für die Bestimmung der experimentellen Wirkungsquerschnitte in Elektronenstreuexperimenten muß die Gesamtzahl der einlaufenden Elektronen bekannt sein. Die Bestimmung dieser Zahl erfolgt über die Ladungsmessung mit Strahlstrommonitoren. Wegen der zentralen Bedeutung dieser Größe für die absolute als auch nur für die relative Bestimmung der Wirkungsquerschnitte wird der Strom mit mehreren Methoden gemessen[10]:

Photoeffekt-Monitor: Der Photoeffekt-Monitor [eut94] nutzt die am letzten Ablenkdipol der Strahlführung vor der Spektrometerhalle emittierte Synchrotronstrahlung für die Strommessung aus. Die Strahlung trifft auf eine Edelstahlfolie, wo sie auf Grund des Photoeffekts Elektronen auslöst, die auf einer auf etwa 50 Volt liegenden Anode gesammelt werden. Der resultierende Photostrom ist proportional zum Strahlstrom. Die relative Genauigkeit der Strommessung ist besser als $\pm 1\%$ im Bereich von 1 nA bis 60 μA Strahlstrom. Auf Grund von Veränderungen der Folienoberfläche können allerdings Langzeitdriften auftreten, die eine absolute Rekalibrierung z.B. mit Hilfe der Förstersonde notwendig machen.

Förstersonde: Mit der Förstersonde [ed72] wird zur Bestimmung des Stroms das absolute Magnetfeld des Elektronenstrahls gemessen. Man erreicht mit diesem Verfahren eine (absolute) Genauigkeit von $\pm 0.3\ \mu A$, es stellt also nur bei großen Strahlströmen eine (absolute) Präzisionsmethode dar.

HF-Cavity: Das Meßprinzip mit einem HF-Cavity, bei dem die in einem Hohlraumresonator durch den hindurchtretenden Elektronenstrahl induzierte Feldstärke zur Strommessung genutzt wird, ist bereits mit Abbildung 5.7 im Abschnitt 5.4 besprochen worden.

Zum Auslesen der Förstersonde und des Photoeffekt-Monitors wird das jeweilige Meßsignal über einen Spannungs-Frequenzwandler in NIM-Impulse umgewandelt, die auf einen CAMAC-Zähler gegeben und über die Datenerfassung eingelesen werden. Durch Differentiation nach der Zeit kann daraus wieder auf den Strom geschlossen werden.
Die Überwachung des Strahlstroms, die durch den Vergleich mit einer oberen und unteren Schranke gemäß dem Testverfahren für einzelne Parameter erfolgt, ist insbesondere bei Messungen mit Flüssigtargets von Bedeutung, um zu jeder Zeit die Targetbelastung zu kennen; dies ist z.B. erforderlich, um ggf. auf stromabhängige Dichteschwankungen korrigieren zu können.
Das Vorhandensein dreier verschiedener Strommonitore legt in besonderer Weise

[10] Der ebenfalls installierte Faraday-Cup eignet sich nicht zur einer genauen Bestimmung des Strahlstroms. Hier wurden bei konstantem Strahlstrom relative Schwankungen um mehr als $\pm$ 2% gemessen.

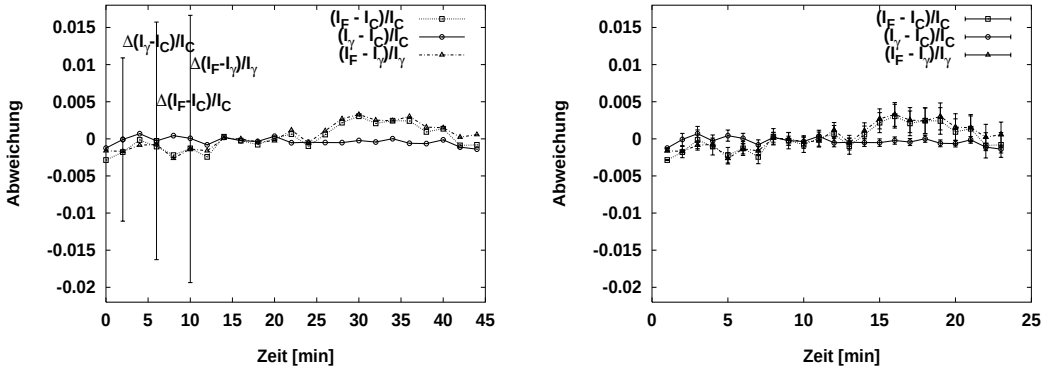


Abb.7.9: Linke Seite Relative Abweichung der drei Strommonitore bei einem Strahlstrom von $\approx 20\ \mu A$ (I_F: Förstersonde, I_γ:Photoeffekt, I_C: HF-Cavity). Die eingezeichneten Fehler stellen die absoluten Meßgenauigkeit dar: $\Delta I_C/I_C = \pm 0.5\%$, $\Delta I_\gamma/I_\gamma = \pm 1.0\%$ und $\Delta I_F = \pm 0.3\mu$A. **Rechte Seite:** Hier sind dieselben Daten zusammen mit ihren statistischen Fehlern, die sich aus der Berechnung des laufenden Mittelwertes ergaben, aufgetragen. Beide Bilder wurden auf Grund des in der Fußnote auf S.11 geschilderten Problems off-line, d.h. anhand der Information aus dem elektronischen Logbuch, erstellt.

eine automatische Überwachung dieser Komponente der Apparatur durch Kontrolle der Korrelation zwischen den Signalen der drei Sonden nahe[11]. Abb.7.9 zeigt ein Beispiel der drei Strommessungen, aus denen geschlossen werden kann, daß die Anzeige der Förstersonde während der Messung gedriftet ist.

7.4.2 Triggerdetektoren

Die Funktionsfähigkeit der Trigger-Detektoren (Szintillatoren und Čerenkov-Detektoren) wird kontinuierlich durch das Einkoppeln von Lichtsignalen in die einzelnen Detektorelemente überwacht. Die in der Registrierung dieser Signale enthaltene Information gilt es auszuwerten.
Die Lichtsignale gehen von einem zentralen Stickstofflaser aus, von dem sie über ein Lichtleit- und -verteilsystem aus Quarzglasfasern zu den einzelnen Detektoren geleitet werden [wag92]. Das Licht wird bei den Čerenkov-Detektoren direkt auf die Kathoden der Photomultiplier gegeben, während es bei den Szintillatoren auf die Szintillationselemente gegeben wird, so daß hier auch noch der Szintillator selbst und seine optische Ankopplung an den Photomultiplier überprüft wird. Alle Lichtsignale treffen gleichzeitig bei den Detektoren ein, sie werden bei der Datenaufname mit Hilfe der Koinzidenz-Logik [ri94] als *Laserereignisse* markiert
.
Zur Überwachung der Triggerdetektoren wird zum einen mit Glg.7.9 die zeitliche Konstanz der Laserrate überprüft. Außerdem wird die Stabilität der Photomultiplier durch Test der über einen ADC umgewandelten Laserereignis-Signale

[11] Dies ist jedoch z.Z. noch nicht möglich, weil die HF-Cavity wegen der langen Meßzeit über das ECS eingelesen wird, die beiden anderen Monitoren werden über die Datenerfassung nicht aber über das ECS eingelesen. Um die Überwachung auf der Basis aller Monitore on-line durchführen zu können und um die Kalibration der Strommonitore zu verbessern, wird z.Z. an einem direkten Auslesemechanismus für die Förstersonde und für den Photoeffekt-Monitor über das ECS gearbeitet.

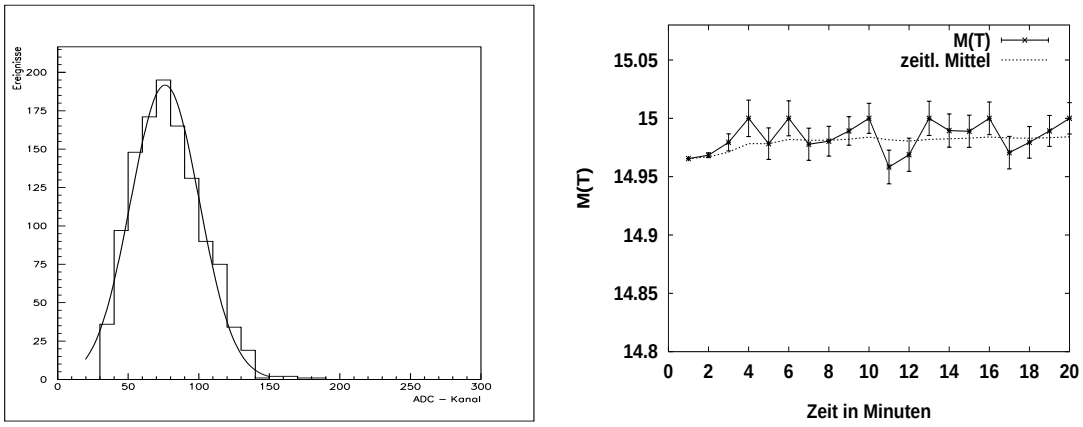


Abb. 7.10: Linke Seite Verteilung der Spannungsamplituden am Photomultiplierausgang für Laserereignisse in einem Szintillator aus der *Time-of-flight*-Ebene. An die Daten wurde eine Gauß-Verteilung angepaßt, die die Messung gut beschreibt. Am linken Rand erkennt man den Einfluß der Schwelle. **Rechte Seite:** Mittlere Multiplizität $M(T)$ als Funktion der Zeit. Die durchgezogende Linie repräsentiert den Mittelwert $\bar{M}(T_k) = 1/k \sum_{\kappa \leq k} M(T_\kappa)$ aus den Einzelmessungen $M(T_\kappa)$. Als Fehler der Einzelmessungen wurde die Wurzel aus den Varianzen $s^2(T_k) = 1/(k-1) \sum_{\kappa <= k} (M(T_\kappa) - \bar{M}(T_k))^2$ angegeben.

(*Intensität*) überprüft[12]. Um das Akkumulieren der Histogramme[13] für die einzelnen ADC-Werte zu vermeiden, erfolgt die Berechnung der mittleren Intensität "rekursiv" (Glg.7.3).
Im Idealfall sprechen alle N eingeschalteten Detektorelemente einer Szintillator-Ebene auf ein Lasersignal an. Bei ungenügender Verstärkung in einem Photomultiplier spricht der betreffende Kanal allerdings nicht an. Ein einfacher Funktionstest besteht also in der Überprüfung der Zahl der Detektoren, die bei einem Laserereignis angesprochen haben. Der zeitliche Verlauf der *mittleren Multiplizität* $M(T)$ ist im rechten Diagramm in Abb.7.10 dargestellt. Diese Messungen zeigen, daß die Multiplizität bei weniger als 0.2% der Laserereignisse unter der hier maximalen Zahl von 15 liegt.
Mit den Laserpulsen kann als weiteres die zeitliche Abstimmung der Koinzidenz-Logik überwacht werden, indem kontinuierlich die Lage des Lasersignals im Koinzidenzzeitspektrum beobachtet wird.
Die Berücksichtigung der Laserereignisse in der Bewertung des Zustands der Anlage muß auf leichte Weise variierbar sein, da bei bestimmten Experimenten kein Gebrauch vom Lasersystem gemacht wird[14] und man nicht unnötig Rechenzeit in die Erarbeitung nicht relevanter Information stecken will. Daher können die entsprechenden Programmteile ggf. mit einem Konfigurationsparameter ausgeblendet werden.

Außer den Laserereignissen werden im Prototypsystem auch die physikalischen Ereignisse zur Bewertung herangezogen. Dazu werden in periodischen Abständen

[12] Die Langzeitstabilität des Lasersystems selbst wurde in [wag92] untersucht.

[13] etwa 50 Histogramme pro Spektrometer

[14] In diese Kategorie fallen z.Z. auch solche Experimente bei denen der Čerenkov-Detektor als Veto-Detektor in die Koinzidenzbedingung eingeht. Damit zukünftig auch in diesen Fällen das Lasersystem zur Überwachung genutzt werden kann, sollte man den Aufbau der Koinzidenz-Logik in Zukunft so entsprechend (variabler) gestalten.

die Zählraten der einzelnen Detektoren ermittelt und - bei gleichbleibenden Meßbedingungen – auf ihre Konstanz überprüft. Die Verteilung der Ereignisse auf die einzelnen Detektoren wird mit Hilfe eines χ^2- bzw. Kolmogorov-Smirnov-Tests regelmäßig überprüft.

7.4.3 Driftkammern

Für die Driftkammern, in denen die Spuren der durchtretenden geladenen Teilchen durch das Messen der Driftzeiten von im Kammergas freigesetzten Ladungen zu den Signaldrähten bestimmt werden, gibt es kein mit dem Lasersystem vergleichbares Verfahren zur Überprüfung der Gesamtfunktionsfähigkeit[15]
Zur Überprüfung werden für jede der vier Drahtebenen eines Spektrometers drei Häufigkeitsverteilungen herangezogen, nämlich das *Drahtspektrum*, das angibt, wie oft die einzelnen Drähte angesprochen haben, das *Driftzeitspektrum*, das angibt, wie häufig die verschiedenen Driftzeiten vorgekommen sind, und das *Multiplizitätsspektrum*, das angibt, mit welcher Häufigkeit die betreffende Anzahl von Drähten bei den einzelnen Ereignissen "gefeuert" hat.
Außerdem wird *on-line* die Zeitinformation für jedes Ereignis in die Bildebenenkoordinaten und daraus in die Targetkoordinaten umgerechnet. Auch diese Verteilungen werden vom Prototypsystem zu Histogrammen verarbeitet und überwacht, wegen des erheblichen Rechenaufwands geschieht dies allerdings nur optional.
Neben diesen Spektren wird für jede Ebene noch periodisch deren Effizienz (s.S.86) und zusätzlich das Verhältnis der gültigen Spuren zur Gesamtzahl der Triggerereignisse berechnet und überwacht.

Auf dieser Basis wird z.B. ein Test durchgeführt, mit dem ein bei den Driftkammern sehr häufig auftretendes Problem erkannt werden soll. Es handelt sich dabei um störende Schwingungen auf einzelnen Vorverstärkerkarten, z.B. auf Grund mangelhafter Massekontakte. Die Folgen dieses Effekts für die Zeitmessung sind in Abb.7.11 dargestellt. Dort ist auf der rechten Seite ein Driftzeitspektrum gezeigt, das die an allen Drähten einer gut funktionierenden Kammer gemessenen Driftzeiten enthält. Die linke Abbildung zeigt dagegen das Spektrum einer Gruppe von Drähten, deren Signale über eine "schwingende Karte" ausgelesen wurden; hier fallen insbesondere die physikalisch nicht sinnvollen langen und kurzen Zeiten ins Auge.
Als ein robustes Maß für diese Funktionsstörung kann die Streuung $\sigma_t(i,j)$ der Driftzeiten der i-ten Karte herangezogen werden:

$$\sigma_t^2 = 1/(N-1)\sum_j (t(i,j) - \bar{t}(i))^2 \ ;$$

dabei ist die Summe über j über alle N Ereignisse zu erstrecken, die über die i-te Karte eingelesen wurden ($\bar{t}(i)$ ist der Mittelwert über diese Ereignisse). Die Abbildung 7.12 zeigt die Verteilung der $\sigma_t(i)$ aller $N_k = 25$ Karten der betrachteten Drahtebene: Die schwingende Karte fällt deutlich aus dem Rahmen.
Es ist zweckmäßig, die Funktionsfähigkeit der Detektoren kontinuierlich auf eine solche Störung in der Elektronik zu überprüfen. Um dabei die Zahl der Parameter

[15] Tests außerhalb der Experimentierzeit können mit Höhenstrahlung durchgeführt werden.

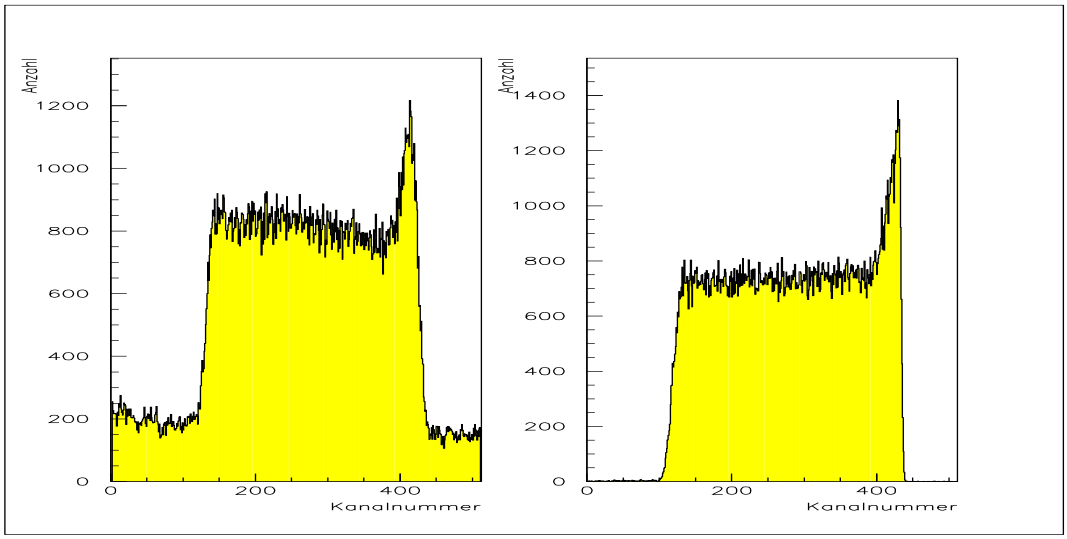


Abb.7.11: *Links:* Driftzeitspektrum einer "schwingenden" Vorverstärkerkarte. *Rechts:* Driftzeitspektrum einer Ebene. Die Driftzeiten sind in Kanälen angegeben. Ein Kanal entspricht 0.75ns.

möglichst klein zu halten, wird für jede Drahtebene kontinuierlich im Analyseprogramm nur der Mittelwert $\bar{\sigma}_t$ über die Karten einer Kammer und die Streuung

$$\sigma_t^2 = \frac{1}{N_k - 1} \sum_i^{N_k} (\sigma_t(i) - \bar{\sigma}_t)^2$$

als Funktion der Zeit verfolgt und bewertet. Beim Auftreten einer Fehlermeldung auf diesem Niveau könnte z.B. das Expertenprogramm eine differenzierte Analyse in Gang setzen.
Bei funktionierender Elektronik wurde typischerweise eine Varianz von 3 Kanälen beobachtet, während die Streuung in dem in Abb.7.12 gezeigten Beispiel mehr 10 Kanäle betrug.

Anhand dieses abschliesenden Beispiels sei nochmal das Zusammenspiel der im Abschnitt 7.3.2 vorgestellten Werkzeuge des Prototypsystems erläutert. Die Driftzeiten werden von einem Objekt, das der Driftkammer (*Eventklasse*) zugeordnet ist, zur Verfügung gestellt. Hieraus werden von der entsprechenden *Statistikklasse* $\bar{t}(i)$, $\sigma_t(i)$ und σ_t berechnet. Aus Gründen der Platzersparnis werden keine Histogramme für alle $\approx$ 280 (!) Karten aller Kammern angelegt, vielmehr werden die $\sigma_t(i)$ direkt über die Summendarstellung berechnet. Diese Information wird als Monitorfeld im Shared-Memory abgelegt, auf das andere Programme mit Hilfe eines symbolischen Namens, z.B. "a/vdc/x1/times" für die *Zeiten* der *x1-Ebene* der *vertikalen Driftkammer* des *Spektrometers A*, zugreifen können. Die Überprüfung durch das Expertenprogramm erfolgt parallel zur Analyse.

7.5 Zukünftige Enwicklung der On-line-Datenkontrolle

In Rahmen der vorliegenden Arbeit wurden die Methoden zur On-line Datenkontrolle entwickelt und als Software-Module nach dem Baukastenprinzip realisiert.

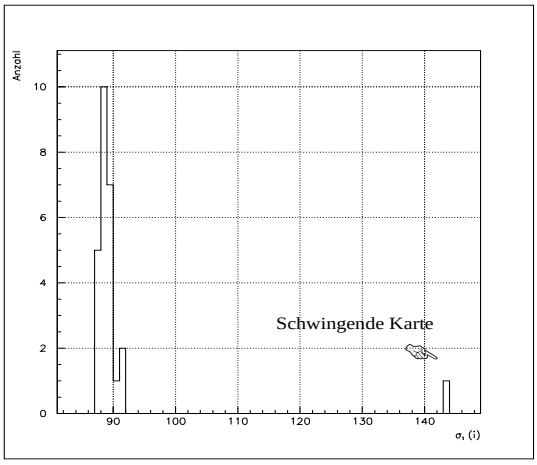


Abb.7.12: Verteilung der beobachteten $\sigma_t(i)$. Die Standardabweichung $\sigma_t(i)$ jeder Karte ist dabei in Kanälen angegeben. Die schwingende Karte hebt sich in diesem Fall deutlich von den übrigen ab.

Zur Demonstration der Funktionsfähigkeit wurde ein Prototypsystem erstellt, das im wesentlichen aus einem Analyse- und einem Überwachungsprogramm besteht. Der Prototyp ist im Hinblick auf das Ziel, *alle* Fehlersituationen unter Kontrolle zu bringen, nur ein erster Schritt. Es enthält aber bereits alle wesentlichen Elemente zur Überwachung der Standardkomponenten, von denen einige in diesem Abschnitt im Detail vorgestellt wurden. Seiner Ausweitung auf eine vollständige Überwachung sollte nichts im Wege stehen.

Einige Teile der Software, auf denen das Prototypsystem aufsetzt, sind schon des längeren Bestandteil des On-line-Systems. Hierzu gehören Module wie z.B. die Histogrammbibliothek, die Tasty-Bibliothek, oder der Algorithmus - die Trace-Machine - zum Zurückrechnen der Bildebenen- auf die Targetkoordinaten. Diese Module wurden bereits zur Auswertung verschiedener Experimente[di95, ko95, kus95, ku95, sch95] benutzt. Sie wurden dabei sowohl auf den UNIX-Workstations des Instituts als auch auf mit LINUX betriebenen PCs eingesetzt.

Das Prototypsystem stellt insgesamt ein Expertensystem zur Bewertung der Funktionsfähigkeit der Drei-Spektrometer-Anlage dar. Es wird über Tabellen gesteuert und kann damit ohne zusätzlichen Programmieraufwand an die jeweils spezifische Situation angepaßt werden. Der Einsatz beim Experimentieren hat sich im Prinzip bewährt, gleichzeitig aber auch noch bestehende Mängel offengelegt. So erwies sich die Handhabung des Systems für den ungeübten Benutzer als schwierig. Die Situation würde sehr viel übersichtlicher und damit leichter handhabbar, wenn bereits die Durchführung der Experimente selbst über eine Datei gesteuert würde, auf die dann auch das Überwachungsprogramm zugreifen könnte, wodurch es zuverlässig mit den relevanten Parametern versorgt würde. Eine datenbankgesteuerte Experimentdurchführung ist derzeit in Vorbereitung [gei94].

Bislang standen die methodischen Entwicklungen und die Erstellung der erforderlichen Software-Bausteine im Vordergrund. Dabei wurden zwar praktische Gesichtspunkte wie die Wartbarkeit und Portierbarkeit der Programme von vorneherein berücksichtigt, es wurden aber keine größeren Anstrengungen zur Optimierung der Analyse-Software unternommen (lediglich der sehr zeitaufwendige Rückrechenalgorithmus wurde hinsichtlich seines Rechenzeitbedarfs optimiert (Anhang A)). In dieser Beziehung sind Verbesserungen erforderlich, damit das Analyseprogramm auch bei hohen Datenraten die anfallende Information parallel zur Messung verarbeiten kann. Aus dem gleichen Grund ist daran zu denken,

die im On-line-Bereich eingesetzten Rechner (DECstation 5000) durch schnellere Maschinen zu ersetzen (siehe auch Tab.4.3).

Kapitel 8

Zusammenfassung

Zur Durchführung von Zwei- und Dreifach-Koinzidenzexperimenten wurde am 855-MeV-Elektronenbeschleuniger MAMI im Rahmen der A1-Kollaboration eine Experimentiereinrichtung aus drei Magnetspektrometern - die Drei-Spektrometer-Anlage - aufgebaut.
Um eine Anlage dieser Komplexität, die u.a. durch die große Anzahl der zu ihrem Betrieb notwendigen Komponenten hervorgerufen wird, effizient einsetzen zu können, wurde ein System zur rechnergestützten Steuerung und Überwachung der Experimente konzipiert und aufgebaut. Das hierfür im Rahmen der vorliegenden Dissertation entwickelte Gesamtkonzept sieht auf einer ersten Stufe - der apparativen Überwachung - eine laufende Kontrolle aller technischen Parameter der Apparatur vor. Diese Aufgabe ist eng verknüpft mit den Aufgaben zur Steuerung der Anlage - beide Aufgaben wurden deshalb im Rahmen eines umfassenden Systems, des Experiment-Kontroll-Systems ECS, gelöst.

Bedingt durch den Aufbau der Anlage wurde das Kontroll-System als dezentrales System konzipiert, das in eine Vielzahl von Prozessen aufgespalten ist, die über ein lokales Netzwerk von Rechnern verteilt sind. Daher waren umfangreiche Synchronisations- und Kommunikationsprobleme zu lösen. Dafür wurde im Rahmen der vorliegenden Dissertation ein Softwarepaket names MUPIX - MUltiProcessor Interprocess Communication System - erstellt, das eine neue und vor allem portable Lösung dieses Problems darstellt, die nicht nur für den Einsatz an der Drei-Spektrometer-Anlage geeignet ist.
Bei MUPIX werden Sender und Empfänger als Objekte betrachtet, die auf verschiedene Prozesse im Netzwerk verteilt sind und untereinander Nachrichten austauschen - es setzt damit das objekt-orientierte Programmierkonzept über die engen Grenzen des Programmkontextes hinaus fort. Die Software basiert auf dem UDP/IP Protokoll und wurde bereits auf den Betriebssystemen UNIX, VMS und OS-9 eingesetzt.
Die Adressierung der Objekte erfolgt mit Hilfe symbolischer Namen. Die Namensauflösung wird durch einen Serverprozeß geleistet, der auch für andere Protokollfamilien, z.B. UDP/IP und TCP/IP, eingesetzt werden kann. MUPIX erlaubt auch die Kommunikation zwischen Objekten innerhalb eines Prozesses.

Basierend auf der Kommunikationssoftware MUPIX wurden im Rahmen der vorliegenden Arbeit ein Konzept für den Aufbau des Experimentkontrollsystems entwickelt und die Software-Werkzeuge zur appprativen Überwachung erstellt, die den aktuellen Zustand der Apparatur erfassen, protokollieren und dem Experimentator verfügbar machen. Darüber hinaus sorgen sie dafür, daß dem Experimentator aufgetretene Fehlersituationen mitgeteilt werden.

Um eine vollständig computer-gestützte Experimentführung zu ermöglichen, müssen alle Meßvorgänge automatisiert ablaufen. Dies gilt insbesondere für die Magnetfeldeinstellung auf Grund der Feldmessung mit dem Kernresonanzverfahren. Des-

halb wurde als weiterer Teil der vorliegenden Dissertation das System zur Magnetfeldmessung aufgebaut und kalibriert. Da einer der Spektrometerdipolmagnete als inhomogener "Clam-Shell"-Magnet konzipiert ist, mußte der Feldgradient am Ort der Kernresonanzprobe durch einen entgegengesetzten Gradienten kompensiert werden. Dabei war es erforderlich, die Richtung des Kompensationsgradienten variabel zu gestalten. Dieses Problem wurde durch die Erzeugung des Kompensationsfeldes durch zwei gegeneinander verschert aufgebaute Quadrupole gelöst. Die mit dieser Anordnung erzielte Genauigkeit ist besser als $5*10^{-6}$ und übertrifft damit die Anforderungen.

Das Gesamtkonzept zur Überwachung der Experimente an der Drei-Spektrometer-Anlage sieht vor, Informationen über den Zustand der Apparatur aus den erfaßten Ereignissen der jeweils untersuchten physikalischen Reaktion selbst zu gewinnen.
In Rahmen der vorliegenden Arbeit wurden die Methoden zur On-line Datenkontrolle entwickelt und als Software-Module nach dem Baukastenprinzip realisiert. Zur Demonstration der Funktionsfähigkeit wurde ein Prototypsystem erstellt, das im wesentlichen aus einem Analyse- und einem Überwachungsprogramm besteht. Die Wirksamkeit der im Prototyp enthaltenen Testfunktionen wurde an verschiedenen Beispielen aus Messungen und aus Datensimulationen überprüft.
Damit ist ein erster Schritt zu einem Expertensystem zur vollständigen Überwachung und ggf. zur Fehleranalyse der komplexen Drei-Spektrometer-Anlage vollzogen.

Die im Rahmen der vorliegenden Dissertation erstellte Software zur Experimentsteuerung und Überwachung der Drei-Spektrometer-Anlage wurde mit objektorientierten Methoden entwickelt und besteht insgesamt aus etwa 80000 Zeilen C- und C++-Programmtext. Sie stellt einen integralen Bestandteil der Drei-Spektrometer-Anlage dar. Ihre Funktionsfähigkeit hat sie durch den Einsatz bei allen bisherigen Experimenten unter Beweis gestellt.

Danksagung

Herrn Prof.Dr.K.Merle danke ich für die Themenstellung, Betreuung und die Förderung der Arbeit, aber auch für den mir überlassenen Spielraum.
Herrn Prof.Dr.J.Friedrich danke ich daneben für seine wertvollen Hinweise bei der Niederschrift dieser Arbeit.

Den Mitarbeitern der Werkstätten danke ich für die schnelle und zuverlässige Ausführung der angefallenen Arbeiten.

Allen Mitgliedern der Kollaboration A1 danke ich für ihren Einsatz beim gemeinsamen Aufbau der Drei-Spektrometer-Anlage .

Insbesondere möchte ich mich bei Andreas, Christoph, Eddy, Jörg-Michael, Klaus-Werner, Manfred, Michael, Stefan, sowie Ingvar Blomqvist und Rainer Geiges bedanken.
Volker Kunde danke ich außerdem für die unkomplizierte Zusammenarbeit und für seine Aufgeschlossenheit gegenüber neuen Ideen und Konzepten.
Stefan Schardt danke ich für die vielen hilfreichen und interessanten Diskussionen, die auch manch längere Nachtschicht bereichert haben.

Schließlich möchte ich meinen Eltern für ihre vielfältige Unterstützung besonders danken.

Literaturverzeichnis

[a1] **A1-Kollaboration:** K.I. Blomqvist[a], W.U. Boeglin[a], R. Böhm[a], O. Denhard[a], M. Distler[a], R. Edelhoff[a], R. Florizone[k], J. Friedrich[a], D. Fritschi[c], R. Geiges[a], R. Gilman[d], M. Jones[d], J. Jourdan[c], M. Kahrau[a], M. Korn[a], H. Kramer[a], K.W. Krygier[a], V. Kunde[a], M. Kuss[e], J. Lác[d], A. Liesenfeld[a], K. Merle[a], C.L. Morris[f], R. Neuhausen[a], E.A.J.M. Offermann[a], Th. Pospischil[a], M. Potokar[g], C. Rangacharyulu[h], R.D. Ransome[d], A. Rokavec[g], A. Richter[e], A.W. Richter[a], B.G. Ritchie[i], S. Robinson[c], G. Rosner[a], P. Rutt[d], P. Sauer[a], A. Sarty[k], S. Schardt[a], J.P. Schiffer[j], G. Schrieder[e], I. Sick[c], P. Trüb[c], Th. Veit[a], A. Wagner[a], Th. Walcher[a], S. Wolf[a]

[a]Institut für Kernphysik, Universität Mainz; [b]Department of Physics, University of New Hampshire, Durham, USA; [c]Institut für Physik, Universität Basel; [d]Physics Department, Rutgers University, Piscataway, USA; [e]Institut für Kernphysik, TH Darmstadt; [f]LAMPF, LANL, Los Alamos, USA; [g] University of Ljubljana, Slovenia; [h] University of Saskatchewan, Saskatoon, Canada; [i] Physics Department, Arizona State University, Tempe, USA; [j]Physics Department, ANL, Argonne, USA; [k] MIT, Cambridge, USA;

[ad92] T.Adye et al., *The Delphi Experiment Control System,* New Computing Techniques in Physics Research 2, ed. D.Perrex-Gallix (London 1992)

[aut90] K.Authenrieth, *Technik verteilter Betriebssysteme,* Hüthig-Verlag (Heidelberg 1990)

[A1-90] A1-proposal A1/1-90: An Investigation of the Electroproduction of Multihadron Final States in Quasielastic, Dip and Δ Resonance Region; Mainz, 1990

[A1-91] Blomqvist et al., *Jahresbericht 1990/91,* S.55 ff., KPH

[A1-93] Blomqvist et al., *Jahresbericht 1992/93,*, S.61 ff., KPH

[A1-94] Blomqvist et al, *High-Momentum Components in the* $1p$ *orbitals of* ^{16}O*,* angenommen zur Veröffentlichung in Phys.Let.B 1994

[be94] U.Behrens et al., *ZEX – An Expert System for ZEUS,* New Computing Techniques in Physics Research 3, ed. K.-H. Becks, World Scientific (London 1994)

[blo91] I.Blomqvist - *private Mitteilung*

[bof92] S.Boffi,C.Giusti und F.D.Pacati, *Nuclear Response in Electromagnetic Interactions with Complex Nuclei,* Physics Reports, **226**, 1993

[boo91] G.Booch, *Object-oriented design and applications,* Benjamin/Cummings (Redwood City 1991)

[bru91] R.Brun, *HBOOK Long Writeup,* CERN Bibliothek, Y250, 1991

[cab88] L.Cabrera et al., *User-Process Communication Perforamnce in Networks of Computers,* IEEE Trans. on Software Engineering, Vol. 14, No. 1, S.38-53 (1988)

[cap95] T.Caprano – private Mitteilung

[coa90] P.Coad and E.Yourdon, *Object-Oriented Analysis,* Prentice-Hall (London 1990)

[com88] D.Comer, *Internetworking with TCP/IP,* Prentice-Hall (London 1988)

[cor91] J.R. Corbin, *The Art of Distributed Applications,* Springer (New York 1991)

[den60] Denisov, Instr. Exptl. Techn., 1, 89, 1960

[di95] M.Distler, *Dissertation in Vorbereitung,* (KPH 1995)

[ead71] Eadie et al., *Statistical Methods in Exp. Physics,* North-Holland (1971)

[ed72] F.Eder, *Moderne Meßmethoden der Physik*, Band 3, S.729 ff., Berlin (1972)

[en64] H.A. Enge, Nucl. Instr. and Methods **28** (1964) 119

[eut94] Euteneuer et. al, *Jahresbericht 1992/93,* S.16 ff.

[gei94] R.Geiges - private Mitteilung

[ger90] M.Gergeleit et al., *SphinX: Ein Konzept zur multiplexfähigen Prozeßkommunikation über Rechnergrenzen,* German National Research Center for Computer Science (1990)

[ha93] A.Hake, *Software-Werkzeuge zur Steuerung an MAMI-Experimenten,* Diplomarbeit (KPH 1993)

[här90] H.Härtig et al., *Architecture of the BirliX Operating System,* German National Research Center for Computer Science (1990)

[hei92] A.Heil - private Mitteilung

[her90] H.Herminghaus et al., Proc. LINAC Conf. 1990, Albuquerque, New Mexico, USA

[hert92] K.Herter, *Entwicklung und Erprobung eines Verfahrens zur Messung der absoluten Energie des Elektronenstrahls von MAMI,* Diplomarbeit (KPH 1992)

[hof56] R.Hofstadter, *Rev. Mod. Phys.* **28** 214-54 (1956)

[hon88] Klaus Honscheid, *Das Trigger- und Datenerfassungssystem des SAPHIR-Detektors,* Diplomarbeit IR-88-61, (Bonn 1988)

[hop79] J.E.Hopcroft und J.D.Ullman, *Introduction to automata theory, languages and computation,* Addison-Wesley, (Reading 1979)

[ka87] P.Karn und C.Partridge, *Improving Round-Trip Estimates in Reliable Transport Protocolls,* Com. Comm. Rev., Vol. 17, No. 5 (1987)

[ker88] Kernighan & Ritchie, *The C-Programming Language,* 2.Auflage, Addison-Wesley (Reading 1998)

[knu68] D.E.Knuth, *The Art of Computer Programming,* Addison-Wesley (1968)

[kos92] B.Kosko, *Neural Networks and Fuzzy Systems,* Prentice Hall (London 1992)

[ko95] M.Korn, *Entwicklung des Bahnrückverfolgungsverfahrens für die Drei-Spektrometer-Anlage und Bestimmung der Abbildungseigenschaften der Spektrometer A und B mit elastischer Elektronenstreuung,* Dissertation (KPH 1995)

[kraM90] H.Kramer und K.Merle, *Voruntersuchungen zur (Neu-)Implementierung eines universellen rechnerübergreifenden Kommunikationssystems,* KPH - Jahresbericht 1988/89

[kra92a] H.Kramer, *MUPIX Version 1.5 – 1.7.1,* Interner Report (KPH 1992)

[kra92b] H.Kramer, *The OPCON Programming Environment,* Interner Report (KPH 1992)

[kra92c] H.Kramer, *TMA - Using the TMA - vertex reconstruction software,* Interner Report (KPH 1992)

[kra92d] H. Kramer, *The Tasty Users Guide - α-Version*, Interner Report KPH (1992)

[kra93a] H.Kramer, *The Mupix Users Guide 2.0*, Interner Report (KPH 1993)

[kra93b] H.Kramer, *MUPIX und X11-Events,* Interner Report (KPH 1993)

[kra93c] H. Kramer und C. Martin. *Tvm744++ – Treibersoftware*, Interner Report (KPH 1993)

[kre89] H.Kreidel - private Mitteilung.

[kry95] K.W.Krygier, *Dissertation in Vorbereitung*

[ku89] V.Kunde, K.W.Krygier und K.Merle, *VMEbus im Institut für Kernphysik,* KPH - Jahresbericht 1988/89

[ku95] V.Kunde - Dissertation in Vorbereitung

[kun92] M.Kunze, *Experience with mixed Language Programming,* New Computing Techniques in Physics Research 2, ed. D.Perrex-Gallix, World Scientific (1992)

[kus95] M.Kuss, *Dissertation in Vorbereitung,* TH Darmstadt

[lef89] S.Leffler, M.McKusick, M.Karels, J.Quaterman, *The Design and Implementation of the 4.3BSD Unix Operating System,* Addison-Wesley (Reading 1989)

[li95] A.Liesenfeld, *Dissertation in Vorbereitung*

[ma93] C.Martin, *Die A1-Targetsystemsoftware und Integration ins Gesamtsystem für die Kontrolle von (e,e'p) - Messungen,* Diplomarbeit (KPH 1993)

[mer84] K.Merle und K.W.Krygier, *Experiment-Datenerfassung unter UNIX,* KPH - Jahresbericht 1982/83

[met89] *PT2025 NMR High Precision Teslameter,* Manual 2025/89/11, Metrolab Instruments S.A., Genf

[mil90] J.S.Milton, *Introduction to Probability and Statistics,* McGraw-Hill (New York 1990)

[mit92] D. Mittwich, *Entwicklung und Erprobung eines Hochfrequenzmonitors zur Messung der Intensität und Lage eines 855 MeV Elektronenstrahls,* Diplomarbeit (KPH 1992)

[neu64] R.Neuhausen und G.Fricke, *Messung inhomogener Magnetfelder mit der Kernresonanzmethode,* Zeitschrift für angewandte Physik, 17. Band, 6.Heft, S. 446-448, Springer (Heidelberg 1964)

[neu94] R.Neuhausen, *The three-spectrometer setup for $(e, e'x)$ coincidence experiments at the Mainz microtron MAMI,* Proc. of the 4th Int.Conf. on Advanced Technolgy and Particle Physics, Como 1994, to be published in Nucl.Phys. B

[nel81] B.J.Nelson, *Remote Procedure Call,* Ph.D. thesis, Carnegie-Mellon University, 1981, CMU-CS-91-119

[off92] E.A.J.M. Offermann *Vertex reconstruction* Interner Report (KPH 1992)

[off93] E.A.J.M. Offermann, *ESPACE — Event Scanning Program for A1 Collaboration Experiments*, Interner Report (KPH 1993)

[osf91] Open Software Foundation, *Guide to OSF/1: A Technical Synopsis*, O'Reilly & Associates

[pr88] W.H.Press et al., *Numerical Recipes in C*, Cambridge Universty Press (1988)

[ri94] A.Richter, *Trennung des longitudinalen, transversalen und longitudinal- transversal interferierenden Anteils des Wirkungsquerschnitts der Reaktion H(e,e'pi+) in der Nähe der Pionenschwelle*, Dissertation (KPH 1994)

[san89] L.L.Santoline et al., *Multiprocessor shared-memory information exchange*, IEEE Trans. on Nuclear Science, Vol 36 (1989)

[sau95] P.Sauer, *Dissertation in Vorbereitung*,

[sch87] A.T.Schreiner, *UNIX-Sprechstunde*, Hanser-Verlag (München 1987)

[sch95] Stefan Schardt, *Aufbau und Erprobung der Drei–Spektrometer–Anordnung für Koinzidenzexperimente mit Elektronen am 855 MeV–Elektronenbeschleuniger MAMI* , Dissertation (KPH 1995)

[schr88] W.Schröder, *PEACE: The distributed SUPRENUM operating system*, Parallel Computing 7, 325-333 (1988)

[schri93] C.Schrimpf, *Entwicklung und Test von Analysemethoden im Rahmen der MECDAS-Datenerfassung*, Diplomarbeit (KPH 1993)

[ste90] W.R. Stevens, *UNIX network programming*, Prentice Hall (London 1990)

[ste92] W.Stevens, *Advanced Programming in the UNIX environment*, Addison-Wesley (Reading 1992)

[stef93] S.Steffens, *Entwicklung und Erprobung einer "X-Windows"-Benutzeroberfläche zur rechnergesteuerten und kontrollierten Durchführung von Koinzidenzexperimenten an MAMI*, Diplomarbeit (KPH 1993)

[str86] B.Stroustrup, *The C++-Programming Language*, Addison-Wesley (1986)

[tan87] A.Tannenbaum, *Operating Systems*, Prentice-Hall (London 1987)

[tat65] J.R.Tatarczuk und B.Ziegler, *Eine Kernresonanzanordnung zur Messung und Stabilisierung von Magnetfeldern zwischen 0.1 G und 20 kG*, Nucl. Instr. and Methods **41** (1966) 45

[vi92] L.Vinot et al. *Safety of High Energy Physics Experiments by an Expert System. Temporal and Statistical Aspects*, New Computing Techniques in Physics Research 2, ed. D.Perrex-Gallix, World Scientific (London 1992)

[wag92] A.Wagner, *Aufbau eines Laser-Monitor-Systems für die Spektrometer-Detektoren*, Diplomarbeit (KPH 1992)

[wat87] W.A.Watson und M.J.LeVine, *Distributed data acquisition for BNL800II: Software*, IEEE Transactions on Nuclear Science, Vol 34 (1987)

[wi83] N.Wirth, *Algorithmen und Datenstrukturen*, Teubner Verlag (Stuttgart 1983)

KPH: Institut für Kernphysik, Mainz

Anhang A

TraceMaschine - ein schneller und flexibler Algorithmus zur Bahnrückrechnung

Bei der On-line Analyse der Streudaten aus den Experimenten mit den Spektrometern muß die zunächst als Koordinaten in der Bildebene vorliegende Information umgerechnet werden auf Koordinaten am Ort des Targets [ko95]. Die Abbildung ergibt sich im Prinzip durch Lösung der Bewegungsgleichung von geladenen Teilchen im Spektrometerfeld. Stattdessen macht man im Sinne einer Taylorentwicklung einen Ansatz der Form

$$q_{tg} = \sum_{j,k,l,m=0}^{n_x,n_y,n_\theta,n_\phi} c^{(q)}_{jklm} x^j_{fp} y^k_{fp} \theta^l_{fp} \phi^m_{fp} \qquad \text{(A.1)}$$

wobei q die Targetkoordinaten durchläuft [ko95], also $\{\delta, y, \theta, \phi\}$ - für die relative Impulsablage, die transversale Abweichung von der Referenzbahn, den Winkel zur Referenzbahn in der dispersiven Richtung und den nicht-dispersiven Streuwinkel am Target (siehe Abb.A.1).

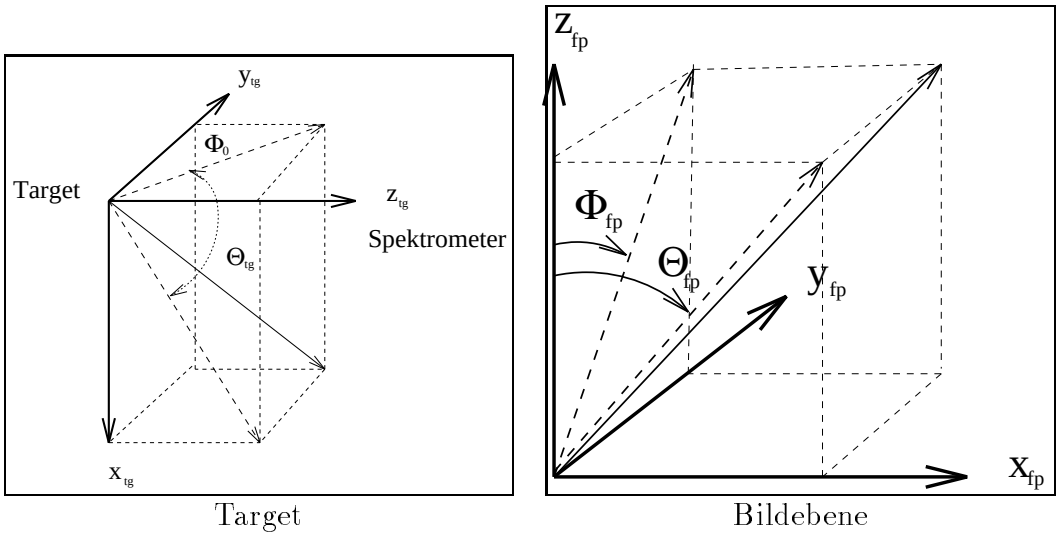


Abb.A.1: Definition von Target- und Bildebenenkoordinatensystem.

Diese Berechnung muß für jedes Ereignis durchgeführt werden. Damit das Analyseprogramm so schnell wie möglich die Datenpuffer an den Eventbuilder zurückgeben kann, ist die Berechnung so effizient wie möglich zu gestalten. Neben der Effizienz muß der Algorithmus so flexibel sein, daß auch ohne Neukompilation von Programmen die Werte der Transferkoeffizienten $c^{(q)}_{jklm}$ oder die Ordnung in der

Taylorentwicklung geändert werden können. Da die vorhandenen Implementierungen in beiderlei Hinsicht nicht befriedigend waren, wurde ein neuer einfacher und dennoch effizienter Algorithmus konzipiert und implementiert.

Grundidee des Algorithmus

Gemäß Glg.A.1 ist die häufigste Operation, die bei der Berechnung von $\vec{q}_{tg}$ angewandt wird, die Multiplikation von möglicherweise doppeltgenauen reellen Zahlen (float). Da die (floating point) Multiplikation eine der rechenintensivsten Operationen (siehe Tabelle A.1) ist, hängt die Geschwindigkeit des Rückrechnungsalgorithmus wesentlich von der Anzahl der auszuführenden Multiplikationen ab. Durch Ausnutzung von im Verlauf der Rechnung bereits gewonnenen Ergebnissen kann diese Zahl verringert werden.

- Zunächst ordnet man alle vorhandenen Transferkoeffizienten in einer Tabelle (Transfermatrix) an:

Targetkoordinate	$x_{fp}^0 y_{fp}^0 \theta_{fp}^0 \phi_{fp}^0$	$x_{fp}^1 y_{fp}^0 \theta_{fp}^0 \phi_{fp}^0$	...	$x_{fp}^{n_x} y_{fp}^{n_y} \theta_{fp}^{n_\theta} \phi_{fp}^{n_\phi}$
δ_{tg}	$c^{(\delta)}_{0,0,0,0}$	$c^{(\delta)}_{1,0,0,0}$	...	$c^{(\delta)}_{n_x,n_y,n_\theta,n_\phi}$
y_{tg}	$c^{(y)}_{0,0,0,0}$	$c^{(y)}_{1,0,0,0}$	...	$c^{(y)}_{n_x,n_y,n_\theta,n_\phi}$
θ_{tg}	$c^{(\theta)}_{0,0,0,0}$	$c^{(\theta)}_{1,0,0,0}$	...	$c^{(\theta)}_{n_x,n_y,n_\theta,n_\phi}$
ϕ_{tg}	$c^{(\phi)}_{0,0,0,0}$	$c^{(\phi)}_{1,0,0,0}$	...	$c^{(\phi)}_{n_x,n_y,n_\theta,n_\phi}$

  In einer Spalte stehen die Koeffizienten zu gleichen Potenzen (j,k,l,m) in $x_{fp}^j y_{fp}^k \theta_{fp}^l \phi_{fp}^m$. In den Zeilen sind die Terme einer Targetkoordinate angeordnet. Läuft man die Tabelle in vertikaler Richtung durch so muß man jeden Term nur einmal auswerten. Dadurch kann man gegenüber dem üblichen Verfahren bei dem reihenweise vorgegangen wird, die Anzahl der Multiplikationen um bis zu 75 % verringern.

- Da die einzelnen Spalten Produkte aus Potenzen der Bildebenenkoordinaten von der Form $x_{fp}^j y_{fp}^k \theta_{fp}^l \phi_{fp}^m$ sind, ist es sinnvoll alle benötigten Einzelpotenzen $x_{fp}^{(j)k}$ vorher zu berechnen, und in einer Tabelle abzulegen. Bei der späteren Berechnung werden die Terme aus der Tabelle abgerufen (lookup - Verfahren).

- Beim Aufbau der lookup-table kann man Zeit durch rekursive Berechnung der Einträge für die Potenzen einer Koordinate sparen: $x_{fp}^{(j)k+1} = x_{fp}^{(j)k} x_{fp}^{(j)}$. Man berechnet deshalb die Einträge günstigerweise in einer Schleife und vermeidet unnötiges Potenzieren.

Tab.A.1: Rechenzeiten, die auf den beiden Testrechnern und einem PC für typische Operationen benötigt werden. Die Variablen sind in (C-Notation) vom Typ `double a,b,c; long i,j,k, *ip`.

Operation	DECstation in μs	486DX/33 in μs
`i = j + k`	0.054	0.43
`a = b + c`	0.289	0.99
`i = j * k`	0.484	0.76
`a = b * c`	0.374	1.04
`*ip`	0.012	0.04

Programm	Brutto in s	Tara in s	Netto in s	Zeit pro Punkt in ms
`trad`	149.3	31.5	117.8	1.178
`tMachineF`	80.8	28.1	51.9	0.519
`tMachine`	71.2	20.9	50.3	0.503

Tabelle A.2: Um die Leistung des neue Algorithmus zu beurteilen, wurde das bisher zur Rückrechnung verwendete Programm (`trad`) mit auf dem neuen Algorithmus beruhenden Programmen `tMachine` und `tMachineF` zur Bahnrückverfolgung auf einer DECstation 5000/200 verglichen. Das Programm `tMachine` ruft die `TraceMachine` aus C heraus auf, `tMachineF` aus Fortran. Zum Test wurden 100000 zufällige Bildebenenkoordinaten eingelesen und zurückgerechnet (Bruttozeiten). Um die Leerlast (Tara) festzustellen, wurden die Daten erneut durch die Programme eingelesen, ohne jedoch die Rückrechnung auszuführen. Der Unterschied in der Verarbeitungszeit zwischen `tMachineF.f` und `tMachine.c` kann damit erklärt werden, daß bei der Fortran - Anbindung noch eine zusätzliche Funktion zur Schnittstellenumsetzung notwendig ist.
Auch bei den Tarazeiten besteht eine Differenz zwischen den beiden neuen Programmen. Dieser ist auf die unterschiedliche I/O-Anbindung zurückzuführen. Fortran-I/O ist unter der UNIX-Variante Ultrix um 40 Prozent langsamer. Bei dieser Anwendung verliert man dadurch bereits 10% der gesamten Zeit.

Implementation

Der erste Schritt besteht darin, die Transferkoeffizienten nach Potenzen zu orden, also eine Tranfermatrix aufzubauen. Dieser Schritt wurde von den eigentlichen Berechnungsroutinen getrennt und in einem speziellen Programm implementiert. Die Ausgabe dieses Programms wird als Datei abgelegt, die von den Rekonstruktionsroutinen bei der Initialisierung eingelesen wird. Das Abtrennen dieses Schrittes hat mehrere Vorteile:

- Das Sortieren für ein gegebens Polynom wird nur einmal gemacht.
- Der zum Sortieren benötigte Code muß nicht in die Analyse-Programme eingebunden werden.
- Die Rekonstruktionsroutinen können davon ausgehen, daß die Parameterdatei syntaktisch in Ordnung ist.

Der Sortieralgorithmus basiert auf einem Baumalgorithmus ("threaded trees"). Die dazu notwendigen Routinen standen schon aus einem anderen Projekt zur Verfügung, so daß die Implementierung sehr schnell durchgeführt werden konnte (siehe Seite 57). Die eigentlichen Rekonstruktionsroutinen wurden objektorientiert implementiert. Die Funktion zur Rückrechnung wird von einer Klasse (`TraceMaschine`) zur Verfügung gestellt. Deshalb ist es leicht möglich, mehrere solcher Rekonstruktionsmaschinen innerhalb eines Programms (für jedes Spektrometer) zu halten, die Organistion wird vom Compiler geleistet. Nach dem Obigen wickelt ein Objekt der `TraceMachine` im wesentlichen drei Aufgaben (Methoden) ab

- Konstruktion der Maschine
  - Einlesen der Parameterdatei

  - Reservierung des notwendigen Speichers
  - Verkettung der Daten
- Rückrechnung
- Vernichtung der Maschine, Freigabe des Speichers

Die Schittstellen zur `TraceMaschine` sind in der Dokumentation [kra92c] beschrieben. Die `TraceMachine` kann sowohl in C- als auch in Fortran-Programmen verwendet werden.

Performance

Der hier vorgestellte Algorithmus und dessen Implementierung ist den ursprünglichen Routinen überlegen. Der Zeitgewinn ist beträchtlich, gemessen an den typischen Operationszeiten (siehe Tab.A.1) auf dem Testrechner, was insbesondere für den Einsatz im On-line Bereich von Bedeutung ist. Ferner zeigte sich, daß man bei der Verwendung von Fortran für I/O-Aufgaben, wie sie sich notwendigerweise immer bei der Analyse großer Datenmengen ergeben, unter UNIX u.U. erhebliche Zeitnachteile gegenüber C in Kauf nehmen muß. Deshalb müssen die herstellerspezifischen Fortran-I/O Bibliotheken jeweils auf ihre Leistungsfähigkeit überprüft werden.

Anhang B

MUPIX

B.1 Das Nameserver-Protokoll im XDR-Format

Mit dem Nameserver werden zwei Typen von Nachrichten ausgetauscht: Anfragen an den Nameserver (`d_query`) und Antworten vom Nameserver (`d_answ`).

```
struct d_query {
          int             q_number;
          string          q_domain_name<>;
          u_short         q_family;
          short           q_pad;
          struct fuic     q_uic;
          queryU          q_info;
};

struct d_answ {
          int             q_number;
          enum q_error    q_err;
          answerU         q_info;
};
```

Verschiedenen Anfragen werden durch eine Flagge (`q_type`) in der Abfragestruktur angezeigt; sie werden in einer *diskriminierenden Union* (`queryU`) zusammengefaßt.

```
enum q_type {
         DST_REMOVE,
         DST_RESOLV,
         DST_ADD,
         DST_DEL,
         DST_GET_PORT
};

union queryU switch (enum q_type q_type) {
          case DST_REMOVE:
            struct aPhyAddr    q_phy;
          case  DST_RESOLV:
            struct aLogAddr    q_log;
          case DST_ADD:
            struct aBindQuery  q_bind;
          case DST_DEL:
            struct aBindQuery  q_del;
          case DST_GET_PORT:
            void;
} ;
```

Beim Anmelden (`aBindQuery`) werden die symbolischen Namen und die dazugehörige Protokolladresse übermittelt:

```
struct aBindQuery {
          struct aPhyAddr    *phy;
```

```
          struct aAlias      log<>;
} ;

struct aLogAddr {
          string name<>;
};

struct aAlias {
          u_short    l_unit ;
          string     l_alias<>;
};

struct aPhyAddr {
          u_long    sms_addr;   /* node */
          u_short   sms_port;   /* port */
          u_short   sms_unit;   /* unit */
          u_short   sms_prot;
          u_short   sms_key ;
          int       sms_pid ;
};
```

Antworten des Nameservers `d_answ` (s.o.) enthalten neben dem Ergebnis (`answerU`) noch eine Fehlernummer (`q_error`) (ggf. `DST_RT_OK` für keinen Fehler):

```
union answerU switch (q_type q_type) {
          case DST_RESOLV:
            struct aPhyAddr    q_phy;
          case DST_ADD:
            struct aPhyAddr    q_add;
          case DST_REMOVE:
            void;
          case DST_DEL:
            void ;
          case DST_GET_PORT:
            struct aPhyAddr    q_prt;
} ;

enum q_error {
          DST_RT_OK = 0,    /*   no error    */
          DST_RT_SF = 1,    /*   server failure          */
          DST_RT_FE = 2,    /*   format error            */
          DST_RT_NN = 3,    /*   name doesn not exist    */
          DST_RT_NE = 4     /*   name already exist    */
};
```

B.2 Die Basisdatentypen

```
#define  aBasicData_CK     0
#define  aPhysicalData_CK  1
#define  aChannel_CK       2
#define  aChannelArray_CK  3
#define  aPhysicalArray_CK 4
#define  aSwitch_CK        7
#define  aStatusReport_CK  8
#define  aIntChannel_CK     12
#define  aIntChannelSet_CK  13
#define  aIntData_CK  14
#define  aIntArray_CK  15
#define  aBufData_CK  16


struct aBasicData {
  int class_type;
};

struct aPhysicalData {
        int class_type;
        int dim;
        float val;
};

struct aChannel {
        int class_type;
        int number;
        float val;
};

struct aChannelArray {
  int class_type;
  int dim;
  u_int len;
  struct aChannel *val;
};

struct aIntData {
        int class_type;
        int dim;
        long val;
};
struct aPhysicalArray {
        int class_type;
        int dim;
        u_int len;
        float *val;
};
```

```
struct aIntArray {
        int class_type;
        int dim;
        u_int len;
        long *val;
};

struct aSwitch {
        int class_type;
        int on_off;
};

struct aIntChannel {
        int class_type;
        int number;
        int val;
};

struct aIntChannelSet {
        int class_type;
        int dim;
        u_int len;
        struct aIntChannel *val;
};

struct  aBufData {
  int   class_type;
  u_int len;
  char *buf;
};


struct aStatusReport {
        int class_type;
        char *submitter;
        int barcode;
        int date;
        int field_type;
        int report_type;
        char *description;
};
```

B.3 Anwendungsbeispiel

```
int mirror_client(caddr_t buf, int len) {
  struct sockaddr_ms mirror; // Bereitstellen einer Adreßstruktur
  // Initialisierung der MUPIX-Bibliothek und
  // Anmelden eines namenlosen Klienten in der Domäne ecs
  mpx_open("ecs", NULL, NULL);
  // Initialisieren der Adreßstruktur mit dem symbolischen Namen des Empfängers
  mpx_initaddr("mirror", &mirror);
  // Senden einer Nachricht
  if (mpx_sendto(buf, len, 0, 0, &mirror, NULL) != OK) {
    perror("mirror_client: mpx_sendto failed") ;
    return ERROR;
  }
  // Warte maximal 10 Sekunden auf eine Antwort des Servers. Dabei wird mit der
  // MS_PEEK  Flagge wird angezeigt, daß nur auf die Nachricht eines
  // bestimmten Absenders gewartet wird.
  if (mpx_recvfrom(buf, len, 10, MS_PEEK, &mirror, NULL) != len)  {
    perror("mirror_client: mpx_recvfrom failed") ;
    return ERROR;
  }
  mpx_close(); //  Schließen des MUPIX-Kanals
  return OK;
}

void mirror_server() {
  caddr_t buf [NMAX];
  int len;
  struct sockaddr_ms from;
  // Anmelden des Servers unter dem Namen "mirror"
  mpx_open("ecs", "mirror", NULL);
  // Warte maximal 120 Sekunden auf eine Nachricht
  while ((len = mpx_recvfrom(buf, NMAX, 120, 0, &from, NULL)) > 0) {
    // Ausgabe auf dem Bildschrim
    cout << "bytes: " << len << " " << buf << endl;
    mpx_sendto(buf, len, 0, 0, &from, NULL); // Einfach zurücksenden.
 }
 mpx_close();
}
```

Abb:B.1: Beispiele für die Verwendung von MUPIX-Routinen. Das Klientenprogramm `mirror_client` sendet die Nachricht `buf` der Länge `len` an den Server `mirror_server`. Dieser empfängt die Nachricht, druckt sie zusammen mit der Länge aus und sendet sie wieder an den Klienten zurück.

Anhang C

Anhang zum NMR-System der Spektrometer

C.1 Elastische Elektronenstreuung an $^{181}Ta(e,e')$

Die in Abschnitt 6.1 beschriebenen Magnetfeld-Eichmessungen konnten nur bei nicht evakuierter Vakuumkammer des Spektrometers durchgeführt werden. Deshalb traten hier die bei den eigentlichen physikalischen Messungen zwischen dem Purcell-Filter und der Vakuumkammer bestehende Druckdifferenz und damit die den magnetischen Kräften entgegengesetzten und in der Größenordnung vergleichbaren Vakuumkräfte [sch95] nicht auf.

Um den Einfluß der Vakuumkräfte auf den Polschuh und damit auf das Magnetfeld zu studieren, wurden bei einer mittleren Feldstärke zwei Messungen mit elastisch an ^{181}Ta gestreuten Elektronen durchgeführt, die sich dadurch unterscheiden, daß bei Messung A das als Druckkammer ausgebildete Purcell-Filter (Abb.C.1) unter Normaldruck stand, während es bei Messung B auf Vorvakuum ausgepumpt wurde. Dadurch entfällt bei der zweiten Messung genau wie bei der seinerzeitigen Eichmessung die zwischen dem Purcell-Filter und der evakuierten Vakuumkammer des Dipolmagneten bestehende Druckdifferenz.

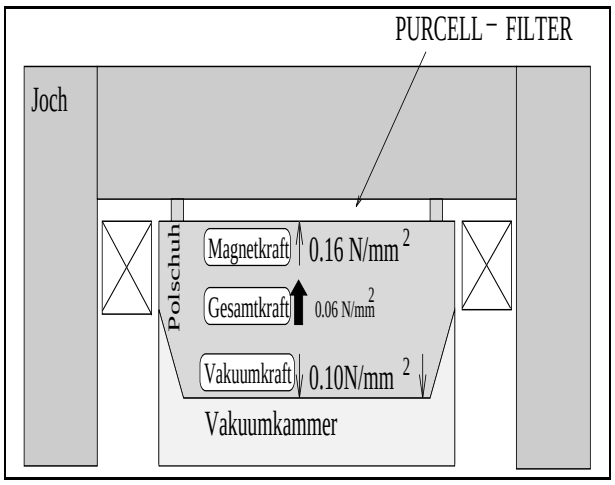


Abb.C.1: Kräfte auf den Polschuh des Dipolmagneten AD2 [sch95]. Die Vakuumkraft ergibt sich aus der Druckdifferenz zwischen Vakuumkammer und Purcell-Filter. Die Magnetkraft durch die "Ausbuchtung" des Felds im Randbereich.

Die beiden Messungen wurden bei einer Energie $E = 495.11\ MeV$ und einem zentralen Streuwinkel $\phi = 32.27°$ durchgeführt. Tantal wurde als Targetmaterial (Targetdicke 60mg/cm^2) ausgewählt, weil es mit einer Massenzahl von 181 ein sehr schwerer Kern ist, bei dem die Änderung der Rückstoßenergie über die Winkelakzeptanz des Spektrometers klein ist, so daß die Peaklage besonders gut zu bestimmen ist.

Zur Untergrundreduktion sind bei der Analyse nur Ereignisse berücksichtigt worden, bei denen auch der Čerenkov-Detektor angesprochen hat. Von den gemes-

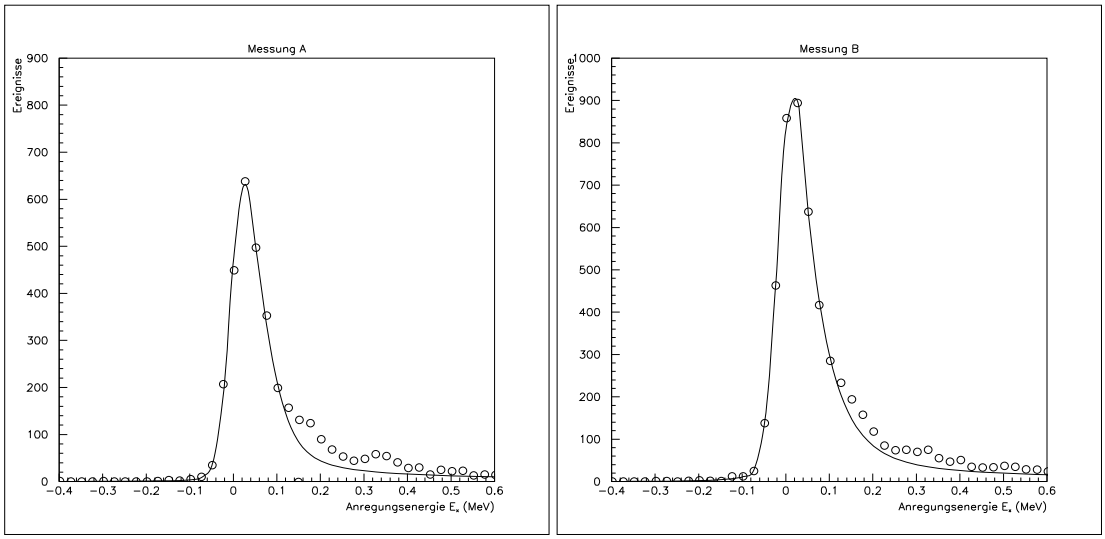


Abb.C.2: Gemessene Anregungsspektren von ^{181}Ta. Messung A mit, Messung B ohne Druckdifferenz zwischen Purcell-Filter und Feldbereich zwischen den Polschuhen.

senen Bildebenenkoordinaten wurde mit den bekannten Abbildungseigenschaften [ko95] das Energiespektrum der gestreuten Elektronen unter Berücksichtigung von Rückstoßkorrekturen erstellt.
Abb.C.2 zeigt die beiden Spektren.
Zur Bestimmung der Peaklage wurde eine analytische Linienform, die auch den Strahlenschwanz berücksichtigt, an die elastische Linie ($I^\pi = \frac{7}{2}^+$) und an die beiden ersten angeregten Niveaus von Tantal ($I^\pi = \frac{9}{2}^+, \omega = 0.136$ MeV, $I^\pi = \frac{11}{2}^+, \omega = 0.301$ MeV) der gemessenen Spektren angepaßt. Die Auswertung erfolgte mit Hilfe der Programme `espace` und `allfit` [off93].
Für die relative Lage E_0 und für die Breite ΔE der elastischen Linien ergaben sich die folgenden Werte (der Nullpunkt wurde willkürlich auf die elastische Linie der Messung A gelegt):

	$E_0[keV]$	$\Delta E[keV]$
Messung A	0.0 ± 1.1	59.6 ± 2.4
Messung B	-1.2 ± 0.7	37.6 ± 1.4

Im Rahmen der Fehler wird also keine Verschiebung beobachtet.

Zum Studium dieses Effekts sollte eine weitere Messung bei einer geringeren Energie, d.h. bei kleineren magnetischen Kräften, durchgeführt werden. Eine solche Messung ist jederzeit möglich, weil im Zuge der hier beschriebenen Messungen das Purcell-Filter an das Standard-Vakuumsystem des Spektrometers angeschlossen wurde.

C.2 Spektrometer B

C.2.1 Schnittzeichnung des Clam-Shell Magneten

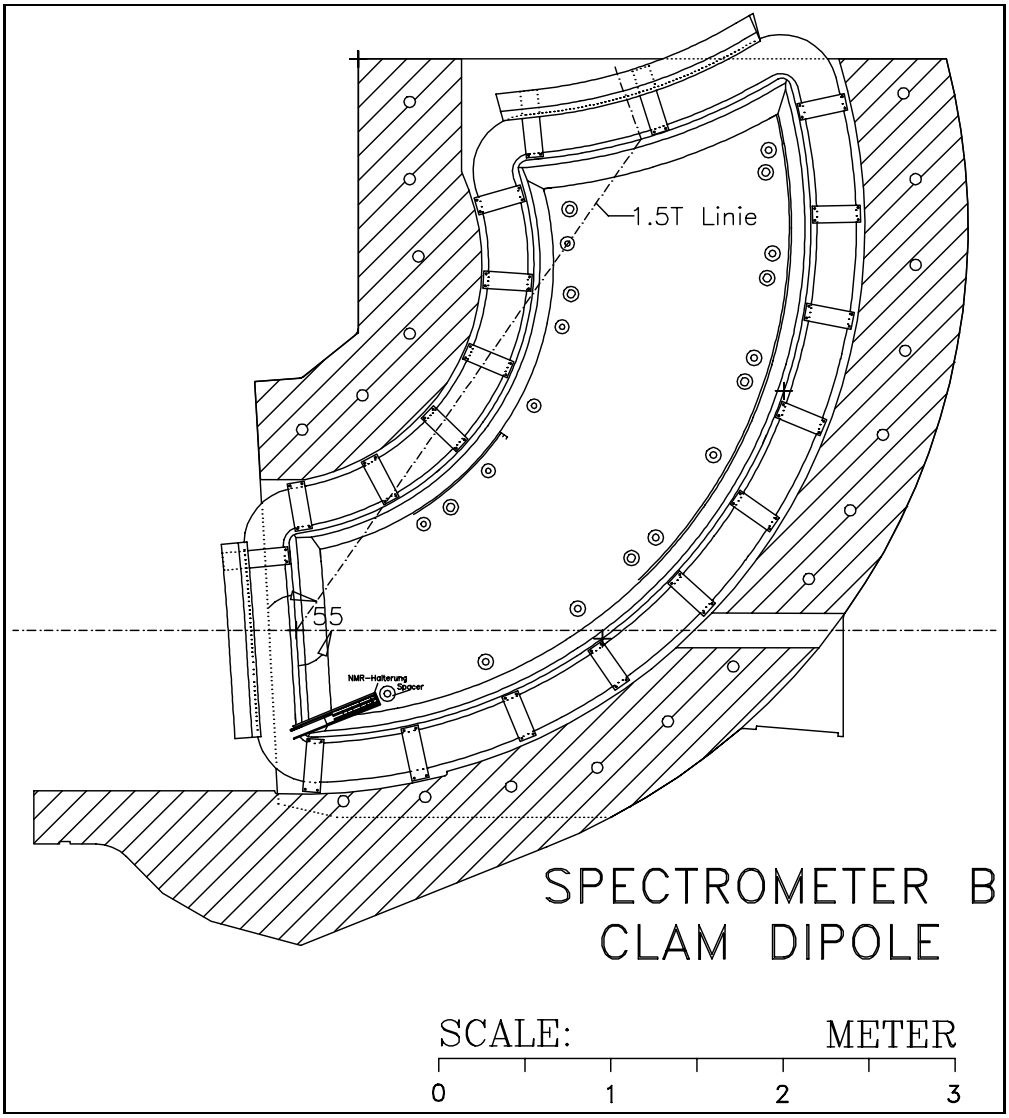


Abb.C.3: Schnittzeichnung von Spektrometer B (Bild aus [ko95]).

C.2.2 Stromversorgung der Kompensationsspulen

Bit	Beschreibung
1	
⋮	DAC Vorgabe (lsb first)
16	
17	Handshake
18	DAC Anwahl
19	Polwender
20, 21	Sondenanwahl
22	
⋮	frei
24	
25	
⋮	Bussteuerung
32	

Bitbelegung der Stromversorgung für die Kompensationsspulen für die NMR-Messung im Spektrometer B.

DAC-Wert	I_1/mA	I_2/mA
1000	31.8	33.1
2000	63.6	65.9
3000	95.2	98.8
4000	126.9	131.6
5000	158.6	164.5
7500	242.0	251
10000	322	334
12500	403	418
15000	483	501
17500	564	584
20000	644	668
22500	725	751
25000	805	835
27500	885	918
30000	966	1002
32500	1045	1086
35000	1124	1169
37500	1202	1252
40000	1280	1335
45000	1425	1500
50000	1592	1650
55000	1690	1772
60000	1776	1862
65000	1785	1906

Die Tabelle zeigt den Strom in den Kompensationsspulen in Abhängigkeit des eingestellten DAC-Wertes. Die Last beträgt 2 Ω.

C.3 Zusammenhang zwischen Magnetfeld im Innenbereich und im Bereich der Meßsonden von Spektrometer A

	NMR-Anzeige in Tesla							
	Referenzsonden				Meßsonden			
I_{DAC} / Amp.	Typ	B/T	Typ	B/T	Typ	B/T	Typ	B/T
196.40959	2	0.244456	3	0.244463	2	0.244416	3	0.244446
200.361969	2	0.249321	3	0.249328	2	0.249279	3	0.24931
206.368698	2	0.256761	3	0.256768	2	0.256718	3	0.256749
393.927307	3	0.489551	4	0.489550	3	0.489515	4	0.489411
402.018799	3	0.499521	4	0.499521	3	0.499484	4	0.499378
410.064667	3	0.509500	4	0.509499	3	0.509462	4	0.509354
466.831085	4	0.579932	-	-	4	0.579779	-	-
497.142761	4	0.617527	-	-	4	0.617368	-	-
507.168274	4	0.629906	-	-	4	0.629741	-	-
542.155823	4	0.673471	-	-	4	0.673311	-	-
553.274231	4	0.687200	-	-	4	0.687038	-	-
564.647156	4	0.701235	-	-	4	0.701070	-	-
576.252502	4	0.715529	-	-	4	0.715361	-	-
587.860596	4	0.729904	-	-	4	0.729732	-	-
609.173279	4	0.756479	5	0.756494	4	0.756312	5	0.756437
621.694397	4	0.771921	5	0.771937	4	0.771752	5	0.771880
634.467346	4	0.787663	5	0.787678	4	0.787492	5	0.787621
647.519714	4	0.803757	5	0.803773	4	0.803584	5	0.803715
660.597046	4	0.819875	5	0.819890	4	0.819698	5	0.819831
668.876648	4	0.830279	5	0.830295	4	0.830108	5	0.830244
682.627686	4	0.847236	5	0.847253	4	0.847064	5	0.847201
696.631836	4	0.864485	5	0.864502	4	0.864311	5	0.864450
710.749390	4	0.881876	5	0.881894	4	0.881700	5	0.881839
736.037964	4	0.913290	5	0.913309	4	0.913119	5	0.913265
751.179260	4	0.931911	5	0.931930	4	0.931739	5	0.931886
766.539185	4	0.950791	5	0.950811	4	0.950617	5	0.950767
782.087219	4	0.969880	5	0.969899	4	0.969702	5	0.969853
827.43365	4	1.02591	5	1.02593	4	1.02574	5	1.0259
844.503418	4	1.046796	5	1.046817	4	1.046626	5	1.046788
861.802856	5	1.067988	-	-	5	1.067956	-	-
930.562012	5	1.152465	-	-	5	1.152443	-	-
949.906067	5	1.175992	-	-	5	1.175973	-	-
969.597412	5	1.199879	-	-	5	1.199846	-	-
988.902710	5	1.223541	-	-	5	1.223520	-	-
1009.610840	5	1.248524	-	-	5	1.248493	-	-
1030.821167	5	1.273895	-	-	5	1.273846	-	-
1083.38757	5	1.33659	-	-	5	1.336490	-	-
1107.233276	5	1.363983	-	-	5	1.363808	-	-
1132.357178	5	1.391935	-	-	5	1.391635	-	-
1159.056763	5	1.420534	-	-	5	1.420021	-	-
1178.34612	5	1.44076	-	-	5	1.44004	-	-
1208.77625	5	1.47114	-	-	5	1.46999	-	-
1239.73254	5	1.50097	-	-	5	1.49933	-	-

Tabelle C.1: Die mit der Meßsonde (Port-Sonde) und der mit der Referenzsonde gemessenen Feldstärken bei Dipol AD1 in Abhängigkeit vom eingestellten Strom und dem verwendeten Sondentyp. Dieser Zusammenhang ist auch in Abb. 6.1 graphisch dargestellt und mit den Polynomanpassungen Tab.6.2 beschrieben.

	NMR-Anzeige in Tesla							
	Referenzsonden				Meßsonden			
I_{DAC} /Amp	Typ	B/T	Typ	B/T	Typ	B/T	Typ	B/T
88.16647	2	0.109983	-	-	2	0.109996	-	-
103.96682	2	0.129489	-	-	2	0.129506	-	-
144.740494	2	0.180436	-	-	2	0.180472	-	-
152.432449	2	0.189923	-	-	2	0.189963	-	-
167.82877	2	0.208932	3	0.208936	2	0.208981	3	0.208975
183.25679	2	0.227939	3	0.227943	2	0.227998	3	0.227991
201.10493	2	0.249926	3	0.24993	2	0.250003	3	0.249997
291.476379	3	0.362454	-	-	3	0.362561	-	-
303.714844	3	0.377581	-	-	3	0.377690	-	-
316.444183	3	0.393304	-	-	3	0.393418	-	-
328.828827	3	0.408627	4	0.408625	3	0.408747	4	0.408706
391.29385	3	0.486303	4	0.486301	3	0.486459	4	0.486411
407.681641	3	0.506545	4	0.506543	3	0.506707	4	0.506657
424.758942	4	0.527620	-	-	4	0.527739	-	-
441.814148	4	0.548753	-	-	4	0.548881	-	-
527.879089	4	0.655524	-	-	4	0.655695	-	-
550.002014	4	0.682818	-	-	4	0.682996	-	-
572.265686	4	0.710385	-	-	4	0.710581	-	-
624.868225	4	0.775473	5	0.775477	4	0.775683	5	0.775706
643.484497	4	0.798414	5	0.798417	4	0.798627	5	0.798651
698.394104	4	0.866458	5	0.866461	4	0.866708	5	0.866734
703.67554	4	0.873007	5	0.873010	4	0.873267	5	0.873293
720.192993	4	0.893282	5	0.893285	4	0.893538	5	0.893564
725.41797	4	0.899750	5	0.899754	4	0.900019	5	0.900045
741.636108	4	0.919664	5	0.919666	4	0.919926	5	0.919951
-	4	0.938122	5	0.938125	4	0.938400	5	0.938426
787.072144	4	0.975864	5	0.975867	4	0.976166	5	0.976193
811.637268	4	1.006051	5	1.006055	4	1.006363	5	1.006391
836.150024	4	1.036110	5	1.036113	4	1.036428	5	1.036456
857.449768	5	1.062452	-	-	5	1.062801	-	-
883.526306	5	1.094408	-	-	5	1.094764	-	-
935.428345	5	1.158285	-	-	5	1.158686	-	-
964.752319	5	1.194053	-	-	5	1.194469	-	-
994.576904	5	1.230332	-	-	5	1.230761	-	-
1065.409546	5	1.316282	-	-	5	1.316847	-	-
1099.550415	5	1.356919	-	-	5	1.357600	-	-
1135.321167	5	1.398397	-	-	5	1.399277	-	-
1158.135010	5	1.424511	-	-	5	1.425588	-	-
1193.51270	5	1.463152	-	-	5	1.464385	-	-
1207.9823	5	1.4784652	-	-	5	1.479732	-	-
1237.387695	5	1.508400	-	-	5	1.509646	-	-
1247.64185	5	1.51845	-	-	5	1.51969	-	-
1290.26619	5	1.55839	-	-	5	1.5595	-	-

Tabelle C.2: Die mit der Meßsonde (Port-Sonde) und der mit der Referenzsonde gemessenen Feldstärken bei Dipol AD2 in Abhängigkeit vom eingestellten Strom und dem verwendeten Sondentyp. Dieser Zusammenhang ist auch in Abb. 6.2 graphisch dargestellt und mit den Polynomanpassungen Tab.6.3 beschrieben.

C.4 Zusammenhang zwischen Magnetfeld im Innenbereich und im Bereich der Meßsonden von Spektrometer C

NMR-Anzeige in Tesla							
Referenzsonden				Meßsonden			
Typ	B/T	Typ	B/T	Typ	B/T	Typ	B/T
5	1.520911	0	0	5	1.51875	0	0
5	1.494337	0	0	5	1.492898	0	0
5	1.477083	0	0	5	1.476095	0	0
5	1.440922	0	0	5	1.440714	0	0
5	1.422410	0	0	5	1.422491	0	0
5	1.383877	0	0	5	1.38434	0	0
5	1.344829	0	0	5	1.345481	0	0
5	1.305016	0	0	5	1.305752	0	0
5	1.284439	0	0	5	1.285184	0	0
5	1.244302	0	0	5	1.245066	0	0
5	1.203852	0	0	5	1.204609	0	0
5	1.18309	0	0	5	1.183832	0	0
5	1.162950	0	0	5	1.163686	0	0
5	1.183103	0	0	5	1.183844	0	0
5	1.142578	0	0	5	1.143296	0	0
5	1.112131	0	0	5	1.112832	0	0
5	1.081577	0	0	5	1.082254	0	0
5	1.050286	4	1.050272	5	1.050928	4	1.051046
5	1.019851	4	1.019837	5	1.020469	4	1.020586
5	0.989294	4	0.989282	5	0.989887	4	0.990001
5	0.958681	4	0.958669	5	0.959249	4	0.959360
5	0.927440	4	0.927429	5	0.927978	4	0.928087
5	0.896888	4	0.896878	5	0.897401	4	0.897507
5	0.876539	4	0.876528	5	0.877035	4	0.877139
5	0.783725	4	0.783717	5	0.784141	4	0.784236
5	0.742885	4	0.742877	5	0.743274	4	0.743366
5	0.727593	4	0.727584	5	0.727972	4	0.728062
0	0	4	0.619219	0	0	4	0.619601
0	0	4	0.588534	0	0	4	0.588897
0	0	4	0.547514	0	0	4	0.547847
3	0.505937	4	0.50593	3	0.506162	4	0.506226
3	0.47522	4	0.475213	3	0.475432	4	0.475493
3	0.454736	4	0.45473	3	0.454936	4	0.454995
3	0.43938	4	0.439373	3	0.439571	4	0.439629
3	0.309917	0	0	3	0.31004	0	0
3	0.279143	0	0	3	0.279255	0	0
3	0.222323	0	0	3	0.222400	2	0.222446
3	0.206953	0	0	3	0.207027	2	0.207071
3	0.191587	0	0	3	0.191654	2	0.191696
3	0.176212	0	0	3	0.176273	2	0.176313

Tabelle C.3: Die mit der Meßsonde (Port-Sonde) und der mit der Referenzsonde gemessenen Feldstärken in Abhängigkeit vom verwendeten Sondentyp für den Dipol 2 von Spekrometer C.

NMR-Anzeige in Tesla					
Referenzsonden		Meßsonden			
Typ	B/T	Typ	B/T	Typ	B/T
5	1.51067	5	1.50822	0	0
5	1.49497	5	1.49293	0	0
5	1.46238	5	1.46102	0	0
5	1.42794	5	1.42714	0	0
5	1.3823	5	1.38192	0	0
5	1.34916	5	1.34895	0	0
5	1.31506	5	1.31494	0	0
5	1.28026	5	1.28021	0	0
5	1.26471	5	1.26467	0	0
5	1.23472	5	1.23472	0	0
5	1.18414	5	1.18416	0	0
5	1.1537	5	1.15374	0	0
5	1.10207	5	1.10212	0	0
5	1.06141	5	1.06146	0	0
5	1.02059	5	1.02065	4	1.02072
5	0.979701	5	0.979759	4	0.979832
5	0.927907	5	0.927967	4	0.928037
5	0.887	5	0.887061	4	0.887123
5	0.856349	5	0.85639	4	0.856462
5	0.825624	5	0.825681	4	0.825742
5	0.800427	5	0.800492	4	0.800546
5	0.776888	5	0.776939	4	0.776999
5	0.755403	5	0.755458	4	0.755515
5	0.732866	5	0.732915	4	0.732963
3	0.515923	3	0.51592	4	0.516012
3	0.500604	3	0.500597	4	0.500686
3	0.485191	3	0.485174	4	0.485253
3	0.464663	3	0.464643	4	0.464728
3	0.406549	3	0.406546	4	0.406616
3	0.392244	3	0.39224	4	0.392302
3	0.381978	3	0.381975	4	0.382036
3	0.371689	3	0.371706	4	0.371756
3	0.360933	3	0.360944	4	0.361012
3	0.350716	3	0.350712	4	0.350775
3	0.340426	3	0.340424	0	0
3	0.252663	3	0.252633	2	0.252687
3	0.243434	3	0.24342	2	0.243487
3	0.195109	3	0.195103	2	0.195186
3	0.187885	3	0.187870	2	0.187941
3	0.179592	3	0.179585	2	0.179637

Tabelle C.4: Die mit der Meßsonde (Port-Sonde) und der mit der Referenzsonde gemessenen Feldstärken in Abhängigkeit vom verwendeten Sondentyp für den Dipol 1 von Spekrometer C.

Glossar

CAMAC Computer Automated Measurement and Control, Bus Standard in der Kernphysik zum Anschluß von Modulen an einen Rechner

Baum Ein Baum ist eine nicht-leere Menge von Vertizes und Kanten, die bestimmten Bedingungen unterliegen. Jeder Vertex (Knoten) ist ein Objekt, das durch einen bestimmten Namen (Schlüssel) identifiziert wird. Eine Kante ist die Verbindung zwischen zwei Vertizes (Abb.5.5). Ein Pfad innerhalb des Baums wird durch die Aufzählung von verschiedenen Knoten, die jeweils durch Kanten miteinander verbunden sind, beschrieben.

Client/Server siehe Server

Deskriptor Eine Zahl, die vom System zugewiesen wird, wenn eine Datei oder ein Socket mit Systemaufrufen geöffnet wird. Die Zahl identifiziert den Zugriffspfad auf die Datei oder das Socket innerhalb eines Prozesses.

ECS Experiment Control System

Interprocess Communication (IPC) Datentransfer-Kommunikation zwischen Prozessen.

IP Datagramm Die elementare Dateneinheit, die über das Internet transferiet wird. Ein Datagramm enthält neben den Daten als Pfadinformation die Adresse des Sende- und Zielrechners.

Klasse Als Klasse bezeichnet man die Menge aller Objekte mit gleicher Struktur und gleichem Verhalten.

Maskiert Wenn ein Signal "maskiert" worden ist, wird seine Weitergabe vom System verzögert, bis es unmaskiert ist. Bei sicheren Signalmechanismen maskiert das System automatisch ein empfangenes Signal, während es abgearbeitet wird. Ansonsten erfolgt das Maskieren durch einen entsprechenden Systemaufruf, der die Nummer des zu maskierenden Signals enthält.

MUPIX MUltiProcessor Interprocess Communication System

Objekt Ein Objekt stellt die Abstraktion eines bestimmten Gegenstands oder einer bestimmten Sache innerhalb eines Problembereichs dar. Jedes Objekt ist durch seinen Zustand, seine Funktionalität und seine Identität charakterisiert. Dee Begriff Objekt ist identisch mit dem aus der englischsprachigen Literatur übernommen Begriff *Instanz*.

Pipe Pipes stellen eine IPC-Einrichtung dar, die den Datenfluß zwischen zwei Prozessen in einer Richtung unterstützt.

Prozeß Ein Prozeß ist eine durch ein Programm spezifizierte Folge von Aktionen, deren erste begonnen, deren letzte aber noch nicht abgeschlossen ist.

Server-Prozeß Ein Prozeß, der Dienste auf dem Weg über eine Interprozeß-Kommunikations-Einrichtung anbietet. Diese Dienste werden von Client-Prozessen in Anspruch genommen (Abb.4.9).

Shared Memory Ein Speichersegment, das von mehreren Prozessen benutzt wird.

Signal Software-Ereignis, das z.B. nach einem Hardware-Interupt eintritt.

Socket Ein Socket bezeichnet einen Endpunkt der Kommunikation. Der Begriff stammt aus dem 4.3BSD-UNIX Kommunikations-Modell. Ein Socket beinhaltet auch die Datenstruktur, mit der die Abstraktion eines Kommunikationsendpunktes implementiert ist. Es wird mit einem Systemaufruf erzeugt.

Systemaufruf Ein Systemaufruf ist eine Dienstanforderung an das Betriebssystem durch einen Prozeß.

Transmission Contol Protocol (TCP) Ein verbindungs-orientiertes Protokoll aus der Internet-Protokollfamilie, das einen zuverlässigen Datentransport garantiert. Außerdem definiert es Mechanismen zur Flusskontrolle.

User Datagramm Protocol (UDP) Ein einfaches, aber unzuverlässiges Datagramm-Protokoll aus der Internet-Protokollfamilie.

Vererbung Eine Beziehung zwischen zwei Klassen, wobei die abgeleitete ("erbende") Klasse die Eigenschaften und Dienste übernimmt, die bereits in einer anderen Klasse definiert wurden. Durch die Vererbung wird eine Hierachie definiert, bei der die abgeleitete Klasse von einer Basisklasse erbt; typischerweise erweitert eine abgeleitete Klasse den Umfang der Eigenschaften und Dienste ihrer Basisklasse oder spezialisiert die bereits eingeführten. Die Basisklasse ist der "größte gemeinsame Nenner" all ihrer abgeleiteten Klassen.

verkettete Liste Eine verkettete Liste ist eine Menge von Datenelementen, die wie ein Feld (Array) sequentiell organisiert ist. Im Gegensatz zum statisch angeordneten Array enthält bei der verketteten Liste jedes Datenelement einen Zeiger zum Speicherplatz des nächsten Elements der Liste (Abb.5.5, Abb.4.2), so daß die Liste dynamisch wachsen und schrumpfen kann.

XDR eXternal Data Representation (siehe S.54)

Zeiger Speicheradresse

Zustandsmaschine Eine Zustandsmaschine (endlicher Automat) ist ein System mit diskreten Ein- und Ausgaben, das sich stets in einer von nur endlich vielen möglichen internen Konfigurationen, den sogenannten Zuständen, befindet. Ein Zustand - und damit die Reaktionen des Systems - ist durch die in allen bisherigen Eingaben enthaltenen Informationen festgelegt.

Index

Lebenslauf

Name:	Helmut Wilhelm Kramer	
Geburtsdatum:	16. Mai 1964	
Geburtsort:	Mainz	
Staatsangehörigkeit:	deutsch	
Familienstand:	ledig	
Wohnort:	Rheinstraße 17 in 65462 Ginsheim	
Schulbildung:	1970 - 1974	Grundschule in Ginsheim
	1974 - 1976	Integrierte Gesamtschule in Ginsheim
	1976 - 1983	Gymnasium am Kurfürstlichen Schloß, Mainz
	1983	Abitur am Gymnasium am Kurfürstlichen Schloß, Mainz
Studium:	1983 - 1986	Physikstudium Johannes Gutenberg-Universität Mainz
	1986 - 1987	Auslandsstudium an der University of Washington in Seattle, USA
	1987 - 1988	Fortsetzung des Physikstudiums in Mainz, Diplomarbeit
	Januar 1989	Abschluß des Physikstudiums mit Diplom
	Januar 1989	Abschluß des Physikstudiums mit Diplom Thema der Diplomarbeit: Pilotstudie zu den Zweischleifenkorrekturen im Müonzerfall
Arbeitsstellen:	April 1988 - Okt. 1989	Wiss. Mitarbeiter am Universitätsklinikum Mainz
	seit Okt. 1989	Wiss. Mitarbeiter und Doktorand am Institut für Kernphysik in Mainz