Konzeption und Realisierung eines Datenerfassungssystems für die Experimente am Mainzer Mikrotron und Inbetriebnahme für Koinzidenzexperimente mit virtuellen Photonen an der Drei-Spektrometer-Anlage

Dissertation

zur Erlangung des Grades

„Doktor der Naturwissenschaften“

am Fachbereich Physik
der Johnnes Gutenberg-Universität
in Mainz

Klaus Werner Krygier
geboren in Dienheim

Mainz 1995

Dekan:	Prof. Dr. F. Scheck
1. Berichterstatter:	Prof. Dr. Th. Walcher
2. Berichterstatter:	

Tag der mündlichen Prüfung:

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Institut für Kernphysik der Universität Mainz befaßt sich mit der Erforschung des Atomkerns und seiner Bausteine. Zu diesem Zweck wurde in den vergangenen Jahren ein neuer Elektronenbeschleuniger, das Mainzer Mikrotron (MAMI), aufgebaut und in Betrieb genommen. Dieser Beschleuniger ist als dreistufigs Rennbahnmikrotron in der Lage, einen Elektronenstrahl mit einer Endenergie von 855 MeV und einem Strahlstrom von bis zu 100 μA im Dauerstrichbetrieb zu liefern. Damit wurde es unter anderem möglich, Koinzidenzexperimente mit bisher nicht erreichter Genauigkeit durchzuführen.

Die Kollaboration A1, die sich mit Experimenten mit virtuellen Photonen beschäftigt, hat dafür eine Drei-Spektrometer-Anlage geplant und aufgebaut, die für Koinzidenzexperimente mit geladenen Teilchen universell eingesetzt werden kann. Die hochauflösenden Detektorsysteme erlauben eine Reihe neuartiger Experimente mit Elektronen zu Untersuchungen der Kernstruktur und im Bereich der Elementarteilchen. Als Beispiel seien hier (e,e'p)-Messungen an ^{12}C und ^{16}O im quasifreien Bereich, eine Trennung von longitudinalem und transversalem Anteil bei der Elektroproduktion von Pionen am Proton und die Untersuchung der Δ-Resonanz in leichten Kernen genannt.

Bei solchen Experimenten ist es erforderlich, eine große Zahl von Meßwerten Ereignis für Ereignis zu erfassen und weiterzuverarbeiten, um später möglichst genau jedes registrierte physikalische Ereignis rekonstruieren und auswerten zu können. Auch in der Vergangenheit ließen sich solche Aufgaben ohne Rechnerunterstützung nicht lösen, mit gesteigerter Leistungsfähigkeit von Beschleuniger und Detektorsystemen sind aber auch die Anforderungen an die Datenerfassung sowohl im Bereich der Rechner-Hardware als auch auf Software-Ebene gestiegen. Um nun diese Experimente durchführen zu können, war hier eine grundlegende Neukonzeption notwendig,

Ziel dieser Arbeit war es, auf der Basis neuer, leistungsfähigerer Rechner und moderner Software-Methoden ein neues Datenerfassungssystem zu konzipieren und zu realisieren, das die Anforderungen der an MAMI durchzuführenden Experimente bestmöglich abdeckt. Dabei konnte auf die im Institut für Kernphysik gemachten langjährigen Erfahrungen sehr gut aufgebaut werden und eine kon-

sequente Weiterentwicklung bereits etablierter Methoden vorgenommen werden, doch war eine von Grund auf neue Umsetzung in Hard- und Software notwendig.

In Kombination mit zwei weiteren Doktorarbeiten [Kund95, Kram95], die die Bereiche Experimentsteuerung und Überwachung des Experiments abdecken, konnte so ein leistungsfähiges und flexibles System, das Mainzer Experimentsteuerungs- und Datenerfassungssystem (MECDAS), geschaffen werden. Diese Arbeit konzentriert sich dabei i. w. auf die direkt mit der ereignisweisen Erfassung der Meßdaten zusammenhängenden Aufgaben.

Das Kapitel 2 geht zunächst auf die Anforderungen ein, die an das Datenerfassungssystem gestellt werden. Dazu wird neben einer kurzen Erläuterung der physikalischen Fragestellung das experimentelle Umfeld bestehend aus Spektrometer-Anlage und Detektorsystemen beleuchtet und die daraus resultierenden Konsequenzen für die Datenerfassung beschrieben. Es folgt mit Kapitel 3 eine Diskussion der für die Neukonzeption eines Datenerfassungssystems zu beachtenden Grundlagen, um die Entscheidungsfindung in Hard- und Software-Fragen klar zu machen. Kapitel 4 stellt in einer Übersicht das Konzept von MECDAS vor, während in den folgenden Kapiteln etwas näher auf die Realisierung der wesentlichen Komponenten des Datenerfassungssystems eingegangen wird. Infolge der Komplexität des Systems konnte dabei eine nur bruchstückhafte Beschreibung vorgenommen werden. Es wird der Aufbau der zentralen Datenerfassungs-Software erläutert (Kap. 5) und die Verfahren, mit denen der Experimentator sie einfach für unterschiedliche Randbedingungen konfigurieren kann (Kap. 6). Im Anschluß folgt die Darstellung ergänzender Software (Kap. 7) bis hin zu einem Programmpaket zur On-line-Analyse (Kap. 8). Kapitel 9 schließlich beschreibt in einem kurzen Abriß die Inbetriebnahme des Datenerfassungssystems und ihren Einsatz bei den Experimenten an der Drei-Spektrometer-Anlage. Abschließend werden in Kapitel 10 Untersuchungen zur Leistungsfähigkeit des Datenerfassungssystems beschrieben, die bisher gewonnenen Erfahrungen diskutiert und in Kapitel 11 Vorschläge für Verbesserungsmöglichkeiten erörtert.

Kapitel 2

Anforderungen an ein Datenerfassungssystem für Experimente an MAMI

2.1 Physikalische Fragestellung

2.1.1 Experimente an MAMI

Am Institut für Kernphysik der Universität Mainz wird seit den 60er Jahren die elektromagnetische Struktur der Kerne und Nukleonen erforscht. Es wird insbesondere untersucht, ob sich die Struktur der freien Nukleonen in der Kernbindung verändert und wie sich die die Wechselwirkung vermittelnden Pionen auf diese Struktur auswirken. Zur Bestimmung solcher Einteilchen-Eigenschaften ist es notwendig, den Zustand der bei einer Streuung am Kern herausgeschlagenen Nukleonen zu vermessen. Gleichzeitig muß man die beim Streuprozeß eines Elektrons oder auch eines Photons übertragene Energie und den übertragenen Impuls genau kennen, was den koinzidenten Nachweis zweier zusammengehörender Teilchen notwendig macht.

Um in vertretbarer Zeit zu statistisch relevanten Daten zu gelangen, erfordern die Koinzidenzwirkungsquerschnitte der interessierenden Reaktionen bei den üblichen Raumwinkelakzeptanzen der Detektoren so hohe mittlere Strahlströme, daß infolge des extrem hohen Stroms während der einzelnen Elektronenpulse eines gepulsten Beschleunigers, wie er ursprünglich in Form eines Linarbeschleunigers (LINAC) eingesetzt wurde, bei einem niedrigen Tastverhältnis eine extrem hohe, technisch nicht mehr realisierbare Zeitauflösung der Detektoren notwendig wird. Daher wurde ein Dauerstrich-Elektronenbeschleuniger entwickelt, das Mainzer Mikrotron, das in der Ausbaustufe A ab 1983 für Experimente genutzt werden konnte. Die ursprüngliche Energie von 180 MeV wurde in einer Erweiterungsstufe des Mikrotrons in den Jahren 1987 bis 1990 auf 855 MeV erhöht. Der Gesamtbeschleuniger besteht nun aus drei hintereinander geschalteten Rennbahnmikrotrons und einem 3.5 MeV-Einschußbeschleuniger, in den

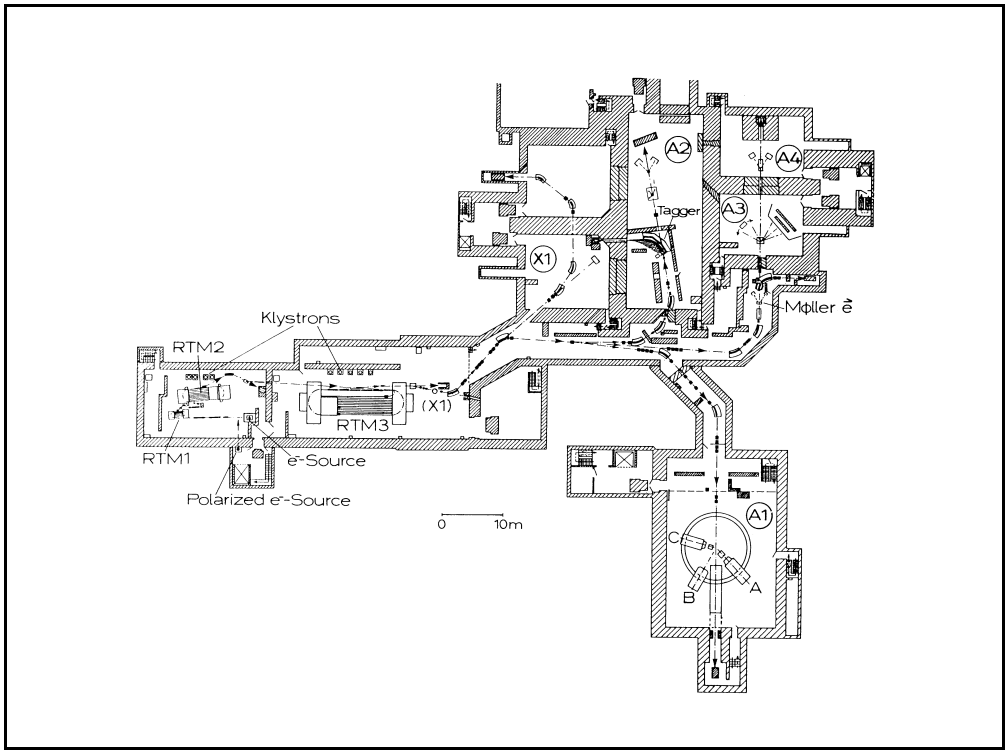

Abb. 2.1: Grundriß des Mainzer Mikrotrons, das aus den drei Rennbahnmikrotrons RTM1 bis RTM3 besteht, und der angegliederten Experimentieranlage.

auch polarisierte Elektronen eingespeist werden können (Abb. 2.1). Das Prinzip des Rennbahnmikrotrons besteht darin, einen Linearbeschleuniger mit kleinem Energiegewinn im Dauerstrich zu betreiben und die gewünschte Endenergie durch wiederholtes Durchlaufen des Strahls durch den gleichen Beschleuniger zu erzielen. Dazu sind zu beiden Seiten des Linearbeschleunigers Ablenkmagnete angebracht, die den Strahl jeweils um 180^o umlenken und über „Rennbahnen“ wieder auf die Beschleunigerachse zurückführen [Herm76].

Verschiedene Arbeitsgruppen innerhalb und außerhalb des Instituts nutzen die Vorzüge des kontinuierlichen Elektronenstrahls von MAMI für eine Vielzahl von Experimenten. Die Kollaboration X1 interessiert sich für hochbrilliante Strahlungsquellen im Bereich weicher und harter Röntgenstrahlung zur Anwendung in verschiedenen Bereichen der Physik, Materialwissenschaft, Medizin und Biologie. In der Kollaboration A4 soll die paritätsverletzende Komponente der Elektronenstreuung untersucht und der „schwache“ elektrische Formfaktor des Nukleons gemessen werden. Das Hauptziel der A3-Kollaboration ist die Messung des elektrischen Formfaktors des Neutrons, des sogenannten Ladungsformfaktors. Die Kollaboration A2 führt Experimente mit reelen Photonen durch. Dazu werden die Elektronen in einem ersten Target abgebremst, wobei hochenergetische Bremsstrahlungsphotonen emittiert werden, deren Energie durch Messung des Impulses des zu jedem individuellen Photon gehörenden abgebremsten Elektrons markiert wird. Die Photonen lösen in einem zweiten Target die interessierende Reaktion aus.

2.1.2 Experimente mit virtuellen Photonen

Ziel der Arbeit der A1-Kollaboration ist die Untersuchung der Wechselwirkung zwischen den Konstituenten des Kerns über deren Anregung durch ein aus der Elektronenstreuung stammendes, virtuelles Austauschphoton. Dazu muß in Koinzidenzexperimenten vom Typ A(e,e'x)B Streuwinkel und Impuls von gestreutem Elektron und der in den zu untersuchenden Reaktionen entstehenden Teilchen (x) bestimmt und eine Teilchenidentifikation vorgenommen werden, wobei als entstehende Teilchen insbesondere Protonen und Pionen in Frage kommen. Abb. 2.2 gibt eine vereinfachte Darstellung eines solchen Streuprozesses. Das wissenschaftliche Programm konzentriert sich zunächst auf die folgenden experimentellen Untersuchungen [A192]:

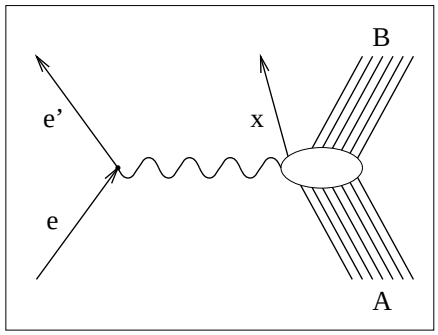

Abb. 2.2: Feynman-Graph der Reaktion A(e,e'x)B in erster Born'scher Näherung

- Koinzidenzexperimente vom Typ (e,e'p) an den Kernen ^{2}H, ^{12}C, ^{16}O, ^{40}Ca und ^{48}Ca zur Trennung der verschiedenen Strukturfunktionen in einem weiten kinematischen Bereich.
- präzise Messung des magnetischen Formfaktors des Neutrons über die Reaktion ^{2}H(e,e'n) als Grundlage für eine entsprechend genaue Bestimmung des elektrischen Formfaktors des Neutrons.
- Trennung von longitudinalem und transversalem Teil des Wirkungsquerschnitts in der Elektroproduktion von Pionen am Proton, p(e,e'π^+)n, zur Ableitung von Axialvektor-Formfaktor und Pioneninhalt des Nukleons.
- Experimente zum Studium der Δ-Resonanz in Kernen, insbesondere die Untersuchung von Dreifach-Koinzidenzen vom Typ (e,e'pp), (e,e'π^+p) oder (e,e'π^-p) mit hoher energetischer Auflösung des Endzustands

2.2 Experimentelles Umfeld

2.2.1 Gemeinsamkeiten der MAMI-Experimente

Um die eben umrissenen physikalischen Fragestellungen untersuchen zu können, werden von den verschiedenen Kollaborationen des Instituts Experimente durchgeführt, die trotz ihrer unterschiedlichen physikalischen Zielrichtung in Hinblick auf die Datenerfassung doch viele Gemeinsamkeiten haben. Die Experimente sind in der Regel Einarm- oder Koinzidenzexperimente, bei denen die wenigen Reaktionsprodukte eines zu untersuchenden Ereignisses nachgewiesen und mit hoher Genauigkeit vermessen werden sollen. Alle Experimente untersuchen Aspekte der Elektronenstreuung in einem Energiebereich bis 855 MeV, der

ebenso wie weitere Experimentparameter vom Beschleuniger MAMI vorgegeben ist. Dazu gehören z. B. auch der maximale Strahlstrom, die Energie- und Ortsunschärfe des Elektronenstrahls oder das Tastverhältnis. All diese Parameter bestimmen in wesentlichen Punkten die Anforderungen an die Detektorsysteme zum Nachweis der Streuprodukte und damit auch an die Datenerfassung.

Während sich die einzelnen Detektoren und Detektorsysteme apparativ stark voneinander unterscheiden, basieren ihre Nachweismethoden prinzipiell auf einer beschränkten Anzahl von physikalischen Effekten. Das ist z. B. der Nachweis von Szintillationslicht oder Čerenkov-Strahlung mithilfe von Photomultipliern oder die Detektion geladener Teilchen aufgrund von Ionisation [Klei87b]. Diese Tatsache spiegelt sich letztendlich auch in der eingesetzten Experimentelektronik wider. Die zur Auslese der Detektoren notwendige Elektronik unterscheidet sich im Grundsatz nur wenig. Sie wird im wesentlichen von Standards aus dem Bereich der Kern- bzw. Hochenergiephysik abgedeckt. Bei allen Detektorsystemen wird CAMAC, Fastbus und VMEbus eingesetzt, und in der Regel werden auch die gleichen oder ähnliche ADC-, TDC- oder Zähler-Module verwendet. Auch in Bezug auf die prinzipiellen Vorgehensweisen und Algorithmen bei der rechnergesteuerten Datenaufnahme unterscheiden sich die einzelnen MAMI-Experimente nur wenig voneinander. Allen gemeinsam ist insbesondere die Tatsache, daß infolge des Duty-Faktors von 1 zu erfassende physikalische Ereignisse mit hoher Rate und ohne eine durch den Beschleuniger aufgeprägte zeitliche Struktur vorkommen. Das definiert eine der härtesten Anforderung an die Datenaufnahme. Sie muß in der Lage sein, sowohl innerhalb möglichst kurzer Zeiten auf die Registrierung eines Ereignisses zu reagieren, als auch die Aufnahme der dabei anfallenden Daten mit möglichst hoher Geschwindigkeit vorzunehmen.

Ein ebenfalls wichtiger Aspekt ist die Tatsache, daß sich die MAMI-Experimente in erster Näherung auch in Umfang und Komplexität der Detektorsysteme sehr ähnlich sind [SFB92]. Es sollte daher mit nicht allzu großem Aufwand möglich sein, ein entsprechend allgemein gehaltenes Datenerfassungssystem zu konzipieren und realisieren, das i. w. alle Anforderungen der verschiedenen Kollaborationen abdecken kann. Als Ausgangspunkt bot sich dabei die Drei-Spektrometer-Anlage der A1-Kollaboration an, da hier ein komplexes Detektorsystem von Grund auf neu geplant und aufgebaut werden mußte. Während bei anderen Detektoren z. T. auf Existierendes zurückgegriffen werden konnte, mußten hier im Bereich der Detektorentwicklung neue Wege beschritten werden. Auch bei der Experimentdatenverarbeitung war dafür eine Neukonzeption notwendig, die allein schon deshalb sinnvoll war, um die in den letzten Jahren stark beschleunigte und auch weiterhin stetig anhaltende Entwicklung im Bereich der Computer-Hardware und -Software konsequent zu nutzen. Gleichzeitig stellte die Struktur und Komplexität der Drei-Spektrometer-Anlage an die Experimentdatenverarbeitung erhöhte Anforderungen, die nicht durch einfaches „Hochskalieren“ existierender Soft- bzw. Hardware abgedeckt werden konnten. Ein auf dieser Basis entwickeltes Datenerfassungssystem sollte dann aber umgekehrt ohne weiteres auch für einfachere Fälle einzusetzen sein. Im folgenden wird nun die Drei-Spektrometer-Anlage etwas näher beschrieben und die sich daraus ergebenden Konzequenzen für das Datenerfassungssystem erläutert.

2.2.2 Die Drei-Spektrometer-Anlage

Das Herzstück der Drei-Spektrometer-Anlage bilden drei Magnetspektrometer [Korn95, Scha95]. Diese sind um eine gemeinsame Drehachse, die gleichzeitig den Targetort definiert, auf Fahrgestellen justierbar angeordnet, so daß mit jedem Spektrometer verschiedene Winkelbereiche abgedeckt werden können. Jedes der Spektrometer besteht aus den magnetoptischen Elementen, die die bei kernphysikalischen Prozessen im Target gestreuten oder freigesetzten geladenen Teilchen nach Impulsen getrennt in der Bildebene abbilden, sowie den Detektorsystemen zum Teilchennachweis (siehe Abb. 2.3). Bei speziellen Experimenten kann dieser Aufbau noch durch andere Detektorsysteme wie ein Neutronendetektor [SFB95] oder ein BGO-Ball [A190] erweitert werden.

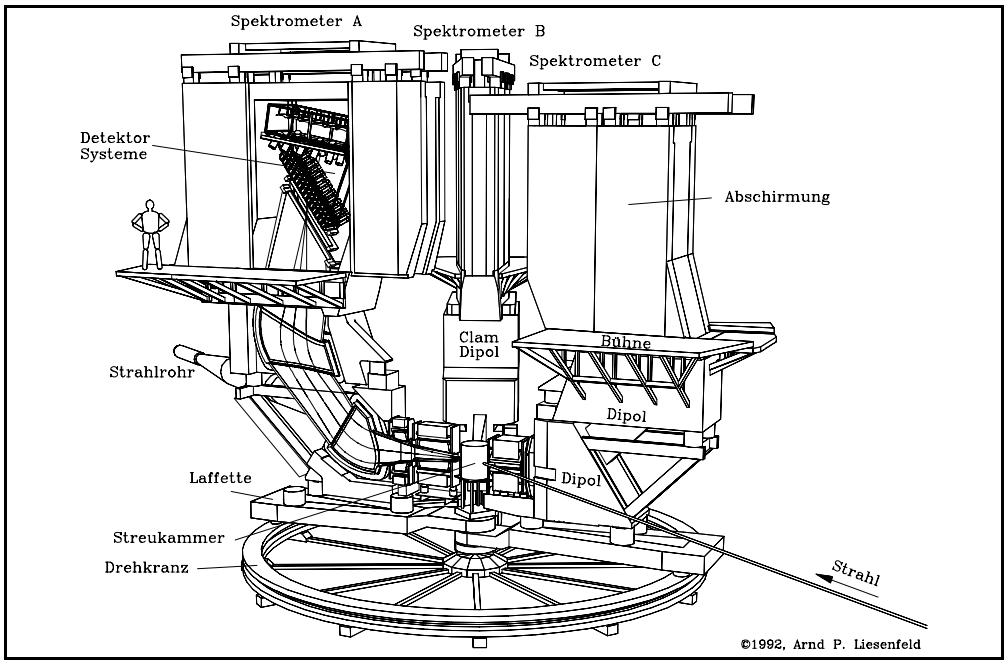

Abb. 2.3: Die Drei-Spektrometer-Anlage [Lies95]

Spektrometer A und C sind sog. QSDD-Spektrometer, d. h., sie bestehen aus einem Quadrupol-, einem Sextupol- und zwei Dipolmagneten, die hintereinander angeordnet sind. Die Dipolmagnete dienen dazu, geladene Teilchen nach ihrem Impuls zu sortieren. Da die vorgesehenen Koinzidenzexperimente in der Regel geringe Wirkungsquerschnitte besitzen und bei der Durchführung der Experimente nur eine begrenzte Strahlzeit zur Verfügung steht, wurde bei der Konzipierung der Spektrometer besonderen Wert auf große Raumwinkel gelegt. Deshalb wird den beiden Dipolmagneten ein Quadrupolmagnet vorangestellt. Zur Kompensation von Abbildungsfehlern zweiter Ordnung dient ein Sextupolmagnet, der zwischen dem Quadrupol- und den Dipolmagneten plaziert ist. Spektrometer B ist ein „clam-shell"-Dipolmagnet, d. h. ein Dipolmagnet, dessen Polschuhplatten nicht parallel, sondern in einem bestimmten Winkel zueinander stehen. Durch diese Konstruktion wird erreicht, daß das Spektrometer sowohl in der dispersi-

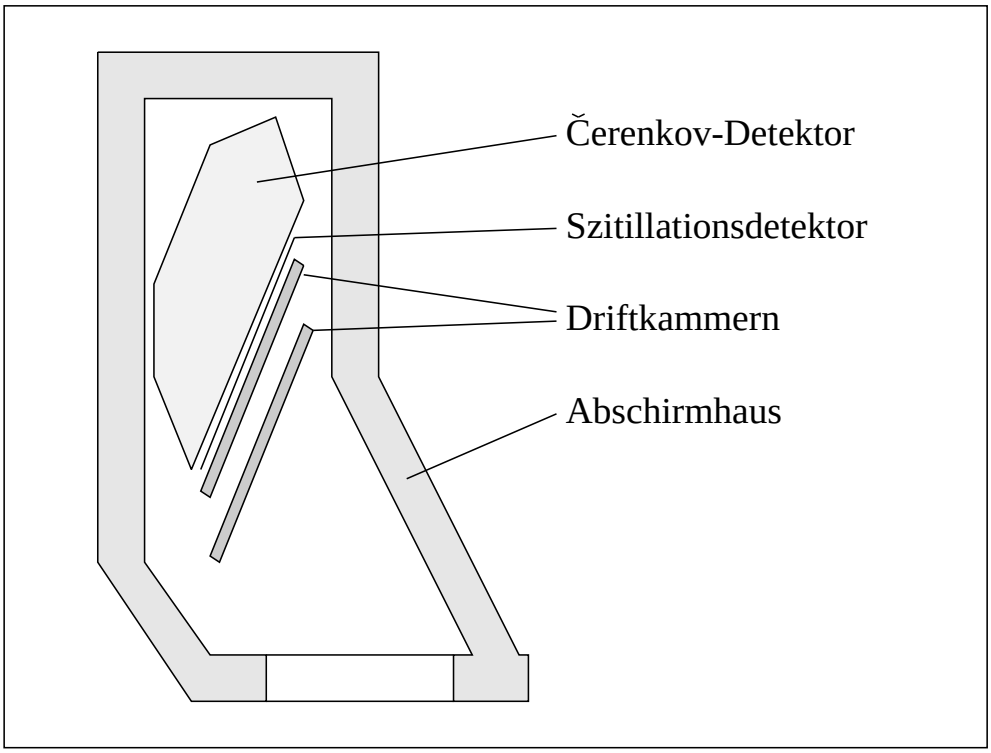

Abb. 2.4: Schematische Darstellung der Detektorsysteme von Spektrometer B. Die Detektorsysteme der beiden anderen Spektrometer sind ganz analog aufgebaut. Sie unterscheiden sich nur in Details durch ihre geometrische Anordnung und Ausmaße.

ven als auch in der nicht-dispersiven Ebene Punkt-zu-Punkt-abbildend ist. Damit wird ein besseres Ortsauflösungsvermögen bei der Verwendung ausgedehnter Targets erreicht.

Die Detektorsysteme (siehe Abb. 2.4) aller drei Spektrometer bestehen jeweils aus zwei Doppel-Driftkammern und den erforderlichen Trigger- und Teilchenidentifizierungsdetektoren. Die Kammern arbeiten nach dem Prinzip der vertikalen Drift [Dist95, Saue95]. Sie erlauben als System von zwei Doppel-Driftkammern die Bestimmung des Durchstoßortes und -winkels jeder Teilchenbahn durch die Bildebene. Aus dieser Information können der Impuls, der Streuwinkel und der Reaktionsort berechnet werden. Zur Erzeugung des Stoppsignals für die Messung der Driftzeit in den Driftkammern, zur Teilchenidentifizierung und zur Untergrundunterdrückung ist jedes Detektorsystem mit zwei Ebenen von Plastikszintillationszählern unmittelbar hinter den Drahtkammern ausgerüstet [Rich95]. Die erste Ebene dient zur Messung des Energieverlustes, die zweite zur Messung des Durchgangszeitpunktes (Flugzeitmessung). Beide Ebenen sind in dispersiver Richtung unterteilt. Die Szintillatorsegmente werden an beiden Seiten an je einen Photomultiplier angekoppelt. Jedes Detektorsystem besitzt zusätzlich einen Gas-Čerenkov-Detektor zur Unterdrückung des Elektronen- bzw. Positronenuntergrundes bei Experimenten, die den Nachweis von Pionen erfordern [Lies95].

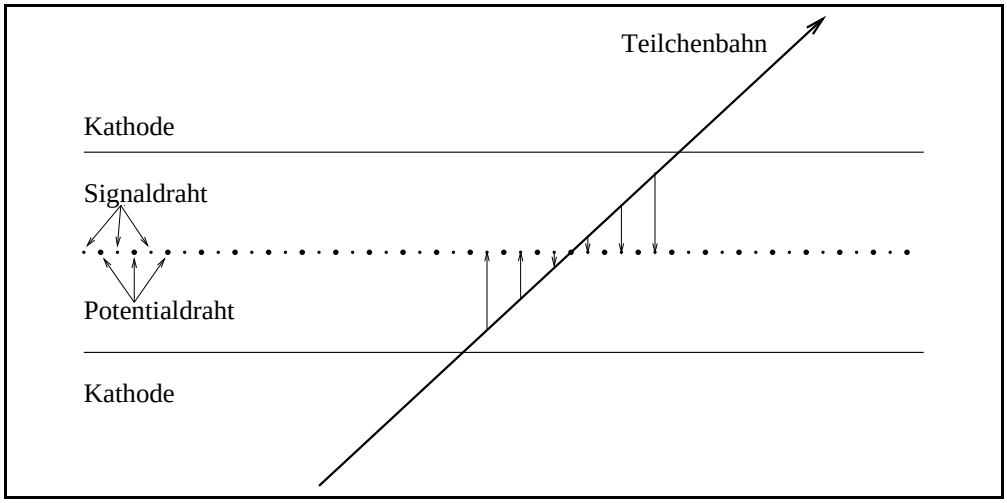

Abb. 2.5: Schematische Darstellung einer vertikalen Driftkammer. Eingezeichnet ist die Durchtrittsbahn eines geladenen Teilchens und in vereinfachter Form die effektiv gemessenen Driftwege

2.2.3 Konsequenzen für die Datenerfassung

Um ein zu untersuchendes Ereignis möglichst gut beschreiben zu können, muß eine Vielzahl von Parametern aufgenommen werden. Für jedes nachgewiesene Teilchen sollte z. B. der Impuls nach Betrag und Richtung am Targetort und die Teilchenart möglichst genau bekannt sein. Da diese Parameter in der Regel nicht direkt bestimmt werden können, ist ein aufwendiger indirekter Meßvorgang notwendig. So wird über die Impuls-Orts-Abbildungseigenschaft der Magnetspektrometer die Impulsmessung auf eine Ortsmessung zurückgeführt, die mithilfe der Driftkammern realisiert wird. Hier wiederum werden die Ortskoordinaten nicht unmittelbar gemessen sondern indirekt durch Bestimmung der Bahnen der nachzuweisenden Teilchen, indem die Zeiten gemessen werden, in denen die Elektronen, die bei der Ionisation des Arbeitsgases beim Durchgang eines geladenen Teilchens freigesetzt wurden, von der Teilchenspur bis zu den Meßdrähten driften (siehe Abb. 2.5).

Da jedes Spektrometer dazu mit zwei Doppeldriftkammern mit jeweils 300–500 Drähten pro Ebene ausgestattet ist, bedeutet das, daß allein zur Impulsmessung etwa 2000 Parameter pro Spektrometer bestimmt werden müssen. Daneben fallen zur Teilchenidentifizierung weitere Daten an. Die zwei 15-fach segmentierten und doppelt ausgelesenen Szintillator-Ebenen liefern 60 Energieinformationen und evtl. zusätzlich noch bis zu 60 Zeitinformationen, die Čerenkov-Detektoren 12 Energiewerte. Je nach Experiment sind bis zu drei Spektrometer und evtl. noch zusätzliche Detektorsysteme beteiligt. Daneben ist es noch notwendig, zusätzliche Informationen aufzunehmen, die z. B. das Target oder den Elektronenstrahl betreffen; das sind u. a. Informationen über den Strahlstrom oder die Strahlposition auf dem Target. Alles in allem wird ein Ereignis durch einige tausend Datenkanäle charakterisiert, deren Auslese und weitere Verarbeitung korrekt und mit genügend großer Geschwindigkeit bewältigt werden muß.

Die Raten, mit denen diese Daten anfallen können, sind infolge des kontinuierlichen Strahlstroms weniger durch den Beschleuniger begrenzt. Sie werden im wesentlichen durch den vom jeweiligen Experiment bestimmten Wirkungsquerschnitt für echte aber auch Untergrundereignisse vorgegeben und werden nicht zuletzt durch die von Detektoren, Elektronik und Rechner verursachte Totzeit beschränkt. Um Strahlzeiten effizient auszunutzen, sollten daher die Hard- und Software-Totzeiten möglichst niedrig sein. Bei zu erwartenden Koinzidenzraten von einigen Hertz bis hin zu einigen 100 Hz bedeutet das, daß die Totzeiten bei der Erfassung eines Ereignisses möglichst kleiner als 1 ms sein sollten. Auch die gleichzeitige Erfassung von Einzel-Ereignissen sollte ohne nachteilige Konsequenzen möglich sein.

2.3 Grundlegende Aufgabenstellung

Die Aufgabe eines Datenerfassungssystems ist es nun, die Daten, die Ereignis für Ereignis von den Detektoren geliefert werden und über die Experimentelektronik dann in digitaler Form zur Verfügung gestellt werden, rechnergesteuert aufzunehmen und entweder direkt auszuwerten oder so aufzubereiten und abzuspeichern, daß eine spätere Rekonstruktion und Auswertung der Ereignisse möglich ist.

Neben der eigentlichen Datenaufnahme, die aus der **Auslese** der Meßelektronik besteht, kann im Rahmen der Datenerfassung eine weitere **Bearbeitung** der Daten (z. B. Offset- oder Eich-Korrekturen), eine **On-line-Reduktion** (z. B. Nullenunterdrückung) oder gar eine **Vorauswertung** der Daten (z. B. Koordinatenberechnung) erfolgen. Auf jeden Fall müssen die Daten, falls sie nicht gleich direkt ausgewertet und in Form von Histogrammen gespeichert werden, so **kodiert** werden, daß ihre spätere Rekonstruktion eindeutig möglich ist. Dabei sollte z. B. auch unterschieden werden, ob ein Parameter den Wert 0 hat oder ungültig ist und daher gar nicht ausgelesen wurde. Die letzte und ebenso wichtige Aufgabe ist die korrekte **Archivierung** der Daten auf einem Massenspeicher einschließlich einer evtl. notwendigen **Datenübertragung**.

Zur Erledigung dieser Aufgaben ist neben der Bereitstellung entsprechender Rechner-Hardware insbesondere die Erstellung von Software notwendig. Die Abbildung 2.6 stellt stark vereinfacht die Grundkomponenten schematisch dar, die an der Datenerfassung beteiligt sind.

2.4 Ergänzende Anforderungen

Neben den Anforderungen, die sich direkt aus den experimentellen Randbedingungen ergeben, sind bei der Konzeption eines Datenerfassungssystems noch eine ganze Reihe weiterer Punkte zu beachten. Hervorzuheben sei dabei die notwendige **Felxibilität**. Schon bei einem Datenerfassungssystem, das nur für die Experimente an der Drei-Spektrometer-Anlage gedacht ist, ist eine feste Kodierung

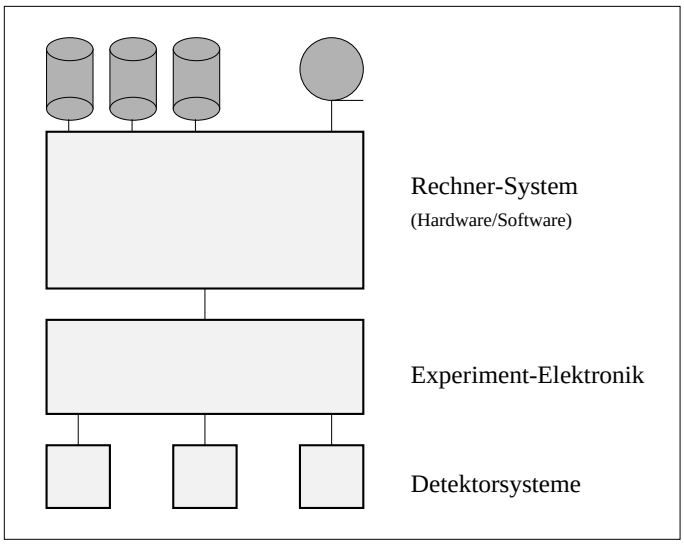

Abb. 2.6: Die an der Datenerfassung beteiligten Grundkomponenten.

der Software für einen bestimmten Experimentaufbau und mit festen Bearbeitungsvorschriften zur Auslese nicht sinnvoll. Die durchzuführenden Experimente sind zwar in Bezug auf die Datenerfassung ziemlich ähnlich, doch in Details können sie sich deutlich voneinander unterscheiden. So muß der Tatsache Rechnung getragen werden, daß in der Regel von Experiment zu Experiment **wechselnde Detektorkombinationen** zum Einsatz kommmen. Wie erwähnt werden nicht allein die fest installierten Spektrometer eingesetzt; sie können je nach Experiment durch weitere, ganz andere Detektoren ergänzt werden. Auch Auslese- und Bearbeitungsvorschriften müssen den aktuellen Gegebenheiten leicht anpaßbar sein. Oftmals ist es sogar notwendig, während eines Experiments im Hard- und Software-Bereich Modifikationen vorzunehmen, um z. B. defekte Elektronik zu ersetzen oder auszublenden oder auf andere Umstände und unvorhersehbare Ereignisse, die beim Experimentierbetrieb auftreten können, zu reagieren.

All das soll ohne eine ständige Neuprogrammierung der Software möglich sein, für die in der Regel tiefergehende Kenntnisse notwendig sind und damit die Fehleranfälligkeit wächst. D. h., das Datenerfassungssystem sollte in weitem Rahmen **konfigurierbar** sein und möglichst einfach an konkrete Gegebenheiten anpaßbar sein. Damit sollte es auch mit nicht allzu großem Aufwand möglich sein, die Datenerfassung für viele andere Experimente, die an MAMI durchgeführt werden, mit diesem System abzuwickeln.

Das Datenerfassungssystem muß in der Lage sein, **unterschiedliche Datenerfassungs-Hardware** zu unterstützen. Für die Experimentierapparaturen an MAMI bedeutet das, daß einerseits Standards wie CAMAC, Fastbus, VME-bus genutzt werden, andererseits aber auch der Einsatz spezieller Elektronik z. B. aus Eigenentwicklung oftmals unumgänglich ist. Im Falle der Drei-Spektrometer-Anlage kommt die genannte Hardware in vollem Umfang sogar gleichzeitig zum Einsatz und muß teilweise auch gegeneinander austauschbar sein. Gerade hierfür sollte kein spezieller Programmieraufwand notwendig werden.

Ein weiterer wichtiger Aspekt war die Notwendigkeit, in der Aufbauphase der Drei-Spektrometer-Anlage möglichst **frühzeitig Unterstützung bei der Entwicklung der unterschiedlichen Detektoren** und deren Tests zu geben. Zum einen ist gerade hier eine hohe Flexibilität notwendig, da in dieser Phase häufig Änderungen im Hardware-Setup vorkommen bzw. Verbesserungen und Anpassungen in den detektornahen Software-Teilen notwendig werden. Andererseits treten hier teilweise neue Anforderungen auf, die für den späteren Experimentierbetrieb weniger von Bedeutung sind. Entwicklung, Aufbau und Test der Detektorteilsysteme wie Driftkammern, Szintillationsdetektoren und Čerenkov-Detektoren wurden bis zu einem bestimmten Punkt unabhängig voneinander bzw. sogar parallel zu laufenden Experimenten mit bereits fertiggestellten Detektoren vorgenommen. Das mußte die Datenerfassungs-Software ebenso leisten wie die Unterstützung beim späteren Zusammenfügen der Einzelkomponenten zu einem Gesamtsystem.

Auch in Zukunft wird das Detektorsystem **ständig erweitert** werden — es ist z. B. vorgesehen, eines der Spektrometer durch ein Polarimeter [A193] zu ergänzen —, so daß diese Anforderungen stets akut bleiben. Desgleichen sollten **zukünftige Änderungen** in den experimentellen Gegebenheiten, die das Anforderungsprofil etwas verschieben könnten (höhere Raten, niedrigere Totzeiten, andere Meßelektronik, intelligente Subsysteme, Software-Trigger), mit vertretbarem Aufwand zu berücksichtigen sein, ohne daß dann eine komplette Neuprogrammierung der Software notwendig wird.

Die Erfahrungen aus der Vergangenheit haben gezeigt, daß eine **Gesamtdokumentation** einer Messung in den archivierten Daten unverzichtbar ist. Nach Möglichkeit sollte ein Meßdatenfile **selbstbeschreibend** sein. Bei der späteren Auswertung der Daten sollte keine zusätzliche Information notwendig sein, um eine vollständige Rekonstruktion der Meßdaten vornehmen zu können.

Eine sehr wichtige Anforderung an das Datenerfassungssystem ist eine **einfache Bedienbarkeit**, die keine Spezialkenntnisse voraussetzt. Während in der Aufbauphase noch genügend Personen mit weitergehenden Kenntnissen verfügbar waren, muß das System in Zukunft von Nicht-Experten konfiguriert und bedient werden können.

Nicht zuletzt sei auch auf Aspekte der Software-Entwicklung selbst hingewiesen. Neben einer **einfachen Programmierbarkeit** mithilfe geeigneter Methoden ist auf eine **gute Wartbarkeit** der Software zu achten, um auch in Zukunft den einwandfreien Betrieb zu gewährleisten, insbesondere falls doch einmal tiefergehende Eingriffe in die Software notwendig werden sollten. Ein weiterer wichtiger Punkt ist **Portierbarkeit** der Software. Die Software sollte z. B. keine speziellen Eigenschaften bestimmter Betriebssystem- oder Compiler-Versionen ausnutzen; vielmehr sollte möglichst von Standards Gebrauch gemacht werden. Auch sollte eine möglichst große Unabhängigkeit von der verwendeten Rechner-Hardware erreicht werden, um z. B. sich ändernden Anforderungen in bezug auf die geforderte Rechenleistung gut begegnen zu können und die Entwicklungen im Bereich der Computer-Hardware zu nutzen.

Kapitel 3

Grundlagen

Ziel dieser Arbeit war es, ein Datenerfassungssystem auf der Basis der im vorhergehenden Kapitel beschriebenen Anforderungen zu realisieren. Bevor jedoch näher auf die konkrete Realisierung eingegangen wird, sollen einige grundlegende Probleme erläutert werden, denn nur wenige Anforderungen implizieren klare Lösungsansätze; einige stehen sogar im Widerspruch zueinander. Bei der Konzeption des Systems hatte daher stets eine Abwägung zu erfolgen, wo die Prioritäten zu setzen waren, und welche Lösungsalternativen gewählt werden sollten. Dabei reichte es nicht aus, sich auf isolierte Details z. B. der Programmierung von Anwendungs-Software zu beschränken. Eine befriedigende Lösung konnte nur durch geeignete Realisierung des Gesamtkomplexes gefunden werden, der sich aus den Bestandteilen

- Experiment-Hardware
- Rechner-Hardware
- Systemsoftware
- problemorientierte Software

zusammensetzt. Während im Hardwarebereich i. w. auf kommerzielle Lösungen zurückgegriffen werden konnte, mußte die Systemsoftware in Teilen noch durch eigene Implementationen ergänzt werden. Auch für die Organisation des Rechnersystems mußten geeignete Konzepte entwickelt und umgesetzt werden.

3.1 Hardware

3.1.1 Die Experimentelektronik

Grundvoraussetzung für eine gut funktionierende Datenerfassung ist natürlich eine Experimentelektronik, die einerseits einen optimalen Betrieb der Detektoren gewährleistet, andererseits sich gut durch die dafür zuständigen Rechner ansteuern läßt. Im kernphysikalischen Bereich existieren hierzu bewährte Standards wie

CAMAC und Fastbus, die bereits in der Vergangenheit im Institut eingesetzt wurden und auch bei den neuen Detektorsystemen zum Einsatz kommen.

Im Falle der Drei-Spektrometer-Anlage wurde eine Mischkonfiguration gewählt, die einen guten Kompromiß in Bezug auf Leistung, Preis und Verfügbarkeit ergab. Hier kommen als Ergänzung zu CAMAC und Fastbus auch Elektronik-Eigenentwicklungen und — insbesondere für Zwecke der Experimentsteuerung — VMEbus-Hardware zum Einsatz. U.a. konnte die Verfügbarkeit intelligenter Auslesesysteme ausgenutzt werden. Bei Fastbus sind das z. B. ADC-Module [Lecr2], die für die Digitalisierung der Energierinformation der Triggerdetektoren eingesetzt werden. Über Pedestal-Subtraction[1] und Nullen-Unterdrückung reduzieren sie die vom Rechner auszulesenden Informationen auf weniger als ein Zehntel der ursprünglichen Datenmenge. Noch extremer sind die Verhältnisse bei den Driftkammern. Hier können immer nur die Drähte sinnvolle Signale liefern, die in unmittelbarer Nähe der Teilchenspur liegen (siehe auch Abb. 2.5). Das sind in der Regel pro Ebene 5–8 Drähte, also bei jedem Spektrometer insgesamt etwa 30 der bis zu 2000 Drähte. Um auch hier den Ausleseaufwand erheblich zu reduzieren, werden zur Messung der Driftzeiten ebenfalls intelligente Systeme eingesetzt. In der Regel findet das kommerzielle TCD Readout-System LeCroy 4299 [Lecr3] Verwendung. Es erkennt die TDCs, bei denen keine Zeitkonversion vorgenommen wurde bzw. ein Überlauf stattfand, und nimmt damit automatisch nur die zu den angesprochenen Drähten gehörigen TDC-Werte auf. Als leistungsfähige und kostengünstige Alternative dazu wurde im Institut ein neues TDC-System auf der Basis einer TDC-Chip-Eigenentwicklung [Geig93a] konzipiert und aufgebaut, das bei Spektrometer C zum Einsatz kommt [Claw95]. Es hat, ebenso wie das etablierte System, die Möglichkeit, nutzlose Daten auszusortieren, bietet aber wegen der Verwendung eines Transputer-Netzwerkes weitaus mehr Möglichkeiten, die Daten zu manipulieren und weiterzuverarbeiten. Es sollte sogar in der Lage sein, Teilaufgaben der herkömmlichen Datenerfassung zu übernehmen.

Die Verwendung solch heterogener Experiment-Hardware hat natürlich auch Konsequenzen für das Datenerfassungssystem. Es muß die Besonderheiten der verschiedenen Systeme beherrschen, aber auch in der Lage sein, Aufgaben, die in einem Fall von einem intelligenten Subsystem übernommen werden, in einem anderen Fall selbst per Software auszuführen.

3.1.2 Die Rechner-Hardware

Bei den im Rahmen der Datenerfassung anfallenden Aufgaben ist es sinnvoll, grundsätzlich zwischen zwei Gruppen zu unterscheiden. Zum einen sind das die hardwarenahen Aufgaben, wo es um das ereignisweise Auslesen und Ansteuern der Experimentelektronik mit Geschwindigkeiten im Bereich von einigen Mikrosekunden pro Zugriff geht; zum anderen treten aber auch übergeordnete Aufgaben im Bereich der Experimentkontrolle und On-line-Analyse auf, wo hohe Anforderungen u. a. an Bedienungskomfort, Rechenleistung und Arbeits- bzw.

[1] automatische Subtraktion eines Offsets vom eigentlichen Meßwert

Massenspeicher gestellt werden. Das mußte auch bei den eingesetzten Rechnern berücksichtigt werden. Es erwies sich daher als angebracht, keine teuren Universalrechner einzusetzen, die all diese Aufgaben, wenn überhaupt, mehr oder weniger befriedigend ausführen können, sondern vielmehr für jedes Aufgabengebiet solche Computer-Systeme, die auf die jeweiligen Anforderungen ausgerichtet sind. Schon in der Vergangenheit wurden in unserem Institut mit dieser Vorgehensweise gute Erfahrungen gemacht [Voge84, Kryg86, Stol88]. Während früher jedoch in der Regel spezielle Prozeßrechner und Minicomputer für diese Einsatzzwecke mit größerem Aufwand aufgerüstet werden mußten, steht seit geraumer Zeit ein wachsendes Angebot kommerziell eingesetzter Rechnersysteme für die verschiedenen Aufgabengebiete zur Verfügung. Hier konnte davon profitiert werden, daß in vielen anderen Bereichen der Datenverarbeitung auch außerhalb der Naturwissenschaften solche auf dezentralen Konzepten basierende Verfahren immer stärker und mit großem Erfolg eingesetzt werden.

3.1.2.1 Einsatz von VMEbus im Frontend-Bereich

Im Frontend-Bereich, d. h. dort, wo die direkte Ansteuerung der Experimentelektronik vorgenommen wird, fallen die typischen Aufgaben der Prozeßdatenverarbeitung [Bolc93] an. Hier konnten die im industriellen Bereich etablierten VMEbus-Systeme (siehe auch Anhang A.1) sehr gut als Prozeßrechner eingesetzt werden. Sie erfüllen folgende Aufgaben der Experimentdatenverarbeitung:

- schnelle Reaktion auf externe Ereignisse, d. h. Starten der Datenaufnahme innerhalb weniger Mikrosekunden nach Registrierung des Trigger-Signals aber auch schnellst mögliche Reaktion auf eine Fehlfunktion der Apparatur
- Aufnahme und Verarbeitung von Analog- u. Digitalinformation der Detektoren, also evtl. selektives Einlesen von Energie- und Zeitinformationen der Triggerdetektoren und Driftkammern oder Angaben über den Strahlstrom
- Steuerung und Überwachung der einzelnen Komponenten der Detektorsysteme wie Hochspannungen von Photomultipliern und Driftkammern, Ströme der Spektrometermagnete oder Position des Targets
- dazu notwendiger direkter und schneller Zugriff auf die Experimentelektronik, also auf ADCs, TDCs, Zähler und andere Module in CAMAC und Fastbus oder anderen Subsystemen mit Zugriffszeiten, die bei der eingesetzten Hardware typischerweise bis in den Mikrosekunden-Bereich reichen

Im Institut für Kernphysik wird der VMEbus seit Mitte der achtziger Jahre erfolgreich eingesetzt. Als Herzstück der Systeme standen ursprünglich nur Motorola-basierte CPUs zur Verfügung. Es wurden CPU-Karten der Mainzer Firma Eltec benutzt, einem Marktführer im VMEbus-Bereich, mit dem das Institut über längere Zeit in enger Zusammenarbeit stand. Die eingesetzten Eurocom-Karten E3, E5, E6 und E7 nutzten die Aufwärtskompatibilität der MC-68000-CPU-Familie, so daß über einen längeren Zeitraum hinweg eine kontinuierliche Soft- und Hardware-Entwicklung bei gleichzeitigem Einsatz neuster

Technologie sichergestellt war. Seit geraumer Zeit gibt es atraktive Alternativen, etwa CPU-Karten auf der Basis von RISC[2]-Prozessoren, die eine gesteigerte Rechenleistung, größeren Arbeitsspeicher und leistungsfähigere I/O-Schnittstellen mit verbesserter Software kombinieren. Sie werden die Eltec-CPUs im Institut in Zukunft ergänzen und langfristig sogar ganz ersetzen. Die Herstellerunabhängigkeit des VMEbus-Standards gewährleistet dabei einen fließenden Übergang.

3.1.2.2 Workstations als übergeordnete Rechner

Der zweite Aufgabenbereich im Rahmen der Experimentdatenverarbeitung umfaßt i. w. übergeordnete Aufgaben. Dazu gehören

- Steuerung und Kontrolle des Experiments und der Apparatur,
- Archivierung und Verwaltung der Meßdaten,
- On-line- und Off-line-Analyse,
- Bereitstellung einer komfortablen Benutzungsoberfläche.

In der Vergangenheit wurden einige der beschriebenen Aufgaben entweder durch den für die Datenerfassung zuständigen Prozeßrechner selbst oder aber daran gekoppelte Minicomputer erledigt. Da aber inzwischen die Anforderungen stark gestiegen sind, reicht in vielen Fällen der Prozeßrechner nicht mehr aus; den Platz der Minicomputer nehmen inzwischen mehr und mehr Workstations ein, die in Rechenleistung und Ausstattung gut mit früheren Großrechnern zu vergleichen sind, darüber hinaus aber noch einige Vorzüge aufzuweisen haben, die sie gerade für den Einsatz im Experiment interessant machen (s. Anhang A.2).

Im Institut für Kernphysik wurden Workstations für eine Vielzahl von Aufgaben bereits frühzeitig eingesetzt. Insbesondere in der näheren Vergangenheit erlaubte ihr im Verhältnis zu konventionellen Minicomputern und Großrechnern bei vergleichbarer Leistung extrem günstiger Preis, sie auch in größerem Ausmaß zu nutzen. Sie wurden für verschiedene Zwecke in einer größeren Anzahl von Form sogenannter Workstation-Cluster installiert.

3.1.3 Erste Schlußfolgerungen

Unter Berücksichtigung der bisher gemachten Überlegungen lassen sich nun die für die Datenerfassung notwendigen Hardware-Komponenten so, wie in Abbildung 3.1 gezeigt, darstellen. Die Abbildung gibt die bereits in Abb. 2.6 gezeigten Hardware-Struktur in verfeinerter Form wieder und konkretisiert die dort nur grob gekennzeichneten Elemente. Sie versucht dabei einerseits noch einmal die sinnvolle Aufspaltung des Rechnersystems in übergeordnete Rechner und Frontendrechner deutlich zu machen, andererseits die enge Ankopplung der Experimentelektronik an das Rechnersystem auf der Basis des VMEbus.

[2]Reduced Instruction Set Computer

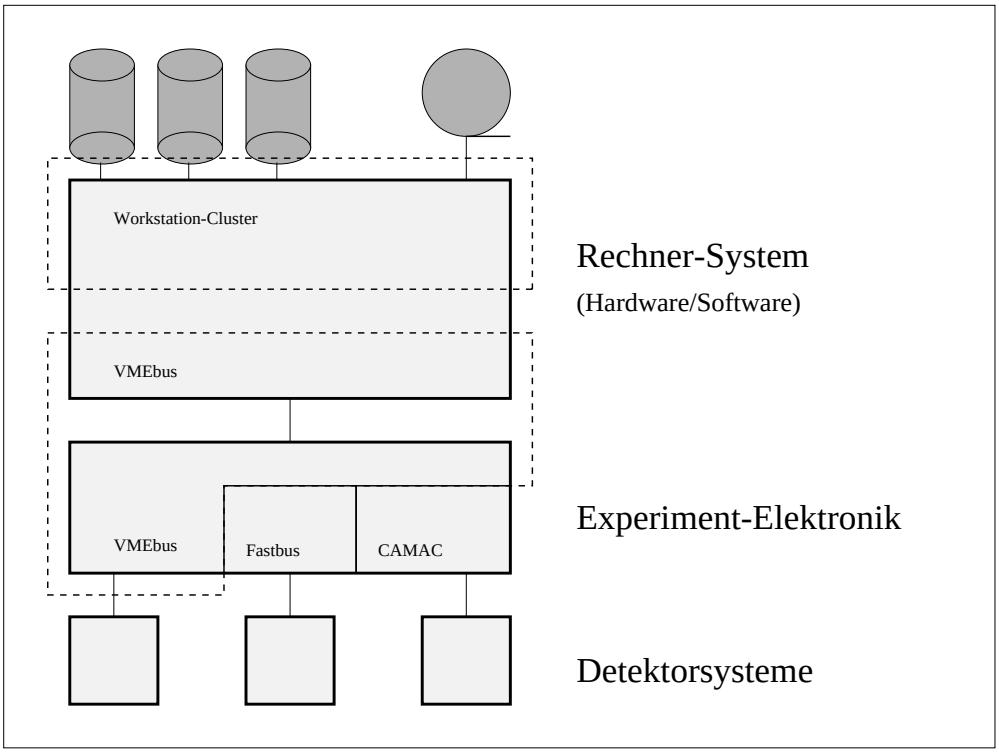

Abb. 3.1: Schematische Darstellung der Hardware-Komponenten, die an der Datenerfassung beteiligt sind, in verfeinerter Form. Vgl. auch Abb. 2.6.

3.2 Software

Neben der Hardware ist natürlich auch Software notwendig, um die schon erwähnten Aufgaben wie Datenaufnahme, Reduktion, Formatierung und Archivierung der Meßdaten, die Bedienung des Systems und On- bzw. Off-line-Analyse zu erfüllen. Unmittelbar damit verbunden sind aber noch weitergehende Anforderungen, die z. B. im Rahmen von evtl. notwendiger Programmentwicklung anfallen. So ist die Verfügbarkeit geeigneter Editoren, Compiler, Debugger und eine ganze Reihe weiterer Programmierhilfsmittel entscheidend, um die eigentlichen Aufgaben zufriedenstellend realisieren zu können. Auch Systemsoftware, die die Hardware bestmöglich unterstützt und an experimentelle Randbedingungen gut angepaßt ist, ist für einen späteren einwandfreien Betrieb unabdingbar.

Umfang und Komplexität der Anforderungen ließen es von Anfang an für sinnvoll erscheinen, zu ihrer Lösung möglichst auch auf bereits Existierendes zurückzugreifen und so weit wie möglich verfügbare Standard-Software auszunutzen. In einigen Fällen bestand die Möglichkeit, für grundlegende Aufgaben kommerzielle Software einzusetzen. Das betrifft insbesondere den Bereich der Betriebssysteme und anderer Systemsoftware, die sehr eng mit der eingesetzten Rechner-Hardware verknüpft ist. Aber auch für viele andere Bereiche wie Graphik- und Datenbankanwendungen können kommerzielle Softwarepakete eine sinnvolle Lösungsalternative bieten.

Eine besondere Grundlage bot auch zu kommerziellen Produkten oftmals gleichrangige oder sogar überlegene Software aus dem Public Domain-Bereich[3], wo inzwischen viele, recht professionelle Pakete existieren. Neben der Tatsache, daß solche Software kostenlos ist, konnte zudem der Vorteil genutzt werden, daß sie infolge der Verfügbarkeit der Quellen auch sehr leicht an spezielle Randbedingungen angepaßt werden kann. Das gleiche gilt in der Regel auch für Software aus anderen wissenschaftlichen Forschungseinrichtungen wie CERN, MIT, oder GSI, die in großem Maße im Institut für Kernphysik in allen Bereichen genutzt wird. Während aber herkömmliche PD-Software oftmals hilfreich bei der Lösung allgemeiner Probleme war, bot solche Software konkrete Lösungsansätze für spezielle Probleme im Bereich der Experimentdatenverarbeitung. Dennoch war es notwendig, mehr oder weniger große Teile der Software selbst zu programmieren, um ein möglichst gut an die konkreten Gegebenheiten angepaßtes Datenerfassungssystem zu realisieren.

3.2.1 Wahl des Betriebssystems

3.2.1.1 Systemsoftware

Wesentliche Teile der für dieses Projekt benötigten Software sind erst einmal unabhängig von der konkreten Problemstellung. Dazu gehört in erster Linie Systemsoftware, die den Betrieb mit der eingesetzten Rechner-Hardware sicherstellt. Je nach Komplexität des Rechnersystems und der zu bearbeitenden Aufgaben kann die Systemsoftware in Form einfacher Unterprogramme aber auch durch ein mehr oder weniger umfangreiches Betriebssystem realisiert sein (siehe auch Anhang A.4).

Im Falle der Experimentdatenverarbeitung können viele Aufgaben bereits durch geeignet gewählte Systemsoftware abgedeckt werden. Dazu gehören neben Routineaufgaben insbesondere ein standardisierter I/O unter Ausnutzung einheitlicher Software-Schnittstellen. Gerade auch die Möglichkeiten von Multitasking, also der Fähigkeit des Betriebssystems, mehrere Programme quasi gleichzeitig auszuführen, oder gar Multiprocessing, bei dem mehrerere Prozessoren parallel eingesetzt werden, sind sehr hilfreich, um mehrere, parallel anfallende Aufgaben, gleichzeitig zu bearbeiten.

Allein im Bereich der Datenerfassung bietet sich für die Vielzahl und Komplexität der Aufgaben eine echte oder zumindest quasi parallele Verarbeitung an. Auslese, Kodieren, Puffern, Transfer und Archivierung der Daten und ihre evtl. on-line notwendige weitere Verarbeitung können natürlich ohne weiteres sequentiell erfolgen. Das bedeutet, daß die beteiligten Rechner nur dann, wenn ein Ereignis registriert wurde, aktiv werden und die Aufgaben streng nacheinander ausführen müssen, den Rest der Zeit aber nichts tun. Damit wird die Rechnertotzeit durch die Summe aller Bearbeitungszeiten bestimmt. Im Idealfall müssen jedoch synchron zum Ereignisnachweis nur die direkt mit der Auslese

[3]Der Begriff Public Domain (PD) Software steht hier allgemein für freie Software. Darunter versteht man Software, die zwar meist durch Copyrights geschützt ist, aber ohne Zahlung von Lizenzgebühren genutzt werden kann und in der Regel auch als Quellcode zur Verfügung steht.

der Meßelektronik verbundenen Vorgänge bearbeitet und eine Pufferung minimal kodierter Daten vorgenommen werden. Die Totzeit wird dann maßgeblich durch die für die Datenaufnahme direkt notwendigen Aktivitäten bestimmt, alles andere kann asynchron dazu abgearbeitet werden in den Zeiten, in denen der Rechner auf das nächste Ereignis wartet.

Durch eine geschickte zeitliche Verteilung der Aufgaben könnte also eine wesentlich gleichmäßigere Auslastung des Computersystems erreicht werden, was nicht nur eine wirtschaftlichere Ausnutzung der Rechner bedeuten würde, sondern insgesamt die Leistungsfähigkeit des Systems steigern könnte. Entscheidend dafür ist dann nicht mehr die Bewältigung höchster Belastungsspitzen über kurze Zeiten hinweg, sondern eine im zeitlichen Mittel hohe Leistung.

Während Multitasking bei der Datenaufnahme also ganz sinnvoll sein kann, ist es anderen Bereichen sogar unentbehrlich. So muß das Experiment ständig überwacht werden, die Datenerfassung muß gesteuert werden und der Benutzer muß in der Lage sein, zu jedem Zeitpunkt korrigierend einzugreifen. Auch für die Steuerung, Regelung und Überwachung der Meßapparatur sind parallele Aktivitäten in großem Umfang nötig [Kund95]. Hier kann ein leistungsfähiges Multitasking-Betriebssystem eine wertvolle Hilfe sowohl bei der Realisierung des Datenerfassungssystems als auch bei seinem späteren Betrieb sein. Der Einsatz eines bestimmten Betriebssystems entscheidet aber auch mit bei der Auswahl der Rechnersysteme, denn leistungsfähige Systemsoftware stellt oft auch nicht zu vernachlässigende Ansprüche an die Hardware. So sind oftmals die Größe des verfügbaren Arbeitsspeichers, Hardware-Unterstützung bei der Speicherverwaltung (Memory Management Unit), die Möglichkeit, evtl. Massenspeicher anschließen zu können, oder die Verfügbarkeit von Netzwerk-Schnittstellen wichtige Entscheidungskriterien.

Mit dem Einsatz von Betriebssystemen können aber auch Nachteile verbunden sein. Im Gegensatz zu einer an Hardware und Problemstellung bestmöglich angepaßten Programmierung bedeutet der Einsatz allgemein gehaltener, standardisierter Software mit automatisch, ohne direkte Benutzereinwirkung ablaufenden Vorgängen stets ein Zusatzaufwand, der sich in erhöhtem Rechenzeitbedarf niederschlägt. Die Bedeutung dieses Effekts ist jedoch abhängig vom eingesetzten Betriebssystem ebenso wie von der zu bearbeitenden Aufgabenstellung. Auch Prozeßumschaltungen im Multitaskingbetrieb kosten Rechenzeit und führen zudem bei herkömmlichen Betriebssystemen zu einem nicht deterministischen zeitlichen Verhalten von Programmen. Gleichzeitig können sich die Reaktionszeiten auf externe Ereignisse von wenigen Mikrosekunden, die durch die Hardware verursacht werden, bis in den Millisekundenbereich erhöhen. Ähnlich wie die oftmals fehlende Möglichkeit, aus Anwenderprogrammen direkt auf Hardware zugreifen zu können, schränkt das ihren Einsatz für die Experimentdatenverarbeitung stark ein.

Bei der Konzeption des Datenerfassungssystems war es also sehr wichtig, wesentliche Aspekte in Bezug auf die einzusetzende Betriebssystem-Software zu untersuchen. Es zeigte sich, daß ebenso wie bei der Hardware eine Differenzierung für die unterschiedlichen Aufgabenbereiche sinnvoll oder sogar notwendig

war. Viele der oben beschriebenen parallel zu bearbeitenden Aufgaben fallen bereits im Frontend-Bereich an. D. h., schon hier wird Systemsoftware mit guten Multitasking-Eigenschaften benötigt. Gleichzeitig werden aber auch höchste Ansprüche an ihre Echtzeitfähigkeit gestellt.

3.2.1.2 Das Betriebssystem OS-9

Zu Beginn der Entwicklung des Datenerfassungssystems in den Jahren 1986/87 [Kund88] war das Angebot an leistungsfähigen Echtzeitbetriebssystemen noch sehr eingeschränkt. Im wesentlichen bot im VMEbus-Bereich nur das Betriebssystem OS-9/68000 [Micr87] befriedigende Eigenschaften, die neben Multitasking und Echtzeitfähigkeit auch eine halbwegs komfortable Software-Entwicklungsumgebung einschlossen (s. Anhang A.5).

OS-9 wurde über längere Zeit im Institut für Kernphysik eingesetzt und bot die Plattform für eine erste Implementation des Datenerfassungssystems. Mit der Zeit zeigte sich jedoch, daß weder die Echtzeitfähigkeiten in Bezug auf absolute und maximale Größe der Interrupt-Reaktionszeit ausreichten noch Qualität und Funktionalität der Systemsoftware den Ansprüchen an ein komfortables Entwicklungssystem genügten. Im Vergleich zu dem Betriebssystem UNIX, das inzwischen ebenfalls auf der eingesetzten Rechner-Hardware verfügbar war [Stol87], bot OS-9 nur sehr unvollständige Möglichkeiten und eine wesentlich geringere Leistung in Bereich von Platten- und Netzwerk-I/O. Auch das Leistungspotential der Prozessoren selbst wurde nicht voll ausgeschöpft.

Trotz aufwendiger Portierung von UNIX-Software auf OS-9 blieb ein deutliches Manko im Bereich von Dienstprogrammen, was besonders durch nicht mehr zeitgemäße Compiler und andere Hilfsprogramme zur Software-Entwicklung verschärft wurde. Auch wurde mit der Zeit deutlich, daß eine parallele Entwicklung von Software auf den VMEbus-Rechnern unter OS-9 und auf anderen für die Datenerfassung eingesetzten Rechnern mit unterschiedlichem Betriebssystemen nur mit starker Programmierdisziplin und hohem Aufwand möglich war. Es traten grundsätzliche Inkompatibilitäten zutage, die sowohl bei der Programmentwicklung als auch beim Einsatz eine ständige Fehlerquelle darstellten.

In diesem Zusammenhang erwies es sich auch als ein deutlicher Nachteil, daß OS-9 ausschließlich die MC 68000-Prozessorfamilie unterstützte und auch in diesem Bereich nur auf einer begrenzten Zahl von Rechnern — im wesentlichen VMEbus-CPUs bestimmter Hersteller — verfügbar war. OS-9 konnte weder für Workstations oder andere Prozeßrechner eingesetzt werden, noch bot es im VMEbus-Bereich die Möglichkeit für einen späteren nahtlosen Übergang auf andere Prozessorarchitekturen.

3.2.1.3 Das Betriebssystem UNIX

Neben OS-9 war für die im Institut eingesetzten VMEbus-Rechner auch das Betriebssystem UNIX schon frühzeitig verfügbar [Stol87]. Es stellte zwar etwas höhere Anforderungen an die Rechner-Hardware, bot jedoch gegenüber OS-9

eine Reihe von Vorteilen (s. Anhang A.6). Im Institut für Kernphysik existierten mit dem Betriebssystem UNIX schon langjährige Erfahrungen. Es wurde bereits seit 1984 erfolgreich bei Experimenten an LINAC und MAMI eingesetzt [Kryg86, Merl83]. Hier wurde es auf einem Prozeßrechner im Rahmen von Experimentsteuerung und Datenerfassung verwendet. Später wurde UNIX auf Minicomputern und VMEbus-Rechnern mehr für Zwecke der Software-Entwicklung benutzt. Mit der breiten Verfügbarkeit von RISC-Workstations schließlich etablierte sich UNIX zum Standard-Betriebssystem auch im Institut für Kernphysik.

3.2.1.4 UNIX und Echtzeit

Neben den vielfältigen nützlichen Eigenschaften zeigt UNIX in seiner Originalform einen gravierenden Schwachpunkt auf dem Gebiet der Prozeßdatenverarbeitung: UNIX ist in erster Linie als ein Mehrbenutzersystem für den Time Sharing-Betrieb konzipiert worden, das darauf ausgerichtet ist, die verfügbaren Ressourcen möglichst fair den parallel arbeitenden Benutzern bzw. Programmen zuzuteilen. Insbesondere sorgen die Scheduling-Mechanismen (s. Anhang A.4) für eine gleichmäßige Zuteilung der CPU-Zeit auf die einzelnen Prozesse und lassen nur in geringem Maße die schnelle Reaktion auf externe Ereignisse und ein deterministisches Zeitverhalten zu. Damit besaßen UNIX-Systeme ursprünglich ein sehr schlechtes Echtzeitverhalten. Auch die UNIX-Varianten, die für die schließlich eingesetzten VMEbus-Rechner verfügbar waren, hatten diesen Nachteil und machten das Betriebssystem erst einmal für die Datenerfassung ungeeignet. Als sich jedoch zeigte, daß auch bei speziellen Echtzeitbetriebssystemen wie OS-9 gravierende Schwächen auftreten können, wurde eine Lösung auf der Basis von UNIX wieder in Betracht gezogen. Während bei OS-9 mit vielen Schwierigkeiten zu kämpfen war, mußte bei UNIX lediglich das, wenn auch schwerwiegende, Echtzeitproblem gelöst werden.

Hierfür gibt es nun mehrere Lösungsansätze, von denen einige mit der Verfügbarkeit UNIX-kompatibler Echtzeitbetriebssysteme wie LynxOs [Lynx92] bzw. spezieller Realtime-Kernels wie z.B. VxWorks [Wind93] erst seit kurzer Zeit in Frage kommen und eher für zukünftige Weiterentwicklungen interessant sein könnten. Aber auch hier ist stets die Gefahr vorhanden, daß die Systeme den Anforderungen im konkreten Einsatz dann doch nicht genügen. Eine zweite Möglichkeit wäre die Eigenimplementation eines auf genau diese Anforderungen optimierten Echtzeitsystems, das ansonsten die positiven Eigenschaften von UNIX zeigt. Doch das scheidet wegen des damit verbundenen hohen Aufwands bei gleichzeitig nur beschränktem Nutzen im Rahmen der Entwicklungen für die Drei-Spektrometer-Anlage von vornherein aus.

Eine dritte Lösungsmöglichkeit wird bei einer weiter ins Detail gehenden Analyse der Problemstellung offensichtlich. Es zeigt sich zwar, daß im Rahmen der Datenerfassung viele Aufgaben parallel bearbeitet werden müssen, hohe Systemanforderungen gestellt werden und Echtzeitanforderungen bestehen, doch muß das nicht zwangsläufig heißen, daß das alles von einem einzigen System gleichermaßen gut erfüllt werden muß. Es wurde ja bereits an früherer Stelle

eine Differenzierung gemacht, wo zwischen Frontend und dem restlichen Bereich unterschieden wurde. Dabei wird deutlich, daß nur im Frontend-Bereich wirkliche Echtzeitanforderungen bestehen, der Rest aber ganz gut durch herkömmliche Multitasking-Systeme wie UNIX abgedeckt werden kann.

Aber auch im Frontend-Bereich ist selbst noch einmal eine Differenzierung angebracht; hier kann man nach harten und weichen Echtzeitanforderungen[4] unterscheiden. Es zeigt sich, daß harte Echtzeitanforderungen nur im Zusammenhang mit der Datenerfassung anfallen, und dort auch nur in einem beschränkten Teilbereich. Hier sind jedoch die restlichen Systemanforderungen so gering, daß für die Bearbeitung der betreffenden Aufgaben sogar ganz auf ein Betriebssystem verzichtet werden kann. Gleichzeitig bleiben jedoch auch weiterhin Aufgaben mit hohen Anforderungen an die Systemsoftware bestehen.

Diese Diskrepanz läßt sich beseitigen, wenn bereits im Frontend-Bereich unterschiedliche Computer für die verschiedenen Aufgabengebiete vorgesehen werden: Während spezielle Rechner mit minimaler Systemsoftware ausschließlich für die harten Echtzeitaufgaben — also die schnelle Reaktion auf das Trigger-Signal mit anschließender Datenaufnahme — zuständig sind, übernehmen andere die Bearbeitung der restlichen, weniger zeitkritischen Aufgaben unter Kontrolle eines leistungsfähigen, aber herkömmlichen Multitasking-Systems; hier wiederum eignet sich UNIX ohne Einschränkungen, dessen gute Programmierumgebung auch als Entwicklungsplattform für die Echtzeitrechner genutzt werden kann. Gleichzeitig bietet ein solcher Rechner eine sehr gute Grundlage, um die anderen zu steuern und bei ihrer Arbeit zu unterstützen. Als sog. Master hat er die volle Kontrolle über einen oder mehrere sog. Slave-Rechner.

Der als Basis für das Frontend-Rechnersystem eingesetzte VMEbus bietet im Gegensatz zu vielen anderen Bussystemen die Möglichkeit, gleichzeitig mehrere CPUs einzusetzen, die über einfache Mechanismen miteinander kommunizieren können. Er erlaubt so die einfache Realisierung eines Multiprozessorsystems. Gleichzeitig gewährleistet er von allen CPUs aus den direkten Hardware-Zugang, der nicht nur für die Datenaufnahme benötigt wird, sondern auch die Grundvoraussetzung ist für viele weitere Aufgaben z. B. zur Steuerung der Meßapparatur.

Herzstück eines VMEbus-Rechners ist in der Regel eine Einschubkarte, die alle zentralen Elemente eines Computer-Systems wie CPU, Arbeitsspeicher, einfache I/O-Interfaces und meist auch Massenspeicher- und Netzwerkschnittstellen enthält. Damit bildet eine solche CPU-Karte schon einen eigenständigen Computer. Mehrere solcher Karten können unter Beachtung einfacher Randbedingungen durch Einstecken in einen Überrahmen installiert werden und bekommen so Zugang zu einem gemeinsamen Bussystem, über das sie nicht nur auf die Hardware zugreifen können, sondern auch untereinander kommunizieren können. Sie sind aber ansonsten unabhängig voneinander. Es ist daher sehr einfach, sowohl CPU-Karten als auch notwendige Systemsoftware jeweils abgestimmt auf die unterschiedlichen Anforderungen zu wählen.

[4] Harte Echzeitanforderungen liegen vor, wenn Reaktions- und Bearbeitungszeiten im Grenzbereich der verwendeten Prozessor-Hardware liegen, also typischerweise im Mikrosekundenbereich, bei weichen sind die Anforderungen deutlich geringer.

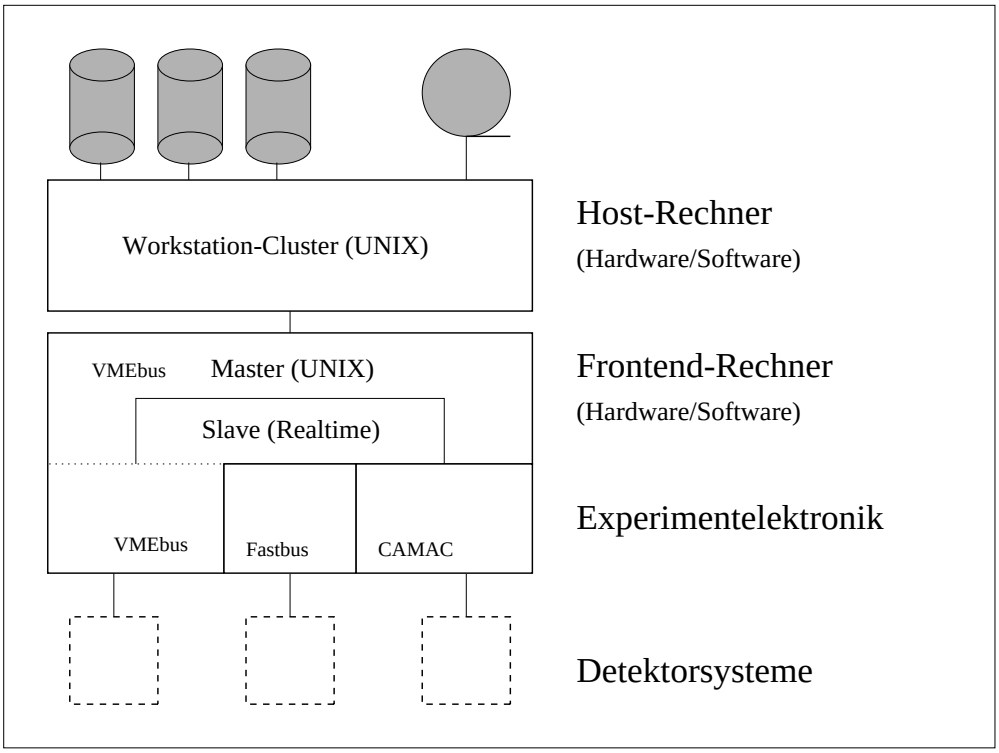

Abb. 3.2: Unterteilung der an der Datenerfassung beteiligten Rechnerkomponenten in drei Grundtypen: Workstations für übergeordnete, zeitunkritische Aufgaben und Master-Slave-Mulitprozessorsysteme für hardwarenahe Echtzeitaufgaben.

Die fürs erste vorgesehenen CPU-Karten der Firma Eltec lassen sich in beiden Bereichen gut einsetzen. Sie können mit wenig Aufwand ohne Betriebssystem genutzt werden, bieten aber auch die Möglichkeit, unter UNIX betrieben zu werden. Die ursprünglich kommerziell erhältlichen UNIX-Versionen [Stol87] waren jedoch sehr fehleranfällig und besaßen in vielen Bereichen eine eingeschränkte Funktionalität. Mit der Verfügbarkeit der UNIX-Quellen — insbesondere der von BSD 4.3[5] — konnte aber eine eigene Portierung dieses leistungsfähigen Betriebssystems auf Eltec-CPUs durchgeführt werden [Kryg91]. Auf diese Weise wurde ein voll zu den auf Workstation-Ebene eingesetzten UNIX-Varianten kompatibles Betriebssystem verfügbar, das keine Software-Kosten verursacht und in Bezug auf Wartung Abhängigkeiten von externen Firmen vermeidet.

Für die inzwischen erhältlichen moderneren VMEbus-CPU-Karten, die mit den existierenden kombiniert werden können oder sie in Zukunft sogar ersetzen werden, wird als Standard-System in der Regel ebenfalls UNIX angeboten. Damit ist der einfache Übergang nicht nur hardwaremäßig sehr einfach, sondern auch in Hinsicht auf die Software gewährleistet.

[5]BSD (Berkeley Software Distribution) ist die an der Universität von Berkeley entwickelte UNIX-Variante, die auf vielen Rechnerarchitekturen verfügbar ist und viele kommerzielle UNIX-Versionen stark beeinflußt hat.

3.2.2 Wahl der Programmiersprache

Auch der Programmiersprache zur Realisierung der Software für das Datenerfassungssystem kommt eine bedeutende Rolle zu. Sie entscheidet mit über die Güte und Effizienz der Implementation, aber auch die zukünftige Wartbarkeit und Übertragbarkeit der Programme.

Die Standardprogrammiersprache unter UNIX und vielen Echtzeitbetriebssystemen ist C [Kern90]. Ihre Eigenschaften machen sie in vielfacher Hinsicht gut geeignet für den Einsatz im Bereich der Experimentdatenverarbeitung, wo es in besonderem Maße auf eine effiziente und hardwarenahe Programmierung ankommt, aber auch viele andere Aspekte der Software-Entwicklung nicht vernachlässigt werden dürfen (s. Anhang A.7). C wird wegen seiner besonderen Vorteile gegenüber anderen Programmiersprachen schon seit vielen Jahre für Programmentwicklungen im Institut für Kernphysik eingesetzt [Kryg86, Kund88, Stol88, Bohn90].

Zur Realisierung des Datenerfassungssystems stellt C eine gute Grundlage dar, die von anderen Programmiersprachen nur unvollständig abgedeckt werden kann. Fortran z. B. ist zwar noch immer die dominierende Programmiersprache im naturwissenschaftlichen Umfeld, doch bietet hier der Standard keine Möglichkeiten der hardwarenahen Programmierung. Für viele Aufgaben müßte dann wieder zwangsläufig auf Assembler ausgewichen werden — verbunden mit Nachteilen wie schwerer Programmierbarkeit, schlechter Wartbarkeit oder fehlender Portierbarkeit. Auch die fehlenden Möglichkeiten, Zeiger zu verwenden oder komplexe Datenstrukturen zu definieren und leicht zu handhaben, machen diese Sprache für viele Probleme, die bei der Datenerfassung anfallen und oftmals bis in den Bereich der Systemprogrammierung hineinreichen, sehr unhandlich. Spezielle Fortran-Derivate (z. B. [vdSc89]), die für diese Zwecke vom Standard abweichende Erweiterungen der Sprache enthalten, lösen diese Problematik meist nur beschränkt und auf Kosten von Kompatibilität und Portabilität. Ihre geringe Verbreitung und gegenseitige Inkompatibilität schränken oft die Verwendbarkeit von vielen Rechnerarchitekturen und Betriebssystemen unnötig ein. Ähnliches gilt auch für viele andere Programmiersprachen. Auch so mächtige Sprachen wie z. B. PL/1 erweisen sich zwar prinzipiell als ähnlich gut oder gar besser geeignet als C, die gestellten Probleme zu lösen, doch erfordern sie in der Regel einen wesentlich höheren finanziellen Aufwand oder sind oftmals für die eingesetzten Computersysteme gar nicht erhältlich. Höhere Komplexität macht sie in der Regel auch schwerer erlern- und beherrschbar, und da sie nicht so weit verbreitet wie C sind, ist meist wesentlich weniger Literatur verfügbar.

Von besonderem Interesse sind seit einiger Zeit die Methoden der objektorientierten Programmierung (OOP). Sie bieten für Software-Entwicklung und -Wartung eine ganze Reihe Vorteile und werden bereits im Bereich der Physik in einigen Software-Projekten verwendet (siehe z. B. [Kunz91, Kunz92, Quar93]). Als Programmiersprache eignet sich für die Experimentdatenverarbeitung wegen ihrer Aufwärtskompatibilität zur Sprache C besonders C++ [Stro92], die auch im Rahmen des A1-Projekts Anwendung findet. Die Sprache wird im Bereich der Steuerung der Meßappartur und der On-line-Analyse erfolgreich einge-

setzt [Kram95, Kund95, Stef93, Hake93, Mart93, Schr93]. Leider waren geeignete Compiler erst zu einem Zeitpunkt verfügbar, als die wesentlichen Arbeiten am Datenerfassungssystem schon abgeschlossen waren. So konnten hier die Vorteile dieser Sprache noch nicht genutzt werden. Dennoch wurden auch ohne dieses Hilfsmittel viele Prinzipien der objektorientierten Programmierung angewendet. In Zukunft ist jedoch eine Weiterentwicklung auf der Basis von C++ sinnvoll.

3.3 Netzwerke und Kommunikation

Der Einsatz von mehr als nur einem Rechner bietet die Möglichkeit, unterschiedliche Aufgaben der Datenerfassung je nach Hard- und Software-Anforderungen auf verschiedene Rechner zu verteilen und dort unter bestmöglichen Bedingungen bearbeiten zu lassen. Auch der damit verbundene Effekt der Parallelisierung kann mit für eine Leistungssteigerung des Gesamtsystems genutzt werden. Bei geeigneter Konzeption kann man schließlich unter Ausnutzung von Redundanzen auch eine Erhöhung der Zuverlässigkeit des Systems erreichen.

Um jedoch die Vorzüge eines verteilten Systems nutzen zu können, müssen auch die Konsequenzen in Form gestiegener Anforderungen an Hard- und Software berücksichtigt werden. In erster Linie tritt dabei das Problem der Kommunikation auf, denn die einzelnen Rechner müssen für eine korrekte Zusammenarbeit ständig Daten und Kontrollinformation austauschen. Aber auch schon innerhalb eines Rechners ist ein Informationsaustausch zwischen verschiedenen Prozessen notwendig, die gemeinsam an derselben Aufgabenstellung arbeiten.

Moderne Betriebssysteme bieten eine ganze Reihe von Verfahren sowohl bei der Interprozeßkommunikation als auch für den rechnerübergreifenden Datenaustausch. Bei der Wahl der einzusetzenden Kommunikationsmechanismen müssen jedoch die von der Aufgabenstellung her bestehenden Anforderungen wie erforderliche Datenübertragungsgeschwindigkeit oder Echtzeitverhalten berücksichtigt werden. Schließlich ist auch noch der Fall zu beachten, daß auch mit Rechnern ohne Betriebssystem eine Kommunikation möglich sein muß; hier kann nämlich nicht mehr auf entsprechende Systemsoftware zurückgegriffen werden.

3.3.1 Interprozeßkommunikation

Die standardmäßig unter UNIX zur Verfügung stehenden Mechanismen zur Interprozeßkommunikation (IPC) (s. Anhang A.8) wie Pipes, Shared Memory und Sockets decken im wesentlichen alle im Rahmen der Datenerfassung anfallenden Anforderungen an die lokale Kommunikation auf den einzelnen Rechnern sehr gut ab. Pipes und Sockets zeichnen sich dabei besonders durch ihre universelle Einsetzbarkeit aus. Neben ihren effizienten Kommunikationseigenschaften kann auch von den Möglichkeiten der symbolischen Adressierung nutzbringend Gebrauch gemacht werden. Shared Memory kann in vielen Bereichen der Datenerfassung genutzt werden, wo es darauf ankommt, z.B. die Meßdaten für verschiedene Aufgaben gleichzeitig zur Verfügung zu stellen. Das ist insbesondere im Rahmen von Archivierung und On-line-Analyse sehr nützlich.

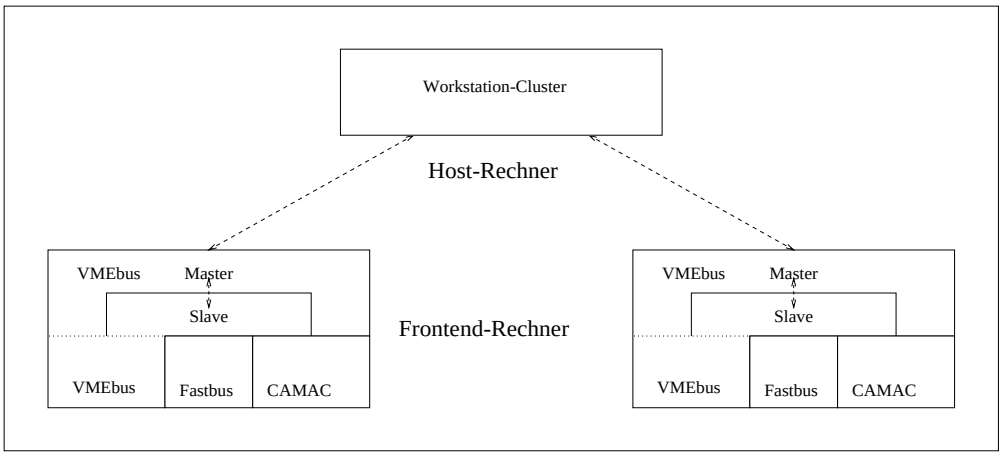

Abb. 3.3: Schematische Darstellung der Rechner, die an der Datenerfassung beteiligt sind, und auf geeignete Weise miteinander kommunizieren müssen.

3.3.2 Rechnerübergreifende Kommunikation

Wie bereits an früherer Stelle erörtert wurde, ist zur Datenerfassung an komplexen Experimentierapparaturen wie der Drei-Spektrometer-Anlage ein Rechnersystem mit recht heterogener Struktur erforderlich, um die vielen, sehr unterschiedlichen Anforderungen bestmöglich abdecken zu können. Nach dem oben beschriebenen Modell besteht es aus mindestens drei Grundtypen von Rechnern, die zur Bearbeitung unterschiedlicher Aufgabenstellungen notwendig sind:

1. Workstations unter einem geeigneten Multitasking/Multiuser-Betriebssystem wie z.B. UNIX
2. VMEbus-Rechner mit übergeordneter Master-Funktion unter einem geeigneten Betriebssystem zur Prozeßdatenverarbeitung
3. VMEbus-Rechner mit untergeordneter Slave-Funktion ohne Betriebssystem oder mit Realtime-Kernel

Für einen sinnvollen Einsatz muß der VMEbus-Master in der Lage sein, sowohl mit den entfernt aufgestellten Workstations zu kommunizieren als auch mit den direkt verbundenen VMEbus-Slaves. Zwischen Workstations und VMEbus-Slaves ist eine direkte Kommunikation nicht unbedingt erforderlich.

Für die Kommunikation zwischen den VMEbus-Rechnern bietet der VMEbus als physikalische Schicht nach dem OSI-Schichtenmodell (Anh. A.9) eine sehr gute Grundlage. Seine Leistungsdaten erlauben eine für die Zwecke der Datenerfassung genügend hohe Datenübertragungsgeschwindigkeit, die weniger vom Bussystem als von der Leistungsfähigkeit der verwendeten CPUs begrenzt wird (Anh. A.1). Die eingesetzten CPU-Karten auf der Basis von 68020- bzw. 68030-Prozessoren sind in der Lage bis zu 6 MB/s über den VMEbus zu transferieren — etwa eine Größenordnung mehr, als bei der beschriebenen Apparatur und den vorgesehenen Experimenten maximal an Datenrate anfallen wird. Die

Arbitrierungslogik des VMEbus regelt den sauberen Buszugriff für alle aktiven Busteilnehmer auf Hardware-Ebene, so daß eine korrekte Datenübertragung zu jedem Zeitpunkt sichergestellt ist. Der Datenaustausch selbst läuft über einfache Speicherzugriffe auf gemeinsam zugänglichen Speicher, der über den VMEbus in den Adreßraum der CPUs eingeblendet ist. Dieser Teil der physikalischen Schicht und alle darüberliegenden Protokollebenen des Schichtenmodells müssen in Form geeigneter Software auf den beteiligten Prozessoren realisiert sein.

Um eine Kommunikation zwischen Workstations und VMEbus-Rechnern zu verwirklichen, ist eine zusätzliche physikalische Verbindung notwendig. Hierfür kommen mehrere Alternativen in Frage, von denen sich in erster Linie einmal Ethernet anbietet, da das ein allgemein eingesetzter Netzwerkstandard ist (s. Anhang A.10). Ethernet-Interfaces gehören zur Standardausstattung von Workstations, und auch die VMEbus-Rechner wurden so gewählt, daß sie ebenfalls über einen Ethernet-Anschluß verfügen. Ethernet wird im Institut für Kernphysik schon seit Mitte der 80er Jahre eingesetzt. Auch die Vernetzung auf dem restlichen Universitätsgelände basiert auf diesem Standard. Fast alle im Institut installierten Rechner sind an das Netzwerk angeschlossen und können untereinander und mit der Außenwelt kommunizieren.

3.3.3 Das Message-System MUPIX

Im Rahmen der Arbeiten an der Software zur Steuerung der Experimentierapparatur wurde ein auf Narchrichten basierendes rechner- und betriebssystemübergreifendes Kommunikationssystem realisiert [Kram95]. Das Message Passing System MUPIX ermöglicht es, die innerhalb eines verteilten, heterogenen Rechnersystems laufenden Teilen der Steuerungs-Software zu synchronisieren und einen geeigneten Informationsaustausch zwischen diesen zu gewährleisten. Es setzt auf dem paketorientierten, ungerichteten Protokoll UPD[6] auf, das wie das verbindungsorientierte TCP[7] zum Standardumfang der Internet-Protokollfamilie (IP) gehört und ohne den Aufbau fester Kommunikationsverbindungen die effiziente Übertragung von Datenpaketen zwischen Prozessen auf verschiedenen Rechnern oder innerhalb eines Rechners erlaubt. Das Message-System nutzt diesen Kommunikationsmechanismus zur Übermittlung von Nachrichten aus und realisiert damit ein eigenes, fehlergesichertes Protokoll, das ohne Eingriffe in das Betriebssystem auf Prozeßebene asynchron arbeitet.

Das Message-System erlaubt die Adressierung der Kommunikationspartner aufgrund symbolischer Namen und schließt dabei auch die Verwendung von Wildcard-Mechanismen ein. Es macht von einem auch für andere Zwecke nutzbaren Nameservice Gebrauch, der die Zuordnung von symbolischen und physikalischen Adressen verwaltet und diese Information Prozessen, die das System benutzen, ebenfalls auf der Basis von UDP zur Verfügung stellt.

Die Paketorientierung von MUPIX, die die Nachrichtenlänge auf etwa 1 KB beschränkt, erlaubt zwar sehr gut die Übermittlung von Steuerungs- und Sta-

[6] User Datagram Protocol
[7] Transmission Control Protocol

tusinformationen, erfordert jedoch für die im Bereich der Datenerfassung notwendige effiziente Übertragung großer Datenmengen zusätzlichen Implementationsaufwand zur Realisierung dazu notwendiger Protokolle. Hier stellt die unmittelbare Verwendung effizienter IPC-Mechanismen oder die Benutzung von TCP die bessere Alternative in Bezug auf Leistung und Programmieraufwand dar. Für die Master-Slave-Kommunikation wäre sogar eine Neuimplementierung des Message-Systems notwendig geworden.

3.4 Alternativen zur Realisierung eines Datenerfassungssystems

Die bisher in diesem Kapitel angestellten grundsätzlichen Überlegungen geben ausgeprägte Hinweise, wie und mit welchen Hilfsmitteln das geforderte Datenerfassungssystem realisiert werden kann, verdeutlichen jedoch auch Komplexität und Schwierigkeitsgrad eines solchen Projekts. Es stellt sich die Frage: ist es notwendig, das System vollständig in Eigenarbeit zu verwirklichen, oder ist es eventuell möglich, auf Existierendes zurückzugreifen?

3.4.1 Ungenügende Verfügbarkeit eines eigenen Systems

Wie bereits an früherer Stelle erörtert war zum Zeitpunkt, an dem MAMI erweitert wurde und die Entwicklungen der neuen Detektorsysteme begannen, ein Datenerfassungssystem, das die gestellten Anforderungen erfüllte, weder innerhalb des Instituts noch bei den sich allmählich etablierenden Kollaborationen vorhanden. Es existierten lediglich erste Ansätze im Rahmen von Diplomarbeiten [Kryg86, Stol88, Kund88], die jedoch zu speziell waren, zum Teil mit inzwischen veralteten Methoden arbeiteten, für einfache und überschaubare Detektorsysteme konzipiert waren und nur geringe Flexibilität und Erweiterungsmöglichkeiten boten. Neben wenigen konkreten Nutzanwendungen konnte daraus in erster Linie ein umfangreicher Erfahrungsschatz mitgenommen werden.

3.4.2 Einsatzmöglichkeit fremder Systeme

3.4.2.1 GOOSY

Als eine mögliche Basis für ein neues Datenerfassungssystem bot sich das Software-System GOOSY[8] der GSI in Darmstadt an [Esse88]. Es besaß als schon längere Zeit existierendes und korrekt arbeitendes System die Vorteile, daß wesentliche Teile bereits intensiv getestet waren, eine gute Unterstützung und ausreichende Dokumentation verfügbar waren. Insbesondere zeigte das System hervorragenden Fähigkeiten bei der Auswertung der Meßdaten und vieler damit verbundenen Aufgabenstellungen.

[8] GSI On-line Off-line SYstem

Dem standen jedoch auch gravierende Nachteile gegenüber. Von besonderer Bedeutung ist die Tatsache, daß die standardmäßig unterstützte Datenerfassung zwar sehr gut an das Analysesystem angekoppelt war, ansonsten jedoch ähnliche Einschränkungen wie die an MAMI verfügbare Datenerfassungs-Software hatte, so daß auch hier neben der Anpassung an die speziellen Detektorsysteme weitergehende Erweiterungen notwendig wurden. Der Einsatz von GOOSY im Rahmen der A2-Kollaboration erforderte schließlich eine vollständige Neuimplementation der Datenaufnahme-Softare im Frontend-Bereich.

Weitere Nachteile dieses Konzepts war die Festlegung auf bestimmte Rechner, Systemsoftware (VAX/VMS) und deren Hersteller im Workstationbereich und auf eine zwar leistungsfähige, jedoch wenig verbreitete und bekannte Programmiersprache für Analysezwecke (PL/1), die zudem auf den Frontend-Rechnern nicht verfügbar war. Daneben verursachte das System einen sehr hohen Verbrauch an Systemressourcen wie Plattenplatz, Arbeitsspeicher und Rechenzeit und erforderte als komplexes System ein spezielles System-Management ähnlich dem für ein Betriebssystem.

Obwohl die im Rahmen dieser Arbeit implementierte Erweiterung von GOOSY über Jahre hinweg für den Spezialfall befriedigend arbeitete, erfüllte dieser Ansatz einige wesentliche Anforderungen, die an das neue Datenerfassungssystem für MAMI gestellt wurden, nicht und hätte für die Zukunft zu starke Einschränkungen bedeutet.

3.4.2.2 Software aus kernphysikalischen Großforschungsanlagen

CERN-Software

Die im kernphysikalischen Bereich weit verbreitete Software aus der CERN-Bibliothek deckt in erster Linie vielfältige Analyse-Aufgaben ab und ist weniger für die anderen Problemstellungen der Datenerfassung geeignet. Die ursprünglich für typische Off-line-Aufgaben konzipierte Software bot zu Beginn der Entwicklungen an dem neu zu schaffenden Datenerfassungssystem für MAMI keinerlei Möglichkeiten für den On-line-Einsatz und wurde erst nach und nach rudimentär im Frontend-Bereich verfügbar.

Spezielle Datenerfassungs-Software

Auch Software, die speziell zur Datenerfassung für Experimente an kernphysikalischen Großforschungsanlagen wie CERN oder DESY realisiert wurde, konnte nicht genutzt werden. Zum einen zeigen sich hier grundsätzliche Probleme. Die oftmals für große Detektorsysteme konzipierten Datenerfassungssysteme können nicht so ohne weiteres nach unten skaliert werden oder haben ganz andere durch Beschleuniger und Experiment vorgegebene Anforderungen zu erfüllen. So zeigt der Vergleich des Zeitverhaltens eines Speicherrings, wie er bei Hochenergieexperimenten eingesetzt wird, mit dem eines Dauerstrichbeschleunigers wie MAMI grundlegende Unterschiede. Während im einen Fall Ereignisse mit einer extrem

großen Datenmenge jedoch sehr niedriger Wiederholfrequenz erzeugt werden, fällt im anderen Fall bei jedem Ereignis nur ein Bruchteil der Datenmenge an, doch kann hier der zeitliche Abstand zweier Ereignisse sehr klein werden. Im ersten Fall kommt es weniger auf eine schnelle Reaktion durch die Rechner an als vielmehr auf eine hohe Geschwindigkeit beim Auslesen und Archivieren der Daten, im zweiten Fall sollte zwar die Datenaufnahme ebenfalls schnellstmöglich abgewickelt werden, doch spielt hier eine schnelle Reaktion eine mindestens ebenso große Rolle. Das macht deutlich, daß ein leistungsfähiges Datenerfassungssystem für Hochenergieexperimente nicht notwendigerweise auch für die Experimente an MAMI geeignet sein muß.

Derartige Software kam auch deswegen nicht zum Einsatz, da sie zum Zeitpunkt der Neukonzeption der MAMI-Datenerfassung zu wenig von neuen Technologien und Methoden Gebrauch machten, wie z. B. der Einsatz von VMEbus, RISC-Workstations, modernen Betriebssystemen und Programmiersprachen. Gleichzeitig war kein allgemein gehaltenes und für verschiedene Einsatzmöglichkeiten konfigurierbares System verfügbar, sondern vielfältige auf spezielle Detektorsysteme, Meßelektronik und Rechnersysteme ausgelegte Datenerfassungsprogramme, die durch einen erheblichen Aufwand auf neue Anforderungen hätten angepaßt werden müssen. Nicht zuletzt spielte auch die finanzielle Kostenfrage eine Rolle, die zu aufwendige, der lokalen Problemstellung nicht angemessene Lösungen von vornherein ausschloß.

3.5 Fazit

Zum Abschluß diese Kapitels sollen noch einmal kurz die Ergebnisse aus den zuvor angestellten grundlegenden Überlegungen zusammengefaßt werden: Es hat sich gezeigt, daß aufgrund fehlender Alternativen eine weitestgehend eigene Implementation von Software zur Durchführung der Datenerfassung für Experimente an MAMI zu leisten war. Lediglich im Bereich der Analyse konnte auf Software aus anderen Forschungseinrichtungen zurückgegriffen werden. Es erwies sich als grundsätzlich sinnvoll, dabei so weit wie möglich Standards auszunutzen, die sowohl in weiten Bereichen der Hardware als auch bei der Software existieren. Damit sollte es möglich sein, mit minimalem zusätzlichem Arbeitsaufwand ein zeitgemäßes und leistungsfähiges System zu erhalten, das auch noch Optionen für die Zukunft offen hält.

Im Bereich der Experimentelektronik finden CAMAC und Fastbus als kernphysikalische Standardausrüstung Verwendung, ergänzt durch den Industriestandard VMEbus, der auch die Basis für die Frontend-Rechner darstellt. Diese sind als Multiprozessorsysteme unter Kontrolle des Betriebssystems UNIX ausgelegt und softwaremäßig eng übergeordnete UNIX-Workstations gekoppelt. Die Software-Entwicklung findet ausschließlich auf Hochsprachenebene mit C unter Ausnutzung der von UNIX zur Verfügung gestellten Werkzeuge statt. Neben den in diesem Betriebssystem verfügbaren Mechanismen zur Interprozeßkommunikation werden zum rechnerübergreifenden Datenaustausch TCP/IP-Standardprotokolle auf der Grundlage von Ethernet genutzt.

Kapitel 4

Das Konzept von MECDAS

Ausgehend von den im vorangegangenen Kapitel erläuterten Grundlagen wurde das Datenerfassungssystem MECDAS (Mainz Experiment Control and Data Acquisition System) konzipiert und realisiert, das der Durchführung der Experimente an MAMI unter besonderer Berücksichtigung der Anforderungen der Koinzidenzexperimente mit der Drei-Spektrometer-Anlage dient. MECDAS ist als ein modulares System aufgebaut, das in weiten Bereichen konfigurierbar ist, um flexibel und ohne Verlust an Effizienz an unterschiedliche Anforderungen angepaßt werden zu können. Es arbeitet mit einem verteilten, heterogenen Rechnersystem auf der Basis von UNIX und nutzt moderne Technologien sowohl im Hardware- als auch im Software-Bereich.

In Kombination mit Doktor- und Diplomarbeiten, die die Aufgabenbereiche Steuerung der Experimentierapparatur [Kund95] und Überwachung von Apparatur und Experiment [Kram95] abdeckten, wurden im Rahmen der vorliegenden Arbeit Verfahren entwickelt und entsprechende Softwarepakete erstellt für die Aufgabengebiete

- Experimentkonfiguration,
- Steuerung der Datenerfassung,
- Datenaufnahme,
- Datentransfer,
- Eventbuilding,
- Datenarchivierung,
- On-line-Analyse.

Sie werden ergänzt durch Vorschriften zur Benutzung des Systems bzw. seiner einzelnen Komponenten und die Definition von Schnittstellen, die es erlauben, vom Benutzer bereitgestellte Software einfach anzukoppeln und damit das System speziellen Bedürfnissen anzupassen bzw. zu erweitern. Die komplette

Software ist in der Hochsprache C geschrieben. Das MECDAS zugrundeliegende Konzept lehnt sich weitestgehend an die Prinzipien der objektorientierten Programmierung (OOP) an, auch wenn zum Zeitpunkt der Software-Implementation entsprechende Hilfsmittel noch nicht direkt genutzt werden konnten.

Um etwaige implementationsspezifische Beschränkungen möglichst gering zu halten, ist MECDAS als offenes System konzipiert. Die Komplexität der Gesamtproblemetik erlaubte es nicht, im Rahmen dieser Arbeit alle Aufgabenstellungen bis in jedes Detail programmtechnisch umzusetzen. Vielmehr erforderte sie, anwendungsspezifische Erweiter- und Anpaßbarkeit zu einem zentralen Element des Konzepts zu machen mit dem Ziel, einige bisher nur in Form von Prototypen realisierte Problemlösungen bei Bedarf mit wenig Aufwand durch verbesserte bzw. besser an geänderte Anforderungen angepaßte Alternativen zu ersetzen.

4.1 Modularität

Die im Rahmen der Datenerfassung zu bewältigenden Aufgaben decken ein weites Problemfeld ab. Um dabei allen Anforderungen möglichst gut gerecht zu werden, ist MECDAS durchgängig als ein modulares System aufgebaut, das aus vielen, unabhängigen Teilen besteht, die miteinander kombiniert werden können und so ein leistungsfähiges Gesamtsystem ergeben. Auf Hardware-Ebene spiegelt sich diese Modularität nicht nur in dem Einsatz modularer Systeme wie CAMAC, Fastbus oder VMEbus wider, sondern auch in der Struktur des Rechnersystems. MECDAS nutzt konsequent die Verteilung der Aufgaben auf mehrere, unterschiedliche Rechner aus, die für die Bearbeitung abgegrenzter Aufgabengebiete in Bezug auf Hardware-Ausstattung und Systemsoftware bestmöglich ausgelegt sind.

4.1.1 Verteiltes System

Um sowohl der räumlichen Trennung der eingesetzen Detektorsysteme als auch den unterschiedlichen Anforderungen der im Rahmen der Datenerfassung anfallenden Aufgaben Rechnung zu tragen, nutzt MECDAS die Eigenschaften eines verteilten, heterogenen Rechnersystem, das aus Gründen der besseren Steuerbarkeit hierarchisch organisiert ist. Es setzt sich je nach Komplexität des Experiments aus einer mehr oder weniger großen Anzahl von Rechnern zusammen, die jeweils unabhängig voneinander Teilaufgaben übernehmen. Zur individuellen Behandlung einzelner, in sich abgeschlossener Detektorsysteme, die z. B. die Arme einer Koinzidenzanordnung repräsentieren, bzw. eigenständiger Detektoreinheiten, aus denen eine komplexe Experimentieranlage bestehen kann, werden jeweils eigene Frontend-Rechner eingesetzt, die die lokale Ansteuerung der jeweiligen Detektorelektronik vornehmen und damit auch unnötigen Verkabelungsaufwand vermeiden helfen. Zu ihrer Koordinierung werden sie ergänzt durch übergeordnete Rechner, die auch die verbleibenden Aufgaben übernehmen. Abbildung 4.1 zeigt eine schematische Darstellung des verteilten Rechnersystems, wie es typischerweise im Rahmen von MECDAS zum Einsatz kommt.

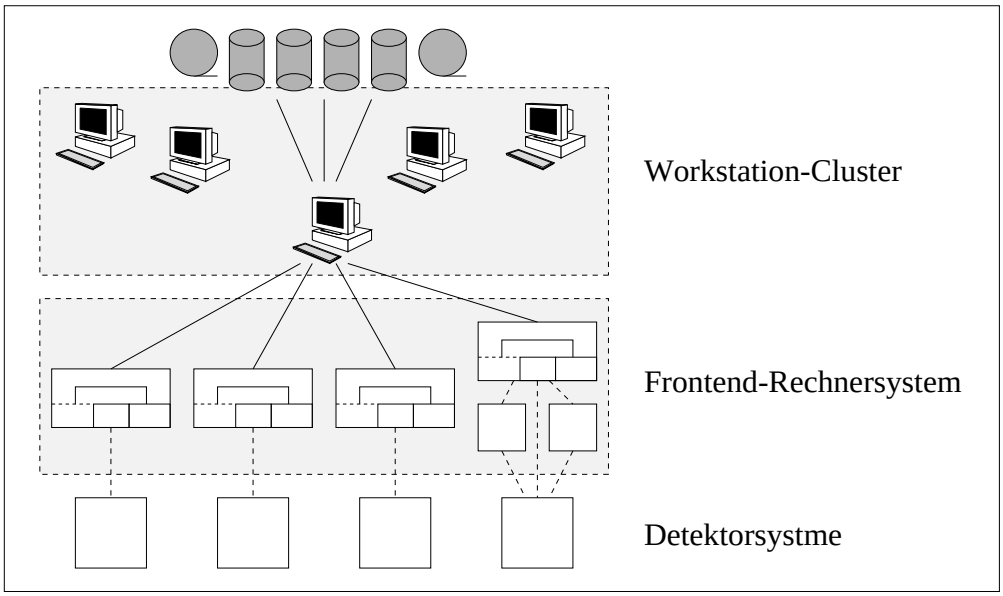

Abb. 4.1: Schematische Darstellung des verteilten Rechnersystems, das sich aus einem Workstation-Cluster und mehreren als Master-Slave-Systeme realisierten Frontend-Rechnern zusammensetzt, die einzelnen Detektorsystemen zugeordnet sind.

In Abhängigkeit von den eingesetzten Detektorsystemen, der Menge der zu verarbeitenden Daten, der erforderliche Datenerfassungsgeschwindigkeit oder sonstiger Anforderungen kann die Struktur eines verteilten Rechnersystems in weiten Grenzen variieren. Die Leistungsfähigkeit des Gesamtsystems läßt sich leicht erhöhen und geänderten Randbedingungen anpassen, indem an der einen oder anderen Stelle Frontend-Rechner oder Workstations ausgetauscht oder hinzugefügt werden. Eine einheitliche Software-Struktur erleichtert dabei die Verlagerung hardwareunabhängiger Aufgaben von einem auf einen anderen Rechner, eine weitere Verteilung oder ein Zusammenzufassen. Aber auch bei direkt mit der Hardware verknüpften Aufgaben ist eine geeignete Aufteilung möglich. Insbesondere der Einsatz intelligenter Subsysteme ist denkbar.

Ähnlich wie bei der Experimentelektronik bietet ein modulares Rechnersystem die Möglichkeit, einzelne Komponenten einfach, mit minimaler Beeinflussung des restlichen Systems auszutauschen. Insbesondere im Hinblick auf die schnelle Fortentwicklung im Computerbereich sind damit die besten Voraussetzungen gegeben, das System sukzessive auf einen moderneren Stand zu bringen, ohne das komplette System zu einem bestimmten Zeitpunkt ersetzen zu müssen.

4.1.2 Software

Im Software-Bereich ist die Modularität auf verschiedenen Ebenen realisiert. Zum einen ist jedes Programm für sich modular programmiert. Anstelle eines unübersichtlichen, monolithischen Programms werden viele unabhängige Unterprogramme und aus einer überschaubaren Anzahl kleiner Routinen und Funktionen bestehende Programm-Module zur Lösung einzelner Probleme verwendet.

Neben dem Vorzug, große Teile derartig aufgebauter Software in unterschiedlichen Bereichen wiederverwenden zu können, hat diese Programmierweise Vorteile bei der Software-Entwicklung und dem Test des Systems. Kleine und übersichtliche Programmeinheiten mit definierten Schnittstellen sind für eine fehlerfreie Programmierung oder nachträgliche Fehlerbeseitigung sehr hilfreich und erleichtern die zukünftige Wartung auch eines in seiner Gesamtheit komplexen Systems wie MECDAS. Durch eine weitgehende Kapselung der Softwaremodule wird erreicht, daß, ähnlich wie bei der Hardware, auch auf Software-Ebene ein fehlerhaftes oder ineffizientes Modul durch eine bessere Implementierung einfach ausgetauscht werden kann, ohne das restliche System zu beeinflussen. Nachträgliche Erweiterungen der Funktionalität sind ebenso leicht möglich.

Modularität zeigt sich auch auf Programmebene. Die Programme selbst stellen einzelne, eigenständige Bausteine für das Gesamtsystem dar. Eine Vielzahl von ihnen steht zur Verfügung, um abgegrenzte Teilaufgaben zu bearbeiten. Sie können einzeln aufgerufen oder in Gruppen benutzt werden und relativ frei zur Lösung komplexer Aufgaben zusammengestellt werden. Noch einfacher als auf Unterprogrammebene sind solche Module unabhängig voneinander zu implementieren und zu testen und bei Bedarf auch auszutauschen. Es ist sogar für einen Anwender ohne viel Aufwand möglich — und in vielen Fällen auch sinnvoll —, einzelne Standard-Programme durch eigene Versionen zu ersetzen oder gar das System durch neue Programme zu erweitern.

4.2 Datenerfassung

4.2.1 Client-Server-Modell

Die Datenerfassung in MECDAS arbeitet nach dem Client-Server-Prinzip. Die zentrale Rolle spielt ein abgeschlossenes Programm, das als einziger Teil innerhalb der Datenerfassungs-Software Kenntnis über experiment- oder detektorspezifische Details hat, um die grundlegenden Aufgaben der Datenerfassung auszuführen und die korrekte Ansteuerung und Auslese der Meßelektronik an einer konkreten Experimentierapparatur vorzunehmen. Es ist in der Lage, jederzeit auf Steueranweisungen durch den Benutzer zu reagieren und sie bei Bedarf auszuführen. Es arbeitet als Server, der alle experimentspezifischen Dienstleistungen zur Verfügung stellt. Seine Klienten, Software zur Steuerung der Datenerfassung, aber auch Archivierungs- und On-line-Analyseprogramme, schicken ihm Anweisungen in Form von Nachrichten, die er bearbeitet und mit einer Antwortnachricht quittiert. Diese Programme sind im wesentlichen sehr allgemein gehalten. Bei Bedarf erhalten sie nötige Informationen vom zentralen Datenaufnahme-Server bzw. lassen diesen die Aktionen zur Datenaufnahme ausführen.

Das Client-Server-Konzept erlaubt eine strikte Trennung zwischen den Bereichen mit harten Echtzeitanforderungen auf der einen Seite und weniger zeitkritischen Anforderungen auf der anderen. Klare Schnittstellen ermöglichen es, diese Software lokal auf einem einzigen Rechner zu betreiben oder auch auf mehrere Rechner zu verteilen.

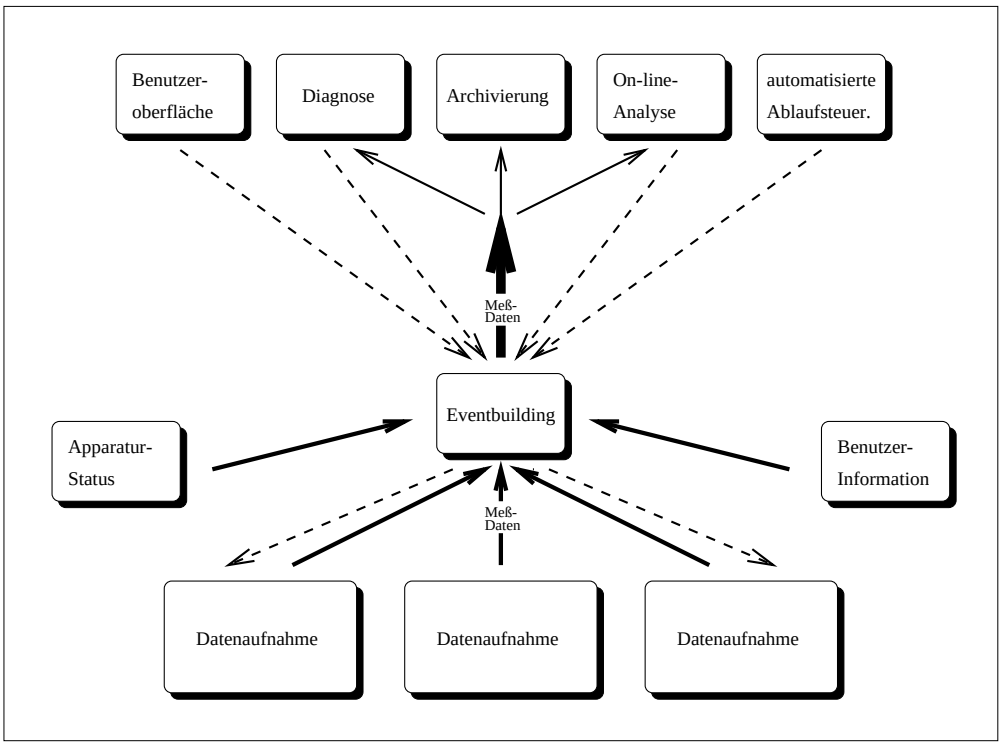

Abb. 4.2: Schematische Darstellung von Kontroll- (- - -) und Datenfluß (——) innerhalb des Client-Server-Systems, das zur Datenaufnahme eingesetzt wird.

4.2.2 Master-Slave-System

Die leistungsfähigste Variante der Umsetzung des Client-Server-Modells basiert auf der Aufgabenverteilung in einem Master-Slave-System: Weniger zeitkritische Anwendungen wie Experimentsteuerung, Datenarchivierung, -vorauswertung oder -reduktion werden durch den unter UNIX arbeitenden Master-Prozessor ausgeführt. Er kontrolliert alle rechnergesteuerten Aktivitäten auf dem Subsystem. Dazu gehört z. B. die Ansteuerung der zum jeweiligen Detektorsystem gehörigen Experimentelektronik oder die Steuerung und Regelung der Spannungs- bzw. Stromversorgung von Detektoren und Spektrometermagneten. Der Master kontrolliert den ausschließlich für die zeitkritischen Aufgaben der Datenerfassung bestimmten Slave-Prozessor und sorgt für die weitere Verarbeitung und insbesondere die Abspeicherung der Meßdaten.

Der Slave-Rechner besitzt selbst kein Betriebssystem und steht daher für eine schnelle Datenaufnahme mit kurzen Totzeiten voll zur Verfügung. Er kann, ebenso wie der Master, über den VMEbus mit hoher Geschwindigkeit auf CAMAC- oder Fastbus-Hardware zugreifen. Auf ihm läuft der Datenaufnahme-Server, der durch den Master geladen und gestartet wurde. Er sorgt für die Kommunikation mit dem Master, während parallel dazu die Datenerfassung interruptgesteuert abläuft. Bei Auftreten eines Trigger-Signals wird innerhalb weniger Mikrosekunden die für das jeweilige Experiment bzw. für die verwendete Hardware speziell konfigurierte Ausleseroutine gestartet. Die Daten werden selektiv ausgelesen,

so formatiert, daß eine spätere Ereignisrekonstruktion einfach möglich ist, zwischengespeichert und von Zeit zu Zeit an den übergeordneten Prozessor weitergeleitet. Dieser hat dann für ihre lokale Archivierung oder den Weitertransport zu einem anderen Rechner zu sorgen, währed der Slave-Prozessor parallel dazu mit der Datenaufnahme fortfährt.

Im Normalfall überträgt der Master die Daten über Ethernet auf der Basis von TCP/IP an einen Rechner, der als Eventbuilder arbeitet. Dieser sammelt alle Daten der einzelnen Arme auf, faßt sie ereignisweise zusammen und sorgt für die endgültige Archivierung. Die Daten können lokal auf dem Eventbuilder abgespeichert werden, oder auch an ein Workstation-Cluster weitergeleitet werden, auf dem die Daten über ein Spooling-System schließlich auf Band archiviert werden. Parallel dazu stehen die Daten zur On-line-Analyse zur Verfügung.

In der Regel erfolgt die Datenaufnahme auf den für die einzelnen Arme einer Koinzidenzanordnung eingesetzten Frontend-Systemen vollkommen entkoppelt. Bis auf die Synchronisation durch die Koinzidenzelektronik findet zwischen diesen unter Echtzeitbedingungen keinerlei Kommunikation statt. Sie tauschen lediglich mit den ihnen übergeordneten Rechnern Information aus. Damit ist für Kontroll- und Datenfluß eine strenge Hierarchie realisiert (Abb. 4.2).

4.3 Experimentkonfiguration

Um konkrete Experimente mit MECDAS durchführen zu können, ist neben allgemeiner Software auch Information über das Experiment notwendig, wie z. B. Art und Umfang der verwendeten Elektronik oder Angaben über die zu erfassenden Daten und, wie sie ausgelesen werden sollen. Diese kann natürlich durch feste Programmierung eingebunden werden, wodurch zwar eine optimale Anpassung an die Anforderungen in allen Bereichen gewährleistet wird, doch bei komplexen Experimentkonfigurationen nur mit hohem Aufwand. Um dennoch mit einfachen Mitteln eine flexible Lösung zu erreichen, wurde ein Konzept entwickelt, mit dem eine einfache Anpassung des Systems an unterschiedliche Gegebenheiten durchführbar wird. Es erlaubt eine Konfigurierbarkeit in weitem Rahmen ohne Neuprogrammierung oder zumindest mit einer Reduzierung der Programmierung auf ein Minimum.

Es wurde ein Programmpaket erstellt, das dem Experimentator die Angabe einer speziellen Experimentbeschreibung in vereinfachter Form als Textdateien erlaubt und daraus Tabellen und effizienten und an die speziellen Randbedingungen angepaßten Programmcode für Auslese, Verpacken, Dekodieren, Eventbuildung und Analyse erzeugt. Damit stehen den an der Datenerfassung beteiligten Programmen zur Laufzeit automatisch alle für ihre Funktion notwendigen experimentspezifischen Informationen zur Verfügung. Die Experimentbeschreibung wird komplett zu den zu archivierenden Daten gepackt, so daß sie z. B. auch für Auswerteaufgaben zur Verfügung steht. Damit sind die Daten, die letztendlich auf einem Magnetband oder einer Plattendatei abgespeichert sind, selbstbeschreibend und zu jedem Zeitpunkt eindeutig identifizierbar. Gleichzeitig ist es

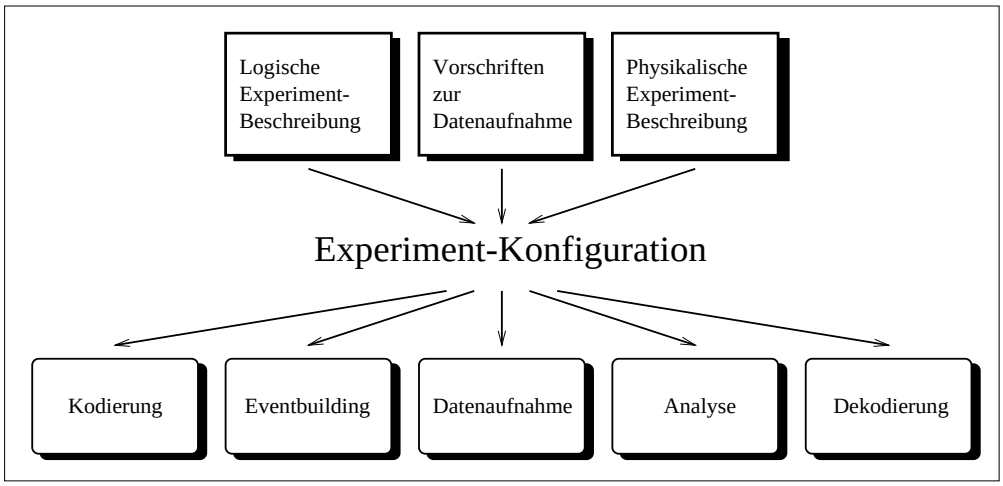

Abb. 4.3: Die zentrale Rolle der Experimentkonfiguration

sehr leicht, sie zur Analyse oder auch zur On-line-Vorauswertung auf zusätzliche Rechner zu verteilen, wo ansonsten keine weitere Information über das Experiment vorhanden ist. Die Experimentbeschreibung stellt die zentrale Datenbasis dar für die wesentlichen Aufgaben von Datenerfassung und Analyse (Abb. 4.3).

4.4 On-line-Analyse

Zur On-line-Überwachung des Experiments ist es notwendig, die aufgenommenen Meßdaten parallel zur Datenerfassung zu verarbeiten. Es muß möglich sein, ein- oder zweidimensionale Histogramme der Rohdaten oder von verknüpften Parametern zu generieren. Auf die Daten müssen einfache Cuts bis hin zu komplexen Auswertebedingungen angewendet werden, um damit Histogramme zu generieren oder die nach bestimmten Kriterien gefilterten Daten für spätere Analyseläufe abzuspeichern. Aus diesem Grund ist On-line-Analyse integraler Bestandteil von MECDAS. Die implementierte Software besteht aus einer Reihe von Standard-Programmen ergänzt durch eine Unterprogrammbibliothek, die es dem Experimentator erlaubt, mit wenig spezifischen Kenntnissen einfach auf die gemessenen Daten zuzugreifen und diese auszuwerten. Wie bei Datenerfassung und Eventbuilding wird auch bei der Analyse von der Experimentbeschreibung Gebrauch gemacht, die u. a. die symbolische Adressierung der Meßdaten über einmal vom Experimentator vergebene Namen erlaubt.

Die Analyse-Software von MECDAS wird abgerundet durch ein umfangreiches Programmpaket zur Verwaltung und graphischen Ausgabe von Histogrammen, das an die speziellen Anforderungen der Experimente an MAMI angepaßt ist. Es wird ergänzt durch Software aus der CERN-Bibliothek [Brun94] und durch das Programmpaket Hippoplotamus der Leland Stanford Junior University [Grav92]. Um sowohl dem Benutzer die einfache Möglichkeit zu geben, spezielle Auswerteprogramme zu implementieren als auch Standard-Software zu integrieren, ist auch die On-line-Analyse als offenes System konzipiert.

4.5 Aufbau des Software-Pakets

Die im Rahmen von MECDAS genutzte Software läßt sich in mehrere Kategorien einteilen. Zum einen gibt es für verschiedene Teilbereiche der Datenerfassung eine Reihe von eigenständigen Programmen, die direkt vom Anwender benutzt werden können bzw. innerhalb komplexer Anwendungen automatisch aufgerufen werden. Sie stehen für viele allgemeine, nicht experimentspezifische Routineaufgaben zur Verfügung, die in der Vorbereitungsphase oder bei der Durchführung eines Experiments anfallen. Dazu gehören z. B. Programme mit komplexen Aufgabenstellungen zum Datentransport, Eventbuilding, zur Pufferverwaltung oder Datenarchivierung bis hin zur graphischen Ausgabe von Histogrammen. Aber auch einfache Hilfsprogramme mit elementaren Aufgaben gehören zur Standardausstattung. Sie werden dazu verwendet, um mit einfachen Mitteln eine allgemeine, leistungsfähige und flexible Benutzerschnittstelle zur Bedienung des Systems zu realisieren. Mittelpunkt ist die Shell, der Standard-Kommandointerpreter von UNIX, deren Eigenschaften hier in umfassendem Maß genutzt werden können. Sie erlaubt nicht nur eine verhältnismäßig komfortable Bedienung, sondern bietet sowohl interaktiv als auch für Kommandoprozeduren sehr einfache und handliche Möglichkeiten, komplexe Aufgaben zu lösen. Gleichzeitig stellt sie mit ihrer Kommandosprache weitreichende Möglichkeiten ihrer Programmierung zur Verfügung, die standardmäßig in MECDAS genutzt werden, aber auch für den einzelnen Anwender sehr hilfreich sein können.

Neben den fertigen Programmen stellt MECDAS noch einen Satz von Unterprogrammbibliotheken zur Verfügung. Sie werden zu einem großen Teil von den oben beschriebenen Programmen benutzt, stellen aber darüberhinaus noch zusätzliche Funktionalität bereit. Sie werden dazu benötigt, um insbesondere im Bereich von Datenaufnahme und On-line-Analyse, wo auch experimentspezifische Besonderheiten in die Software einfließen müssen, Standardaufgaben innerhalb der entsprechenden Programme abzudecken. Dazu gehört z. B. die Behandlung von Ein- und Ausgabe oder anderer Routineaufgaben, das Verpacken und Dekodieren der Meßdaten, elementare Aufgaben bei der Datenaufnahme und Analyse oder die Verwaltung von On-line-Histogrammen.

Die letzte Kategeorie ist die experimentspezifische Software. Sie kann in zwei Formen auftreten. Zum einen sind das Programmteile, die direkt vom Anwender implementiert werden können, um z. B. technische Besonderheiten der Apparatur oder spezielle Algorithmen im Rahmen von Datenaufnahme oder Analyse in die Datenerfassungs-Software zu integrieren. Aufgrund der Modularität des Software-Systems ist das sehr einfach möglich, im Normalfall jedoch nicht notwendig. Von größerer Bedeutung ist dagegen generierter Programmcode, der von standardmäßig verfügbaren Konfigurationswerkzeugen erzeugt wurde und alle wesentlichen experimentabhängigen Parameter enthält.

Kapitel 5

Die zentrale Datenerfassungs-Software

5.1 Überblick

Die Analyse der bereits erläuterten Aufgaben, die das aufzubauende Datenerfassungssystem abdecken sollten, zeigte, daß bei der Datenaufnahme neben den experiment- und detektorspezifischen Aufgaben eine Vielzahl allgemeiner Aufgaben anfallen, die unabhängig von der konkreten Struktur des Experiments und der Konfiguration der Meßelektronik sind. Diese Aufgaben sind in vielen Fällen recht komplex und bedürfen einer aufwendigen Programmierung, auch wenn sie nur Routinefunktionen erfüllen. Um dem Benutzer die Arbeit mit der Datenerfassungs-Software beim Einrichten eines neues Experiment oder Detektorsystem möglichst zu erleichtern und ihn von Routinearbeit zu befreien, empfahl es sich daher, diese Aufgaben weitestgehend allgemein zu lösen und entsprechende Software für die spätere Benutzung zur Verfügung zu stellen.

Um das möglichst gut zu erreichen, wurde die Software so organisiert, daß sie sich in mehrere deutlich abgegrenzte Teilbereiche gliedert, von denen die meisten allgemeinen Charakter haben. Die wenigen Bereiche, die experimentspezifische Aufgaben abdecken, sind über klar definierte Schnittstellen von den anderen getrennt. Ausgehend von diesem Konzept wurde für die elementaren Aufgaben der Datenerfassung wie Datenaufnahme, Archivierung und Steuerung des Systems ein Software-Paket entwickelt, das das Herzstück von MECDAS bildet. Es besteht aus allgemeiner Software in Form experimentunabhängiger Programme und Unterprogrammbibliotheken und wird ergänzt durch Schnittstellen, die es erlauben, experimentspezifischen Programmcode einzubinden, der entweder direkt vom Anwender erstellt werden kann oder mithilfe entsprechender Werkzeuge aus einer Experimentbeschreibung (s. Kapitel 6) generiert wird.

5.1.1 Schichtenmodell für die Datenerfassung

Um den unterschiedlichen Anforderungen, die im Rahmen der Datenerfassung auftreten, bestmöglich Rechnung zu tragen, ist die zentrale Datenerfassungs-Software in einem Schichtenmodell hierarchisch organisiert. Sie setzt sich aus einzelnen Schichten mit deutlich abgegrenzter Funktionalität zusammen, zwischen denen klare Schnittstellen existieren. Jede Schicht baut unmittelbar auf der nächst niedrigeren auf, ohne deren Interna zu kennen, und stellt selbst die Grundlage für die nächst höhere dar. Es war angestrebt, eine möglichst große Unabhängigkeit der einzelnen Schichten voneinander zu erreichen. Das erleichterte einerseits die Software-Entwicklung, machte es aber andererseits möglich, auch in Zukunft bei Bedarf die Lösung einzelner Aufgaben zu verbessern oder sogar neu zu implementieren und so Funktionalität und Leistungsfähigkeit zu steigern, ohne am Gesamtsystem tiefgreifende Änderungen vornehmen zu müssen. Innerhalb des Schichtenmodells nehmen Funktionalität und Abstraktion der Software nach oben hin zu, während Hardware- und Systemabhängigkeiten abnehmen. Es lassen sich folgende Schichten unterscheiden:

1. Bedienoberfläche

Als oberste Schicht stellt die Bedienoberfläche unmittelbaren Kontakt zwischen dem Meßsystem und dem Benutzer her. Sie ist, so wie auch in den anderen Bereichen von MECDAS, besonders deutlich von der restlichen Anwendung getrennt, um die Vermischung anwendungsspezifischer Aufgaben mit den qualitativ stark abweichenden Problemstellungen zu vermeiden, die für die Bedienung eines Software-Systems zu lösen sind. Durch eine klare Schnittstellendefinition auf der Basis des UNIX-Kommando-Konzepts [Bana84] war es möglich, Anwendung und Bedienoberfläche unabhängig voneinander zu implementieren. Insbesondere erlaubte diese Vorgehensweise im Sinne des Rapid Prototyping eine erste Realisierung der Datenerfassungs-Software mit einer einfacheren Benuzterschnittstelle, die erst in einem zweiten Schritt durch eine komfortable, auch von Nicht-Experten benutzbare Oberfläche ersetzt werden mußte. Hier finden Lösungen auf der Basis des herkömmlichen UNIX-Kommandointerpreters und des X Window Systems Verwendung (s. Abschnitt 7.6). Aber auch für die Zukunft bleibt die Möglichkeit erhalten, die Benutzerschnittstelle je nach Anforderung mit wenig Aufwand auszutauschen. Schließlich konnte so auch der Tatsache Rechnung getragen werden, daß die hier angesprochene Software nur ein Teil des Gesamtsystems darstellt, für das es sinnvollerweise eine einheitliche Bedienoberfläche geben sollte.

2. Benutzerspezifische Anpassungen

Die zweite Schicht repräsentiert Software, die benutzerspezifische Anpassungen erlaubt, um auf spezielle Gegebenheiten eines Experiments oder besondere Anforderungen des Experimentators einzugehen. Sie besteht i. w. aus einer Reihe von Kommandoprozeduren, die komplexe Aufgabenstellungen im Rahmen der Steuerung der Datenerfassung oder der Datenarchivierung mithilfe elementarer Kommandos aus der nächsten Schicht

bearbeiten. Sie stehen für den Standardfall zur Verfügung, können aber sehr leicht an spezielle Anforderungen angepaßt werden. Diese Programme werden in der Regel durch die Bedienoberfläche aktiviert.

3. Elementare Kommandos

In der dritten Schicht sind im wesentlichen elementare Kommandos in Form abgeschlossener Programme enthalten zur Durchführung und Steuerung der Datenerfassung, zum Datentransport und zur Archivierung der Meßdaten. Sie sind die Grundlage der nächsthöheren Schicht, können aber bei Bedarf auch direkt vom Anwender benutzt werden. Sie stellen ein allgemeines Baukastensystem dar, dessen Komponenten unter Zuhilfenahme der UNIX-Shell zu komplexen und leistungsfähigen Programmen zur Lösung spezieller Probleme kombiniert werden können.

4. Klienten-Schnittstelle

Die nächste Schicht enthält Unterprogramme in Bibliotheken, die zur Realisierung der in der Kommandoschicht genannten Programme benötigt werden. Sie stehen aber auch dem Anwender direkt zur Verfügung, um damit für bestimmte Zwecke angepaßte Alternativlösungen zu den Programmen der übergeordneten Schicht zu schaffen. Besonders wichtig ist dabei ein verallgemeinertes I/O-Interface, das die Eigenschaften der wichtigsten, aber teilweise sehr unterschiedlichen Kommunikationsmechanismen von UNIX miteinander kombiniert. Darüber wurde dann auch eine Schnittstelle realisiert, um eine einfache Anbindung des experimentspezifischen Teils der eigentlichen Datenaufnahme an die bis dahin i. w. experimentunabhängigen Software-Komponenten vorzunehmen. Der experimentspezifische Teil konnte so als ein einziges Programm realisiert werden, das alle Informationen über die spezifischen Anforderungen und Randbedingungen eines konkreten Experiments enthält. Ein solches Programm kann vollkommen autark und — das ist ein besonderer Vorzug dieses Verfahrens — sogar auf einem anderen Rechner als die restliche Software die Datenaufnahme für ein beliebiges Experiment vornehmen.

5. Dispatcher

Die fünfte Schicht ist bereits Bestandteil des Datenaufnahmeprogramms. Sie bildet zusammen mit der übergeordneten Schicht die Schnittstelle zwischen allgemeiner Software und einem experimentspezifischem Systemkern. Sie setzt die abstrakten Systemaufrufe aus Schicht 4 zur Steuerung der Datenaufnahme in konkrete, zu deren Durchführung notwendige Anweisungen um bzw. gibt die erfaßten Meßdaten an die übergeordnete Schicht weiter.

6. Zustandsmaschine

Alle Aktivitäten innerhalb des Datenaufnahmeprogramms werden durch eine Zustandsmaschine gesteuert, die aufgrund der aus Schicht 5 kommenden Anweisungen in verschiedene Zustände übergehen kann. Die Anzahl der Zustände, die die Zustandsmaschine einnehmen kann, ist nur sehr begrenzt; sie ist damit ein sog. endlicher Automat[1]. Jede Anweisung wird

[1] oder auch Finite State Machine (FSM)

überprüft und wird, falls sie einen illegalen Zustandsübergang hervorrufen würde, zurückgewiesen. Damit ist sichergestellt, daß sich sowohl das Datenaufnahmeprogramm als auch das restliche System stets in einem eindeutigen Zustand befindet. Das ist wichtig, da Zustandsübergänge auch durch externe Ereignisse, wie sie in Form der Trigger-Signale vom Experiment auftreten, veranlaßt werden können.

7. Experimentunabhängige Verwaltung

Bei jedem erlaubten Zustandsübergang werden entsprechende Unterprogramme aufgerufen. Dabei werden insbesondere zwei, von der konkreten Experimentkonfiguration unabhängige Aufgabenbereiche abgedeckt. Der erste umfaßt die Verwaltung von Rechnerressourcen, die direkt bei der Datenerfassung genutzt werden. Dazu gehören z.B. die im Rahmen der Interruptverarbeitung erforderlichen Maßnahmen zur Kontextumschaltung oder auch nur das einfache Maskieren von Interrupts; auch die Verwaltung von Arbeitsspeicher zur Aufnahme von Meßdaten oder allgemeine statistische Aufgaben wie Zählen von Interrupts sind Bestandteil dieses Aufgabenbereichs. In MECDAS stehen hierfür eine Reihe von Standardroutinen zur Verfügung. Ebenso vom speziellen Experiment unabhängig sind einige Routineaufgaben im Bereich der Meßelektronik. Diese ist zwar für jedes Experiment bzw. Experimentieranlage individuell, doch erledigen einzelne Komponenten übergeordnete Aufgaben, die stets identisch behandelt werden müssen. Dazu gehört in erster Linie die Verriegelungselektronik, deren Zustand ganz eng an den des Datenaufnahmesystems gekoppelt sein muß. Auch das ist bereits in MECDAS standardmäßig implementiert.

8. Experimentspezifische Aktivitäten

Bei der Durchführung eines Experiments fallen experiment- bzw. detektorspezifischen Aufgaben an, bei denen spezielle Experimentierelektonik auf individuelle Art und Weise angesteuert werden muß. Diese Aufgaben müssen in den einzelnen Phasen der Datenerfassung korrekt ausgeführt werden, um einerseits bei einem Übergang der Zustandsmaschine die Apparatur in den verlangten Zustand zu bringen, andererseits die korrekten Verfahren bei der Datenaufnahme anzuwenden. Hierfür existieren definierte Schnittstellen für Unterprogramme, die vom Experimentator beim Aufsetzen des Experiments zur Verfügung gestellt werden müssen. Er kann dabei jedoch auf Unterprogramme aus der folgenden Schicht zurückgreifen, die ihm die Erledigung vieler Routineaufgaben abnehmen oder erleichtern.

9. Gerätetreiber

Neben anderen Unterprogrammbibliotheken für allgemeine Routinearbeiten ist eine Bibliothek noch von besonderer Bedeutung. Sie enthält in Form initialisierter Datenstrukturen und Programmcode Beschreibungen für alle bisher in MECDAS eingesetzen Geräte vom einfachen ADC-Modul bis hin zum Frontend-Rechner. Diese Gerätebeschreibungen werden konsequent in Schicht 7 und 8 zur Erledigung hardwarespezifischer Aufgaben genutzt und vereinfachen die hier notwendige Software deutlich.

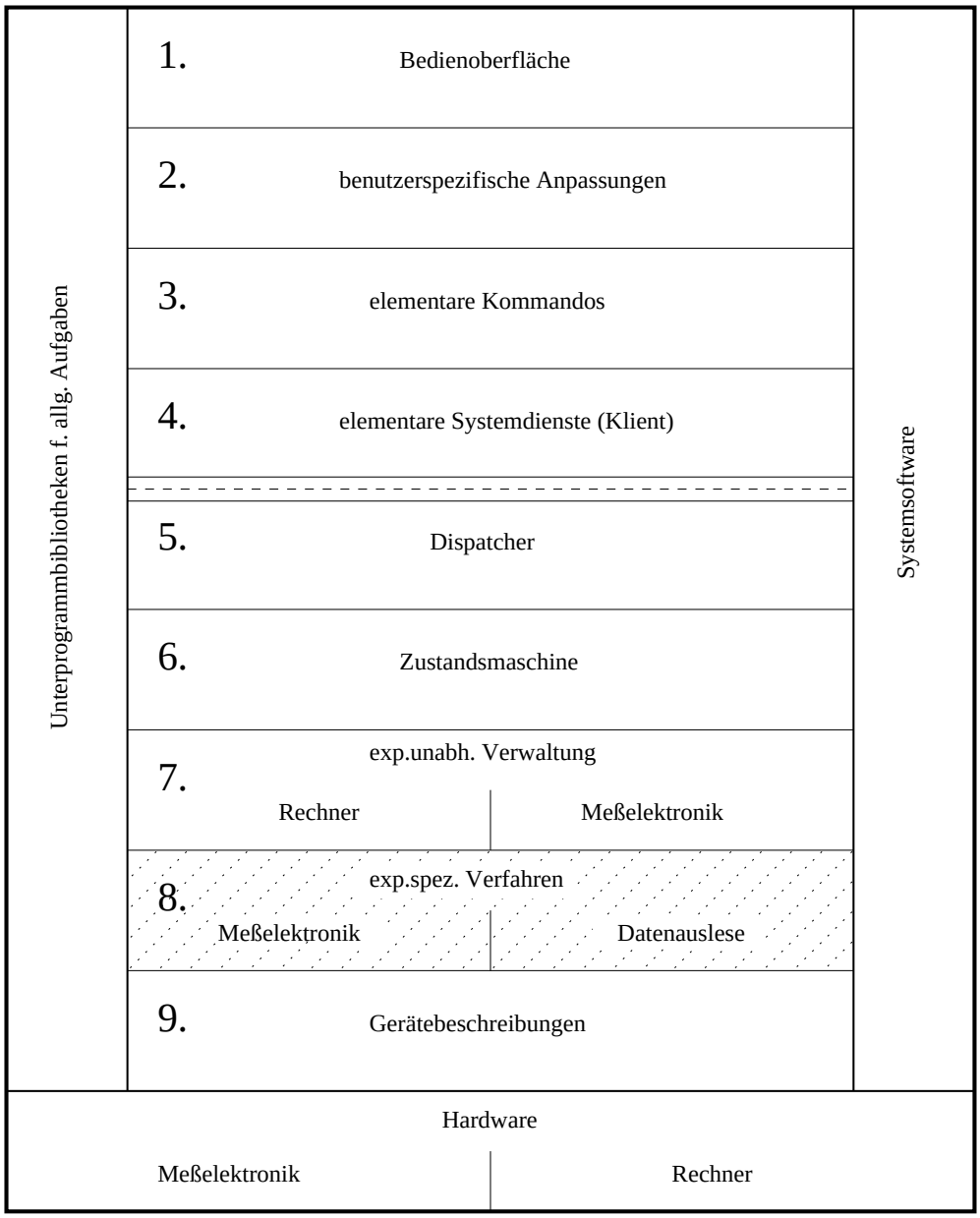

Abb. 5.1: Schichtenmodell der zentralen Datenerfassungs-Software von MECDAS. Der schraffierte Bereich kennzeichnet den Teil der Software, der experiment- bzw. detektorspezifische Komponenten enthält

10. Hardware
Die unterste Schicht wird durch Hardware repräsentiert, die zur Datenerfassung notwendig ist. Dazu gehören neben der Meßelektronik auch die Rechner, die hierzu eingesetzt werden.

Abbildung 5.1 gibt eine schematische Darstellung des beschriebenen MECDAS-Schichtenmodells wieder. Hier wird zusätzlich deutlich, daß auf allen Ebenen neben einem Satz aus Unterprogrammen für allgemeine Aufgaben stets auch von herkömmlicher Systemsoftware Gebrauch gemacht wird, um Systemressourcen zu nutzen.

5.1.2 Das Client-Server-Konzept

Die Schnittstelle zwischen den Schichten 4 und 5, die auch in Abbildung 5.1 besonders gekennzeichnet ist, hat eine herausragende Bedeutung. Sie trennt, so wie das auch zwischen einem Betriebssystemkern und Anwender-Software der Fall ist, zwei komplett unterschiedliche Software-Bereiche voneinander ab, die nach dem Client-Server-Prinzip arbeiten. Oberhalb dieser Schnittstelle überwiegen allgemeine Aufgaben, die vom konkreten Experiment und dem apparativen Aufbau unabhängig sind und auch nicht mit hohen Echtzeitanforderungen verbunden sind. Sie sind in einer Reihe einzelner, unabhängiger Programme realisiert.

Die Software des zweiten Bereichs ist für die eigentliche Datenaufnahme zuständig. Sie besitzt dazu alle nötigen Informationen über Art und Umfang der Experimentelektronik und die Verfahren, wie sie anzusprechen ist und wie die eingelesenen Daten weiterverarbeitet werden sollen; die auszuführenden Aufgaben müssen unter harten Echtzeitbedingungen bearbeitet werden. Diese Software ist als ein in sich abgeschlossenes Programm realisiert, das zwei grundlegende Aufgaben zu erfüllen hat. Zum einen führt es die Datenaufnahme aus, andererseits aber stellt es seine Funktionalität den übergeordneten Software-Schichten zur Verfügung. Aus deren Sicht arbeitet es als ansonsten passiver Server, der auf Anforderung Dienstleistungen im Rahmen der Datenaufnahme an die übergeordneten Klienten bereitstellt.

In Abbildung 5.2 ist die Client-Server-Programmstruktur der Datenerfassungs-Software schematisch dargestellt. Sie zeigt in einer etwas vereinfachten Form das Schichtenmodell aus Abb. 5.1 kennzeichnet jedoch deutlich die Bereiche, die durch abgeschlossene Programme abgedeckt werden. Aufgrund des Client-Server-Prinzips, das auch in anderen Bereichen der Datenverarbeitung sehr erfolgreich eingesetzt wird, sind Funktionalität und Implementation von Klient und Server klar voneinander abgegrenzt. Zwischen ihnen besteht eine recht lose Kopplung, die jedoch über eindeutige Schnittstellen streng definiert ist. Damit lassen sich sowohl Klienten als auch Server relativ einfach gegen kompatible Gegenstücke austauschen.

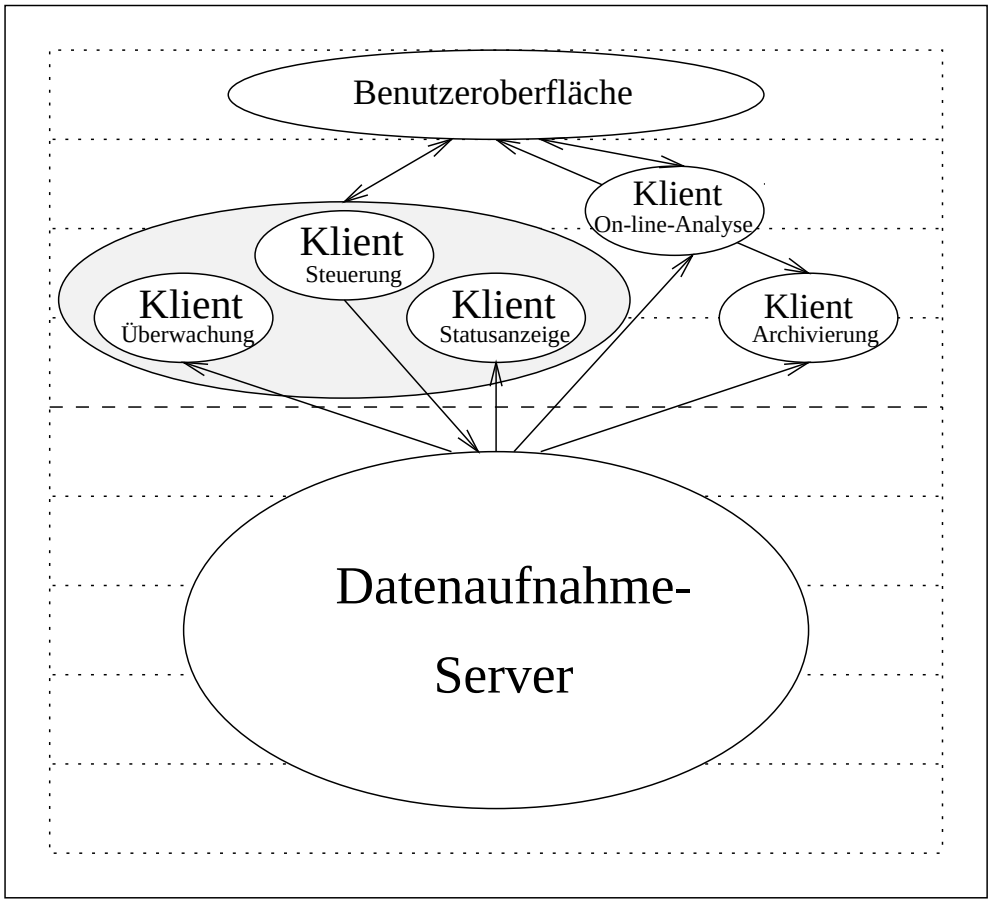

Abb. 5.2: Die Client-Server-Struktur der zentralen Datenerfassungs-Software

5.2 Der Datenaufnahme-Server

Der Datenaufnahme-Server ist der Teil von MECDAS, der die zentralen Aufgaben der Datenaufnahme ausführt. Er vereinigt als einzelnes, eigenständiges Software-Modul in sich alle Informationen, die notwendig sind, um die korrekte Ansteuerung und Auslese der Meßelektronik an einem konkreten Detektorsystem vorzunehmen. Es ist damit der einzige Teil innerhalb der Datenerfassungs-Software, der Kenntnis über experiment- oder detektorspezifische Details hat. MECDAS ist so konzipiert, daß diese Information an keiner anderen Stelle mehr für die Datenerfassung gebraucht wird.

Das Datenaufnahmeprogramm stellt im Sinn der objektorientierten Programmierung ein typisches Objekt dar. Es ist sehr gut gekapselt. Für eine korrekte Funktion benötigt es weder selbst Information von außen, noch sind zu seiner Benutzung Kenntnisse seiner Interna notwendig. Es kommuniziert lediglich durch Austausch von Nachrichten mit seiner Umwelt; nur diese Schnittstelle muß bekannt sein, um das Objekt sinnvoll einsetzen zu können. Das Programm besitzt eine beschränkte Anzahl von Methoden, die durch bestimmte Nachrichten aktiviert werden können.

Neben der Auslagerung vieler Aufgaben gemäß der Schichten 1 bis 4, die nicht direkt synchron mit der ereignisweisen Datenaufnahme bearbeitet werden müssen, wurde auch im Datenaufnahmeprogramm eine klare Abgrenzung unterschiedlicher Aufgabenbereiche nach dem Schichtenmodell vorgenommen, die eine Trennung experimentspezifischer Aktivitäten von allgemeinen Routineaufgaben zuläßt. Es wurde Software für diese allgemeinen Aufgaben — im folgenden mit dem Begriff Kernsoftware bezeichnet — in Form eines Programmrahmens realisiert, der durch Ergänzung mit experimentspezifischen Komponenten zu einem vollwertigen Programm gemacht werden kann.

5.2.1 Die Aufgaben der Kernsoftware

5.2.1.1 Datenaufnahme

Neben den bei der Datenaufnahme anfallenden experiment- und detektorspezifischen Aufgaben, gibt es eine ganze Reihe allgemein zu lösender Routineaufgaben, ohne deren korrekte Erledigung die Datenaufnahme nicht arbeiten würde.

Interruptverarbeitung
: Der Rechner muß in der Lage sein, eine Unterbrechungsanforderung aufgrund eines vom Experiment kommenden Trigger-Signals zu erkennen und korrekt zu bearbeiten. Dazu gehört die korrekte Initialisierung der Interrupt-Quelle und des Interrupt-Systems der CPU. Beim Auftreten eines Interrupts müssen Vorkehrungen zur Kontextumschaltung getroffen werden. Die experimentspezifische Auslese-Routine muß gestartet werden und nach deren Beendigung der Kontext des unterbrochenen Programms wiederhergestellt werden. Während des laufenden Betriebs müssen Interrupts an kritischen Stellen gesperrt und wieder frei gegeben werden können.

Routineaufgaben beim Auslesen bestimmter Meßelektronik
: Beim Auslesen von Standard-Meßelektronik in CAMAC oder Fastbus gibt es eine Vielzahl von Aufgaben, die allgemein abgehandelt werden können und keine individuelle Implementation erfordern.

Verpacken der Meßdaten
: Bei Experimenten, wie sie z. B. mit der Drei-Spektrometer-Anlage durchgeführt werden, die eine große Menge an Daten in variabler Anzahl liefern, ist eine eineindeutige Kodierung der Daten notwendig, die es erlaubt, den Datensatz in seine Originalform zu rekonstruieren, ohne unnötig Tausende an unnützer Daten mit abzuspeichern. Daneben ist es auch sinnvoll, noch Zusatzinformation zu den Daten zu packen, wie z. B. die Beschreibung der Experimentkonfiguration, regelmäßige Zeitstempel usw.

Steuerung der Verriegelungselektronik

Es müssen Mechanismen existieren, die es erlauben, sowohl durch den Benutzer bzw. die übergeordnete Software, als auch aufgrund interner Vorgänge, die für den korrekten Betrieb des Experiments notwendige Verriegelungselektronik zu steuern. Auch eine selbstverriegelnde Elektronik muß zumindest nach erfolgreicher Bearbeitung eines Ereignisses wieder freigegeben werden. Es sollte aber auch möglich sein, die Verriegelung zu irgendeinem Zeitpunkt rechnergesteuert ein- und auszuschalten.

Verwaltung der Meßdaten

Die während der Datenaufnahme erfaßten Daten müssen zwischengespeichert und anschließend auf geeignetem Weg zur Archivierung weitergeleitet werden. Auch Daten aus anderen Quellen, die z. B. den Status der durch die Datenaufnahme nicht direkt zugänglichen Experimentierapparatur beschreiben, müssen in den Meßdatenstrom eingereiht werden. Für diese Zwecke ist es notwendig, entsprechend Speicherplatz zur Verfügung zu stellen und zu verwalten. Zur Archivierung ist ein asynchrones Verfahren unter Verwendung mehrerer Datenpuffer sinnvoll, bei dem die Datenaufnahme bereits neue Daten in einen Puffer abspeichern kann, während der Weitertransport der vorher erfaßten Daten noch läuft. Daneben ist es ganz hilfreich, wenn die Daten auch für die On-line-Analyse noch verfügbar sind.

Behandlung von Fehlern

Eine sehr wichtige und kritische Angelegenheit ist die Behandlung von Fehlern, die in unterschiedlichster Art und Weise während der Datenerfassung auftreten können. Dabei kann man unterscheiden zwischen Laufzeitfehlern innerhalb der Software, Fehler durch defekte Rechner-Hardware und Fehler durch nicht korrekt arbeitende Experimentelektronik. Hier sind z. T. sehr aufwendige Maßnahmen notwendig, um die einzelnen Fehler geeignet zu behandeln und entsprechende Reaktionen zu veranlassen.

Sammeln von statistischer Information

In vielen Fällen kann es ganz sinnvoll sein, im Laufe eines Experiments oder einer Messung neben den Meßdaten anfallende Kenngrößen festzuhalten. Dazu gehören z. B. Start- und Stoppzeiten von einzelnen Messungen, Anzahl der erfaßten Ereignisse evtl. sortiert nach bestimmten Kriterien, Fehlerzähler usw.

Zusammenarbeit mit dem Server

Die genannten Aufgaben laufen zwar in einem gewissen Maße automatisch, gesteuert durch die vom Experiment kommenden Signale ab, doch muß ihre Abarbeitung durch die Software, die die Verbindung zur Außenwelt herstellt, koordiniert und gesteuert werden können. Es ist ein ständiger Datenaustausch zwischen diesen beiden Bereichen notwendig.

5.2.1.2 Service für die höheren Software-Schichten

Die Funktionalität der Datenaufnahme muß der übergeordneten Datenerfassungs-Software zugänglich gemacht werden. Dazu fallen ausschließlich allgemeine Routineaufgaben an:

Steuerung der Datenaufnahme
: Der Benutzer bzw. die übergeordnete Software muß in der Lage sein, die im vorangegangenen Abschnitt beschriebenen Aktivitäten bei der Datenaufnahme zu steuern. Messungen müssen gestartet und gestoppt und bei Bedarf kurzzeitig unterbrochen werden können. Hard- und Software muß über einfache Methoden in einen definierten Grundzustand gebracht werden können. Es muß möglich sein, Statusinformation auszutauschen.

Datentransfer
: Die aufgenommenen Daten müssen der übergeordneten Software zur Verfügung gestellt werden, damit sie dort archiviert und anderweitig weiterverarbeitet werden können. Dazu gehört z. B. eine On-line-Analyse der Daten oder ihre Auswertung für Zwecke der Experimentüberwachung. Aber auch ein Datentransport in die umgekehrte Richtung ist notwendig, um unabhängig von der ereignisgesteuerten Datenaufnahme erfaßte Daten einzubinden.

Client-Server-Kommunikation
: Sowohl bei einer Master-Slave-Konfiguration als auch in dem Fall, daß der Datenaufnahme-Server auf demselben Rechner wie die Klienten läuft, ist Software notwendig, die den Datenaustausch zwischen Server und Klienten erlaubt. Hierbei werden zum einen Steuerungsanweisungen an das Datenaufnahmeprogramm übertragen bzw. Statusinformationen zurückgeliefert, andererseits muß aber auch der Transfer der Meßdaten, die bei der Datenaufnahme anfallen, gewährleistet sein.

I/O-Service
: Einige Teile der Experimentelektronik — im wesentlichen im CAMAC-Bereich — werden nicht nur für die Datenaufnahme sondern auch für die davon unabhängige Steuerung der Experimentieranlage verwendet. Datenaufnahme und Steuerung werden aber von unterschiedlichen Rechnern aus vorgenommen. Um Kollisionen zu vermeiden, dennoch aber prinzipiell die Unabhängigkeit zwischen Datenaufnahme und Steuerung zu gewährleisten, muß durch entsprechende Software auf dem Datenaufnahmerechner für eine Koordinierung gesorgt werden.

Daneben fallen in beiden Bereichen allgemeine Verwaltungsaufgaben an. So muß z. B. die aktuelle Uhrzeit bei Bedarf festgestellt werden, und die Aufbereitung und Ausgabe von Diagnose- und Fehlermeldung ist zu erledigen.

5.2.2 Realisierung und Funktionsweise

Das Kernsoftware bildet die Grundlage für ein abgeschlossenes Programm, das die Aufgaben der Datenaufnahme abwickelt und gleichzeitig in der Lage ist, jederzeit auf Steueranweisungen durch den Benutzer zu reagieren oder andere der oben aufgeführten Serviceleistungen auszuführen. Erreicht wird das durch eine spezielle Struktur des Programmes, so wie in Abbildung 5.3 in vereinfachter Form dargestellt, in der eine ereignisgesteuerte Zustandsmaschine die zentrale Rolle spielt. Sie erlaubt es, die asynchronen Aktivitäten beider Aufgabenbereiche zu koordinieren.

Der Datenaufnahme-Server ist in zwei unterschiedlichen Varianten realisiert. Die erste ist seine Implementierung als herkömmliches Anwenderprogramm zur Ausführung auf einem Frontend-Rechner, der unter einem Betriebssystem wie UNIX oder OS-9 betrieben wird. In diesem Fall läuft das Programm wie jedes andere Anwenderprogramm ohne besondere Privilegien unter Kontrolle dieses Betriebssystems; es wird kein weiterer Rechner benötigt. Das Datenaufnahmeprogramm kann i. w. ohne Einschränkungen von den Diensten des Betriebssystems Gebrauch machen, ist aber auch an dessen Beschränkungen gebunden. Das bedeutet z. B., daß zum Informationsaustausch mit anderen Teilen der Datenerfassungs-Software komfortable Standard-Kommunikationsmechanismen verwendet werden können. Auf der anderen Seite jedoch kann damit, wenn es das Betriebssystem nicht anders vorsieht, evtl. der Verzicht auf die Ausnutzung der Interruptverarbeitung oder ein sehr eingeschränktes Echtzeitverhalten verbunden sein. Um diese Probleme zu mindern, ist in vielen Fällen ein Eingriff ins Betriebssystem unvermeidbar. Für OS-9 wurde zu Beginn der Entwicklung von MECDAS aus diesem Grund ein spezieller Gerätetreiber implementiert und in das Betriebssystem integriert. Er erlaubt die Verwendung von Interrupts bei der Datenaufnahme und stellt eine effiziente Kommunikationsschnittstelle zu den Datenerfassungsklienten zur Verfügung. Ohne eine solche Betriebssystemerweiterung, auf die unter UNIX bisher aus Portabilitätsgründen verzichtet wurde, bleibt bei dieser Variante zur Registrierung eines Triggersignals nur die rechenzeitintensive und unhandliche Polling-Methode und für die Kommunikation mit anderen Prozessen für Echtzeitverhältnisse weniger effiziente IPC-Mechanismen.

Die zweite Variante läuft in einer Master-Slave-Konfiguration. Sie nutzt die Verfügbarkeit mindestens eines weiteren Rechners (Slave) im Frontend-Bereich aus, der unter Echtzeitbedingungen arbeitet und in der Regel kein Betriebssystem besitzt, jedoch durch einen übergeordneten Rechner (Master) kontrolliert wird. In diesem Fall wird der in Bezug auf Echtzeitanforderungen anspruchsvolle Datenaufnahme-Server auf den Slave geladen und dort gestartet. Der Rest der Software bleibt weiterhin auf dem Master. Es findet damit eine Verteilung der Software auf verschiedene Rechner statt, die den unterschiedlichen Anforderungen wesentlich besser gerecht werden als vorher. Während übergeordnete Aufgaben komfortabel unter UNIX abgearbeitet werden können, läuft das Datenaufnahmeprogramm auf einem Rechner mit direktem Zugriff auf die Meßelektronik unter Echtzeitbedingungen. Dabei stellt es aber den unter UNIX arbeitenden Klienten seine Funktionalität voll zur Verfügung.

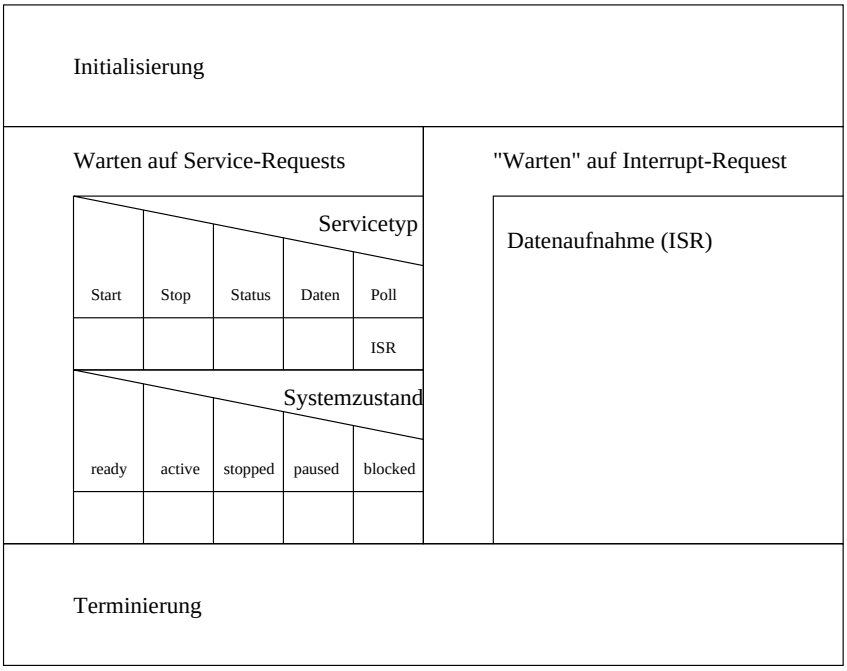

Abb. 5.3: Schematische Darstellung der Aktivitäten innerhalb des Datenaufnahmeprogrammes in Form eines vereinfachten Struktogramms. Es sind die beiden gleichzeitig zu bearbeitenden Aufgabenbereiche des Programms dargestellt: zum einen eine symbolische Schleife, die in der Interrupt-Hardware der Rechner-CPU realisiert ist, zum zweiten der Server-Teil, der auf Service-Requests wartet, und jede einzelne Anweisung individuell aber unter Berücksichtigung des Systemzustandes bearbeitet.

Aufgrund des Client-Server-Konzeptes unterscheiden sich die beiden Varianten nur minimal. Sie sind in ihrer Funktionalität vollkommen identisch und benötigen nur unterschiedliche Anbindungen an die verschiedenen Betriebssystemumgebungen. Unter UNIX (oder OS-9) stehen dafür die Standard-Bibliotheken des Systems zur Verfügung. Für den betriebssystemlosen Slave wurde ein eigenes an die Anforderungen der Datenaufnahme angepaßtes Systemsoftware-Paket implementiert, das sich auf nur wenige benötigte Funktionen beschränkt (siehe Abschnitt 5.6.2). Der wesentliche Unterschied zwischen den beiden Varianten liegt in der Kommunikation zwischen Klienten und Server. Während in dem einen Fall dafür Mechanismen zur Interprozeßkommunikation genutzt werden konnten, war in dem anderen Fall spezielle Software notwendig, die eine geeignete rechnerübergreifende Kommunikation — in diesem Fall über den VME-bus — gewährleistet. Für die Klienten ist diese Software gemeinsam in einem Kommunikationsmodul untergebracht, so daß sie zur Laufzeit von verschiedenen Verfahren Gebrauch machen können. Damit sind die Klienten in beiden Fällen identisch.

5.2.3 Datenaufnahme

5.2.3.1 Programmstruktur

Das Datenaufnahmeprogramm durchläuft im wesentlichen drei Phasen:

1. Initialisierungsphase
2. Bearbeitungsphase
3. Terminierungsphase

In der ersten und letzten Phase erfolgt die streng sequentielle Abarbeitung der oben beschriebenen Aufgaben zur Vorbereitung bzw. zur Beendigung der Datenaufnahme in einer vorgegebenen Abfolge. Neben der Bearbeitung einer Reihe von Standard-Aufgaben erfolgt hier auch die experimentspezifische Initialisierung von Rechner-Hardware und Experimentelektronik. Aber auch verwaltungstechnische Aufgaben zur Vorbereitung der Ausführung spezieller Auslese-Software werden hier vorgenommen.

Neben allgemeiner Software für Routineaufgaben werden eine Reihe von Unterprogrammen aufgerufen, die vom Experimentator zur Verfügung gestellt werden müssen. Sie sind über eindeutige Schnittstellen definiert und können einfache Variableninitialisierungen enthalten, aber auch ganz spezielle Algorithmen zur Bearbeitung der einen oder anderen Aufgabe umfassen. In der Regel kann der Experimentator dabei jedoch auf die Standard-MECDAS-Software mit ihren Konfigurationsmöglichkeiten zurückgreifen (s. Kap. 6).

Im Gegensatz zur Bearbeitungsphase sind hier keine Steuerungsmöglichkeiten durch den Benutzer vorgesehen, außer durch Kommandozeilenparameter, die beim Starten des Programmes übergeben werden. Erst in der Bearbeitungsphase ist das Programm in der Lage, einerseits auf Steueranweisungen des Benutzers zu reagieren und andererseits die Datenaufnahme durchzuführen.

5.2.3.2 Interruptgesteuerte Bearbeitung eines Ereignisses

Die Datenaufnahme findet in der Regel interruptgesteuert (entsprechend dem linken Teil von Abbildung 5.3) statt. Im folgenden seien kurz die Vorgänge beschrieben, die dabei zur Bearbeitung ein kernphysikalisches Ereignisses ablaufen:

1. Der Ablauf beginnt mit dem Auftreten eines zu untersuchenden **kernphysikalischen Ereignisses**, z. B. die Streuung eines Elektrons an einem Proton mit bestimmtem Energie- und Impulsübertrag.
2. Die **Detektoren** weisen diese Reaktion nach; d. h., im genannten Beispiel treten Elektronen und Protonen unter einem bestimmten Winkel und in einem begrenzten, von den Spektrometern überdeckten Energiebereich in die Detektoren ein, werden in Driftkammern und Szintillatoren registriert und durch die Čerenkov-Detektoren identifiziert.

3. Die Signale, die die Detektoren liefern, werden über Diskriminatoren und zusätzliche Logik dazu verwendet, das Ereignis nach weiteren Kriterien auszuselektieren. Gültige Ereignisse führen zu einem **Trigger-Signal**.

4. Aufgrund des Trigger-Signals wird die **Digitalisierung** der bis dahin nur analog vorliegenden Meßwerte gestartet. Spannungssignale von Photomultipliern werden durch ADCs umgewandelt, Zeiten bei Driftkammern und Triggerdetektoren durch TDCs konvertiert. Gleichzeitig wird die Meßelektronik verriegelt. Dadurch wird die Registrierung der während der nächsten Phasen der Datenaufnahme auftretenden Ereignisse unterdrückt.

5. Nach Beendigung der Konversionen wird ein **Signal** generiert, das dem **Frontend-Rechner** anzeigt, daß nun alle Meßwerte in digitaler Form vorliegen. Im wesentlichen ist dieses Signal das um eine gewisse Zeit verzögerte Trigger-Signal.

6. Mithilfe dieses Signals wird in der Regel eine **Unterbrechungsanforderung (Interrupt Request)** für den Prozessor generiert, die den Interrupt-Mechanismus zur Reaktion auf ein externes Ereignis einleitet. Ist aus Hard- oder Software-Gründen Interrupt-Verarbeitung nicht möglich, kann dieses Signal aber auch zum Polling benutzt werden.

7. Bis unmittelbar zu dem Zeitpunkt, wo der Interrupt-Request auftritt, arbeitet der Frontend-Rechner in der Regel die Server-Schleife ab. Falls die Interrupt-Verarbeitung für den Frontend-Rechner korrekt aufgesetzt und freigegeben ist, führt der Interrupt-Request unmittelbar nach Beendigung des gerade abgearbeiteten Maschinenbefehls zu einer **Unterbrechung der aktuellen Aktivitäten**.

8. Anschließend laufen **automatisch einige Aktionen zur Interruptbehandlung** ab. Der Prozessor wird in einen privilegierten Mode versetzt, rettet in der Regel Statusregister und Programmzähler (Program Counter), lädt die Adresse eines speziellen Unterprogrammes, der sog. Interrupt Service Routine (ISR), die ihm in der Initialisierungsphase auf geeignete Art bekannt gemacht wurde, in den Programmzähler und beginnt mit der Abarbeitung dieser Routine.

9. Um nach Beendigung der Interrupt Service Routine das unterbrochene Programm korrekt wieder fortsetzen zu können, muß der **Programmkontext dieses Programmes gesichert** werden. Neben den vom Prozessor selbsttätig geretteten Daten werden dazu einige, teilweise auch anwendungsspezifische Informationen durch explizite Anweisungen zu Beginn der ISR abgespeichert. Wichtig sind insbesondere die Prozessor-Register, die in der ISR verwendet werden und dadurch ihren ursprünglichen Inhalt verlieren. Mit dieser geretteten Information kann dann bei Beendigung der ISR der Programmkontext wieder vollständig restauriert werden.

10. Während der Datenaufnahme können **Laufzeitfehler** auftreten, die einer gesonderten, vom eingesetzten Prozessor abhängigen Behandlung

bedürfen. Da diese kontextabhängig sind, müssen bei jeder Kontextänderung dafür einige **Initialisierungsaufgaben** verrichtet werden.

11. Ansonsten fallen noch einige **allgemeine Aufgaben** wie Initialisieren von Variablen für die Pufferverwaltung oder statistische Zwecke an. Es wird überprüft, ob ein gültiger Datenpuffer vorhanden ist, um die nun zu erfassenden Daten aufzunehmen, und bei Bedarf neu zur Verfügung gestellt.

12. Schließlich wird die **Ausleseroutine gestartet**, die speziell auf das laufende Experiment und die individuelle Konfiguration des Detektorsystems zugeschnitten ist.

13. Es folgt die konkrete **Auslese der Meßelektronik** basierend auf den vom Experimentator vorgegebenen Anweisungen und Informationen. ADCs, TDCs, Zähler oder andere I/O-Einheiten werden sukzessive über CAMAC, Fastbus oder andere Schnittstellen angesprochen und liefern nach und nach Informationen, die das registrierte Ereignis beschreiben.

14. Hand in Hand mit der Auslese geht die **Zwischenspeicherung der aufgenommenen Daten**, um mit ihnen on-line arbeiten zu können und sie für die spätere Archivierung aufzubewahren. Dabei wird auf ein Minimum beschränkte, jedoch zur eindeutigen Identifizierung der Daten ausreichende Information mit festgehalten, um bei der Auswertung der Daten eine vollständige Rekonstruktion des erfaßten Ereignisses zu gewährleisten. Auch hier liegt im allgemeinen Art, Umfang und Integrität der Daten in der Verantwortung des Experimentators.

15. Nach dem Auslesen der **Meßelektronik** muß diese **zurückgesetzt** werden, um sie wieder in der Lage zu versetzen, die Daten für das nächste Ereignis aufzunehmen. Auch hier sind oftmals indiviuelle Vorgehensweisen notwendig mit Kenntnissen über Struktur und Interna der Meßelektronik.

16. Die aufgenommenen Daten werden schließlich mit **zusätzlicher Information** versehen, die für eine eventuell noch erforderliche, asynchron zur Datenaufnahme ablaufenden Formatierung und die anschließende Archivierung der Daten benötigt wird.

17. Damit sind die experimentspezifischen Aufgaben bei der Datenaufnahme abgeschlossen. Es folgen noch einige allgemeine Tätigkeiten zur **Abschlußbearbeitung** des Ereignisses.

18. Der **Datenpuffer**, in dem die verpackten und zu archivierenden Daten abgelegt sind, wird überprüft. Ist er voll, wird er zur Archiverung, die dann auf Server-Ebene stattfindet, freigegeben.

19. Unter der Voraussetzung, daß bei der Pufferverwaltung keine Fehler auftraten, und es mit dem aktuellen Zustand der Zustandsmaschine verträglich ist, wird die zuvor automatisch mit der Registrierung des Ereignisses vorgenommene **Verriegelung der Meßelektronik wieder aufgehoben**; die Meßelektronik kann wieder ein neues Ereignis registrieren.

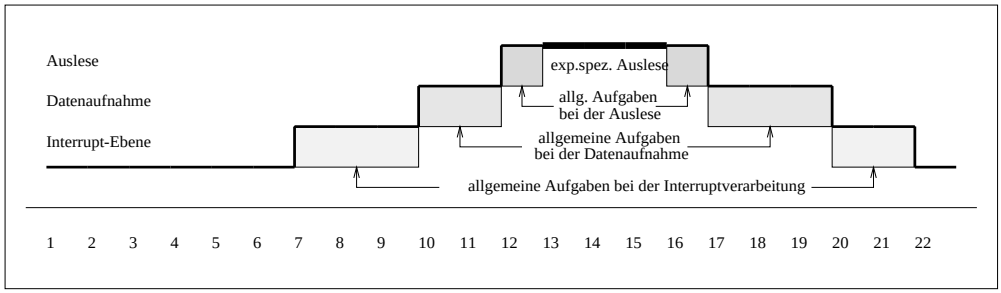

Abb. 5.4: Die wesentlichen Phasen bei der Datenaufnahme für ein kernphysikalisches Ereignis. Die Rechneraktivitäten bewegen sich auf vier Ebenen: Server-, Interrupt-, Datenaufnahme- und Auslese-Ebene.

20. Es folgt die in Schritt 9 vorbereitete **Restaurierung des Kontextes** des unterbrochenen Programmes. Alle wichtigen Variablen und Register-Inhalte werden wieder in den Zustand gebracht, den sie unmittelbar vor der Unterbrechung hatten.

21. Über einen entsprechenden Maschinenbefehl wird schließlich die privilegierte **Interrupt-Ebene verlassen**. Programmzähler und Statusregister werden in den Zustand vor der Unterbrechung gebracht und damit das Programm exakt an der Stelle weitergeführt, wo es vorher unterbrochen wurde. Bei korrekt restauriertem Kontext wirkt sich die Unterbrechung bis auf die zeitliche Verzögerung in keiner Weise auf das unterbrochene Programm aus.

22. Das Programm, in diesem Fall die Server-Schleife, wird so **fortgeführt** als wäre nichts geschehen, kümmert sich weiterhin um Service-Anforderungen der übergeordneten Software und „wartet“ auf den nächsten Interrupt.

Diese Vorgänge wiederholen sich für jedes Ereignis immer wieder bis die Messung beendet wird. Abbildung 5.4 veranschaulicht diese Vorgänge in einer vereinfachten, schematischen Darstellung. Sie macht deutlich, daß sich die Rechneraktivitäten bei der Datenaufnahme auf vier verschiedenen Ebenen bewegen:

Nach Ablauf der Vorgänge innerhalb des Detektorsystems und der Meßelektronik ohne direkte Rechnereinwirkung finden alle Aktivitäten auf Interrupt-Ebene statt (Schritt 7 bis 21). Sie lassen sich grundsätzlich unterteilen in allgemeine, für eine korrekte Interrupt-Verarbeitung notwendige Funktionen, die in der Regel eine starke Maschinenabhängigkeit aufweisen, und einen problemorientierten Bereich. Dieser Bereich (Schritt 10 bis 19) entspricht einer zweiten logischen Ebene. Er umfaßt in diesem Fall alle Aktivitäten für die eigentliche Datenaufnahme. Diese Software ist nicht mehr maschinenabhängig. Auch geht hier, bis auf prinzipielle Einschränkungen, die mit der Interrupt-Verarbeitung stets verbunden sind, nicht die Tatsache ein, daß der Code Bestandteil einer Interrupt Service Routine ist. Vielmehr läßt er sich vollständig unverändert auch im Polling-Betrieb verwenden.

Auch auf der Datenaufnahme-Ebene läßt sich wieder eine Unterteilung in allgemeine, anwendungsorientierte, aber nicht experimentspezifische Aufgaben machen und den Teil der Software, der die konkreten Vorgänge beim Ansprechen der individuellen Meßelektronik beschreibt. Diese Software entspricht schließlich der dritten Ebene (Schritt 12 bis 16). Sie ist in aller Regel durch ein Unterprogramm realisiert, das vom Experimentator bereitgestellt werden muß. Während bis hierhin alle anderen Aufgaben so allgemein sind, daß sie von Standard-Software abgedeckt werden können, sind hier nun ganz individuelle Aktionen notwendig. Sie hängen zum einen von verwendeten Detektoren und Ausleseelektronik ab, sind zum anderen aber auch durch spezielle Bedürfnisse des Experiments vorgegeben. Der Experimentator muß Programmcode zur Verfügung stellen, der die experimentellen Anforderungen widerspiegelt. Der Code muß angeben, welche Daten, wie und in welcher Reihenfolge aufgenommen werden sollen. Spezielle Vorschriften zur selektiven Auslese müssen ebenso explizit programmiert sein wie eine individuelle Verarbeitung der Daten.

Im Standard-Fall, insbesondere bei den Experimenten an der Drei-Spektrometer-Anlage, muß der Experimentator dieses Unterprogramm jedoch in der Regel nicht selbst programmieren, sondern kann von eigens dafür konzipierter Konfigurations-Software Gebrauch machen, die ihn so weit wie möglich beim Aufsetzen eines neuen Experiments unterstützt, und wesentliche Teile des experimentspezifischen Programmcodes generiert (siehe Kapitel 6).

Die letzte Ebene ist die Server-Ebene. Hier wird auch für das endgültige Verpacken der Daten und ihren Weitertransport zur Archivierung und On-line-Auswertung gesorgt.

5.2.3.3 Verwaltung der Meßdaten

Hauptaufgabe der Datenaufnahme-Software ist die Erfassung der Meßdaten. Diese müssen jedoch auch archiviert und u. U. auch anderweitig weiterverarbeitet werden. Um die Beeinflussung der Datenaufnahme durch die anschließende Bearbeitung der Daten möglichst gering zu halten, wird die Weiterverarbeitung der aufgenommenen Meßdaten unter MECDAS grundsätzlich asynchron und gepuffert vorgenommen. So wird eine sehr gute Enkopplung zwischen Datenaufnahme und Archivierung oder On-line-Analyse erreicht. Durch die Pufferung werden statistische Schwankungen bei der Datenaufnahme in bezug auf Ereignisrate und Datenmenge ausgeglichen. Transport und Archivierung unterliegen damit keinen Echtzeitanforderungen und behindern, solange die Daten im zeitlichen Mittel schnell genug abtransportiert werden können, die Datenaufnahme nicht. Verbessert wird dieses Verhalten noch dadurch, daß eine einfache Verteilung dieser Aufgaben auf mehrere Rechner möglich ist. Damit kann der unter Echtzeitbedingungen arbeitende Datenaufnahmerechner fast vollständig von Archivierungsaufgaben entbunden werden und steht fast ausschließlich für die Datenaufnahme zur Verfügung.

Der eingesetzte Pufferungsmechanismus verwendet mehrere Datenpuffer, deren Anzahl und Größe nicht fest vorgegeben sind, sondern an besondere An-

forderungen angepaßt und in der Initialisierungsphase des Systems eingestellt werden können. Die Puffer sind aber stets so groß, daß die Daten einer größeren Anzahl von Ereignissen hineinpassen. Die Datenpuffer sind in sog. Queues[2] organisiert. Das sind Verwaltungseinheiten, mit denen jeweils mehrere Puffer zu Gruppen zusammengefaßt werden können. Es existieren Mechanismen, mit denen ein Puffer in eine solche Gruppe eingefügt oder aus ihr entfernt werden kann. Die Besonderheit einer Queue ist, daß die Puffer nur in der Reihenfolge herausgenommen werden können, in der sie auch eingefügt wurden. Sie realisieren damit das sog. FIFO-Prinzip[3]. Es gewährleistet, daß die Daten, wenn auch asynchron, stets streng in ihrer zeitlichen Abfolge verarbeitet werden.

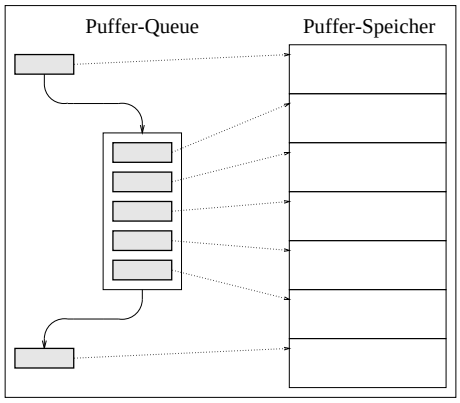

Abb. 5.5: Schematische Darstellung einer als verkettete Liste realisiert Queue, die i. w. Zeiger auf die zu verwaltenden Speicherbereiche enthält.

In MECDAS sind die Queues als lineare Listen implementiert. Zusammen mit einem Satz von Unterprogrammen zu ihrer Manipulation definieren sie eine in sich abgeschlossene Objektklasse, die in vielen Bereichen der Datenerfassungs-Software ihren Einsatz findet.

Mit dem Ziel, die vier Aufgabengebiete Datenaufnahme, Verpacken, Archivierung und On-line-Analyse der Daten möglichst gut zu entkoppeln, wurden innerhalb des Datenaufnahmeprogrammes mehrere Verwaltungseinheiten aus jeweils zwei bzw. drei Queues vorgesehen. Die erste verwaltet die Puffer, die unmittelbar zur Datenaufnahme benötigt werden. Sie besteht aus der sog. *Empty Queue* zur Aufnahme von freien Puffern und der *Full Queue* für gerade von der Datenaufnahme gefüllte Puffer, die zur weiteren, asynchron zur Datenauslese ablaufenden Verarbeitung bereit stehen. Dieser zweite Aufgabenbereich dient der Erledigung von Aufgaben, die nicht zwingend synchron zur Datenauslese bearbeitet werden müssen. Sie werden in den Zeiten verrichtet, in denen der Prozessor nicht mit der Auslese eines Ereignisses beschäftigt ist, so daß die Bearbeitungszeiten nicht mit in die Totzeit eingehen, solange der Prozessor nicht überlastet ist. Zu diesen Aufgaben gehört insbesondere das endgültige Verpacken der Daten und die Vorbereitung zu ihrer Archivierung. In der Regel werden die minimal kodierten Daten mit zusätzlicher Information zur eindeutigen Kennzeichnung der Daten versehen (siehe auch Abschnitt 6.8) und in den Kreislauf der zweiten Queue-Gruppe gegeben. Während auch hier eine Queue der Verwaltung freier Puffer dient (*Free Queue*), hat eine zweite, die *Archive Queue*, die Aufgabe, die Daten zur asynchronen Archivierung zur Verfügung zu stellen. Die *Analyse-Queue* schließlich macht die Daten parallel zur Archivierung auch für eine On-line-Analyse verfügbar.

[2] deutsch: Schlangen bzw. Warteschlangen
[3] First In First Out

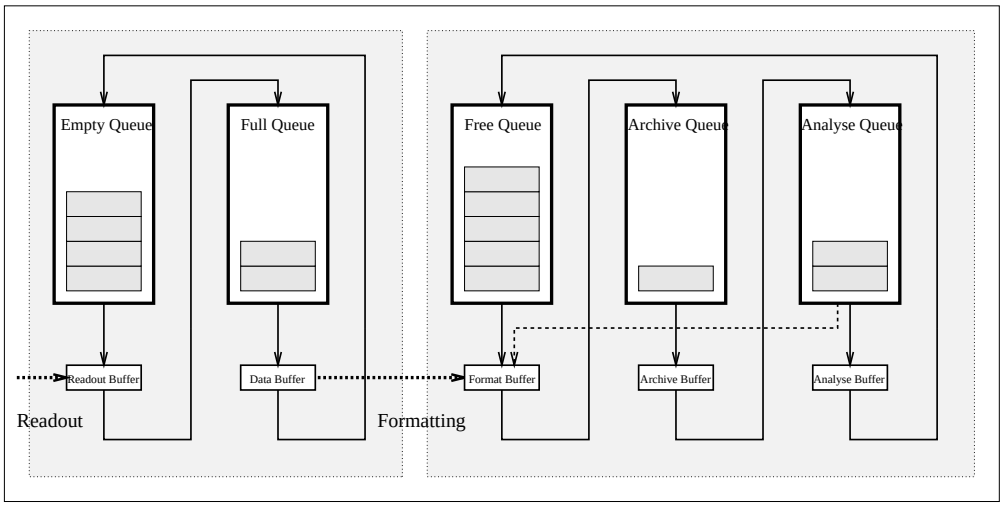

Abb. 5.6: Die prinzipielle Pufferorganisation bei der Datenaufnahme

Einfache Verwaltungsmechanismen sorgen für eine wirksame Flußkontrolle zwischen Datenaufnahme, Verpackung, Archivierung und On-line-Analyse. Sie stellen einerseits eine Synchronisierung mit der externen Archivierung sicher, so daß alle erfaßten Daten auch archiviert werden können, andererseits aber ist die On-line-Analyse nur lose angekoppelt und kann die Datenaufnahme nicht beeinträchtigen. Eine detailliertere Beschreibung der Pufferverwaltung kann der MECDAS-Dokumentation [Kryg95a] entnommen werden.

Neben den unmittelbar durch die ereignisweise arbeitende Datenaufnahme erfaßten Meßdaten fallen bei der Durchführung von Messungen auch Daten und Informationen aus anderen Quellen an, die dem System durch entsprechende Software von außen zur Verfügung gestellt werden. Von besonderer Bedeutung ist dabei der Zustand der Experimentierapparatur, der wichtige Informationen zur späteren Auswertung der Daten enthält. Um auch diese Daten in den Meßdatenstrom einzugliedern, werden dieselben Mechanismen wie für die unmittelbar aufgenommenen Daten verwendet. Das System besitzt damit mehrere prinzipiell gleichwertige Datenquellen, die ganz analog behandelt werden.

5.2.4 Die Server-Funktion

Die Aufgaben des Datenaufnahmeprogramms konzentrieren sich in seinem Hauptteil ausschließlich auf eine reine Server-Funktionalität, um der übergeordneten Software die Möglichkeit zu geben, die Datenaufnahme zu steuern, eine asynchrone Datenarchivierung vorzunehmen oder andere Dienstleistungen zu nutzen — unabhängig davon, ob Interrupt-Verarbeitung oder Polling zur Datenaufnahme eingesetzt wird. Nach Abschluß der Initialisierungsphase befindet sich das Programm i. w. in einer Schleife, in der es auf Service-Anforderungen verschiedenster Art wartet. Bei ihrem Auftreten werden sie in der Regel unter Zuhilfenahme der Zustandsmaschine indirekt bearbeitet: Ein Service-Request stößt, falls mit dem aktuellen Zustand verträglich, Zustandsübergänge an, die

ihrerseits instantan oder auch zu einem späteren Zeitpunkt für die notwendigen Aktivitäten sorgen, um die von den Klienten geforderten Aufgaben auszuführen.

Das Programm kontrolliert ständig seine Kommunikationskanäle, deren Funktionsweise in Abschnitt 5.4 beschrieben wird, und prüft nach, ob eine Nachricht von einem Klienten eingetroffen ist. Solche Nachrichten bestehen aus einer geringen Anzahl von 32-Bit-Worten und besitzen ein festes Format, dessen Struktur in Abbildung 5.7 wiedergegeben ist. Das Format wird sowohl für Service-Requests als auch für die dazugehörigen Antworten des Servers, die an die Klienten zurückgeschickt werden, verwendet. Es beschreibt die zu bearbeitende bzw. abgearbeitete Dienstleistung.

Servicecode
Servicetyp
Kommunikationskanal
Kommunikationskennung
Prozeßnummer
Datenadresse
Datenlänge
Fehlercode
Reservepuffer 1 . . . 7
Checksumme

Abb. 5.7: Format der ACQ-Nachrichten

Service-Code und -Typ spezifizieren dabei die Art des Dienstes. Die konkrete Bedeutung der übrigen Parameter variiert für die verschiedenen Dienstleistungen und wird letzendlich durch Service-Code und -Typ bestimmt. Die Dienstleistungen, die der Server zur Verfügung stellt, lassen sich entsprechend Abschnitt 5.2.1 zu folgende Gruppen zusammenfassen (Abb. 5.8):

- Steuerung der Datenaufnahme
- Transfer und Archivierung der Meßdaten
- On-line-Analyse der Meßdaten
- Integration externer Daten
- Zustandsüberwachung
- Software-Kontrolle
- I/O-Service

5.2.4.1 Steuerung der Datenaufnahme

Die erste Gruppe umfaßt eine Reihe elementarer Anweisungen, mit der die Datenaufnahme direkt kontrolliert werden kann. Das sind im einzelnen:

Initialisierung des Meßsystems, wodurch Hard- und Software generell in die Lage versetzt werden, Messungen durchzuführen. In der Regel ist die Initialisierung direkt mit dem Start des Datenaufnahmeprogrammes verbunden. Erst dann ist es überhaupt in der Lage, andere Service-Requests anzunehmen und zu verarbeiten und auf Signale der Meßelektronik zu reagieren.

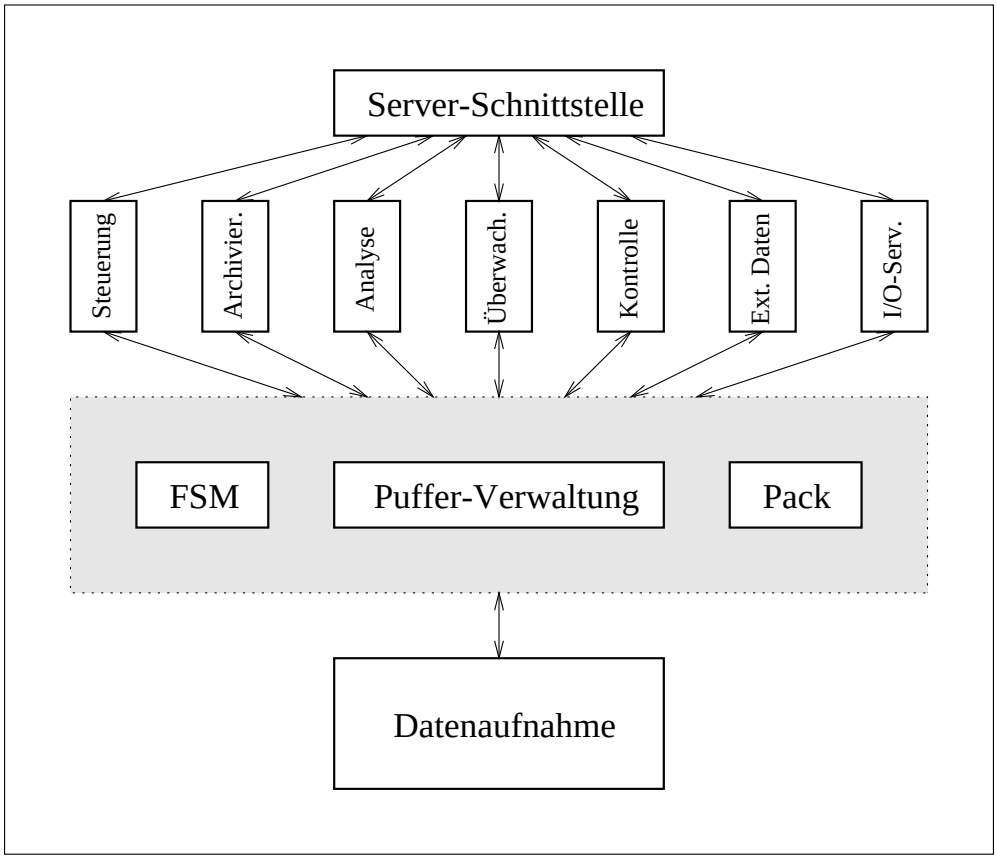

Abb. 5.8: Die wesentlichen Funktionselemente des Datenaufnahme-Servers

Terminierung des Meßsystems zur Desaktivierung von Hard- und Software; d. h., die Meßelektronik wird in einen Zustand versetzt, in dem sie keine weiteren physikalischen Ereignisse mehr aufnimmt, das Datenaufnahmeprogramm und alle davon abhängige Software terminiert auf herkömmliche Art und Weise. Anschließend können weder Signale von der Meßelektronik noch Service-Anforderungen von Klienten verarbeitet werden.

Start einer Messung nach erfolgter Initialisierung. Es wird eine explizite Freigabe der bis dahin gesperrten Verriegelungselektronik vorgenommen. Erst jetzt ist die Meßelektronik in der Lage, die Daten physikalischer Ereignisse zu verarbeiten und an den Rechner weiterzuleiten. Nach einigen Vorbereitungen kann nun der Rechner asynchron zu den restlichen Aktivitäten des Datenaufnahme-Servers per Interrupt oder Polling wie in Abschnitt 5.2.3.2 beschrieben die ereignisweise Datenaufnahme vornehmen.

Stopp einer laufenden Messung; dadurch werden Maßnahmen veranlaßt, die Meßelektronik wieder zu verriegeln und eine softwaremäßige Endbearbeitung der Messung vorzunehmen.

Unterbrechung (Pause) des Meßablaufs; i. w. wird dafür gesorgt, daß die Meßelektronik temporär verriegelt wird. Dadurch wird die Datenaufnahme

kurzzeitig angehalten, während sich am Zustand der laufenden Messung weiter nichts ändert.

Fortführung (Continue) einer unterbrochenen Messung; die Verriegelung wird aufgehoben, wodurch die Meßelektronik wieder sensitiv auf die Registrierung neuer Ereignisse wird — die Datenaufnahme wird fortgeführt.

Restart für Ausnahmesituationen, wenn aufgrund von fehlerhafter Hard- oder Software die Freigabe der automatisch verriegelten Elektronik nicht korrekt funktioniert und dadurch die Messung offensichtlich unmotiviert zum Stehen kommt. Das ist insbesondere für Tests wichtig, wenn an Hard- oder Software noch Arbeiten vorgenommen werden

Die Ausführung der mit den Steueranweisungen verbundenen Aktionen wird nie unmittelbar vorgenommen, sondern stets unter Berücksichtigung des Zustands, in dem sich das System gerade befindet. Dieser umfaßt Informationen über den Status der Messung, d. h. also, ob eine Messung gestartet oder gestoppt wurde und auch tatsächlich läuft, über die Datenarchivierung oder den Status der Verriegelungselektronik. Der Systemzustand wird durch einen Satz statischer Variablen innerhalb des Datenaufnahmeprogramms beschrieben, die nach der Initialisierung bis zur Terminierung permanente Gültigkeit besitzen und bei jeder Zustandsänderung aktualisiert werden. Solche Zustandsänderungen können durch Steueranweisungen oder andere Serice-Anforderungen angestoßen werden, können aber auch Folge der Aktivitäten im Rahmen der dazu asynchron ablaufenden Datenaufnahme sein. Damit ist letztendlich eine einfache Zustandsmaschine realisiert, die nicht nur alle Aktivitäten des Datenaufnahmeprogrammes steuert, sondern auch die anderen Komponenten des Systems koordiniert.

Jede Steueranweisung wird zuerst einmal auf Verträglichkeit mit dem aktuellen Systemzustand überprüft. Ist sie in diesem Zustand nicht erlaubt, wie z. B. ein Stopp bei noch nicht gestarteter Messung oder die Terminierung, während eine Messung noch läuft, wird sie sofort zurückgewiesen. Ansonsten bewirkt jede akzeptierte Steueranweisung in Abhängigkeit des aktuellen Zustands einen entsprechenden Zustandsübergang unter Ausführung der beschriebenen Aktionen. Unmittelbar im Anschluß daran wird die Steueranweisung in Form einer Nachricht beantwortet, die über den entsprechenden Kommunikationskanal an den Klienten zurückgeschickt wird. Diese Antwort besitzt dieselbe Struktur wie die Anweisung. Sie enthält u. a. einen gültigen Fehler-Code, der Auskunft darüber gibt, ob die Steueranweisung korrekt ausgeführt wurde und, falls das nicht der Fall war, was die Fehlerursache war.

5.2.4.2 Transfer und Archivierung der Meßdaten

Eine wichtige Aufgabe, die das Datenaufnahmeprogramm leisten muß, ist es, die Archivierung der erfaßten Meßdaten sicherzustellen, die letztendlich darin besteht, die Daten auf Massenspeicher wie Festplatte oder Magnetband abzulegen. Um jedoch Einschränkungen infolge schlechter Verfügbarkeit und Zugänglichkeit

z. B. im Frontend-Bereich oder ungenügender Leistungsfähigkeit der Datenerfassungsrechner bzw. starker Beeinträchtigung der Datenaufnahme zu vermeiden und gleichzeitig die Implementationsarbeit auf ein Minimum zu reduzieren, ohne an Flexibilität zu verlieren, wird der Massenspeicherzugriff nicht unmittelbar durch das Datenaufnahmeprogramm ausgeführt, sondern mithilfe eines zusätzlichen Prozesses, der ausschließlich für die Archivierung zuständig ist und mit dem Datenaufnahmeprogramm auf einfache Art und Weise kommuniziert. Dabei wird derselbe Mechanismus der Client-Server-Kommunikation ausgenutzt, der bereits für die Steuerung der Datenaufnahme verwendet wird. Damit funktioniert das Verfahren ohne besondere Vorkehrungen sowohl für die lokale als auch die Master-Slave-Variante des Datenaufnahmeprogramms gleichermaßen.

Um einen effizienten Datenaustausch zwischen Klient und Server zu gewährleisten, wird bei diesem Verfahren von der Möglichkeit Gebrauch gemacht, daß mehrere Programme, und insbesondere auf dem VMEbus sogar von verschiedenen Rechnern aus, auf denselben Speicherbereich (sog. Shared Memory) zugreifen können. Das Datenaufnahmeprogramm nimmt dabei eine passive Rolle ein und wird wie bei Steuerung und Datenaufnahme nur durch äußere Einwirkungen aktiv. Es gibt die aufgenommenen und zwischengespeicherten Meßdaten zu ihrer Archivierung nur dann weiter, wenn es durch ein übergeordnetes Programm dazu angewiesen wurde. Das Archivierungsprogramm wendet sich wie jeder andere Klient an den Datenaufnahme-Server und fordert mit jeder Nachricht einen Datenpuffer an, der im Laufe der Datenaufnahme asynchron durch die Ausleseroutinen gefüllt wurde.

Da die Archivierung bei diesem Verfahren voll unter Kontrolle eines herkömmlichen Anwenderprogramms steht, ist dieses Verfahren sehr flexibel und erlaubt einfach Eingriffe von Benutzerseite. Es ist dann möglich nicht nur standardmäßig zur Archivierung vorgesehene Programme, sondern auch andere Software, z. B. auch Programme zur On-line-Analyse, einzusetzen, um die aufgenommenen Daten weiterzuverarbeiten.

Besonders deutlich zeigen sich die Vorteile dieses Verfahrens in der Master-Slave-Konfiguration. Durch die klare Aufgabenteilung konnten auch hier wieder die Vorzüge eines verteilten Systems genutzt werden. Während der Slave weiterhin ohne Einschränkungen sich im wesentlichen auf die Datenaufnahme unter Echtzeitbedingungen konzentrieren kann, wird die Archivierung den Rechnern überlassen, die zum einen weniger stark in die Erledigung von Echtzeitaufgaben eingebunden sind, andererseits aber auch über die entsprechenden Systemressourcen verfügen.

5.2.4.3 On-line-Analyse

Mit denselben Mechanismen wie bei der Archivierung werden die Meßdaten auch zur On-line-Analyse zur Verfügung gestellt. Dazu stehen zwei weitere unabhängige Kommunikationskanäle zur Verfügung. Auf die Anforderung eines Klienten hin liefern sie Daten aus der in Abschnitt 5.2.3.3 beschriebenen *Analyse Queue*.

5.2.4.4 Integration externer Daten

Während für Archivierung und On-line-Analyse es lediglich notwendig war, daß Daten vom Server geliefert werden, wird für andere Aufgaben auch der Datentransport in die umgekehrte Richtung benötigt. Hier ist insbesondere die Einspeisung von Daten aus anderen Quellen von Interesse, die der herkömmlichen Datenaufnahme nicht dirket zugänglich sind. Dazu gehören in erster Linie Statusinformationen über die Experimentierapparatur wie z. B. Magnetfeldstärken der Spektrometer, Photomultiplier-Hochspannungen, Detektor-Schwellen oder Angaben über Koinzidenzeinstellungen [Kram95]. Auch für diesen Zweck werden die bisher erläuterten Kommunikationsmechanismen des Servers genutzt.

5.2.4.5 Zustandsüberwachung

Weitere Dienste, die der Datenaufnahme-Server zur Verfügung stellt, dienen der Zustandsüberwachung von Experiment und Datenerfassungssystem. Die wichtigsten von ihnen sind:

Status des Experiments; in diesem Fall werden vom Benutzer definierte, experimentspezifische Informationen von geringem Umfang, die im Rahmen der Datenaufnahme gesammelt werden und für die Kontrolle des Experiments wichtig sein können, der übergeordneten Software zur Verfügung gestellt. Typischerweise sind das globale Experimentparameter wie Meßzeit, Totzeit, Strahlstrom, Anzahl der Ereignisse unterschiedlicher Art usw. Auch hier wieder enthält die als Antwort zurückgeschickte Nachricht Adresse und Länge des Speicherbereichs, in dem diese Information abgelegt ist. A priori sind keine Randbedingungen an Inhalt oder Form gestellt. Es liegt im wesentlichen in der Verantwortung des Benutzers, die Daten z. B. in bezug auf Byte-Orientierung[4] oder Fließkommadarstellung so zu kodieren, daß sie auch auf den übergeordneten Rechnern mit anderen Architekturen

[4] maschinenabhängige Wertigkeit der einzelnen Bytes innerhalb eines aus mehreren (in der Regel 2, 4 oder auch 8) Bytes bestehenden Wortes

```
Processed Events   :   182765   (116.4 Hz)
dE/ToF Coincidences:       30 k (191.1 Hz)
Cerenkov           :        0 k (0.0 Hz)
Foerster           :      145 k (9.450 kHz (uA))
Photoeffect        :       70 k (9.434 kHz (uA))
Faraday            :       15 k (0.049 kHz (uA))
Real Time          :      104 s (89.2 %)
Dead Time          :       53 s (10.8 %)
Run Time           :      157 s (00:20:37 h)
```

Abb. 5.9: Typische Status-Meldung. Sie enthält experimentspezifische Information über globale Experimentparameter. Das Textlayout kann beliebig gestaltet werden.

```
State of data acquisition             : started - running (31)
Information about current run:
    Run Name       : 940613210305
    Sequence Number: 1
    Started at     : Mon Jun 13 21:03:05 1994
```

Abb. 5.10: Typischer Text, der bei einem State-Request zurückgeliefert wird. Die erste Zeile beschreibt den Status der Zustandsmaschine. Anschließend folgt Information zur aktuellen Messung.

lesbar sind. Als relativ einfache und universelle Form der Kodierung kann dabei mit der ASCII-Darstellung der Information, also der Darstellung als lesbaren Text, gearbeitet werden. Zu deren Umsetzung kann auf einfach zu handhabende Systemsoftware in Form der C-Bibliothek zurückgegriffen werden, die bei jeder herkömmlichen Anwendungs-Software Verwendung findet. Ein Vorteil der ASCII-Darstellung ist neben der Lesbarkeit auch die einfache Weiterverarbeitbarkeit mithilfe von Standard-UNIX-Werkzeugen. Abbildung 5.9 zeigt eine typische Statusmeldung.

State liefert allgemeine Angaben über den Zustand des Datenerfassungssystems in kompakter Form. Sie informieren z. B. darüber, ob eine Einzelmessung gerade im Gang ist, wann sie gestartet oder gestoppt wurde, oder stellen andere, nicht experimentspezifische Verwaltungsinformation zur Verfügung. Auch hier wird die Information in ASCII-Kodierung dargestellt. Abbildung 5.10 zeigt ein Beispiel.

Message History; liefert eine chronologische Liste der zuletzt aufgetretenen Fehler in Form von Klartextmeldungen. Auch Diagnose- und Debug-Meldungen (s. Abschnitt 5.3.1.1) können enthalten sein. Ein Beispiel dafür ist in Abbildung 5.11 gezeigt.

```
Jun 14 00:57:02 speka a: DEBUG started
Jun 14 00:57:02 speka a: DEBUG buffer size set to 65536
Jun 14 00:57:02 speka a: NOTICE e5_setup: slave:a
Jun 14 00:57:02 speka a: NOTICE e5_main boot 0
Jun 14 00:57:02 speka a: DEBUG slave:a
Jun 14 01:05:37 speka a: WARNING illegal interrupt
Jun 14 01:14:53 speka a: ERROR timeout in bbk_v2
Jun 14 01:21:17 speka a: ERROR timeout in bbk_v2
```

Abb. 5.11: Typische Liste an Diagnosemeldungen

```
Buffer size             :      64K        Total number of buffers :       16
Total memory size       :    1024K        Free limit              :        8
Number of frontends     :       1

                        FD BUSY FIRS FREE        KEY FROM  NIO  EXT  INT FREE
server                   4    0    0   32  771555119    3    0    0    0   31
analyse                 -1    0    0    0          0    0    0    0    0    0
ctrlclient              -1    0    0    0          0    0    0    0    0    0
ctrlserver               5    1    1    0          0    0    0    0    0    0
ctrldirect               6    1    0    1          0    0    0    0    0    0
reserve                 -1    0    0    0          0    0    0    0    0    0
special                 -1    0    0    4          0    0    0    0    0    0
runinfo                 -1    0    0    1          0    0    0    0    0    0
diagnose                -1    0    0    0          0    0    0    0    0    0
eventbuilder            -1    0    0    0          0    0    0    0    0    0
archive0                -1    0    0    0          0    0    0    0    0    0
archive1                -1    0    0    0          0    0    0    0    0    0
archive2                -1    0    0    0          0    0    0    0    0    0
archive3                -1    0    0    0          0    0    0    0    0    0
frontend0               -1    0    0    0          0    0    0    0    0    0
frontend1               -1    0    0    4          0    0    0    0    0    0
frontend2               -1    0    0    4          0    0    0    0    0    0
frontend3               -1    0    0    4          0    0    0    0    0    0
```

Abb. 5.12: Ausgabe der Information zur Pufferverwaltung

5.2.4.6 Kontrolle des Servers

Einige Dienste wurden nur implementiert, um bei der Software-Entwicklung Hilfestellung zu geben. Sie haben für den Normalbetrieb keinerlei Bedeutung.

Halt erzwingt das definierte Anhalten des Server-Programmes ohne Rücksicht auf den aktuellen Systemzustand.

Debug bietet die Möglichkeit, die Ausgabe von Diagnosemeldungen ein- bzw. auszuschalten. Fehlermeldungen bleiben davon unbeeinflußt.

Buffers liefert interne Informationen über den Zustand der Pufferverwaltung. Die Information liegt als ASCII-Text in tabellarischer Form vor, dessen Speicheradresse und Länge in der Antwortnachricht untergebracht ist.

In allen Fällen wird die Bearbeitung der Dienstleistungen unabhängig vom aktuellen Systemzustand vorgenommen. Der Zustand wird dadurch nicht verändert.

5.2.4.7 I/O-Service

Wie bereits bei der Aufgabenbeschreibung des Kernsystems deutlich gemacht wurde, werden einige Teile der Experimentelektronik sowohl für die Datenaufnahme als auch für die Steuerung zumindest einzelner Teile der Experimentieranlage verwendet. Um Kollisionen zwischen den beiden unabhängigen Software-Systemen zu vermeiden, stellt der Datenaufnahme-Server auch in diesem Bereich Dienstleistungen zur Verfügung, die dann vom Steuerungssystem [Kund95] verwendet werden können.

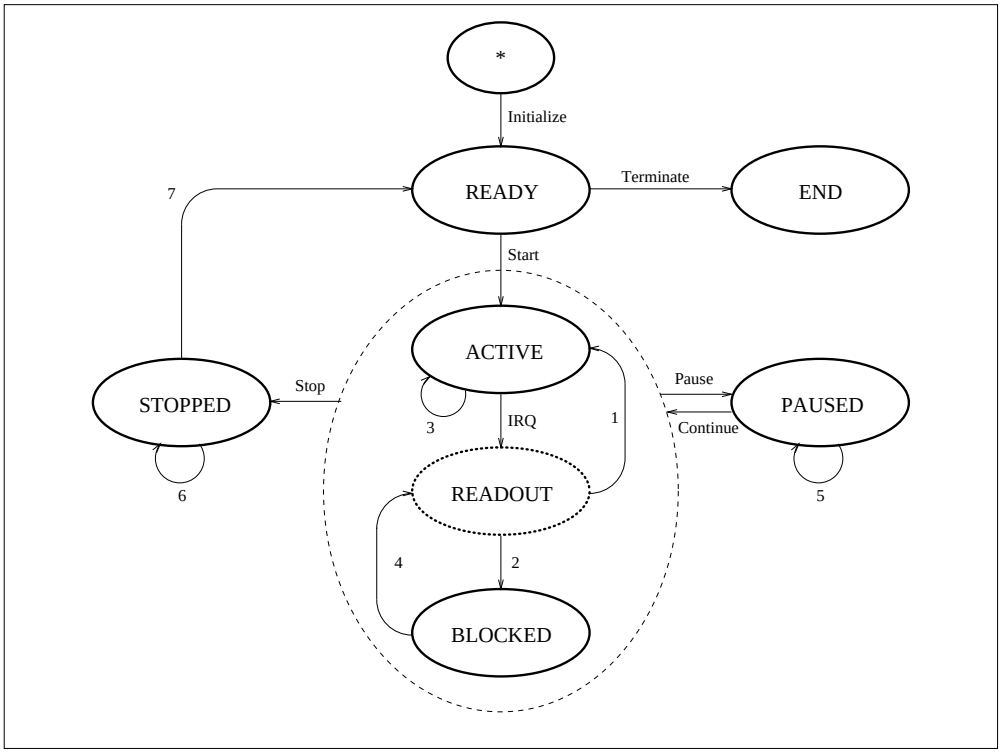

Abb. 5.13: Zustandsdiagramm des Datenaufnahmeprogramms. Neben den durch Steuerungsanweisungen bzw. Interrupts veranlaßten Zustandsübergängen sind von besonderer Bedeutung die mit 1–7 gekennzeichneten internen z. T. durch die Experimentverriegelung bzw. die Datenarchivierung veranlaßte Übergänge.

5.2.5 Zustandsdiagramm

Im Detail werden die Vorgänge innerhalb des Datenaufnahmeprogramms durch das in Abbildung 5.13 gezeigte Zustandsdiagramm beschrieben. Es repräsentiert eine Zustandsmaschine, die durch eine Reihe von Zuständen gekennzeichnet ist. Zustandsübergänge finden in der Regel durch äußere Einwirkungen statt:

kernphysikalische Ereignisse, bei deren Registrierung durch die Meßelektronik Signale an den Frontend-Rechner weitergeleitet werden.

Steuerungsanweisungen unmittelbar durch den Benutzer oder von übergeordneter Software, die den Ablauf des Experiments kontrolliert.

Datenfluß bzw. der Zustand der asynchronen Datenarchivierung

Timeouts, d. h. nach Ablauf bestimmter Zeitspannen automatisch durchgeführte Aktionen, um mögliche Verklemmungen zu vermeiden.

Die Abbildung zeigt alle Zustände, die bei der Datenaufnahme möglich sind, und die erlaubten Zustandsübergänge, mit denen ein Wechseln von einem zu einem anderen Zustand durchgeführt werden kann. Eine ausführliche Beschreibung des Zustandsdiagramms ist in der MECDAS-Dokumentation [Kryg95a] enthalten.

5.3 Datenerfassungs-Klienten

Wie bereits erwähnt ist die zentrale Datenerfassungs-Software von MECDAS in zwei grundlegende Software-Bereiche unterteilt. Der experimentnahe Bereich entsprechend den Schichten 5 bis 9 des MECDAS-Schichtenmodells ist durch ein in sich abgeschlossenes Datenaufnahmeprogramm realisiert, für das das im vorangegangenen Abschnitt beschriebene Kernsystem die Grundlage bildet. Es wird ergänzt durch die Software der Schichten 1 bis 4, die letztendlich dem Benutzer die Dienstleistungen des Datenaufnahmeprogramms zur Verfügung stellen. Diese Software besteht aus einer Reihe von Unterprogrammbibliotheken und darauf aufbauenden Programmen.

Die Bibliotheken enthalten neben Unterprogrammen für allgemeine Routineaufgaben einheitliche I/O-Mechanismen zum Datentransfer bzw. zur Kommunikation mit dem Datenaufnahmeprogramm. Sie kommen in den Standard-Dienstprogrammen von MECDAS zu Einsatz, können aber auch für eigene Anwendungen des Experimentators benutzt werden. Die Dienstprogramme stellen jedoch ein recht flexibles Baukastensystem zur Verfügung, so daß eine weitergehende Programmierung in der Regel nicht notwendig ist. Die Realisierung neuer Anwendungen beschränkt sich dann im wesentlichen auf das Schreiben geeigneter Kommadoprozeduren.

5.3.1 Unterprogrammbibliotheken

5.3.1.1 Allgemeine Routineaufgaben

Zur einheitlichen Bearbeitung allgemeiner Routineaufgaben stehen eine Reihe von Unterprogrammen zur Verfügung. Sie stellen elementare Dienstleistungen zur Verfügung und werden zu einem Teil bereits in Datenaufnahmeprogramm genutzt. Im folgenden seien die wichtigsten von ihnen beschrieben.

Optionen
: Eine wichtige Aufgabe ist die einheitliche Behandlung von Kommandozeilenargumenten von Programmen. Damit lassen sich die Programme sehr einfach mithilfe von Optionen steuern. Einige dieser Optionen besitzen bei allen Programmen die gleiche Bedeutung. Andere Optionen können ganz speziell für das jeweilige Programm sein. Sie alle bieten die Möglichkeit, ein Programm bei seinem Start zu beeinflussen und ihm variable Parameter mitzugeben, mit deren Hilfe der Ablauf des Programmes gesteuert werden kann.

Voreinstellungen
: Es stehen Mechanismen zur Verfügung, mit deren Hilfe die Voreinstellungen bestimmter Programmparameter verändert werden können. Hierbei werden zum einen die oben beschriebenen Kommando-Optionen verwendet, andererseits aber auch die in UNIX verfügbare Möglichkeit, sog.

Environment-Variablen zu definieren. Wurde ein Parameter explizit in einer Option spezifiziert, wird dieser Wert als Voreinstellung genommen. Ist das nicht der Fall, überprüft die Software, ob eine entsprechende, in aller Regel programmspezifische Environment-Variable gesetzt ist. In diesem Fall wird deren Wert zur Initialisierung des fraglichen Parameters verwendet. Erst wenn auch diese Alternative nicht zutrifft, werden die im Programmcode festgelegten Voreinstellungen benutzt.

Meldungen

Mithilfe spezieller Unterprogramme ist es möglich, auf einfache Art und Weise einheitliche Meldungen für Diagnosezwecke zu generieren. Dabei wird unterschieden zwischen Fehlermeldungen, Warnungen, Notizen, Informationshinweisen und Debug-Meldungen. Sie besitzen alle unterschiedliche Prioritäten, mit denen sie ausgegeben werden sollen. Fehlermeldungen werden stets ausgegeben; ihre Ausgabe läßt sich nicht beeinflussen. Die Ausgabe aller anderen Meldungstypen wird über eine globale Variable gesteuert, die jedes Programm besitzt und unter anderem durch die oben beschriebenen Optionen von außen beeinflußt werden kann.

Die Meldungen bestehen jeweils aus einer Zeile und haben folgende feste Struktur [Syslog]:

```
Mon dd hh:mm:ss host prog [pid]: TYPE message text
```

sie enthalten also die Informationen:

- Datum und Uhrzeit, wann die Meldung generiert wurde
- Name des Rechners, auf dem das Programm läuft
- Name des Programmes
- optionale Prozeß-Nummer in eckigen Klammern
- Typ der Meldung (`ERROR`, `WARNING`, `NOTICE`, `INFO`, `DEBUG`)
- kurze Situationsbeschreibung im Klartext

Dieses einheitliche Format erleichtert nicht nur dem Benutzer die Identifikation und Zuordnung von Meldungen, sondern bietet auch die Möglichkeit, diese durch übergeordnete Software mit einfachen Mitteln zu analysieren. Abbildung 5.14 zeigt einige typische Beispiele für im Rahmen von MECDAS verwendeter Meldungen. Die Meldungen werden in der Regel wie in UNIX üblich über die Diagnose-Ausgabe des jeweiligen Programmes ausgegeben. Durch ihre strenge Kapselung ist es aber auch mit minimalem Aufwand möglich, sie über einen beliebigen anderen Kommunikationskanal zu versenden.

VMEbus-Schnittstelle

Um Maschinen- oder Betriebssystemabhängigkeiten zu verbergen, stehen Funktionen zur Verfügung, die eine einheitliche Schnittstelle für VMEbus-Zugriffe gewährleisten. Dabei werden die unterschiedlichen Adressierungsmodi des VMEbus unterstützt.

```
Jan 04 09:37:02 spekb b: DEBUG buffer size set to 65536
Dec 02 10:53:54 a1d archive: ERROR acqOpen: can't access acq dev
Mar 06 23:25:02 speka a: NOTICE e5_main boot 0
May 13 00:57:02 a1d eventbuilder [4711]: DEBUG slave:a
Sep 03 01:05:37 speka a: WARNING illegal interrupt
Jun 14 01:14:53 spekc c: ERROR timeout in bbk_v2
Jun 28 05:37:15 krygier decode: ERROR init_input: bad transfer m
```

Abb. 5.14: Liste typischer Diagnosemeldungen

Daneben steht ergänzende Software für verschiedene Aufgabenbereiche zur Verfügung, wie z. B.

- elementare Aufgaben der Pufferverwaltung
- Anlegen und Initialisieren von Dateien
- einfache Zeichkettenmanipulationen
- Prozeßverwaltung
- Verwaltung von Shared Memory

Ebenso wie bei den vorher beschrieben Mechanismen ist auch die Software für diese Aufgaben in Form von in sich streng abgeschlossenen Programmmodulen realisiert, die zwar in der Programmiersprache C implementiert sind, jedoch die Grundlage für unabhängige Objektklassen darstellen. Sie stellen eine Reihe von Methoden zur Verfügung und besitzen eindeutige Eigenschaften, die ausschließlich über klar definierte Schnittstellen verändert werden können.

5.3.1.2 Datenaustausch

Zentrale Bedeutung in einem verteilten System, als das MECDAS realisiert ist, hat der Austausch von Daten. Dazu gehört neben dem Transfer von Meßdaten auch die Übertragung von Kontroll- und Statusinformationen zur Steuerung des Experiments und zur Überwachung von dessen Ablauf. Zur Bearbeitung aller Aufgaben, die hierbei anfallen und nicht bereits durch Standard-Systemsoftware abgedeckt sind, wurde spezielle Software implementiert und in einer Bibliothek zusammengefaßt. Sie enthält Mechanismen zur Kommunikation mit dem Datenaufnahmeprogramm, stellt Ein- und Ausgabemöglichkeiten für herkömmliche Dateien oder I/O-Geräte zur Verfügung und unterstützt den Datenaustausch zwischen Rechnern innerhalb eines Netzwerks.

Die Software realisiert dabei eine **einheitliche Schnittstelle** für die unterschiedlichen Kommunikations- und Ein/Ausgabe-Verfahren, die über den

Standard-I/O-Mechanismus von UNIX hinausgeht und an die speziellen Anforderungen der Datenerfassung angepaßt sind. Im einzelnen werden dabei folgende Möglichkeiten beim Datenaustausch abgedeckt:

- Lesen beliebiger Daten von einer Datei, einem herkömmlichen Eingabegerät wie z. B. Magnetband oder einer Pipe
- Schreiben von beliebigen Daten auf Datei, Ausgabegerät oder Pipe
- Übertragung von Daten über eine fehlergesicherte Netzwerkverbindung auf der Basis von TCP/IP
- Übertragung von Daten von und auch zu dem Datenaufnahmeprogramm unter Ausnutzung von global zugreifbarem Speicher, wobei nicht unterschieden wird, ob das Datenaufnahmeprogramm auf demselben Rechner oder in einer Master-Slave-Konfiguration auf einem anderen Rechner läuft

Es wurden Unterprogramme implementiert, die ein einheitliches Öffnen und Schließen dieser Datenkanäle und damit ihre homogene Verwaltung gewährleisten, aber auch die Durchführung des Datentransfers auf einheitliche Art und Weise vornehmen. Damit konnte die restliche Software vollkommen unabhängig von Quelle und Ziel der Daten gehalten werden. Sie muß sich lediglich auf ihre unmittelbaren Aufgaben konzentrieren. Gleichzeitig erleichtert es auch die Realisierung spezieller eigener Anwendungen durch den Experimentator.

Diese Schnittstelle wird sowohl in Programmen verwendet, die in erster Linie den Transfer und die Archivierung der Meßdaten während der Messung sicherzustellen haben, als auch in Software zur weiteren Verarbeitung der Daten für Zwecke der On-line- und auch Off-line-Analyse.

5.3.1.3 Die ACQ-Schnittstelle

Während fast alle der oben genannten Verfahren herkömmlichen I/O-Mechanismen entsprechen, stellt das letzte eine Besonderheit dar. Hierunter verbirgt sich einerseits eine Methode zur Interprozeßkommunikation zum anderen ein Mechanismus zur rechnerübergreifenden Kommunikation zwischen Master und Slave. Es stellt ganz allgemein die Klientenschnittstelle einer Client-Server-Kommunikation zur Verfügung, die notwendig ist, um den Zugriff auf die von der asynchron arbeitenden Datenaufnahme-Software erfaßten Meßdaten zu gewährleisten. Die Verwendung gemeinsam ansprechbaren Speicherplatzes ermöglicht in beiden Fällen einen sehr effizienten Datenaustausch zwischen dem Datenaufnahme-Server und den Klienten.

Diese Schnittstelle dient jedoch nicht nur dem Transfer der Meßdaten sondern auch der Übermittlung von Steueranweisungen an den Datenaufnahme-Server und der Rückübermittlung von Kontroll- und Statusinformation. Sie ist im wesentlichen durch drei Unterprogramme realisiert: `acqOpen()` dient dem Öffnen eines Kommunikationskanals, über den `acqIO()` die Kommunikation

durchführen kann, bis er durch `acqClose()` wieder geschlossen wird. Die Funktionen verwenden die in UNIX üblichen Mechanismen, um die korrekte oder fehlerhafte Ausführung zu kennzeichnen. Eine ausführlichere Beschreibung dieser Funktionen liegt in [Kryg95a] vor.

Die Ausführung von Steueranweisungen arbeitet generell nach dem Prinzip des Remote Procedure Calls (RPC). Die Klienten rufen dazu einfach Unterprogramme auf, die für die Bearbeitung dieser Steueranweisungen sorgen. Die Steueranweisungen werden jedoch nicht vom Klienten ausgeführt sondern im Server, einem getrennten Programm, das noch dazu auf einem separaten Rechner laufen kann. Kennzeichen von RPCs im Gegensatz zum Message-Passing-Verfahren ist, daß ein solches Unterprogramm im Klienten solange aktiv bleibt, wie die Bearbeitung der jeweiligen Steueranweisung im Server andauert. Es wird erst dann verlassen, wenn die Bearbeitung beendet oder ein Fehler aufgetreten ist. Für das aufrufende Programm zeigt sich dabei kein Unterschied, ob die auszuführende Funktion direkt durch das Programm selbst oder durch ein Server-Programm bearbeitet wird, das auf diesem oder einem anderen Rechner läuft. Es benutzt lediglich Unterprogramme, bei deren Aufruf wie üblich Parameter übergeben werden, und die nach Ausführung der Dienstleistung das Ergebnis auf herkömmliche Art zurückliefern. Damit ist die Nutzung von RPCs sehr einfach. Sie bieten eine einfach zu handhabende Ankopplung von Klienten und Server bei gleichzeitig bestmöglicher Trennung der verschiedenen Aufgabenbereiche.

5.3.2 Dienstprogramme und Kommandos

Auf der Basis der oben beschriebenen ACQ-Schnittstelle wurde für jede Steueranweisung, mit dem die Zustandsmaschine des Kernsystems beeinflußt werden kann, ein Dienstprogramm implementiert, das es ermöglicht, die Anweisung auf der Benutzerebene interaktiv als Kommandos einzugeben und auszuführen.

Unter Ausnutzung der Eigenschaften der standardmäßig unter UNIX verfügbaren Kommando-Interpreter (sog. Shells) konnte damit eine einfache, kommandoorientierte Bedienoberfläche geschaffen werden, die es ermöglicht, jedes einzelne Kommando direkt ohne zusätzliche Hilfsmittel einzugeben und ausführen zu lassen. Es besteht die Möglichkeit, auf dieser Ebene einzelne Kommandos mit Standard-UNIX-Kommandos zu kombinieren und zu leistungsfähigen Kommandoprozeduren zusammenzufassen. Davon wurde auch in vielen Fällen Gebrauch gemacht, um teilweise recht komplexe Vorgänge mit wenig Programmieraufwand zu steuern. Solche Kommandoprozeduren können zu jeder Zeit an spezielle Problemstellungen angepaßt werden. Insbesondere ist das einem nur wenig erfahrenen UNIX-Benutzer schon nach kurzer Einarbeitungszeit möglich.

Die Verwendung elementarer Kommandos bietet grundsätzlich den Vorteil, daß sich das System mit minimalem Aufwand einfach und direkt steuern läßt. Neue Aufgaben können oftmals ohne zusätzliche Programmierung durch Kombination der verfügbaren Kommandos und einfache Shell-Prozeduren erledigt werden. Im Einzelfall kann es aber auch sinnvoll sein, aufwendigere Probleme durch explizite, problemangepaßte Programmierung unter Verwendung der ent-

`start`	Start einer Messung
`stop`	Stopp der aktuellen Messung
`restart`	Restart einer Messung
`terminate`	Terminierung der Datenerfassung
`pause`	temporäre Unterbrechung einer Messung
`cont`	Fortführung einer unterbrochenen Messung
`status`	Ausgabe des Experimentstatus
`halt`	Abschalten des Datenerfassungs-Kontrollsystems
`debug`	Ein- bzw. Ausschalten des Debug-Modes
`buffers`	Ausgabe von Informationen zur Puffer-Verwaltung
`flush`	Flush der Datenpuffer
`state`	Ausgabe des Datenaufnahmezustands
`messages`	Ausgabe des Message-Puffers

Tab. 5.1: Erlaubte MECDAS-Steuerkommandos

sprechenden Bibliotheksfunktionen effizient zu lösen. Beide Möglichkeiten stehen zur Verfügung und können bei Bedarf genutzt werden.

5.3.2.1 Steuerung der Datenaufnahme: das Kommando `acq`

Eine direkte Umsetzung der ACQ-Schnittstelle, die im vorangegangenen Abschnitt beschrieben wurde, wurde in Form des Programmes `acq` realisiert. Bei seinem Aufruf mit einem Argument, das eine auszuführende Steueranweisung spezifiziert, wird im wesentlichen nichts anderes getan, als die Anweisung mithilfe der beschriebenen Funktionen abwickeln zu lassen. D. h., mit dem Kommando[5]

```
$ acq start
```

wird dem Datenaufnahme-Server eine Start-Anweisung übermittelt und, falls mit dem Systemzustand verträglich, eine Messung gestartet. Ursprünglich zur Vereinfachung der Benutzung wurden eine Reihe von einfachen Kommandoprozeduren mit den Namen der Steueranweisungen erstellt, die i. w. ausschließlich das Programm `acq` mit ihrem eigenen Namen als Argument aufrufen. Zum Start einer Messung reicht also das Kommando

```
$ start
```

aus. Tabelle 5.1 zeigt eine Übersicht der erlaubten Steuerkommandos mit kurzer Funktionsbeschreibung entsprechend den in Abschnitt 5.2 beschriebenen

[5] Das in den Kommandobeispielen vorangestellte „$“-Zeichen entspricht dem Shell-Prompt und soll lediglich andeuten, daß es sich bei dem nachfolgenden Text um ein Kommando handelt, das man innerhalb des Standard-Kommandointerpreters wie jedes andere Kommando auf herkömmliche Weise benutzen kann.

Dienstleistungen des Datenerfassungs-Servers. Sie stellen die offiziellen Kommandos und damit die definierten Schnittstellen zur Steuerung der Datenerfassung dar und bieten als Kommandoprozeduren, die lokal verändert werden können, die Möglichkeit, in Sonderfällen die unmittelbaren Anweisungen an den Datenaufnahme-Server durch spezielle Aktivitäten zu ergänzen (siehe beispielsweise [Claw95]). Alle diese Kommandos können zu einem beliebigen Zeitpunkt beliebig oft ausgeführt werden. Entscheidend für ihre Wirksamkeit ist der aktuelle Zustand der Zustandsmaschine innerhalb des Datenaufnahmeprogrammes, der den Gesamtsystemzustand repräsentiert. Ist eine Steueranweisung mit dem aktuellen Zustand unverträglich, wird sie nicht weiter abgearbeitet und mit einer entsprechenden Fehlermeldung abgebrochen.

Eine Besonderheit sind die Kommandos `status`, `state`, `messages` und `buffers`. Während alle anderen Kommandos ausschließlich Steueranweisungen sind, um den Zustand des Datenaufnahmesystems zu beeinflussen, liefern sie Informationen über den Systemzustand. Wegen der einfachereren Handhabbarbeit werden diese Informationen als ASCII-Texte ausgegeben und können bei Bedarf anschließend mit Standard-UNIX-Werkzeugen oder eigenen Filterprogrammen weiterverarbeitet werden (siehe auch Abschnitt 5.2.4.5).

5.3.2.2 Datenarchivierung: das Kommando `archive`

Die Archivierung der Meßdaten bei MECDAS wird wie bereits oben beschrieben grundsätzlich asynchron vorgenommen. Das Datenaufnahmeprogramm stellt dazu die Daten lediglich zu Verfügung. Sie müssen von einem geeigneten Programm abgeholt werden, das sich dann auch um die weitere Abspeicherung der Daten kümmert. Für diesen Zweck wurden eine Reihe von Bibliotheksfunktionen implementiert, die den einfachen Zugriff und die Abspeicherung der Meßdaten gewährleisten. Sie können bei der Realisierung von Programmen, die mit den Meßdaten in irgend einer Form arbeiten, verwendet werden, bilden aber auch die Grundlage für das Programm `archive`, das in MECDAS standardmäßig für die Archivierung der Meßdaten zuständig ist.

Das Programm `archive` erlaubt den Datentransport von einer beliebigen Quelle zu einem beliebigen Ziel, soweit das im Rahmen der Datenerfassung sinnvoll ist. Entsprechend der in Abschnitt 5.3.1.2 dargestellten Übersicht sind das herkömmliche Dateien, I/O-Geräte, Pipes, Standard-Input/Output, TCP/IP-Verbindungen und schließlich der Datenaufnahme-Server. Ziel und Quelle für den Datentransfer können beim Aufruf des Programmes in beliebiger Kombination gewählt werden. Die Daten werden dabei grundsätzlich als Datenstrom ohne Berücksichtigung einer inneren Struktur übertragen. Sie werden vollkommen transparent vom Ziel zur Quelle durchgereicht und dabei nicht modifiziert. Wie bereits innerhalb des Datenaufnahmeprogramms werden auch hier die Daten unabhängig davon, ob und in welchem Format sie verpackt sind, transportiert. `archive` ist letztendlich ein relativ allgemeines Werkzeug, um beliebige Daten von einem Ort zu einem anderen Ort, auch Rechnergrenzen übergreifend, zu transferieren, angepaßt an die speziellen Bedürfnisse der Datenerfassung.

5.3.2.3 On-line-Analyse und Diagnose

Die Klienten-Software bietet neben der Nutzung der Standard-Schnittstelle des Datenaufnahme-Servers zur Archivierung der Meßdaten, die aus der Sicht des Klienten synchronisiert[6] arbeitet, auch die Möglichkeit, die für Zwecke der On-line-Analyse und Diagnose vorgesehenen nicht synchronisierten Schnittstellen zu nutzen. Entsprechende Software ist Bestandteil der oben beschriebenen Bibliotheken. Auch das Programm `archive` erlaubt über spezielle Kommandoparameter die Verwendung dieser nicht synchronisierten Schnittstellen.

Besonders interessant ist diese Möglichkeit jedoch für spezielle On-line-Analyse- und Diagnose-Software. MECDAS stellt zu diesem Zweck einen Programmrahmen zur Verfügung, der durch experimentspezifischen Programmcode ergänzt werden kann und so ein beliebiges Auswerteprogramm ergibt. Es verwendet dieselben Kommunikationsmechanismen wie das Programm `archive`, benutzt jedoch im Normalfall die nicht synchronisierte Datenschnittstelle des Servers. Auf dieser Basis wurde bereits ein abgeschlossenes Diagnoseprogramm (`decode`) implementiert, das lediglich die Meßdaten dekodiert und in lesbarer Form ausgibt. Es ist in erster Linie für Debugging-Zwecke konzipiert, kann jedoch auch als Schnittstellenumsetzer verwendet werden und bietet so ein sehr einfaches Interface, um fremde Analyse-Software an MECDAS anzukoppeln. Eine ausführlichere Beschreibung der On-line-Analyse-Software ist in Kapitel 8 zu finden.

5.3.2.4 I/O-Service: das Kommando `camreq`

Ein weiteres Hilfsprogramm, das Dienstleistungen durch den Datenaufnahme-Server abwickeln läßt, ist das Programm `camreq`. Es ist weniger ein unverzichtbarer Bestandteil der im Rahmen der Steuerung der Datenerfassung notwendigen Software, als vielmehr ein Beispiel für einen weiteren Klienten, der eine extrem beschränkte Teilaufgabe innerhalb des komplexen Systems auf einfach zu handhabende Art und Weise erfüllt. Das Programm bietet die Möglichkeit, jeden beliebigen CAMAC-Request unter Kontrolle des Datenaufnahme-Servers und damit ohne die Gefahr der Kollision mit der Datenerfassung auszuführen.

Weitere I/O-Services, die in Zukunft evtl. im Datenaufnahme-Sever implementiert werden, können durch entsprechende Hilfsprogramme, die sehr einfach auch noch nachträglich zu programmieren sind, genutzt werden. Das Client-Server-Konzept bietet dabei u. a. den Vorteil, daß ein solches Programm weitgehend unabhängig von der restlichen, bereits existierenden oder noch zu erstellenden Software ist.

[6] Die Archivierung läuft zwar innerhalb des Servers asynchron zur Datenaufnahme ab, doch synchronisiert sich die Kommunikations-Software des Servers mit dem Klienten, um sicherzustellen daß dieser jeden zur Archivierung bestimmten Puffer auch tatsächlich erhält.

5.4 Die Client-Server-Kommunikation

Grundlage für das Funktionieren des in den vorangegangenen Abschnitten beschriebenen Client-Server-Konzeptes ist eine leistungsfähige und korrekt arbeitende Kommunikation zwischen Klient und Server und damit den einzelnen Komponenten eines verteilten Systems, das aus mehreren in der Regel auf unterschiedlichen Rechnern aktiven Programmen besteht. Sie dient einerseits zum Austausch von Steueranweisungen und Statusmeldungen, andererseits aber auch zum Transfer der Meßdaten, was die Anforderungen erheblich verschärft.

5.4.1 Problemstellung

Grundsätzlich lassen sich bei dem für die Datenerfassung eingesetzten Client-Server-System zwei verschiedene Fälle unterscheiden, die insbesondere für die Kommunikation von Bedeutung sind:

1. Klienten und Server laufen auf demselben Rechner
2. Klienten und Server sind auf unterschiedlichen Rechnern aktiv

Bereits an früherer Stelle wurde erwähnt, daß für beide Fälle eine softwaremäßige Umsetzung vorgenommen wurde, womit es möglich ist, die Datenerfassung unter eingeschränkten Bedingungen mit einem einzigen Rechner durchzuführen, aber auch die Vorzüge eines Mehrrechnersystems voll zu nutzen. Die Verwendung eines Client-Server-Konzepts ermöglichte eine einheitliche Realisierung beider Problemstellungen und bietet auch dem Anwender einen einfachen Übergang von einem zum anderen Lösung. Eine definierte Schnittstelle (Abschn. 5.2.4) sorgt hierbei für eine klare Trennung zwischen Klienten und Server, unabhängig davon, ob beide auf demselben Rechner laufen oder nicht. Gleichzeitig stellt sie ein für die Anwendung transparentes Bindeglied zwischen den beiden Softwareteilen dar.

Die Unterschiede zwischen den beiden Verfahren reduzieren sich schließlich nur noch auf die Kommunikation zwischen Klient und Server. Im ersten Fall beschränkt sie sich auf eine reine Interprozeßkommunikation, für die auf die im Anhang beschriebenen Mechanismen, die das Betriebssystem UNIX zur Verfügung stellt, zurückgegriffen werden kann. Ist aber das System auf mehrere Rechner verteilt, ist eine geeignete rechnerübergreifende Kommunikation notwendig, die je nach verfügbarer Kommunikations-Hardware und abhängig von den beteiligten Rechnern sehr unterschiedlich realisiert sein kann. Im Frontend-Bereich kommt hierfür jedoch in erster Linie der VMEbus in Frage, der den gemeinsamen Betrieb mehrerer Recher zuläßt und zwischen ihnen eine enge und effiziente Kopplung ermöglicht. Aber auch wenn die beteiligten Rechner keine so enge Verbindung besitzen, lassen sich — wenn auch mit teilweise deutlichem Mehraufwand — leistungsfähige Alternativen realisieren.

Das im Rahmen der Entwicklung des A1-Steuerungssystem implementierte Message-System MUPIX [Kram95] wurde hier aus mehreren Gründen nicht eingesetzt.

1. MUPIX basiert auf dem Internet-Protokoll UDP, für das auf den betriebssystemlosen Slave-Rechnern keine Software zur Verfügung stand. Die Portierung dieses Standard-Protokolls hätte einen im Vergleich zum erwarteten Nutzen unverhältnismäßig hohen Aufwand bedeutet.

2. Der VMEbus als Kommunikationsmedium würde jedoch nicht nur eine Implementation von UDP/IP für den Slave-Rechner erfordern, sondern ebenso auch für den Master-Rechner erhebliche Eingriffe in das Betriebssystem erforderlich machen. Trotz Verfügbarkeit von Quellcode wäre auch hier ein hoher Implementationsaufwand notwendig geworden.

3. MUPIX ist auf die asynchronen Ein/Ausgabe-Möglichkeiten und Signal-Behandlungs-Möglichkeiten von BSD-UNIX angewiesen. Auch diese stehen auf dem Slave nicht zur Verfügung und hätten mit nicht zu vertretendem Aufwand implementiert werden müssen.

4. Eine alternative Realisierung von MUPIX auf dem VMEbus, die zwar kompatible Anwendungs-Schnittstellen besitzt, jedoch intern ein vereinfachtes, dem VMEbus angepaßtes Protokoll benutzt, könnte den Implementationsaufwand reduzieren, hätte jedoch eine Modifikation der allgemeinen MUPIX-Software auf der Master-Seite zur Folge.

5. Unabhängig vom Medium ist das Message-System infolge der Verwendung von UDP auf Paketgrößen von 1 KB beschränkt und daher nicht direkt für die effiziente Übertragung großer Datenmengen geeignet, wie sie bei der Datenerfassung anfallen. Aus diesem Grund wäre eine Modifikation bzw. Erweiterung des MUPIX-Protokolls notwendig geworden, verbunden mit erheblichem Mehraufwand bei nur beschränktem Funktionalitätsgewinn.

6. Die Alternative zur Modifikation des MUPIX-Protokolls für den effizienten Transfer großer Datenmengen wäre die Kombination von Message-System und TCP/IP-Verbindungen bzw. Shared Memory. In diesem Fall jedoch ergäben sich weder im Bereich von Funktionalität noch bei der Implementation Vorteile gegenüber einer direkten Nutzung des Socket-I/Os.

7. Das bei MUPIX benutzte Protokoll ist nicht verbindungsorientiert. Für die Kommunikationsanforderungen im Rahmen der Datenerfassung bieten sich jedoch die Vorzüge eines verbindungsorientierten Protokolls an (s. 5.4.4). Diese Eigenschaften müßten zusätzlich implementiert werden.

8. Das MECDAS zugrunde liegende Kommunikationskonzept basiert auf einer streng hierarchischen Struktur mit definierten Kommunikationspartern, so daß die Vorzüge von MUPIX gar nicht genuzt würden.

9. Die Verwendung des Message-Systems innerhalb der Anwendungen erfordert die Rücksichtnahme auf dessen asynchronen Arbeitsweise, um erhebliche Beeinträchtigungen der Funktionalität zu vermeiden. Anstelle von Standard-Bibliotheksfunktionen und Systemaufrufen müssen für spezielle Zwecke entsprechende MUPIX-Funktionen verwendet werden. Das macht die Programmierung etwas aufwendiger.

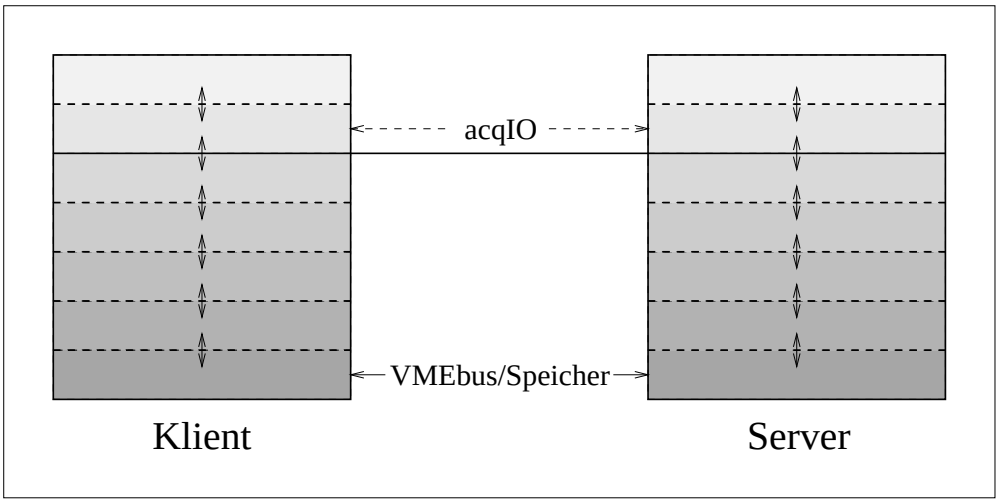

Abb. 5.15: Die Umsetzung des OSI-Schichtenmodells für die Kommunikation zwischen Datenerfassungs-Server und seinen Klienten

5.4.2 Anwendung des OSI-Schichtenmodells

Auch wenn sich die Implementation der Software im konkreten Fall neben der lokal unter UNIX arbeitenden Variante vorerst auf den VMEbus beschränkte, so wurde sie doch so strukturiert, daß ein Austausch des Übertragungsmediums oder des Kommunikationsverfahrens mit wenig Aufwand möglich ist. Sie orientiert sich dabei ganz grob an dem in Anhang A.9 beschriebenen OSI-Schichtenmodell und deckt darin die Schichten 1 bis 5 ab Die darüberliegenden Schichten bleiben der anwendungsorientierten Software des Datenaufnahmeprogrammes und seiner Klienten vorbehalten:

Klienten und Server sind in verschiedene, jeweils korrespondierende Schichten aufgeteilt, wobei die höheren Schichten der eigentlichen Anwendung entsprechen und die unteren die verschiedenen Aspekte der Kommunikation zwischen Klient und Server abdecken. Jede Schicht der Klienten tauscht Informationen ausschließlich mit der entsprechenden Schicht des Servers aus. Der eigentliche Datenfluß findet jedoch immer nur zwischen den benachbarten Schichten innerhalb eines Programmes und schließlich über das Kommunikationsmedium statt. Somit wird nicht nur die Anwendung unabhängig von den Details des Datentransfers, sondern auch innerhalb der Kommunikations-Software existieren unterschiedliche Abstraktions- und Funktionalitätsebenen. In Abbildung 5.15 ist die Umsetzung des OSI-Schichtenmodells auf das Client-Server-Konzept von MECDAS in grober Form graphisch dargestellt.

Die einzelnen Schnittstellen werden auf Klienten- und Server-Seite jeweils durch einen Satz von Unterprogrammen definiert, die etwa für den Verbindungsauf- und -abbau und das Verschicken und Empfangen der Nachrichten zuständig sind. Die Kommunikations-Software sorgt nicht zuletzt für die korrekte Übermittlung der Information, also der Nachrichten und evtl. zu übertragender weiterer Daten und nimmt schließlich die individuelle Ansteuerung der

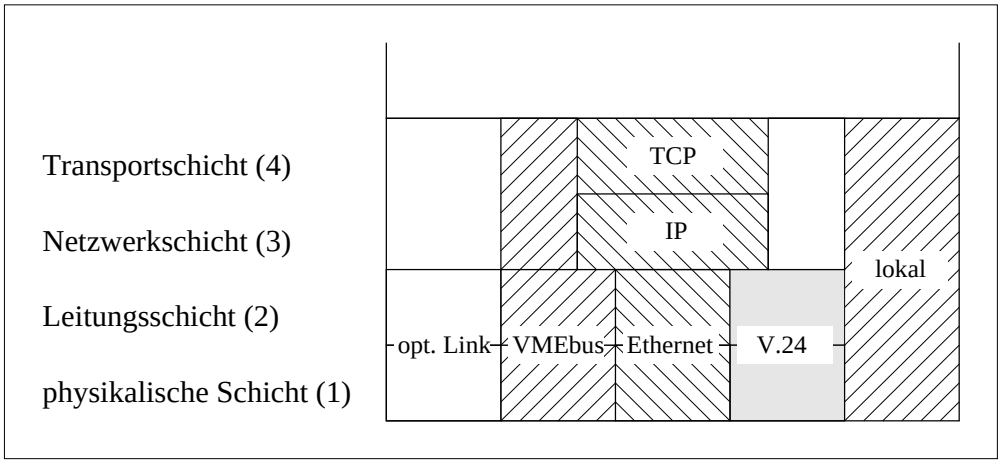

Abb. 5.16: Gegenüberstellung der verschiedenen, z. Z. bereits verfügbaren bzw. in Zukunft denkbaren Kommunikationsmechanismen

Kommunikationshardware vor. Je nach verwendetem Medium leitet sie Maßnahmen ein, um eine sichere und fehlerfreie Übertragung zu gewährleisten. Daneben stellt sie mehrere verschiedene logische Kanäle zur Verfügung, über die eine i. w. unabhängige Kommunikation zwischen einem Server und mehreren Klienten zur gleichen Zeit abgewickelt werden kann. Die verschiedenen Übertragungsmechanismen stellen eine nach oben hin identische Schnittstelle zur Verfügung, auf der die Anwendungs-Software aufsetzten kann, die es aber auch erlaubt, die unterschiedlichen Verfahren leicht auszuwechseln.

Für den Fall der rechnerinternen Kommunikation bzw. bei Verwendung von TCP/IP-Protokollen über Ethernet werden z. B. die Schichten 1 bis 4 bereits durch Systemsoftware abgedeckt, lediglich die darüberliegenden Schichten mußten für die spezielle Anwendung neu implementiert werden. Diese wiederum konnten zum Einsatz anderer Kommunikationsmechanismen verwendet werden, während dafür lediglich die entsprechenden Teile der unteren Schichten realisiert werden mußten. Voraussetzung dabei war, daß unterschiedliche Realisierungen einer Schicht natürlich identische Schnittstellen zumindest zu den höheren, sie nutzenden Schichten haben mußten und funktionell kompatibel zu sein hatten. Lediglich Effizienzgesichtspunkte bzw. ein unnötig hoher Implementationsaufwand ließen es teilweise als sinnvoll erscheinen, von den strengen Schichteneinteilung, die das OSI-Modell vorgibt, abzuweichen.

Abbildung 5.16 stellt die verschiedenen Alternativen, die zur Kommunikation zwischen Klienten und Server eingesetzt werden können, gegenüber. Sie zeigt einerseits die Struktur der bereits implementierten bzw. verfügbaren in Hinblick auf das Schichtenmodell, andererseits auch weitere, denkbare Mechanismen, die in Zukunft Verwendung finden können, jedoch noch zusätzlichen Implementationsaufwand erfordern. Von den in der Abbildung dargestellten Möglichkeiten wurden besonderer Wert einerseits auf den rechnerübergreifenden Datenaustausch über den VMEbus im Rahmen einer die Master-Slave-Lösung gelegt und andererseits auf die lokale, rechnerinterne Kommunikation.

5.4.3 Das Master-Slave-Modell

Das Master-Slave-Modell von MECDAS nutzt den Einsatz mehrerer, in der Regel eng gekoppelter Rechner im Frontend-Bereich aus, die sich die einzelnen Aufgaben, die bei der Datenerfassung anfallen, teilen. Ein Rechner — der Master, der üblicherweise unter UNIX betrieben wird — ist für alle übergeordneten Aufgaben zuständig und kontrolliert einen oder mehrere Slaves, die in erster Linie für die Datenaufnahme unter Echtzeitbedingungen verantwortlich sind. In der derzeitigen Implementation besitzt der Slave kein eigenständiges Betriebssystem und verfügt a priori auch nicht über Standard-Kommunikationsmechanismen wie TCP/IP. Neben der Bereitstellung eines einfachen Systemsoftwarepakets, das einfache Systemdienste zur Verfügung stellt, war daher die Implementation von Software zur Kommunikation zwischen Master und Slave notwendig, die die Schichten 1 bis 4 abdeckt. Da in der Regel Master und Slave innerhalb eines VMEbus-Systems untergebracht werden konnten, bot sich als gut geeignetes Kommuniaktionsmedium zwischen beiden der VMEbus an.

5.4.3.1 Kommunikation über den VMEbus

Der VMEbus bietet für eine effiziente Kommunikation gute Voraussetzungen (s. Anhang A.1). Dabei kann ausgenutzt werden, daß allen innerhalb eines VMEbus-Überrahmens installierten Rechner ein gemeinsamer Adreßraum zur Verfügung steht, den die einzelnen CPUs über direkte Speicherzugriffe und damit effizient und flexibel ansprechen können. Befindet sich innerhalb dieses Adreßraums Schreib/Lese-Speicher, auf den jeweils mindestens zwei CPUs zugreifen können, dann ist darüber ein universeller Datenaustausch möglich, indem die zu übertragenden Daten von der sendenden CPU in diesen Speicherbereich geschrieben und von der empfangenden aus diesem gelesen werden.

Die im konkreten Fall als Master bzw. Slave eingesetzten CPU-Karten besitzen den besonderem Vorteil, daß ihr Arbeitsspeicher als „dual ported RAM“ ausgelegt ist. D. h., der auf den CPU-Karten installierte Speicher kann nicht nur durch die lokale CPU selbst benutzt werden, sondern er ist auch über den VMEbus ansprechbar. Damit kann zur Realisierung der Kommunikation vollständig auf zusätzliche Hilfsmittel verzichtet werden.

Auf dieser Basis wurde Software implementiert, die eine verbindungsorientierte Punkt-zu-Punkt-Kommunikation zwischen Master und Slave erlaubt. Sie deckt elementare Aspekte der Schichten 2–4 des OSI-Schichtenmodells ab, ist jedoch zur Vermeidung unnötigen Overheads und wegen der eingeschränkten Anforderungen der Anwendung auf ein Minimum an notwendiger Funktionalität beschränkt. Besonderen Wert wurde auf hohe Effizienz bei der Übertragung großer Datenmengen gelegt. Es konnte erreicht werden, daß bei Puffergrößen, wie sie den standardmäßigen Voreinstellungen entsprechen, sich der Verwaltungsaufwand auf weniger als 1 % des Aufwands beläuft, der durch das Kopieren der Daten verursacht wird. Nähere Details über das Funktionsprinzip und die konkrete Realisierung der Kommunikations-Software kann der MECDAS-Dokumentation entnommen [Kryg95a] werden.

5.4.3.2 Alternativen

Neben dem VMEbus sind für Master-Slave-Kommunikation eine Reihe weiterer Alternativen denkbar, die in dem einen oder anderen Sonderfall nutzbringend eingesetzt werden können und durch das MECDAS zugrundeliegende Konzept abgedeckt werden.

Ethernet

Alle zur Zeit und wohl auch in Zukunft eingesetzten Frontend-Rechner verfügen über ein Ethernet-Interface. Das ermöglicht es ihnen, über das allgemeine Institutsnetz oder auch spezielle Meßnetze mit anderen, auch entfernt aufgestellten Rechnern zu kommunizieren. Insbesondere für die Datenerfassung eines isolierten Subdetektorsystems ist über Ethernet eine leichte Ankopplung an das restliche System möglich. Damit müßten also Master und Slave nicht mehr in einem VMEbus-Überrahmen untergebracht werden, sondern könnten über Ethernet miteinander kommunizieren. Aufgrund der klaren Trennung von Anwendung und Kommunikation im Client-Server-System von MECDAS ist dazu lediglich der Austausch der Kommunikations-Software notwendig. Für diesen Fall wird Software benötigt, die eine fehlergesicherte Übertragung von Steuerungsinformationen und Meßdaten mit einem hohen Durchsatz und mit möglichst geringer Beeinträchtigung der Echtzeitfähigkeit über Ethernet erlaubt und sowohl für den Master als auch den Slave verfügbar ist. Während auf der Seite des Masters diese Anforderungen sehr gut durch TCP/IP abgedeckt wird, müßte für den Slave spezielle Software mit verhältnismäßig hohem Aufwand entwickelt bzw. portiert werden. Als Alternative könnte auch ein spezielles, einfacher zu realisierendes Protokoll implementiert werden, das nur den benötigten Bruchteil der Funktionalität von TCP/IP abdeckt. In beiden Fällen wurden entsprechende Vorarbeiten geleistet. Dabei zeigte sich jedoch, daß die noch aufzuwendende Arbeit ohne konkreten Einsatz nicht zu rechtfertigen ist, wobei zusätzliche noch berücksichtigt werden muß, daß die Datentransfergeschwindigkeit im Vergleich zum VMEbus mindestens eine Größenordnung geringer ist und die Echtzeitfähigkeit — abhängig von der Implementation der Software — evtl. stark beeinträchtigt werden kann.

Optische Verbindungen

Auch die Kommunikation auf der Basis von Lichtleitern kommt als Alternative in Frage. Zwar bedeutet ihr Einsatz entsprechende Mehrinvestitionen für spezielle VMEbus-Karten, doch sind im Zusammenhang mit Ethernet genannte Nachteile wie niedrigere Geschwindigkeit und Verschlechterung der Echtzeitfähigkeit für optische Verbindungen weit weniger gravierend. Der Aufwand zur Implementierung entsprechender Software ist infolge der zur Realisierung von Punkt-zu-Punkt-Verbindungen notwendigen einfacheren Hardware geringer, muß aber doch geleistet werden. Ansonsten bieten die beschriebenen Schnittstellen in der MECDAS-Software einfache Integrationsmöglichkeiten an.

SCSI-Bus

Ähnlich wie bei Ethernet besitzt jeder eingesetzte Frontendrechner ein SCSI-Interface [Elte90], das eigentlich zum Anschluß von Massenspeicher gedacht ist, grundsätzlich aber auch zur Rechnerkopplung verwendet werden kann. Der SCSI-Bus [SCSI] vereint einige Vorteile von VMEbus und Ethernet: Als Bus mit parallelen Datenleitungen erreicht er je nach Version Datenraten von einigen MB/s und kann damit trotz seiner nominell deutlich niedrigeren Geschwindigkeit ähnlich schnell Daten transferieren, wie effektiv über den VMEbus übertragen werden können. Während hier die CPU die Geschwindigkeit begrenzt und mit dem Datentransfer belastet ist, nutzen die SCSI-Controller so wie bei Ethernet DMA[7] aus. In Bezug auf Software zur Datenübertragung und Fehlersicherung ist weniger Aufwand als bei Ethernet, aber dennoch deutlich mehr als für den VMEbus notwendig. Eine SCSI-Kopplung macht zwar die Installation der CPUs in einem gemeinsamen Überrahmen nicht mehr notwendig, doch lassen sich mit seiner Hilfe nur Entfernungen von wenigen Metern überbrücken. Bis auf die Verfügbarkeit besitzt der SCSI-Bus im Vergleich zu optischen Verbindungen deutliche Einschränkungen.

Serielle Schnittstelle

Grundsätzlich läßt sich auch eine herkömmliche serielle Schnittstelle (V.24) einsetzen. Wenn überhaupt kommt sie aber höchstens in extremen Ausnahmesitutationen in Frage, da ihre niedrige Datentransfergeschwindigkeit zwar zur Übermittlung von Informationen zur Experimentsteuerung ausreicht, nicht jedoch die Datenraten zuläßt, die in der Regel zur Übertragung von Meßdaten benötigt werden. Im Rahmen von früheren Tests konnten serielle Schnittstellen in ähnlichem Zusammenhang aber schon erfolgreich eingesetzt werden. Grundlegende Arbeiten dafür, wie z.B. die Implementation eines fehlergesicherten Protokolls, wurden bereits geleistet, doch rechtfertigt auch hier der geringe Nutzen keinen weiteren Aufwand zur endgültigen Implementation, solange keine konkrete Anwendung vorliegt.

Transputer-Links

Insbesondere beim Einsatz von Transputer im Frontend-Bereich wie im Institut für Kernphysik bereits realisiert [Geig93a, Venu91] bietet sich auch die Verwendung von Transputer-Links an. Bei den bisherigen Anwendungen wurde zwar auch bei den Transputern von der Kommunikation über den VMEbus Gebrauch gemacht, doch erfordert das teure Transputer-CPU-Karten für den VMEbus, die für die eigentliche Anwendung oftmals nicht direkt benötigt werden. Ein einfaches Link-Interface für den VMEbus wäre ausreichend, um beliebige Transputer-Systeme an MECDAS anzukoppeln. Auch hier kann in Zukunft von bereits geleisteten Vorarbeiten profitiert werden.

[7] Direct Memory Access; direkter Speicherzugriff von peripheren Geräten unter kontrollierter Umgehung der CPU

5.4.4 Ein-Rechner-Variante

Das Master-Slave-Konzept von MECDAS wurde entwickelt, um mit der im Frontend-Bereich verfügbaren Hard- und Software eine bestmögliche Lösung der im Rahmen der Datenerfassung anfallenden Aufgaben zu realisieren. Dennoch ist der Einsatz mehrerer Frontend-Rechner nicht immer möglich oder sinnvoll. In vielen Anwendungen, wie Experimenten mit niedrigen Ereignisraten oder bei Testaufbauten, oder gar bei Einsatz moderner, schnellerer Rechner reicht oftmals eine CPU aus, um alle Aufgaben gut zu erfüllen. Auch der Einsatz intelligenter Subsysteme, die bereits die Echtzeitaufgaben übernehmen, kann einen speziellen Rechner dafür überflüssig machen. Aus diesen Gründen beschränkt sich MECDAS nicht auf die Master-Slave-Lösung, sondern nutzt die Vorteile einer allgemeineren Client-Server-Konzeption, wie sie bereits in den vorangegangenen Abschnitten beschrieben wurde. Sie erlaubt es, den Datenerfassungs-Server sowohl auf einen Slave-Rechner zu verlagern, als auch auf dem Rechner zu halten, auf dem der Rest der Software aktiv ist. Bei identischer Anwendungs-Software ist für die lokale Variante lediglich andere Kommunikations-Software als für den Master-Slave-Fall notwendig.

Die Implementation nutzt dabei konsequent die IPC-Möglichkeiten von UNIX aus. Die Grundlage bilden einerseits UNIX-Domain-Sockets und andererseits Shared Memory. Die Sockets erlauben den einfachen Aufbau einer fehlergesicherten Kommunikationsverbindung zwischen Klient und Server, über die im Regelfall Steuerungsnachrichten ausgetauscht werden. Im Gegensatz zur Master-Slave-Kommunikation muß weder für die Fehlersicherung noch die Synchronisierung der Kommunikation explizit gesorgt werden. Diese Aufgaben übernimmt das Betriebssystem automatisch. Auch sorgt es für die Generierung und Verwaltung paralleler Kommunikationskanäle und stellt einfache Mechanismen zu deren Benutzung zur Verfügung. Nich zuletzt wird ausgenutzt, daß Verbindungsabbrüche aufgrund beliebiger Ursachen stets an die Kommunikationspartner gemeldet werden, womit eine unmittelbare Reaktion auch auf Fehler wie z. B. Absturz eines Kommunikationspartners möglicht ist.

Durch die Verwendung der UNIX-Domain-Sockets als Kommunikationsbasis ist jedem Server-Prozeß ein eindeutiger symbolischer Name zugeordnet, über den er sich ansprechen läßt. Es ist der Pfadname des Sockets innerhalb des UNIX-Filesystems. Für die Klienten reicht die Kenntnis dieses Namens aus, um mit dem Server in Kontakt zu treten und eine anschließende Kommunikation durchzuführen. Die Sockets decken vollständig alle Anforderungen an die Client-Server-Kommunikation zur Übermittlung von Nachrichten ab. Aufgrund des einheitlichen Socket-Interfaces kann die Kommunikation vom rechnerinternen Datenaustausch mit wenig Aufwand auch erweitert werden, so daß sie rechnerübergreifend auf der Basis von TCP/IP möglich ist und somit wiederum für eine Master-Slave-Variante brauchbar ist.

Die rechnerinterne Kommunikation nutzt neben den Sockets noch eine weitere Betriebssystemeigenschaft aus, um insbesondere die Übertragung großer Mengen von Meßdaten effizient zu gestalten. Analog zur VMEbus-Kommunikation werden die Daten im Normalfall selbst nicht über den beschriebenen Kommuni-

kationsweg übertragen, sondern über gemeinsam zugreifbaren Speicher verfügbar gemacht. Es werden lediglich Nachrichten ausgetauscht, die Informationen über Größen und Position innerhalb des Shared Memory enthalten. Damit ist es möglich, unnötiges Umkopieren von Daten zu vermeiden; für den eigentlichen Datentransfer wird im Idealfall keine Zeit benötigt. Ebenso wie bei der VMEbus-Lösung wird dieser Datenbereich ausschließlich durch den Server verwaltet. Durch Austausch entsprechender Nachrichten kann er die Kontrolle und damit Lese- und Schreibrechte einzelner Speichersegmente kurzzeitig an die Klienten übergeben und vermeidet so Inkonsistenzen in der Verwaltung.

Im Gegensatz zur Slave-Variante arbeitet die derzeitige Implementation des Datenerfassungs-Servers unter UNIX im Polling-Mode, also noch nicht bzw. nur eingeschränkt interruptgesteuert. Dafür wären schwer portierbare Eingriffe in das Betriebssystem notwendig, deren Aufwand bezogen auf den zu erwartenden Nutzen bisher nicht zu rechtfertigen war, denn es stand mit der Master-Slave-Varianten eine funktionsfähige Lösung als Standard-Werkzeug zur Verfügung. Dieser gegenüber wäre die UNIX-Lösung unter den Randbedingungen, die durch die bisher verfügbare Rechner-Hardware und Systemsoftware bestimmt werden, mit starken Leistungseinbußen verbunden — aus Gründen, die schon früherer Stelle ausführlich diskutiert wurden.

Inzwischen kommt jedoch mit der Verfügbarkeit leistungsfähigerer Hard- und Software die Realisierung einer Ein-Rechner-Varianten unter Ausnutzung von Interruptverarbeitung wieder in Betracht. Hier bietet sich analog zu der ersten MECDAS-Realisierung unter dem Betriebssystem OS-9 (s. Abschnitt 5.2.2 u. [Kryg95a]) die Implementation eines speziellen System-Gerätetreibers an, der die Aufgaben der Kernsoftware abdeckt. Dazu gehört nicht nur die Interruptverarbeitung, die bei einem zu UNIX weitgehend kompatiblen Betriebssystem in der Tat nur mithilfe eines Gerätetreibers zu realisieren wäre, und für eine effiziente Lösung damit zusammenhängend die unmittelbare Durchführung der Datenaufnahme, sondern insbesondere auch die Server-Funktionalität, die aus Anwendersicht generell die grundlegende Aufgabe eines Gerätetreibers darstellt. Damit untrennbar verbunden ist die Client-Server-Kommunikation, die in diesem Fall durch die Kommunikation zwischen Anwenderprogramm und Gerätetreiber und damit durch herkömmliche Ein/Ausgabe-Systemaufrufe repräsentiert wird. Sie erfordert aus der Sicht des Klienten i. w. identische Mechanismen wie bei der implementierten Lösung auf der Basis der Socket-Schnittstelle, bei der ebenfalls ausschließlich von elementaren Standard-Ein/Ausgabe-Mechanismen Gebrauch gemacht wird. Das war u. a. einer der Gründe, weswegen UNIX-Domain-Sockets für diese Aufgabe ursprünglich gewählt wurden.

Wie bereits erwähnt liegt bisher noch die Bedeutung der Ein-Rechner-Varianten für die Datenaufnahme eher im Bereich von Spezialanwendungen und Testanordnungen. Außerhalb des hardwarenahen Frontend-Bereichs findet sie aber eine wichtige und auf den ersten Blick ganz andere Anwendung. Sie bildet nämlich auf übergeordneten Rechnern die Grundlage für das Zusammenfassen von Meßdaten (Eventbuilding) von mehreren, gleichzeitig an einer Messung beteiligten Frontend-Rechnern; dazu aber mehr in Kapitel 7.

5.5 Fehlerbehandlung

Im Rahmen der Datenerfassung können eine ganze Reihe von teilweise sehr unterschiedlichen Fehlern auftreten. Man kann dabei unterscheiden zwischen Fehlern aufgrund defekter Rechner-Hardware oder nicht korrekt funktionierender Meßelektronik und Fehlern aus dem Software-Bereich, wozu Laufzeitfehler der Systemsoftware aber auch Probleme und auftretende Inkonsistenzen bei Zugriffen auf die Meßelektronik in Ausnahmesituationen gehöhren. Programmierfehler, die natürlich ebenfalls eine mögliche Fehlerquelle darstellen und zu einem fatalen Verhalten des Systems führen können, dürfen hier nicht dazugerechnet werden. Bei ihrem Auftreten müssen sie unbedingt beseitigt werden, um so sukzessive die Zuverlässigkeit des Systems zu erhöhen. Dasselbe gilt natürlich auch für Defekte in der Hardware, doch ist es hier in bestimmten Fällen möglich, auftretende Probleme, softwaretechnisch zumindest zu erkennen und dem Experimentator Diagnosehilfen zu geben.

5.5.1 Fehlermeldung

Analog zu den Funktionen der UNIX-Standard-Bibliotheken verwendet MECDAS bei Unterprogrammen ein einheitliches Fehlerschema. Jedes Unterprogramm liefert soweit sinnvoll einen Rückgabewert, aus dem sich immer erkennen läßt, ob innerhalb des Unterprogramms ein Fehler aufgetreten ist. Ähnliches gilt für abgeschlossene Programme. Hier wird der sog. Exit-Code zur Kennzeichnung von fehlerhafter Programmausführung verwendet. Bei Bedarf werden die Fehler genauer spezifiziert durch Klartext-Fehlermeldungen, die nach festen Vorschriften aufgebaut sind (siehe Abschnitt 5.3.1.1). Sie geben dem Benutzer nicht nur individuelle Auskunft über die aufgetretenen Fehler, sondern können auch von übergeordnete Software verwendet und evtl. ausgewertet werden. Die Fehlermeldungen werden in der Regel wie in UNIX üblich über die Diagnose-Ausgabe des jeweiligen Programmes ausgegeben. Treten Fehler in unmittelbar an der Datenaufnahme beteiligten Programmen auf, sorgt die Software dafür, daß die Meldungen für Dokumentationszwecke in den Meßdatenstrom mit aufgenommen werden und so bei der späteren Auswertung der Daten zur Verfügung stehen. Daneben existieren einfache Logging-Mechanismen, mit deren Hilfe Fehler- und andere Diagnose-Meldungen in speziellen Dateien abgelegt werden können.

5.5.2 Ausnahmebehandlung

Aus Effizienzgründen ist es nicht sinnvoll, jede mögliche Fehlersituation durch explizites Abprüfen bestimmter Bedingungen abzuprüfen. Viele Fehler entziehen sich sogar grundsätzlich dieser Möglichkeit. Zu diesem Zweck sind bereits durch die Prozessor-Hardware Vorkehrungen getroffen, um verschiedenste Fehler zu erkennen und zu bearbeiten. Treten solche sog. Ausnahmesituationen wie z. B. die Division durch Null oder der Versuch, fehlerhaften Programmcode mit illegalen Maschineninstruktionen auszuführen, auf, führen sie in Form sog. Traps zu synchronen Unterbrechungen der Programmausführung, deren Bearbeitung

genauso abläuft wie bei asynchronen Unterbrechungen von außen. Art und Umfang von Traps sind von Prozessor zu Prozessor sehr unterschiedlich. Auch ihre Bearbeitung ist extrem maschinenabhängig, da hier prozessorinterne Information ausgewertet werden muß, um die Fehlersituation korrekt zu analysieren und zu behandeln. Sie besitzen jedoch den Vorteil, daß sie mit minimalem Aufwand während des Programmablaufs auskommen. Er beschränkt sich auf das korrekte Aufsetzen der Traps und ihre Behandlung im Fehlerfall. Ansonsten wird keine Rechenzeit zur Fehlerüberprüfung benötigt.

Von besonderer Bedeutung im Rahmen der Datenaufnahme sind Traps, die beim Zugriffsversuch auf fehlerhafte oder nicht existente Hardware oder beim falschen Ansprechen der Hardware ausgelöst werden. Bei MC 68000-Prozessoren, die bisher ausschließlich zur Datenaufnahme eingesetzt werden, sind das die sog. Bus- und Adreßfehler. Sie lassen bei ihrem Auftreten im Rahmen der Datenaufnahme oftmals direkte Rückschlüsse auf Fehler in der Meßelektronik zu und werden daher auch im Datenaufnahmeprogramm bei der Ansteuerung der Elektronik bearbeitet. In der Regel werden bei der Trap-Verarbeitung Fehlermeldungen generiert, die den Experimentator über die Fehlersituation informieren.

In der Initialisierungsphase führt das Auftreten eines Traps ebenso wie bei anderen unmittelbar erkannten Fehlern zu Abbruch des Datenaufnahmeprogrammes. Während der eigentlichen Datenaufnahme wird zwischen verschiedenen Situationen unterschieden. Bei sporadisch auftretenden Fehlern wird eine laufende Messung in der Regel nicht abgebrochen sondern lediglich die Bearbeitung eines Ereignisses vorzeitig beendet. Erst dann, wenn die Fehlerhäufigkeit ein bestimmtes, vom Experimentator konfigurierbares Maß überschreitet, wird die Zustandsmaschine in einen Zustand versetzt, der zu einer Beendigung der Messung führt. Dasselbe geschieht, wenn Fehler permanent auftreten [Kryg95a].

5.6 Systemsoftware

5.6.1 UNIX auf dem Master-Rechner

Die Grundlage der Systemsoftware auf dem Master bildet das Betriebssystem UNIX. Als etabliertes Betriebssystem stellt es die benötigte Funktionalität weitestgehend zur Verfügung und erfordert grundsätzlich keine spezielle Systemsoftware-Entwicklung. Für die einzusetzenden VMEbus-Rechner war keine ausreichende kommerzielle Lösung verfügbar, doch konnte hier auf eine eigene Portierung des Betriebssystems zurückgegriffen werden, die aufgrund der legalen Verfügbarkeit der UNIX-Quellen und der bereits existierenden Unterstützung der weitverbreiteten MC 680xx-Prozessoren mit vertretbarem Aufwand möglich war [Kryg91]. Das im folgenden als VMEbsd bezeichnete Betriebssystem basiert auf der von der Universität in Berkeley entwickelten UNIX-Variante BSD 4.3 [BSD86a, BSD86b, Bach86]. Es besitzt zwar keine kommerzielle Unterstützung, doch wird das hier sehr gut durch Software aus dem Public Domain-Bereich kompensiert, die auch bei den anderen Rechnern eine große Rollte spielt.

Um dieses Betriebssystem für die angestrebten Zwecke der Datenerfassung und Experimentsteuerung auch tatsächlich einsetzen zu können, war noch einiges an Zusatzaufwand zu leisten. Während viele der damit verbundenen Arbeiten sich im wesentlichen auf das Editieren und Ergänzen von Konfigurationsdateien und das Erstellen oder Modifizieren von Kommandoprozeduren beschränkte, war für bestimmte Zwecke eine explizite Systemprogrammierung mit Eingriffen in den Betriebssystemkern angebracht. Eine der wichtigsten Aufgaben war dabei die Inbetriebnahme des sogenannten Diskless-Betriebs, also der Fähigkeit des Systems, vollständig ohne lokalen Massenspeicher zu arbeiten, der üblicherweise zum Betrieb von UNIX benötigt wird. Damit wurde es möglich, den Einsatz fehleranfälliger, mechanischer Komponenten wie Plattenlaufwerke im schlecht zugänglichen und radioaktiver Strahlung ausgesetzten Frontend-Bereich zu vermeiden und so die Anzahl der aktiven Komponenten auf das absolute Minimum zu beschränken. Zur Funktion des Frontend-Rechners ist einzig und allein die CPU-Karte innerhalb des VMEbus notwendig. Die Systemmodifikationen von VMEbsd wurden ergänzt durch Software auf den übergeordneten Workstations, die als NFS-Server den Diskless-Betrieb der VMEbus-Rechner sicherstellen.

Genauso wie die Workstations wurden auch die VMEbus-Systeme in Rechner-Clustern organisiert, um eine einheitliche mit minimalem Aufwand verbundene Verwaltung der Rechner zu gewährleisten. Sie wurden so weit wie möglich in das allgemeine Workstation-Cluster integriert. Workstations und VMEbus-Rechner unterliegen damit einer gemeinsamen Benutzer-Verwaltung und nutzen gleichermaßen netzwerkweit verfügbare Dienste von NFS[8] über Name Service (BIND[9] und YP/NIS[10]) bis hin zur Datensicherung. Am Netzwerk angeschlossene X-Terminals bzw. viele der im Institut eingesetzten Drucker können von allen UNIX-Rechnern gleichberechtigt genutzt werden. Damit steht auf allen Ebenen eine identische Benutzer- und Entwicklungsumgebung zur Verfügung.

5.6.2 Stand-alone-Library für Slave-Rechner

Während auf dem Master von der Funktionalität von UNIX sehr gut Gebrauch gemacht werden konnte, mußte für die Slave-Rechner, die in der derzeitigen Ausbaustufe kein Betriebssystem besitzen, grundlegende Systemsoftware neu implementiert werden. An erster Stelle wurde hier rechnerabhängige Software zum Starten und Initialisieren der Anwendung auf dem Slave-Prozessor, zur Interrupt-Verarbeitung, zur Ausnahmebehandlung etwa bei Speicherzugriffsfehlern und für elementare Ein- und Ausgabe erstellt. Sie wurde ergänzt durch i. w. hardwareunabhängige Software z. T. aus dem Public-Domain-Bereich zur Speicherverwaltung, Bereitstellung einer Systemzeit, für die formatierte Ausgabe von Fehlermeldungen und Debug-Informationen und verschiedene Hilfsroutinen für sonstige Zwecke. Die Software aus beiden Bereichen wurde in entsprechen-

[8] Network File System [Sant90, Ster93]

[9] Berkeley Internet Name Domain: Hierarchisch organisiertes, weltweit verteiltes System zur Verwaltung von Internet-Namen und -Adressen [Albi92].

[10] Yellow Pages/Network Information System: System zur Verwaltung von Namens- und Adreßinformation für ein verteiltes Rechner-System ([Ster91].

asctime	getenv	localtime	signal	strerror
atoi	gethostname	malloc	slaveboot*	strlen
atol	getopt	memchr	slavehalt*	strncmp
bcopy	getpid	memcpy	slavemem*	strncpy
calloc	gettimeofday	memset	slavemode*	strrchr
ctime	gmtime	perror	sprintf	system
execute*	iexecute*	printf	sscanf	time
exit	ioctl	putchar	strcat	vfprintf
fprintf	irqMask*	scanf	strchr	vprintf
free	irqSetmask*	settimeofday	strcmp	vsprintf
fscanf	irqUnmask*	siginterrupt	strcpy	write

Tab. 5.2: Übersicht über die in der Stand-alone-Library verfügbaren Funktionen. Die mit einem Stern gekennzeichneten Funktionen sind abweichend von der Standard-C-Bibliothek hardware- bzw. problemspezifische Ergänzungen.

den Unterprogrammbibliotheken (zusammengefaßt unter dem Stichwort „Stand-alone-Library“) verfügbar gemacht Tabelle 5.2 gibt eine kurze Übersicht über die in der Stand-alone-Library verfügbaren Funktionen.

Ein wichtiger Aspekt bei der Realisierung dieser Software war es, auch auf dem Slave eine klare Trennung zwischen allgemeinen, typischerweise von Systemsoftware zu erledigenden Aufgaben und der mit der speziellen Anwendung verbundenen Problemen vorzunehmen. Insbesondere mit der Isolierung rechnerspezifischer Besonderheiten wurde die Voraussetzung geschaffen, eine zukünftige Portierung der Slave-Software mit minimalem Aufwand durchführen zu können. Soweit notwendig orientierte sich die Implementation der Slave-Systemsoftware konsequent an den entsprechenden UNIX-Schnittstellen. Damit konnte es möglich gemacht werden, die Anwendungs-Software so einheitlich zu gestanlten, daß lediglich durch Binden mit unterschiedlichen Bibliotheken daraus entweder ein unter UNIX ausführbares Programm oder ein voll funktionsfähiges und eigenständig laufendes Programm für den Slave wird. Beide Programme sind dann zueinander funktional vollständig kompatibel.

Diese Tatsache bot insbesondere bei der Software-Entwicklung viele Vorteile, da hier ausgiebig von den Entwicklungswerkzeugen, Debugging-Hilfsmitteln und anderen hilfreichen Betriebssystemeigenschaften von UNIX Gebrauch gemacht werden konnte. Sie erlaubten bei der Realisierung fast aller Teile der Datenaufnahme-Software die Nutzung herkömmlicher Entwicklungsmethoden für Anwendungs-Software, ohne stets mit den Einschränkungen eines betriebssystemlosen Rechners konfrontiert zu werden. Für den Slave-Betrieb mußten keine speziellen Debugging-Mechanismen und unterstützende Software installiert werden, um Fehler zu diagnostizieren und zu beheben. Statt dessen konnte z. B. auf Source Code Debugger unter UNIX zurückgegriffen oder dessen Speicherschutz genutzt werden. Lediglich vom Timing des asynchron arbeitenden Programmes abhängige Fehler oder Seiteneffekte mußten auf dem Slave selbst untersucht werden.

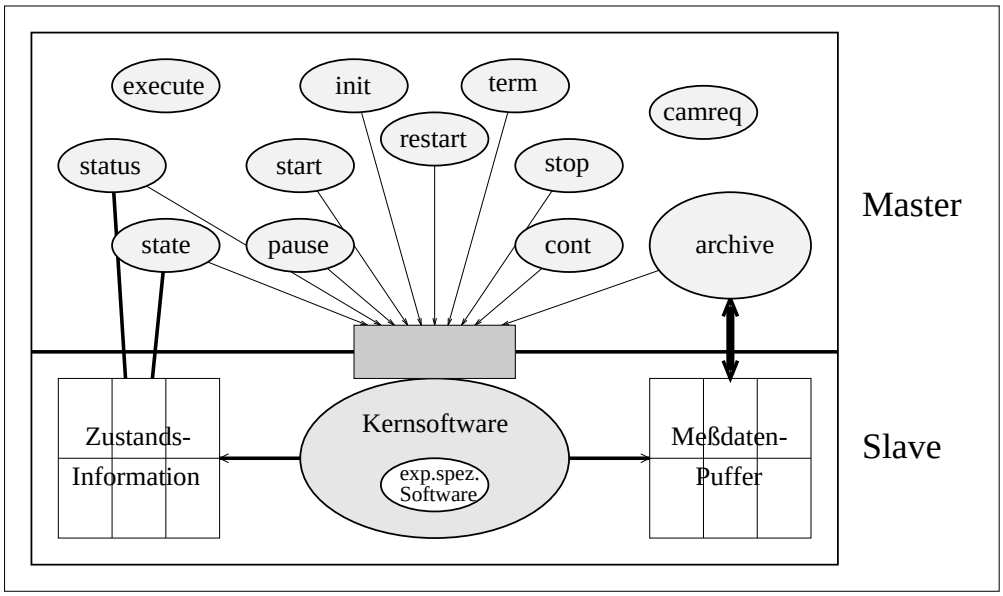

Abb. 5.17: Struktur der zentralen Datenerfassungs-Software

5.6.3 Das Programm `execute`

Die Stand-alone-Library wird ergänzt durch ein auf dem Master lauffähiges Programm mit dem Namen `execute`, das ein korrekt gebundenes Slave-Programm auf den Slave lädt und dort zur Ausführung bringt. Wie auch an anderen Stellen beschränkt sich die derzeitige Implementation vorerst auf den VMEbus. Nach seinem Start liest es die Programmdatei ein, extrahiert daraus Größe, Lade- und Startadresse des Programmcodes und Informationen über die Programmdaten und kopiert Code und Daten an die entsprechenden Stellen in den Arbeitsspeicher des Slaves. Diese werden durch Kommandozeilenargumente, wie sie bei herkömmlichen UNIX-Programmen üblich sind, ergänzt. Zum Abschluß wird das Slave-Programm mithilfe spezieller, von der Hardware des verwendeten Rechners abhängiger Mechanismen gestartet. Nähere Details über Eigenschaften und Realisierung von `execute` können [Kryg95a] entnommen werden.

5.7 Minimal-Datenerfassungssystem

Die bisher vorgestellte Software bestehend aus Datenerfassungs-Server, Klienten und Hilfsprogrammen stellt bereits ein vollständiges Datenerfassungssystem dar, das in dieser Form für Detektortests oder auch Ein- oder einfache Mehrarmexperimente eingesetzt werden kann. Abbildung 5.17 stellt noch einmal die Struktur der zentralen Datenerfassungs-Software von MECDAS dar, die aus einzelnen Bausteinen zusammengesetzt ist, selbst aber wiederum nur einen Baustein für ein komplexes Gesamtsystem bildet (s. Kapitel 7). Neben dem Datenerfassungs-Server sind zur Steuerung der Datenerfassung die Programm bzw. Kommandos `init`, `start`, `stop` usw. notwendig. Sie werden ergänzt durch Hilfsprogramme

wie `status` oder `execute`. Komplettiert wird das System durch das Programm `archive` zur Archivierung der Meßdaten. Zur Durchführung einer Reihe von Messungen muß der Experimentator nun nichts anderes tun, als vom Standard-UNIX-Kommandointerpreter Shell beispielsweise folgende Kommandos aufzurufen:

```
$ init
$ start
$ archive -o file1 &
$ stop
$ start
$ archive -o file2 &
$ stop
...
$ start
$ archive -o filen &
$ stop
$ term
```

Mit `init` wird das Datenerfassungsprogramm gestartet. Es benutzt dazu in der Regel das Programm `execute`. `start` startet eine Einzelmessung, deren Daten mit Hilfe des Programmes `archive` vom Datenerfassungsprogramm entgegengenommen und z. B. in eine Datei des UNIX-Rechners abgelegt werden können. Das Programm bleibt dabei — im Gegensatz zu den anderen Kommandos — für die Dauer der Einzelmessung aktiv. Diese wird mit dem Kommando `stop` beendet. Nach Abspeichern der besonders gekennzeichneten letzten Datenpuffers terminiert dann auch das Archivierungsprogramm automatisch. Das Datenerfassungsprogramm ist dann wieder im Bereitschaftszustand, aus dem heraus analog zu eben weitere Einzelmessungen gestartet werden.

Während des Meßablaufs können zu jedem Zeitpunkt auch andere Steuerungskommandos ausgeführt werden. Mit `status` bzw. `state` kann z. B. der Zustand der Messung überwacht werden, `pause` und `cont` dienen der kurzzeitigen Unterbrechung. Für eine weitergehende Diagnose des Experimentzustands bzw. zur On-line-Analyse der Daten kann analog zu `archive` ein zweites bzw. drittes Programm gestartet werden, das die Meßdaten über die asynchrone Datenschnittstellen des Servers anfordert und sie unabhängig von der Archivierung bearbeiten und auswerten kann (s. Kap. 8).

Die Archivierung der Meßdaten wurde bewußt von der restlichen Experimentsteuerung getrennt, um dem Benutzer die Möglichkeit zu geben, für jede Einzelmessung Archivierungsmethode und Ziel individuell zu spezifizieren. Doch läßt sich hier durch die Verwendung einfacher Kommandoprozeduren eine gute Automatisierung erreichen. Auch der Meßablauf selbst läßt sich sehr gut durch Kommandoprozeduren steuern. Abb. 5.18 zeigt dafür ein Beispiel. Es werden hintereinander automatisch 16 Einzelmessungen von jeweils 15 Minuten Dauer ausgeführt und für die korrekte Archivierung der Daten gesorgt. Vor jeder Messung können mithilfe des fiktiven Kommandos `experiment` z. B. Einstellungen an der Meßapparatur vorgenommen werden.

```
init
for runnumber in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
do
        experiment $runnumber
        start
        archive -o run$runnumber &
        sleep 900
        stop
        wait
done
term
```

Abb. 5.18: Shell-Script zur automatisierten Ablaufsteuerung einer Meßreihe.

Das Beispiel demonstriert deutlich das Bausteinkonzept von MECDAS, das nicht nur die speziell für die Datenerfassung implementierte Software nutzt, sondern auch von allgemeinen Hilfsmitteln Gebrauch macht und so die Funktionalität des Systems steigert, ohne daß dafür zusätzlicher Implementierungsaufwand notwendig wäre. So kann das UNIX-Kommando `sleep` z. B. durch eine Schleife mit regelmäßiger Statusausgabe (Abb. 5.19) oder eine weitere Kommandoprozedur ersetzt werden, die die Länge einer Messung nach anderen Kriterien definiert. Auch bei der Archivierung der Daten bieten sich Möglichkeiten an, sie anstelle direkt in Dateien über UNIX-Pipes an andere Programme weiterzuleiten, die für eine On-line-Bearbeitung der Daten und ihre spezielle Archivierung sorgen.

5.8 Experimentspezifische Software

Wie schon erwähnt wird die Kernsoftware ergänzt durch wenige experimentspezifische Softwareteile, die notwendig sind, um damit ein konkretes Experiment durchführen zu können. Sie müssen die Information über Art und Umfang der Meßelektronik enthalten und, wie im Rahmen der Datenerfassung darauf zuzugreifen ist. Für diese Zwecke besitzt die Kernsoftware definierte Schnittstellen. Sie erfordern das Einbinden eines Satzes von Unterprogrammen, die im wesentlichen zu den in Abschnitt 5.2 beschriebenen Zustandsübergängen des Datenerfassungs-Servers korrespondieren.

5.8.1 Aufgaben

Die MECDAS-Kernsoftware benötigt in den einzelnen Phasen der Datenerfassung experiment- bzw. detektorspezifische Software, die folgende Aufgaben abdeckt:

Initialisierung und Terminierung
Zu Beginn einer Meßreihe muß die Meßelektronik in einen definierten Anfangszustand gebracht werden. Es müssen evtl. Einstellungen und Eichun-

```
I.   t=0
     while [ $t -lt 900 ]
     do
             status
             sleep 10
             t=`expr $t + 10`
     done

II.  while [ `events` -lt 10000 ]
     do
             sleep 10
     done

III. archive -o - | rsh asterix dd of=/dev/nrmt0h bs=8k
```

Abb. 5.19: Einfache Beispiele für Kommandoprozeduren, die bei der Datenerfassung verwendet werden können. I.: Wiederholte Statusausgabe während einer Zeit von 900 Sekunden. II.: Warten, bis die Anzahl der registrierten Ereignisse größer als 10000 ist. III.: Archivierung der Meßdaten über Netz auf ein Magnetband, das an einem entfernten Rechner angeschlossen ist.

gen der Meßapparatur vorgenommen werden. Dazu sind spezielle Detailinformationen über die verwendete Hardware notwendig. Oftmals müssen dabei wie auch bei späteren Zugriffen auf die Meßelektronik bestimmte Methoden angewendet und Bearbeitungsvorschriften berücksichtigt werden. Zu diesem Zweck müssen in der Initialisierungsphase auch rein softwaretechnische Maßnahmen getroffen werden, von der einfachen Initialisierung von globalen Variablen über Speicherallokierungen bis hin zum Anlegen von Tabellen und Listen, die später u. a. auch für eine effizientere Bearbeitung der auszuführenden Aufgaben benötigt werden. Auch zur Beendigung einer Meßreihe fallen Aufgaben an, um die Meßelektronik in einen definierten Zustand zu bringen. Im Software-Bereich sind evtl. Aufräumarbeiten notwendig.

Start und Beenden einer Einzelmessung

Auch hier sind evtl. spezifische Aktionen an der Meßelektronik notwendig. Globale Daten, die nur für die Dauer einer Einzelmessung Gültigkeit haben, müssen zurückgesetzt, statistische Informationen aktualisiert, Startzeit oder laufende Nummer der Messung festgestellt und eventuell weitere Verwaltungsaufgaben ausgeführt werden. Analog zum Start einer Messung fallen auch bei der Beendigung Aufgaben an. Sie werden noch ergänzt durch Vorkehrungen zur Archivierung der im Rahmen der Messung angesammelten Informationen z. B. statistischer Art wie Anzahl von Ereignissen unterschiedlichen Typs, aufakkumulierten Zählerinhalten, Ereignisraten, Totzeit usw.

Durchführung der Datenaufnahme

Nach dem Nachweis eines physikalischen Ereignisses durch die Detektorsysteme wird Software benötigt, die individuell und abgestimmt auf die Meßapparatur die Auslese der Meßelektronik vornimmt. Dazu gehöhrt u. a. das spezielle Ansprechen einzelner Geräte über CAMAC, Fastbus, den VME-bus oder andere Schnittstellen. Auch eine evtl. notwendige Kommunikation mit intelligenten Subsystemen über spezielle Hardware (z. B. optische Verbindungen, Transputerlinks) muß korrekt durchgeführt werden. Wichtig ist oftmals auch die Reihenfolge, in der die einzelnen Aktionen abgearbeitet werde müssen. Speziell an Experiment und Hardware angepaßte Methoden erlauben in vielen Fällen ein selektives Auslesen und Bearbeiten der Meßdaten und helfen bei der Reduzierung der Totzeit bzw. der zu archivierenden Datenmenge. Eine Datenreduktion kann auch erreicht werden durch eine an die Auslese anschließende Nachbehandlung der Daten. Auch eine On-line-Vorauswertung ist denkbar, soweit ihr Rechenaufwand das Totzeitverhalten nicht nachhaltig stört. Neben der Auslese gehören auch das Zurücksetzen der Meßelektronik und Vorbereiten für die Aufnahme des nächsten Ereignisses zum Aufgabenbereich der experimentspezifischen Software.

Verpacken von Meßdaten

Die bei der Datenaufnahme erfaßten Meßdaten müssen so kodiert werden, daß sie bei ihrer Auswertung ohne Informationsverlust wieder restauriert werden können. Dazu müssen die ausgelesenen Werte abgespeichert und mit eindeutigen Adressierungsinformationen versehen werden, um sie später wieder richtig zuordnen zu können. Dafür ist experimentspezifische Kenntnis über jedes einzelne Datum unerläßlich.

Steuerung der Verriegelungselektronik

Die Eigenheiten der Verriegelungselektronik, die bei Zustandsübergängen der Datenerfassungs-Software angesteuert werden muß, müssen berücksichtigt werden, um den Datenfluß zwischen Meßelektronik und Rechner durch Freigeben bzw. Blockierung der Verriegelung korrekt zu steuern.

Weitere Aufgaben

Neben den genannten, für eine korrekte Durchführung der Datenerfassung unerläßlichen Aufgaben gibt es noch eine Reihe weiterer Aufgabenstellungen, die evtl. ebenfalls einer experiment-, detektor- oder hardwarespezifischen Erledigung bedürfen. Dazu gehören:

- kurzzeitiges Unterbrechen einer Messung
- Ermittlung des aktuellen Zustands von Messung und Apparatur
- Bearbeitung von Fehlern beim Zugriff auf die Meßelektronik

Auch beim Starten und Beenden der Datenerfassungs-Software selbst können u. U. vom Experiment bzw. von speziellen Eigenschaften der verwendeten Rechner-Hardware abhängige Vorgehensweisen notwendig werden.

5.8.2 Schnittstellen

Für all die genannten Aufgaben ist in der Kernsoftware von MECDAS der Aufruf von jeweils eines Unterprogrammes vorgesehen. Tabelle 5.3 gibt eine Übersicht mit kurzer Funktionserläuterung wieder. Die Unterprogramme lassen sich in zwei Gruppen unterteilen. Die Funktionen der ersten Gruppe werden in der Regel mit zwei Parametern aufgerufen, die Adresse und Größe eines Speicherbereichs spezifizieren, den die Funktion mit Daten aufffüllen kann. Mit Ausnahme von `readoutStatus()` ist dieser Speicherbereich Bestandteil des aktuellen Datenpuffers, der in erster Linie zur Aufnahme der in `readoutMain()` eingelesenen Meßdaten dient, aber natürlich auch andere Informationen aufnehmen kann. Für Konsistenz und Format der abgespeicherten Daten hat im allgemeinen der Anwender zu sorgen. Nach Einfüllen der Daten in den Puffer gibt die Funktion die Adresse des nun noch verbleibenden Speicherbereichs zurück. Die Funktionen der zweiten Gruppe dienen ausschließlich der Steuerung des Meßablaufs durch die Verriegelungselektronik.

Funktion	**Kurzbeschreibung**	**ZÜ**
`readoutInit`	Initialisierung von Hard- und Software	INIT
`readoutTerm`	Terminierung von Hard -und Software	TERM
`readoutStart`	Start einer Einzelmessung	START
`readoutStop`	Stopp einer Einzelmessung	STOP
`readoutPause`	kurzzeitige Meßunterbrechung	PAUSE
`readoutCont`	Fortführung einer unterbrochenen Messung	CONT
`readoutStatus`	Ausgabe des Zustands der Auslese	
`readoutState`	Ausgabe des Zustands der Messung	
`readoutMain`	Auslese-Routine	IRQ
`readoutRecover`	Behandlung von Hardware-Fehlern	ERROR
`readoutError`	Kennzeichnung der Meßdaten bei Fehlern	ERROR
`readoutBuf`	Vorbereiten der Auslese	IRQ
`readoutBoot`	Start des Server-Programmes	BOOT
`readoutHalt`	Beenden des Server-Programmes	HALT
`readoutFormat`	Formatierung der Meßdaten	
`controlInit`	Initialisierung der Verriegelungselektronik	INIT
`controlTerm`	Zurücksetzen der Verriegelungselektronik	TERM
`controlOpen`	Freigabe der Messung	
`controlClose`	Verriegelung der Messung	
`controlPause`	globale Verriegelung der Elektronik	PAUSE
`controlCont`	Freigabe nach globaler Verriegelung	CONT
`controlPoll`	Abfrage der Triggerbedingung	

Tab. 5.3: Übersicht der Funktionen, die die experimentspezifische Schnittstelle der Kernsoftware definieren ergänzt durch eine kurze Funktionsbeschreibung und den jeweils korrespondierenden Zustandsübergang.

Kapitel 6

Die Experimentkonfiguration

6.1 Das Konzept

Bei der Konzeption von MECDAS wurde besonderes Augenmerk auf hohe Flexibilität des Datenerfassungssystems gelegt. Ziel war es, das System bei einer Vielzahl ähnlicher, aber im Detail doch recht unterschiedlicher Experimente an MAMI einsetzen zu können, ohne daß dabei von Seiten des Benutzers, wenn überhaupt, tiefergehende Eingriffe in die Software notwendig werden. Das konnte, wie bereits im vorangeganenen Kapitel beschrieben, nach der Analyse der im Rahmen der Datenerfassung anfallenden Aufgaben und ihrer strengen Aufgliederung in experimentunabhängige und experimentspezifische Gruppen zu einem großen Teil dadurch erreicht werden, daß alle wesentlichen experimentunabhängigen Aufgaben durch die Implementation entsprechender allgemeiner Software abgedeckt werden konnte. Lediglich für einen sehr kleinen, aber entscheidenden Teil der zu erledigenden Aufgaben sind Detailkenntnisse über das Experiment notwendig, die dann in entsprechende, bestmöglich an das jeweilige Experiment angepaßte Software umgesetzt werden müssen.

Die Kernsoftware von MECDAS definiert hierfür Schnittstellen, läßt aber dem Anwender ansonsten große Freiheiten. So ist es möglich, das System in einem weiten Anwendungsbereich einzusetzen, der deutlich über die an MAMI durchzuführenden Experimente geht. Beschränkt man sich jedoch auf diese, zeigen sich auch in den experimentspezifischen Details eine Vielzahl von ähnlichen, charakteristischen Eigenschaften. Diese Ähnlichkeiten können zwar nicht unmittelbar durch allgemeinen und gleichzeitig effizienten Programmcode abgedeckt werden, doch erlauben sie es, dem Experimentator Hilfsmittel zur einfachen Anpassung des Systems an experimentspezifische Gegebenheiten an die Hand zu geben. Mit der Entwicklung solcher Hilfsmittel konnte erreicht werden, daß nicht mehr für jedes Experiment bzw. für jeden experimentellen Aufbau die individuelle Implementation experimentspezifischer Software notwendig wird, die zwar die optimale Anpassung an die speziellen Anforderungen und Gegebenheiten erlaubt, doch trotz bereits verfügbarer Software für allgemeine Aufgaben u. U. nur mit großem Aufwand und unter Erhöhung der Fehleranfälligkeit zu realisieren ist.

Aus diesem Grund wurde für MECDAS ein Verfahren und die dazu notwendige Software entwickelt, die von dem Experimentator i. w. nur noch verlangt, die experimentspezifische Information in einer nach bestimmten systematischen Kriterien aufbereiteten Form anzugeben, so daß sich eine vollständige Beschreibung des experimentellen Aufbaus und anderer für die Durchführung des Experiments wichtiger Tatsachen ergibt. Damit ist entsprechende Konfigurations-Software in der Lage, einerseits Tabellen, andererseits aber auch effizienten, leistungsfähigen und an die speziellen Randbedingungen angepaßten Programmcode für Auslese, Verpacken, Dekodieren, Eventbuilding und Analyse zu erzeugen.

Nach näherer Betrachtung zeigt sich, daß sich ein kernphysikalisches Experiment in Bezug auf die Datenerfassung in der Regel nach drei unterschiedlichen Gesichtspunkten beschreiben läßt, die jeweils nur einen Teilaspekt der Gesamtproblematik abdecken, jedoch zusammen eine sehr gute Beschreibung des Gesamtsystems für die Datenerfassung bieten. Zum einen ist das die sogenannte **physikalische Beschreibung** des experimentellen Aufbaus. Sie macht Aussagen über Struktur und Art der Experimentelektronik und stellt damit wichtige Informationen insbesondere für die Datenauslese zur Verfügung. Als zweites stellt die sogenannte **logische Beschreibung** eine hardwareunabhängige Charakterisierung des Experiments dar, die unverzichtbar zur systematischen Bearbeitung vieler Aufgaben im Bereich der Datenerfassung ist und gleichzeitig eine für den Benutzer besser zu handhabende Schnittstlle bildet. Diese statischen Beschreibungen werden ergänzt durch spezielle **Bearbeitungsvorschriften**, die das dynamische Verhalten in den verschiedenen Phasen der Datenerfassung beschreiben. In MECDAS werden diese Informationen genutzt, um auf weitestgehend automatisierte Art und Weise die experimentspezifischen Softwareteile zu erhalten, die die Kernsoftware zu einem in sich abgeschlossenen, leistungsfähigen Datenaufnahmeprogramm für ein konkretes Experiment vervollständigt.

Der Experimentator muß diese Beschreibung in einer stark an die Programmiersprache C angelehnten Form spezifizieren und in einer oder mehreren Textdateien zur Verfügung stellen. Die Syntax von C erlaubt es, eine beliebig komplexe Experimentkonfiguration in Form von C-Strukturen eindeutig und sehr kompakt zu beschreiben und in eine rechnerlesbare Form zu bringen. Auch Bearbeitungsvorschriften lassen sich damit einfach spezifizieren. Sie bestehen letztendlich aus C-Anweisungen, mit denen sich auch sehr komplizierte Algorithmen formulieren lassen. Mithilfe eines speziellen C-Parsers und unter Ausnutzung von Standard-Hilfsmitteln wie C-Compiler und C-Präprozessor wird diese experimentspezifische Konfigurationsinformation analysiert und in eine Form gebracht, die es der allgemeinen Software von MECDAS erlaubt, damit zu arbeiten.

Die Experimentbeschreibung ist nicht nur die Grundlage für die Datenaufnahme; sie wird auch komplett zu den zu archivierenden Daten gepackt, so daß sie z. B. ebenso für Auswerteaufgaben unmittelbar zur Verfügung steht. Damit sind die Daten, die letzendlich auf einem Magnetband oder einer Plattendatei abgespeichert sind, selbstbeschreibend und zu jedem Zeitpunkt ohne zusätzliche Information eindeutig identifizierbar. Die Experimentbeschreibung bildet somit die experimentspezifische Datenbasis für alle Aktivitäten von MECDAS von der Datenaufnahme bis hin zur Off-line-Analyse (Abb. 6.1).

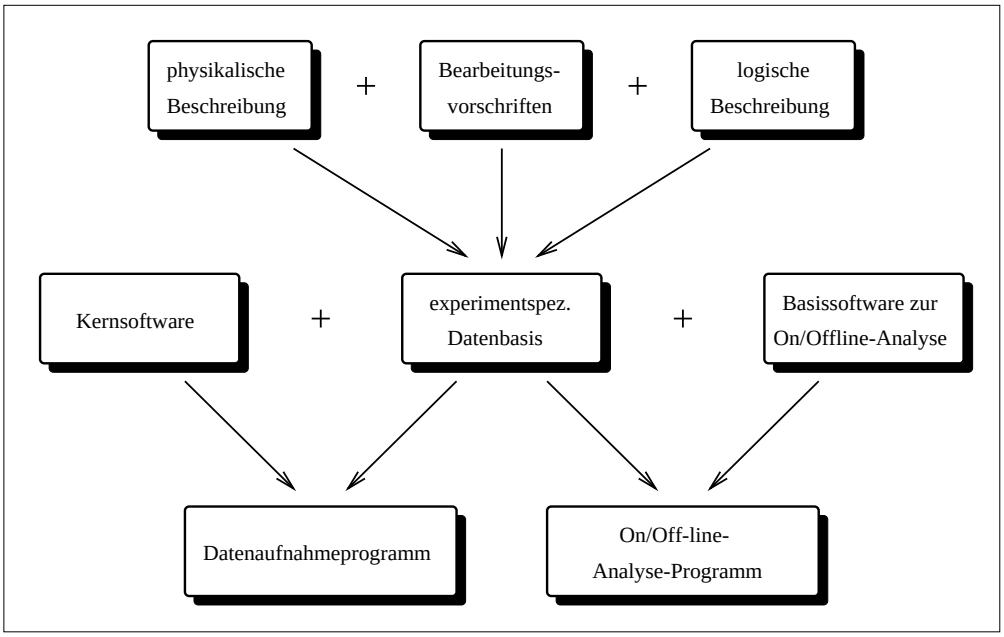

Abb. 6.1: Die Experimentbeschreibung und ihre funktionale Einbettung in die restliche MECDAS-Software

6.2 Die physikalische Beschreibung

6.2.1 Definition

Die physikalische Experimentbeschreibung vermittelt Informationen über den konkreten Aufbau der Experimentelektronik, die insbesondere für die Datenaufnahme unerläßlich sind. Sie beschreibt die vom Rechner aus anzusprechende Meßelektronik des Experiments bzw. der Detektorsysteme und spezifiziert dabei die verwendeten Geräte. Sie gibt Auskunft, welche Gerätetypen eingesetzt werden, und an welcher Stelle CAMAC, Fastbus, VMEbus oder andere Meßelektronik Verwendung findet. Auch Typ und Name der verwendeten Rechner können ebenso wie die verwendeten Kommunikationsmechanismen zur Beschreibung gehören. Besonders wichtig sind schließlich Informationen über Bauart, Anzahl und Adressen von Interfaces, Controllern und Modulen und die Mechanismen, wie mit ihnen zu arbeiten ist. Diese Informationen sind nötig, um zu Beginn eines Experiments eine korrekte und vollständige Initialisierung der Meßelektronik vornehmen zu können und bei der Auslese in korrekter Weise auf die richtige Hardware zuzugreifen, ohne daß der Experimentator spezielle Software dafür schreiben muß.

6.2.2 Die Hardware-Datenstruktur

Die Beschreibung nutzt zur Vereinfachung die oftmals vorhandene hierarchische Hardware-Struktur aus, wie sie beispielsweise bei CAMAC sehr gut ausgeprägt ist (Abb. 6.2). Programmtechnisch ist in MECDAS die physikalische Beschrei-

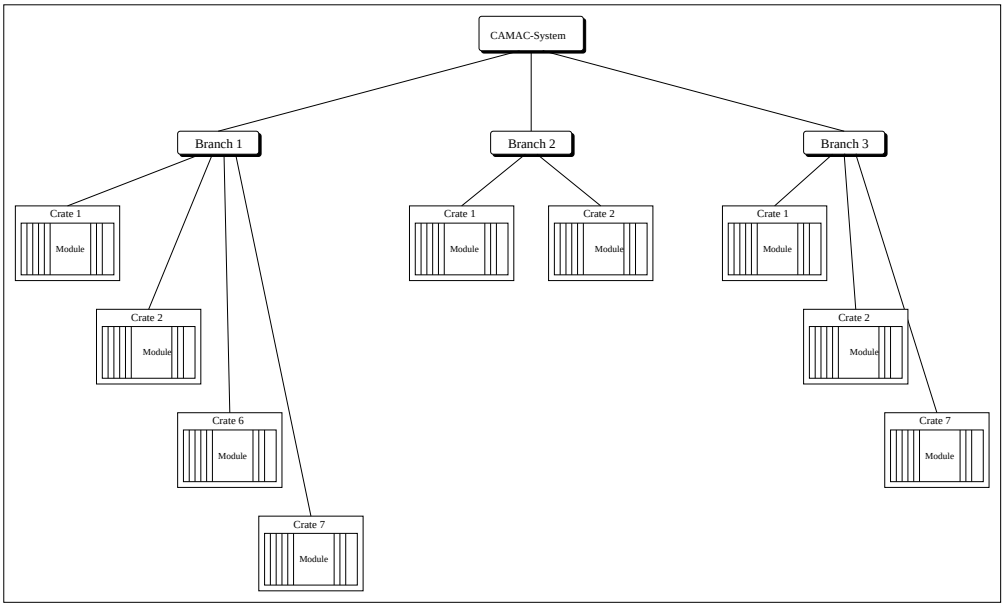

Abb. 6.2: Die hierarchische Struktur eines CAMAC-Systems.

bung als ein logischer Baum realisiert, bestehend aus verschiedenen Teilbäumen, die jeweils die hierarchisch organisierte Information über CAMAC-, Fastbus- und VMEbus-Hardware und die direkt an der Datenerfassung beteiligten Rechner enthalten. Sie sind im wesentlichen ein abstrahiertes Abbild des jeweiligen Hardware-Systems. Die bei all diesen Systemen in der einen oder anderen Form vorkommenden einzelnen kleinsten Hardware-Einheiten (in der Regel Module) entsprechen den Blättern des Baums, übergeordnete Einheiten mit Controllern oder Interfaces stellen die Knoten dar.

Alle Teilbäume sind aus denselben Grundelementen aufgebaut, die letztendlich die Information über die eingesetzten Geräte enthalten. Für jedes Gerät steht die Information über den Typ, seine relative Adresse innerhalb eines Teilbaums und die Zuordnung zu einer Gerätegruppe zur Verfügung. Dieses Grundelement läßt sich sehr einfach durch eine C-Datenstruktur beschreiben. Sie ist in Abbildung 6.3 dargestellt.

```
#define Hardware        struct hardware

struct hardware {
        long     h_addr;                    /* relative Geraeteadresse    */
        Device   *h_device;                 /* Verweis auf Geraetebeschr. */
        Hardware *h_up;                     /* uebergeordneter Baumknoten */
        void     *h_down;                   /* Geraetegruppe              */
        Hardware *h_next;                   /* Nachbarblatt im Baum       */
};
```

Abb. 6.3: C-Struktur zur Beschreibung der zur Datenerfassung benutzen Hardware.

6.2.3 Die Beschreibung durch den Experimentator

Die im vorangegangenen Abschnitt erläuterte Hardware-Datenstruktur ist auch die Basis für die physikalische Experimentbeschreibung, wie sie vom Experimentator zur Verfügung gestellt werden muß. Effektiv besteht diese aus mehreren miteinander verknüpften Vektoren dieser Datenstruktur, die gemäß der relativen Geräteadressen mit den konkreten Parametern initialisiert sind. Abbildung 6.4 zeigt ein Beispiel, wie die physikalische Beschreibung für eine einfache Konfiguration aussehen kann. In diesem Fall enthält die Meßelektronik ein CAMAC-System bestehend aus einem Branch mit einem Crate, das vier Module enthält, und ein VMEbus-System mit einer CPU, die im Rahmen der Datenerfassung als Slave eingesetzt wird. Im Anhang ist ein komplexeres Beispiel wiedergegeben.

Zum Erstellen einer solchen Beschreibung muß der Experimentator die Interna jedoch nicht kennen. Unter Zuhilfenahme einiger C-Präprozessor-Makros sind lediglich einfache Konstruktionsvorschriften zu beachten. Für jedes Gerät ist, in der Regel indirekt, eine relative Adresse, der Gerätetyp und eine Gerätegruppe zu spezifizieren. Damit enthält die physikalische Beschreibung implizit oder auch explizit die Adreßinformation für jede darin enthaltene Hardware, aus der die Konfigurations-Software unter Ausnutzung der Baumstruktur alle benötigten Adressen in konkreter Form für die Datenerfassung generieren kann, so z.B. der sog. CNAF[1] für CAMAC-Zugriffe oder die Internet-Adresse eines Rechners. Weitergehende Informationen sind für jeden Gerätetyp in speziellen Gerätebeschreibungen vorhanden, die in Form einer dynamisch erweiterbaren Bibliothek vorliegen. Mithilfe der Gerätegruppen schließlich lassen sich mehrere, jeweils gleichartige Geräte logisch zusammenfassen. Eine solche Gruppe wird durch einen symbolischen Namen identifiziert, mit dessen Hilfe die einzelnen Datenkanäle an anderer Stelle einheitlich angesprochen werden können. Sie bilden den Anknüpfungspunkt zur logischen Experimentbeschreibung.

6.3 Die Gerätebibliothek

6.3.1 Aufteilung der Hardware-Beschreibung

Die zur Charakterisierung der Meßelektronik im Rahmen der Datenerfassung notwendige Information läßt sich grundsätzlich in zwei Gruppen aufteilen. Die erste entspricht der physikalischen Beschreibung und umfaßt Angaben über die Struktur und Zusammensetzung der Meßelektronik eines speziellen Experiments. Neben Adreßinformationen gehört dazu auch die Spezifizierung des Typs der verwendeten Geräte. Die Hardwaredetails für diese Geräte selbst werden separat gehalten. Sie bilden die zweite Gruppe in Form gerätespezifischer Programm-Module. Durch diese Aufteilung ist es nicht nur möglich, die physikalische Beschreibung übersichtlich und frei von redundanter Information zu halten und damit den Benutzer bei ihrer Erstellung zu entlasten, sondern sie bietet zusätzliche Vorzüge u. a. auch in softwaretechnischer Hinsicht.

[1] Tupel aus Crate-Nummer, Einschub-Nummer, Subadresse und Funktionscode

```
/* --- Verweise auf Geraete/Hardware-Beschreibungen aus Bibliothek --- */

extern Device   lecroy_2228;            /* ADC von LeCroy, Typ 2228    */
extern Device   lecroy_2249;            /* TDC von LeCroy, Typ 2249    */
extern Device   microbusy;              /* Event-FF KPH-Microbusy      */
extern Device   pattern_unit;           /* Pattern-Unit, Standard-Typ  */
extern Device   ces_a2;                 /* A2-Crate-Controller von CES */
extern Device   cbd_8210;               /* Branch-Controller CBD 8210  */
extern Device   e5;                     /* Eltec E5 VMEbus CPU         */

/* ------------------- Definition von Modulgruppen ------------------- */

Group   adc;                            /* logische Gruppe der ADCs    */
Group   tdc;                            /* logische Gruppe der TDCs    */
Group   eventff;                        /* Verriegelungselektronik     */
Group   pattern;                        /* logische Gruppe Hit-Pattern */
Group   slave;                          /* Slave-Rechner zur Datenerf. */

/* ------------------------ Hardware-Setup -------------------------- */
/* ----------------- CAMAC-Crate mit Readout-Hardware ---------------- */

Crate crate1  = {
        MODULE(lecroy_2228, adc),       /* ADC in Slot 1               */
        EMPTY,                          /* Slot 2 leer                 */
        EMPTY,                          /* Slot 3 leer                 */
        EMPTY,                          /* Slot 4 leer                 */
        MODULE(lecroy_2249, tdc),       /* TDC in Slot 5               */
        EMPTY,                          /* Slot 6 leer                 */
        EMPTY,                          /* Slot 7 leer                 */
        EMPTY,                          /* Slot 8 leer                 */
        EMPTY,                          /* Slot 9 leer                 */
        EMPTY,                          /* Slot 10 leer                */
        EMPTY,                          /* Slot 11 leer                */
        MODULE(microbusy, eventff)      /* Eventff in Slot 12          */
        EMPTY,                          /* Slot 13 leer                */
        MODULE(pattern_unit, pattern)   /* Pattern Unit in Slot 14     */
};

/* ------------------------- CAMAC-Branch --------------------------- */
Branch branch = {
        CRATE(ces_a2, crate1)           /* obiges CAMAC-Crate m. A2-CC */
};

/* ------------------------- CAMAC-System --------------------------- */
Camac camac = {
        BRANCH(cbd_8210, branch)        /* obiger Branch mit CBD 8210  */
};

/* --------------- An Datenaufnahme beteiligte Rechner --------------- */
Computer computer = {
        FRONTEND(e5, slave)             /* E5 als Slave-Prozessor      */
};
```

Abb. 6.4: Beispiel für eine physikalische Experimentbeschreibung.

Für jeden Gerätetyp ist nur eine einzige, etwas allgemeinere Beschreibung von Hardware-Eigenschaften und -Funktion notwendig, auch wenn mehrere Exemplare von Geräten der gleichen Bauart eingesetzt werden. Durch ihre weitgehende Kapselung in einzelne Programm-Module ist ein sehr individuelle Beschreibung auch von Details möglich. Gleichzeitig können Seiteneffekte vermieden werden. Eine wohldefinierte Schnittstelle sorgt für eine einheitliche Anknüpfung dieser Informationen an die allgemeine Datenerfassungs-Software.

6.3.2 Die Gerätebeschreibung

Grundlage der Gerätebeschreibung von MECDAS stellt die in Abbildung 6.5 wiedergegebene Datenstruktur dar. Sie untergliedert sich in drei Bereiche. Der erste dient zur Klassifizierung des zu beschreibenden Geräts nach

- Gerätefamilien wie CAMAC, Fastbus, VMEbus oder auch Computer
- familienspezifische Geräteklassen zur Unterscheidung zwischen einfachen Modulen oder übergeordneten Controllern
- allgemeine Gerätespezifizierungen, die die Funktion des Gerätes kennzeichnen, wie z. B. ADC, TDC, Zähler usw.

Anschließend folgt eine Liste mit allgemeinen Geräteeigenschaften, die bei Bedarf gerätespezifisch erweiert werden kann. Sie enthält standardmäßig neben dem Namen des Geräts Informationen über Anzahl und Wortbreite der Datenkanäle, die dieses Gerät zur Verfügung stellt, und wird ergänzt durch einen typspezifischen Satz von Flaggen für verschiedenste Zwecke. Der dritte Bereich besteht aus einer Liste von Informationen über die Zugriffsmechanismen auf die Hardware. Sie umfaßt acht Komponenten, die jeweils aus einem Funktionscode und einem optionalen Verweis auf ergänzende Software bestehen und damit gerätespezifische Informationen und Methoden zur Auslese, zum Initialisieren, Zurücksetzen und andere Mechanismen des Geräts repräsentieren. Im Falle einfacher CAMAC-Module (s. Beispiel in Abb. 6.6) reicht hier in der Regel die Angabe des entsprechenden CAMAC-Funktionscodes, der zur Ausführung einer Lese-, Schreib- oder Kontroll-Operation verwendet wird. Er ergänzt die aus der physikalischen Beschreibung erhältlichen Adreßinformation zu einem kompletten CNAF und definiert damit für einen CAMAC-Datenkanal vollständig den Zugriffsmechanismus.

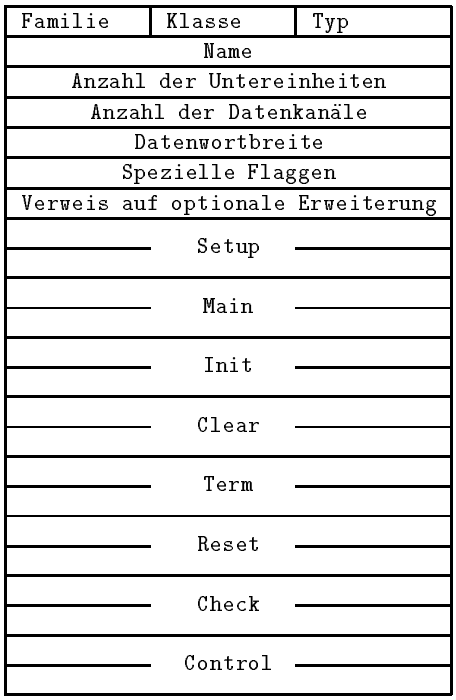

Abb. 6.5: Aufbau der Datenstruktur zur Gerätebeschreibung.

```
Device  lecroy_2228a = {
        MCAMAC, MMODULE, MTDC,          /* module type                        */
        "lecroy_2228a",                 /* module name                        */
        8,                              /* number of subaddresses             */
        1,                              /* 1 word to read, no block transfer */
        2048,                           /* range 0 ... 2047                   */
        CAMAC16 | CAMACx | CAMACq | CAMAClam | DEVoverflow(11),   /* flags */
        NULL,                           /* no expansion                       */
        FNULL,                          /* no setup function                  */
        F(0),                           /* read CNAF                          */
        F(24),                          /* initialization CNAF (disable lam) */
        F(9),                           /* clear module (and lam) CNAF        */
        FNULL,                          /* no special termination function   */
        FNULL,                          /* no special reset function          */
        FNULL,                          /* no special check funtion           */
        FNULL                           /* no special control funtion         */
};
```

Abb. 6.6: Beispiel für eine konkrete Geräte-Beschreibung

Die Datenaufnahme-Software benötigt damit keine weiteren Informationen, um individuell auf die CAMAC-Hardware zuzugreifen.

Im allgemeinen Fall reicht jedoch ein einziger Parameter und oftmals sogar auch ein Satz aus mehreren Parametern nicht immer aus, um speziell auf die Hardware abgestimmte Zugriffe — insbesondere auch effizient — vornehmen zu können. Eine weitergehende Parametrisierung der Zugriffsmechanismen würde trotz des damit verbundenen zusätzlichen Aufwands den allgemeinen Fall nur unzureichend abdecken können und in Sonderfällen unnötige Einschränkungen bedeuten. Statt dessen bietet sich hier die Verwendung von herkömmlichem Programmcode an, mit dem beliebige hardwarespezifische Zugriffmechanismen formuliert werden können. Für jede Ein/Ausgabe-Aktion steht dann ein Unterprogramm zur Verfügung, dessen Adresse in das entsprechende Feld der Datenstruktur zur Gerätebeschreibung eingetragen ist. Damit ist die übergeordnete Software beim Zugriff auf beliebige Hardware in der Lage, diese ganz individuell anzusteuern, ohne selbst tiefergehende Informationen darüber zu besitzen.

6.3.3 Objektorientierte Realisierung

Die beschriebene Datenstruktur stellt das Bindeglied zwischen der allgemeinen Software zur Datenaufnahme und der hardwarespezifischen Routinen dar. Die Schnittstelle wird lediglich ergänzt durch die einheitliche Definition von Übergabeparametern für den Aufruf dieser Routinen und ihren Rückgabewert.

Hier findet sich ein Beispiel, wie mit der konsequenten Anwendung der Ideen der objektorientierten Programmierung eine saubere Trennung von unterschiedlichen Software-Elementen, für die aber die Notwendigkeit der engen Zusammenarbeit besteht, erreicht werden konnte, auch wenn OOP-Hilfsmittel nicht unmittelbar genutzt werden konnten. Die einzelnen Gerätebeschreibungen stel-

len jeweils eine in sich abgeschlossene Objektklasse dar, die Methoden zur Manipulierung ihrer mit konkreter Hardware verknüpften Instanzen bereitstellt. Gerade in einem besonders kritischen Bereich der MECDAS-Software, in dem bei Bedarf eine eigene Programmierung des Anwenders notwendig wird, hilft dieser Ansatz, mögliche Fehler und ungewollte Seiteneffekte auf ein Minimum zu beschränken, ohne dabei an Effizienz zu verlieren.

Softwaretechnisch ist dieses Konzept umgesetzt, indem jedem im Rahmen von MECDAS eingesetzten Gerätetyp ein Programm-Modul zugeordnet ist, das eine mit den das Gerät beschreibenden Werten initialisierte Geräte-Datenstruktur enthält und bei Bedarf eine Reihe von lokalen, also nur innerhalb des Moduls bekannten Funktionen. Nach dessen Compilation kann es wie jedes andere Programm-Modul mit der Software, die es benötigt, zusamengebunden werden.

Für einen großen Teil der im Institut für Kernphysik eingesetzten Geräte wurden solche Beschreibungsmodule implementiert und zu einer Bibliothek zusammengefaßt, die Standardbestandteil der MECDAS-Software ist. Diese Bibliothek ist bei Bedarf sehr leicht erweiterbar. Aber auch die Verwendung individueller Gerätebeschreibungen unabhängig davon ist mit den vorhandenen Hilfsmitteln unter Ausnutzung von Compiler und Linker einfach. Tabelle 6.1 zeigt einen Überblick der bisher in der Bibliothek verfügbaren Gerätebeschreibungen.

Gerätetyp	**Kurzbeschreibung**
CBD 8210	CAMAC-Branch-Controller als VMEbus-Karte
CES A2	A2-Crate-Controller
E5	Frontend-Rechner Eurocom 5
E5timer	Timerbaustein auf Eurocom 5
E6	Frontend-Rechner Eurocom 6
KPH 0815	Einfaches Beispiel
LeCroy 1882n	Fastbus-ADC-Modul
LeCroy 2228a	CAMAC-TDC-Modul
LeCroy 2229	CAMAC-TDC-Modul
LeCroy 2249a	CAMAC-ADC-Modul
LeCroy 2249w	CAMAC-ADC-Modul
LeCroy 2551	CAMAC-Scaler-Modul
LeCroy 4299	TDC-Readout-System
LeCroy 4434	CAMAC-Scaler-Modul
LeCroy 4434-24	CAMAC-Scaler-Modul mit 24-Bit-Untersttzung
LeCroy 4448	CAMAC Input Coincidence Register/Latch
Microbusy	CAMAC-Modul zur Steuerung der Verriegelungselektronik
Pattern Unit	CAMAC-Pattern-Unit
Scaler KPH	CAMAC-Scaler-Modul (Instituts-Eigenbau)
SMI 1821	Fastbus Segment Manager/Interface

Tab. 6.1: Übersicht über die standardmäßig verfügbaren Gerätebeschreibungen.

6.4 Die logische Beschreibung

6.4.1 Definition

Die logische Beschreibung gibt Auskunft über die Zusammensetzung des experimentellen Aufbaus aus den einzelnen Detektorsystemen wie z. B. den einzelnen Armen eines Koinzidenzexperiments, den sie bildenden Untereinheiten bis hin zu den einzelnen Detektorkanälen. Während die physikalische Beschreibung mehr technische Details abdeckt, die für das korrekte Ansprechen der Meßelektronik wichtig sind, dient sie zur Beschreibung der Struktur einer Experimentieranlage nach inhaltlichen Gesichtspunkten. Sie vermittelt die Bedeutung, die hinter den einzelnen Teilen steckt, welche physikalischen Parameter gemessen werden und welche Rolle der jeweilige Meßwert im Gesamtsystem spielt. Die logische Beschreibung bietet die Möglichkeit, inhaltlich zusammengehörige Teile zusammenzufassen, auch wenn die entsprechende Meßelektronik und ihre Bearbeitung im Rahmen der Datenaufnahme vollkommen verschieden ist. Die logische Beschreibung verdeckt damit im wesentlichen alle Hardware-Eigenschaften. Sie stellt eine weitgehend hardwareunabhängige Beschreibung des Experiments dar, die in vielen Fällen einer systematischen Bearbeitung des ganzen Experiments oder Teilen davon wesentlich besser gerecht wird. Das kann bereits bei der Auslese hilfreich sein, spielt aber danach beim Weiterverarbeiten der Daten wie z. B. Eventbuilding oder Analyse eine große Rolle. Nicht zuletzt unterstützt die logische Beschreibung unmittelbar den Benutzer bei der Erstellung von Datenaufnahme- und Analyse-Software, wo der Zugriff auf die Meßdaten mithilfe einer prägnanten symbolischen Adressierung vorgenommen werden kann.

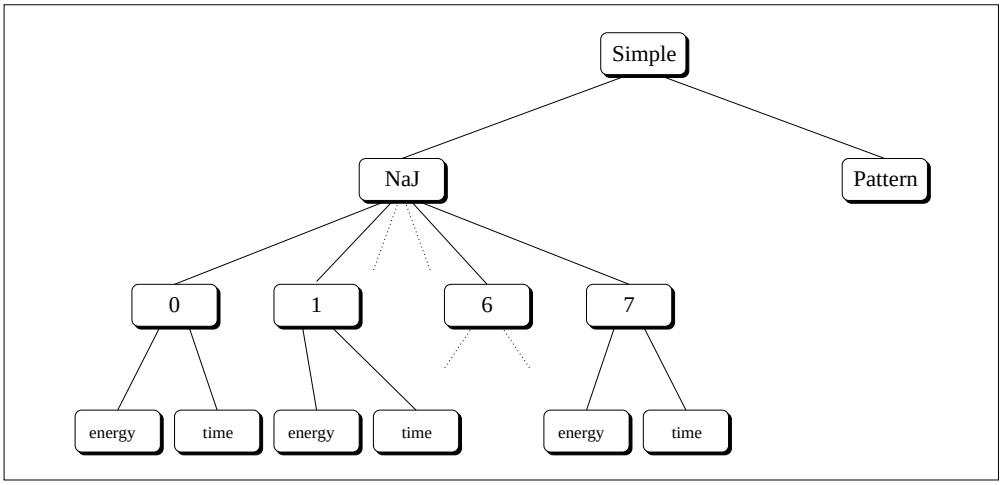

Abb. 6.7: Schematische Darstellung der logischen Struktur eines Experiments

Es zeigt sich, daß sich die logische Struktur eines Experiments in der Regel durch eine hierarchische Baumstruktur beschreiben läßt (Abb. 6.7). Im konkreten Fall läßt sich jedem Element einer Experimentkonfiguration, ebenso wie dem ganzen Experiment oder einzelnen Teilbäumen, ein symbolischer Name zuordnen, mit dessen Hilfe eine Identifizierung dieses Elements bei Datenerfassung oder Analyse möglich ist.

```
typedef struct descriptor       Descriptor;

struct  descriptor {
        int         type;                   /* Datentyp                     */
        char        *name;                  /* Name                         */
        int         number;                 /* relative Adresse             */
        int         len;                    /* Anzahl untergeord. Knoten    */
        int         level;                  /* Baum-Niveau                  */
        int         offset;                 /* Position in Ereignisstruktur */
        int         index;                  /* laufende Nummer eines Blatts */
        long        ok;                     /* Flaggen                      */
        Descriptor *parent;                 /* uebergeordneter Knoten       */
        Descriptor **first;                 /* erstes und letztes Element   */
        Descriptor **last;                  /* einer untergord. Knotenliste */
        int         (*fun)();               /* Verweis auf Unterprogramm    */
        void        *param;                 /* Parameterliste f. Unterprog. */
        void        *databuf;               /* Verweis auf Datenpuffer      */
        int         datalen;                /* Groesse des Datenpuffers     */
        Descriptor *desc[1];                /* Liste untergeord. Knoten     */
};
```

Abb. 6.8: C-Struktur zur Beschreibung der logischen Elemente eines Experiments.

6.4.2 Die Descriptor-Datenstruktur

In MECDAS übernimmt die logische Beschreibung eine grundlegende Funktion von der Auslese bis hin zur Datenanalyse. Softwaremäßig ist die logische Beschreibung als ein einziger, mehrfach verketteter Baum realisiert, dessen Knoten und Blätter die logischen Elemente des Experiments beschreiben. Dabei wird, ähnlich wie bei der physikalischen Beschreibung, eine einheitliche C-Datenstruktur (s. Abb. 6.8) verwendet, an deren Komplexität ihre zentrale Rolle für die Datenerfassungs-Software deutlich wird.

Die Descriptor-Datenstruktur, die die Grundlage der logischen Beschreibung bildet, definiert u. a. den logischen Typ des zu beschreibenden Baumelements. Das ist im wesentlichen eine Kodierung für den Datentyp, also entweder ein elementarer Datentyp der Programmiersprache C, wie z. B. `short`, `long`, oder `float`, oder eine C-Struktur, die sich selbst aus elementaren Datentypen oder anderen C-Strukturen zusammensetzen kann. Dieser Parameter enthält implizit Informationen über die Wortgröße, den Wertebereich und andere wichtige Eigenschaften, deren Kenntnis bei der Ausübung vieler Aufgaben wichtig ist.

Jedes Element besitzt sowohl einen symbolischen Namen als auch eine relative Adresse, die beide eine unabhängige Adressierung innerhalb eines Teilbaums erlauben. Ein weiterer wichtiger Parameter ist die Information, ob das Element ein Blatt des Baums ist oder ein Knoten. In Abhängigkeit davon wird die diesem Element zugeordnete Information unterschiedlich interpretiert. Im Falle eines Knotens finden sich darin weitere Descriptor-Informationen auf untergeordnete Baumelemente. Wenn es jedoch ein Blatt ist, entspricht dieses logische Element einem konkreten Gerät bzw. Teil eines Gerätes, in der Regel einem einzelnen

Kanal eines CAMAC- oder Fastbus-Moduls. In diesem Fall enthält der Descriptor Verweise auf die dieses Gerät spezifizierende Hardware-Datenstruktur aus der physikalischen Beschreibung. So ist es möglich, für jedes einzelne Element des logischen Baums die jeweils damit verknüpfte Hardware-Beschreibung aufzufinden. Man besitzt damit schließlich alle relevanten Informationen, die dieses Element für die Datenerfassung charakterisieren. Die anderen Komponenten der Descriptor-Datenstruktur dienen in unterschiedlichsten Anwendungsbereichen speziellen Zwecken, deren Beschreibung hier zu weit führen würde [Kryg95b].

6.4.3 Die Beschreibung durch den Experimentator

Der Experimentator muß sich nun bei der Erstellung der logischen Beschreibung nicht selbst um die Generierung eines solchen Baums kümmern. Dafür wurde Software entwickelt. Die Aufgabe des Experimentators beschränkt sich lediglich auf die Angabe einer C-Struktur, die ein Abbild der logischen Experimentstruktur sein soll. Auch hier gelten einfache Regeln zu ihrer Konstruktion, und es kann von den Möglichkeiten des C-Präprozessors Gebrauch gemacht werden.

Abbildung 6.9 zeigt ein Beispiel für eine logische Konfiguration, wie sie vom Experimentator zur Beschreibung des Experiments angegeben werden muß. Sie entspricht der schematischen Darstellung aus Abb. 6.7 und beschreibt einen Experimentaufbau aus acht identischen NaJ-Detektoren mit jeweils einem ADC und TDC zur Messung von Energie- und Zeitinformation ergänzt durch ein Hit-Pattern. Die dazugehörige Meßelektronik kann physikalisch so, wie bereits in Abb. 6.4 gezeigt, beschrieben werden. Die einzelnen Strukturkomponenten wurden mit Namen gekennzeichnet, die die Bedeutung der entsprechenden Teile der Experimentierapparatur veranschaulichen. Wie man aus dem Beispiel erkennt, ist jedes Element mit einem Datentyp versehen, der entsprechend der Sprachdefinition von C Definitions- und Wertebereich des entsprechenden Meßwerts definiert. In diesem Fall werden ausnahmslos `short`-Werte verwendet, d. h. vorzeichenbehaftete ganze Zahlen zwischen $-2^{15} = -32768$ und $2^{15} - 1 = 32767$.

```
struct simple {                          /* Definition der log. Konfig. */
        struct {                         /* Definition eines Detektors  */
                short energy;            /* Energieinformation  (ADC)   */
                short time;              /* Zeitinformation (TDC)       */
        } naj[8];                        /* acht NaJ-Detektoren         */
        short pattern;                   /* Hit-Pattern                 */
} simple;                                /* symbolischer Experimentname */
```

Abb. 6.9: Beispiel für eine logische Experimentbeschreibung

6.4.4 Einsatzgebiete der logischen Experimentbeschreibung

Mithilfe des bereits angesprochenen C-Parsers wird die logische Beschreibung in den dazu äquivalenten Baum überführt. Diese ist dann die Grundlage zur Bearbeitung vieler Aufgabenstellungen in MECDAS. Er wird verwendet

1. zur Generierung experimentspezifischer Software für die Datenaufnahme,
2. bei selektivem Auslesen und Manipulieren von Daten (Nullenunterdrükkung, Offset-Korrekturen, Datenreduktion) bei der Datenaufnahme,
3. zum hardwareunabhängigen Verpacken der Daten,
4. zur Beschreibung der verpackten Daten,
5. zum Auspacken der Daten bei der Ereignisrekonstruktion,
6. zum individuellen Zugriff auf die Meßdaten bei der Analyse.

In der Regel reicht bis auf Punkt 1 die logische Beschreibung zur Erfüllung der genannten Aufgaben voll und ganz aus. Die Software von MECDAS versucht diese Tatsache soweit wie möglich auszunutzen, um damit auch den experimentspezifischen Anteil der Aufgaben 2 bis 6 bestmöglich zu automatisieren und für den Benutzer zu vereinfachen. Lediglich für die Datenaufnahme wird neben der logischen auch die physikalische Beschreibung zwingend benötigt.

6.4.5 Zuordnung von logischer und physikalischer Beschreibung

Solange physikalische und logische Beschreibung zwar existieren aber nicht miteinander in Bezug gesetzt werden können, stellen sie keine gegenseitige Ergänzung dar und machen nur wenig Sinn. Es ist daher notwendig, neben ihrer Spezifizierung auch die Zuordnung der einzelnen logischen Elemente zu den physikalischen vorzunehmen. Auch dafür wurden Vorkehrungen getroffen. Der Experimentator muß die physikalische und logische Beschreibung durch eine Folge von symbolischen C-Zuweisungen ergänzen, wobei jeweils ein einzelnes Element oder eine Gruppe von physikalischen Elementen den korrespondierenden logischen zugeordnet werden. Das ist für das bereits in den Abbildungen 6.4 und 6.9 erläuterte Beispiel nun in Abbildung 6.10 dargestellt. Sie enthält die Zuordnung der Energie- und Zeit-Komponenten der logischen Beschreibung zu den entsprechenden ADC- bzw. TDC-Kanälen.

Hierbei wird konsequent von der kompakten Darstellung von C-Datenstrukturen Gebrauch gemacht. Auf der linken Seite stehen die entsprechend spezifizierten logischen Elemente, rechts die Mitglieder der als Datenvektor formulierten Hardwaregruppe, die jeweils einem physikalischen Kanal entsprechen.

```
simple.naj[0, 3].energy = adc[0, 3];          /* Energie-Info. stammt aus */
simple.naj[4].energy    = adc[8];             /* ADC-Modul(en), dessen 4. */
simple.naj[5, 7].energy = adc[5,7];           /* Kanal defekt ist.        */
simple.naj[0, 7].time   = tdc[0, 7];          /* Zeiten werden von TDC-   */
                                              /* Modul geliefert          */
simple.pattern          = pattern[0];
```

Abb. 6.10: Beispiel für die Zuordnung von logischen und physikalischen Konfigurationselementen. Es demonstriert globale aber auch individuelle Zuordnungen.

Die Software überprüft die Einhaltung von unteren und oberen Feldgrenzen sowohl bei den logsichen als auch den physikalischen Elementen und erzwingt eine vollständige Zuordnung der in der logischen Beschreibung definierten Elemente. Während auch hier wieder die C-Schreibweise verwendet wird, wurde jedoch von der normalen C-Semantik abgewichen. Im Unterschied zu Standard-C spezifiziert der Komma-Operator hier eine von Fortran her bekannte sog. implizite DO-Loop. Das bedeutet, Zuordnungs-Anweisungen, bei denen auf der linken und rechten Seite Feldelemente in definierte Folge der Indizes benutzt werden, können zu einer Anweisung zusammengefaßt werden, wobei die konstanten Idizes nun durch einen Kommaausdruck zur Spezifizierung des ersten und letzten zuzuordnenden Elements ersetzt werden müssen. Optional kann durch ein weiteres Komma getrennt, auch eine Schrittweite, die von der Voreinstellung 1 abweicht, angegeben werden. Im allgemeinen Fall ist also der Ausdruck

```
x[begin, end, step]
```

äquivalent zu einer Folge von Ausdrücken

```
x[begin]
x[begin + step]
x[begin + 2 * step]
...
x[begin + n * step]
```

wobei n die kleinste ganze Zahl ist, für die die Bedingung $(begin + (n+1) * step > end)$ gilt ist. Die Komma-Ausdrücke auf der linken und rechten Seite müssen nicht identisch sein, lediglich die Anzahl der zu durchlaufenden Schritte muß übereinstimmen. In

```
x[xbegin, xend, xstep] = y[ybegin, yend, ystep];
```

muß also gelten: $((xend - xbegin)/xstep = (yend - ybegin)/ystep)$

Diese Besonderheit wurde eingeführt, um den Experimentator die Zuordnung insbesondere für sehr umfangreiche Experimentkonfigurationen zu erleichtern. Sie stellt einen guten Kompromiß dar zwischen der Notwendigkeit, die Zuordnung für jeden einzelnen Kanal zwar mit hohem Aufwand aber doch individuell vornehmen zu können, und einer vollständigen Automatisierung, die jedoch an vorgegebene, feste Regeln gebunden ist und die Flexibilität stark einschränkt. Das in der Abbildung dargestellte Beispiel demonstriert den nicht seltenen Fall, wo eine beliebige Zuordnung ganz hilfreich ist, nämlich wenn einzelne Hardware-Kanäle defekt sind und durch andere, evtl. noch verfügbare ersetzt werden müssen. Durch einfache Änderungen in den Zuordnungsanweisungen werden damit notwendige Umverkabelungen für die Software vollkommen transparent und müssen dort nicht mehr weiter berücksichtigt werden.

6.5 Bearbeitungsvorschriften

6.5.1 Beschreibung des dynamischen Verhaltens

Das letzte Element der Experimentbeschreibung ist eine Zusammenstellung von Bearbeitungsvorschriften für die im Rahmen der Datenaufnahme anfallenden Aufgaben. Sie stellen im Gegensatz zu den anderen Teilen der Beschreibung, die eine statische Charakterisierung der Experimentkonfiguration bilden, Anweisungen für das dynamische Verhalten der Software in den verschiedenen Phasen der Datenaufnahme dar. Sie erlauben unter Einbeziehung von logischer und physikalischer Konfiguration die individuelle Bearbeitung der erforderlichen Aufgaben. Dazu gehört u. a.:

- das individuelle Ansprechen spezieller Hardware bei der Initialisierung, der Auslese und zum Zurücksetzen
- die beliebige Formulierung auch komplexer Algorithmen bei der Auslese oder Nachbearbeitung der Daten
- die freie Wahl der Bearbeitungsreihenfolge
- die Orientierung des Ablaufs an technischen oder logischen Gegebenheiten
- eine einfache Realisierung selektiver Bearbeitungsmechanismen wie Abprüfen von Treffermustern oder Auswertung komplexer Bedingungen
- die einfache Einbeziehung von Datenreduktion und Vorauswertung

6.5.2 Die Angaben des Experimentators

Wie auch in den anderen Teilen der Beschreibung muß der Benutzer die dazu notwendige Information in der Sprache C spezifizieren. Während zur Angabe der logischen und physikalische Beschreibung ausschließlich Definitionen, Deklarationen und Zuordnungen von Datenstrukturen und Variablen Verwendung finden, steht für die Bearbeitungsvorschriften im wesentlichen die volle Syntax der Programmiersprache C zur Verfügung. Sie erlaubt es, komplexe Anweisungen unter Einbeziehung der in C bekannten elementaren und daraus ableitbaren Datentypen, Adressierungsmechanismen und Kontrollstrukturen zu verwenden. Mit ihrer Hilfe sind letztendlich eine Reihe von Unterprogrammen korrespondierend zu den in Abschnitt 5.8.2 beschriebenen Schnittstellen zu definieren, die dazu dienen, experimentspezifische Aufgaben zu erledigen, für die die alleinige Kenntnis der in den anderen Teilen der Beschreibung spezifizierten statischen Experimentkonfiguration nicht ausreicht.

Tabelle 6.2 gibt eine Übersicht über die Unterprogramme, die, falls sie durch den Benutzer zur Verfügung gestellt wurden, von der Datenaufnahme-Software in Anspruch genommen werden. Bis auf eines sind alle optional und müssen nicht explizit durch den Benutzer angegeben werden. Für sie existieren in der

Funktion	Kurzbeschreibung
`rdtInit`	Initialisierung von Hard- und Software
`rdtTerm`	Terminierung von Hard -und Software
`rdtStart`	Start einer Einzelmessung
`rdtStop`	Stopp einer Einzelmessung
`rdtPause`	kurzzeitige Meßunterbrechung
`rdtCont`	Fortführung einer unterbrochenen Messung
`rdtStatus`	Ausgabe des Zustands der Auslese
`rdtMain`	Auslese-Routine
`rdtRecover`	Behandlung von Hardware-Fehlern
`rdtError`	Kennzeichnung der Meßdaten bei Fehlern
`rdtFormat`	Formatierung der Meßdaten
`ctlInit`	Initialisierung der Verriegelungselektronik
`ctlTerm`	Zurücksetzen der Verriegelungselektronik
`ctlOpen`	Freigabe der Messung
`ctlClose`	Verriegelung der Messung
`ctlPause`	globale Verriegelung der Elektronik
`ctlCont`	Freigabe nach globaler Verriegelung
`ctlPoll`	Abfrage der Triggerbedingung

Tab. 6.2: Übersicht der durch den Experimentator anzugebenden rdt/ctl-Funktionen mit den Bearbeitungsvorschriften für die verschiedenen Phasen der Datenaufnahme.

Datenaufnahmebibliothek von MECDAS Voreinstellungen. Sie sind nur bei ausdrücklichem Bedarf vorzusehen, falls die aufgrund der Beschreibung automatisch veranlaßten Vorgänge nicht ausreichen sollten. Eine Ausnahme bildet die Funktion **rdtMain()**. Diese Routine stellt die eigentliche Ausleseroutine dar, die von der allgemeinen Software im Laufe der Datenaufnahme bei Auftreten eines physikalischen Ereignisses aufgeruften wird. In ihr muß der Experimentator alle Auslesevorschriften spezifizieren, die zur individuellen Erfassung der Meßdaten benötigt werden. Eine Voreinstellung dafür besteht nicht.

6.5.3 Auslese-Primitiven

Für Routineaufgaben und zum einfachen Zugriff auf Information aus der statischen Experimentbeschreibung steht dem Experimentator eine Reihe von Bibliotheksfunktionen bzw. Präprozessor-Makros zur Verfügung, Von zentraler Bedeutung für die wesentlichen Aufgaben der Auslese sind folgende fünf Funktionen:

`lread()`

Diese Funktion erlaubt die Auslese des kompletten Experimentaufbaus oder einzelner seiner Teile ausschließlich auf der Basis der logischen Experimentkonfiguration. Sie stellt die Grundlage insbesondere zur selektiven Auslese und der bedingten Bearbeitung einzelner Teile des Detektorsystems, soweit es die anzusprechende Meßelektronik zuläßt, dar und verdeckt hardwarespezifische Eigenheiten. Die Auslesereihenfolge wird durch die Ordnung innerhalb der logischen Konfiguration vorgegeben.

```
static int      events;

Data *
rdtStart(Data *buf, size_t n)
{
        events = 0;

        return (buf);
}

Data *
rdtMain(Data *buf, size_t n)
{
        int     i;
        int     type = 1;

        ++events;
        lread(simple.pattern);
        for (i = 0; i < 8; ++i)
                if (simple.pattern & (1 << i))
                        lread(simple.naj[i]);
        buf = pack(simple, &simple, buf, type, events);

        return (buf);
}

Data *
rdtStatus(Data *buf, size_t n)
{
        sprintf(buf, "Events: %d\n", events);

        return (buf);
}
```

Abb. 6.11: Beispiel für Bearbeitungsvorschriften

`pread()`

Hier orientiert sich die Auslese ausschließlich an den technischen Gegebenheiten, die in der physikalischen Beschreibung in Kombination mit den Gerätebeschreibungen vorgegeben ist. Die Funktion dient der Auslese jeweils einer kompletten Gerätegruppe, die aus einer mehr oder weniger großen Anzahl von einzelnen Hardware-Kanälen bis hin zu komplexen Subsystemen bestehen kann. Dabei bildet zwar die logische Zuordung der Datenkanäle die Grundlage für die Kodierung der ausgelesenen Daten, sie bleibt für den Auslesevorgang selbst jedoch unberücksichtigt. Vielmehr hält sich dieser streng an die physikalische Struktur der Meßelektronik und macht von den Vorzügen der Hardware wie z. B. Blocktransfer unmittelbar Gebrauch. Insbesondere die Nutzung einer sequentiellen tabellengesteuerten Bearbeitung erlaubt sowohl bei der Verwendung intelligenter Subsysteme als auch bei einfacher Hardware eine effiziente Auslese.

`dread()`
: Während bei der eben beschriebenen logischen wie auch bei der physikalischen Auslese jeder eingelesene Meßwert auf der Basis der logischen Experimentstruktur zwischengespeichert wird, und so bei der nachfolgenden Auslese anderer Hardware oder für Vorauswerteaufgaben bei Bedarf genutzt werden kann, reduziert sich die Bearbeitung in diesem Fall ausschließlich auf die Auslese. Diese direkte Auslese realisiert das absolute Minimum an Bearbeitungsaufwand, der unter Berücksichtigung einiger weniger, aber notwendiger Verwaltungsaufgaben zur eindeutigen Kennzeichnung der Daten, erforderlich ist. Ansonsten verhält sie sich wie die physikalische Auslese.

`clear()`
: Diese Funktion sorgt für das Löschen bzw. Zurücksetzen der Bestandteile einer Gerätegruppe, falls diese Aufgabe nicht schon im Zusammenhang mit der Auslese erledigt wurde.

`pack()`
: Diese Funktion packt die im Laufe des Auslesevorgangs angesammelten Daten formal zu einem Datensegment zusammen, indem sie die in einem Puffer abgelegten Daten mit globaler Information in Form des Datensegment-Kopfes versieht und der nachfolgenden, asynchron zur Auslese arbeitenden Formatierung und Archivierung zuführt.

Diese Funktionen sind in der Regel als Präprozessor-Makros realisiert, die die Handhabung komplexer Funktionsaufrufe innerhalb der aus der statischen Experimentbeschreibung generierten Software erleichtert. Abbildung 6.11 zeigt ein einfaches Beispiel für die Angabe der Bearbeitungsvorschriften.

6.6 C als Beschreibungssprache

6.6.1 Wahl der Sprache

Die Syntax von C als Grundlage einer Beschreibungssprache für die physikalische und logische Konfiguration in MECDAS wurde aus mehreren Gründen gewählt:

1. Es wurde eine Beschreibungsmöglichkeit benötigt, die es erlaubt, einfach und kompakt
   (a) statische Strukturen unterschiedlichster Komplexität für den experimentellen Aufbau eindeutig zu spezifizieren,
   (b) simple Anweisungen, aber auch
   (c) komplizierte Algorithmen als Bearbeitungsvorschriften für die Datenaufnahme zu formulieren, wobei besonderen Wert darauf gelegt wird,
   (d) direkte Hardwarezugriffe effizient vorzunehmen.

2. Die Beschreibungssprache sollte die unter 1. genannten Anforderungen möglichst vollständig erfüllen. Der Rückgriff auf eine andere Sprache sollte auch in Sonderfällen nicht notwendig sein, um die Handhabung für den Benutzer nicht unnötig kompliziert zu gestalten.

3. Punkte 1 und 2 zusammengenommen bedeuten, daß die Beschreibungssprache eine ähnliche Flexibilität und Leistungsumfang wie eine herkömmliche, vollständige Programmiersprache haben muß.

4. Dennoch sollte sie möglichst einfach zu erlernen sein.

5. Entsprechende Hilfsmittel zur Benutzung dieser Sprache wie Editor, Compiler, Interpreter oder Parser sollten verfügbar oder mit vertretbarem Aufwand implementierbar sein.

6. Insbesondere sollten diese Werkzeuge auf den Frontend-Rechnern lauffähig sein bzw. dafür Code generieren können.

7. Der generierte Code sollte möglichst effizient sein, da er gerade die besonders zeitkritischen Programmteile der Datenerfassung repräsentiert.

8. Die Beschreibungsmöglichkeit sollte sich gut in das restliche Datenerfassungssystem einbetten lassen in Bezug auf

   (a) den generierten Code,

   (b) die verwendete Sprache,

   (c) die notwendigen Hilfsmittel,

   um den Aufwand bei Implementation und Benutzung gering zu halten.

9. Software-Entwicklung, Wartung und Portierbarkeit sollten einfach sein.

10. Nicht zuletzt sollten sowohl Sprache als auch Werkzeuge ausreichend dokumentiert sein.

Alle Punkte bis auf 1a) werden durch die Programmiersprache C sehr gut abgedeckt. Wegen ihrer Verwendung als Implementationssprache von MECDAS wären die benötigten Hifsmittel vorhanden, Leistungsfähigkeit, Flexibilität und Effizienz würden nicht eingeschränkt, die Integration optimal, Software-Entwicklung, Wartung und Portierbarkeit nicht verschieden, Dokumentation vorhanden. Doch wäre die Formulierung der statischen Experimentbeschreibung in Standard-C für eine direkte Bearbeitung durch den Compiler nur mit großem Aufwand und tiefergehenden Kenntnissen der Programmiersprache bei gleichzeitig hoher Fehleranfälligkeit zu erreichen. Die herkömmliche Verwendung von C kam also nicht in Frage.

Leider gab es zum Zeitpunkt der Implementation auch keine ausreichende Alternative. Die wenigen existierenden Beschreibungssprachen für kernphysikalische Experimente [Kühn88, Schm89] hatten in aller Regel einen wesentlich geringeren Leistungsumfang als gefordert und besaßen in Bezug auf Verfügbarkeit,

Integrier-, Wart- und Portierbarkeit und insbesondere auch bei der Dokumentation große Schwächen. In allen Fällen wäre noch Portierungs- und Implementationsaufwand in größerem Ausmaß notwendig geworden, auch wenn die genannten Anforderungen reduziert worden wären.

Noch viel weniger kamen herkömmliche Programmiersprachen in Betracht, deren Sprachumfang für eine komplexe Programmierung zwar ausreichend ist, die jedoch weder als Beschreibungssprache gut geeignet sind, noch die anderen Anforderungen ausreichend erfüllen. Lediglich die Sprache Lisp wäre eher sowohl als Beschreibungssprache als auch als leistungsfähige Programmiersprache in Frage gekommen. Doch die fehlende Möglichkeit direkte Hardwarezugriffe, eine im Falle der Verwendung eines Interpreters mit Sicherheit niedrige Effizienz und mangelnde Echtzeitfähigkeit und zu erwartende Integrationsprobleme in größerem Umfang ließen neben anderen Gründen auch diesen Ansatz scheitern.

Eine Eigenimplementation einer neuen Beschreibungssprache hätte zwar eine sehr gute Anpassung an das Problem erlaubt, schied jedoch ebenso aus, da hierfür der Aufwand zu groß geworden wäre. Allein die Definition einer neuen Sprache mit einwandfreier Syntax und Semantik, zu C vergleichbarem Leistungsumfang, aber besseren Möglichkeiten zur Experimentbeschreibung und die Überprüfung auf Korrektheit wäre mit hohem Arbeitsaufwand verbunden. Auch die Implementation eines Compilers bzw. Interpreters mit korrekt arbeitender Codegenerierung wäre sehr aufwendig.

Eingehende Untersuchungen zeigten jedoch, daß die Neuentwicklung einer Beschreibungssprache nicht notwendig war, sondern die Syntax der Sprache C, losgelöst von ihrer herkömmlichen Semantik, zur Beschreibung von Experimentkonfigurationen genügend Möglichkeiten bietet. Somit war lediglich die Entwicklung eines neuen Übersetzers notwendig, der speziellen C-Code nicht als herkömmliches Programm sondern als Experimentbeschreibung interpretiert und entsprechenden Code generiert. Vorerst reichte es sogar aus, diese Funktionalität auf den statischen Teil der Experimentbeschreibung zu beschränken, da der Rest durch den herkömmlichen Compiler abgedeckt wurde. Mit der Implementation eines speziellen C-Parsers und ergänzender Software konnten so alle Punkte der o. g. Anforderungsliste zufriedenstellend erfüllt werden.

6.6.2 Der C-Parser `mcc`

Der C-Parser `mcc` [Kryg95b] bildet das Herzstück eines Programmpakets, das dazu entwickelt wurde, in C-Syntax formulierte experimentspezifische Konfigurationsinformation zu analysieren und in eine Form zu bringen, die es den allgemeinen, experimentunabhängigen Teilen der MECDAS-Software erlaubt, damit zu arbeiten. Der Parser wurde als allgemeines Werkzeug realisiert und ist in der Lage, C-Code fast beliebiger Art zu übersetzen. Er unterstützt nur mit wenigen Einschränkungen beinahe den vollständigen Sprachumfang von ANSI-C [Kern90]. Bei seiner Bearbeitung von C-Programmcode nimmt er eine Syntaxüberprüfung vor, generiert dabei einen vollständigen Syntaxbaum und gibt ihn schließlich aus. Er erzeugt damit eine sprachunabhängige Repräsentation

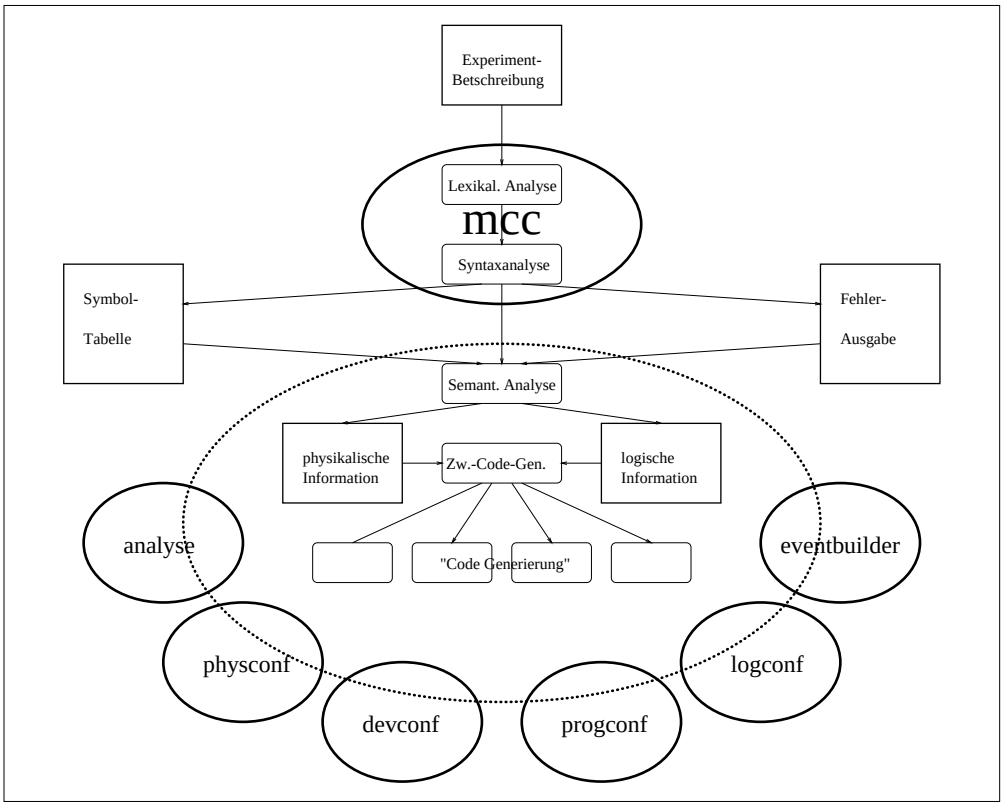

Abb. 6.12: Schematische Darstellung der Software zur Analyse und Verarbeitung der MECDAS-Experimentbeschreibung auf der Basis eines Compiler-Modells.

des analysierten Programmcodes, die mit wenig Aufwand von nachfolgenden Programmen weiterverarbeitet werden kann. Diese Eigenschaft wird an vielen Stellen von MECDAS für die unterschiedlichsten Zwecke ausgenutzt.

Im Rahmen der Experimentkonfiguration wird der nach der Analyse der Experimentbeschreibung von `mcc` produzierte Syntaxbaum durch verschiedene, separate Programme nach unterschiedlichsten Gesichtspunkten untersucht. Mit der so erhaltenen Information wird nach und nach experimentspezifischer Code generiert, der nach seiner Übersetzung mit herkömmlichen Compilern die Kernsoftware zu einem lauffähigen Programm ergänzt, das alle Informationen über das konkrete Experiment besitzt.

`mcc` wurde mithilfe der UNIX-Werkzeuge `lex` [Lesk78] und `yacc` [John78] erstellt und besitzt fast vollständig die Eigenschaften eines herkömmlichen C-Compilers. Das Programm wird ergänzt durch Bibliotheksfunktionen und Programme, die die von `mcc` generierte Konfigurationsinformation für die Datenaufnahme, zur Formatierung und Dekodierung von Meßdaten oder auch für Analysezwecke verfügbar macht. Abbildung 6.12 demonstriert die Struktur dieses Softwarepakets. Eine detailierte Information über Realisierung und Funktionsweise von `mcc` und der dazugehörigen Hilfs-Software kann der MECDAS-Dokumentation [Kryg95b] entnommen werden.

6.7 Die Konfigurations-Software

In diesem Kapitel wurden bisher einzelne, wesentliche Elemente von MECDAS beschrieben, die es erlauben, experiment-, detektor- bzw. hardwarespezifische Besonderheiten auf weitgehend automatisiertem Weg abzuwickeln. Abschließend soll noch einmal zusammenfassend ihr Zusammenwirken im Rahmen der Datenerfassung dargestellt werden. Die bereits beschriebenen Programme bzw. Bibliotheken werden ergänzt durch eine Reihe von Hilfsprogrammen. Sie bilden gemeinsam ein modulares Softwarepaket, das dazu dient, aus der vom Experimentator vorgegebenen Experimentbeschreibung die experimentspezifischen Teile der Datenerfassungs-Software zu generieren.

6.7.1 Das Programm `configure`

Der Experimentator besitzt zwar die Möglichkeit, die einzelnen Programme zur Konfiguration der Datenaufnahme in eigener Regie einzusetzen, braucht sich jedoch in der Regel nicht um Details zu kümmern und kann auf das Programm `configure` zurückgreifen, das vollautomatisch für die korrekte Generierung des Datenaufnahmeprogramms aus der vom Experimentator bereitgestellten Experimentbeschreibung sorgt.

Das Programm `configure` ist eine Shell-Kommandoprozedur, die unter Zuhilfenahme einiger MECDAS-Programme und einer Vielzahl von Standard-UNIX-Werkzeugen die Generierung des Datenaufnahmeprogramms vornimmt. Bei seinem Aufruf muß in der Regel der Name der Experimentbeschreibung angegeben werden, die bearbeitet werden soll. Dieser Name spielt sowohl in der Konfigurationsphase als auch bei der Datenerfassung eine zentrale Rolle. Er wird daher im folgenden auch mit „Experimentname“ oder auch „Setup-Name“[2] bezeichnet. Nach Konvention sollte die das Experiment beschreibende C-Datenstruktur diesen Namen besitzen. Nur dann ist gewährleistet, daß alle Automatismen bei der Konfiguration korrekt bearbeitet werden.

6.7.2 Beschreibungs-Dateien

`configure` erwartet, daß fünf Dateien, deren Namen im wesentlichen durch den Setup-Namen bestimmt werden und sich nur in den Endungen unterscheiden. Die Dateien enthalten die verschiedenen Bestandteile der Experimentbeschreibung. Das sind im einzelnen:

Header-Datei,
die unter dem Namen *setup*`.h` alle für die logische Beschreibung notwendigen Strukturdefinitionen enthalten sollte. Zur Erhöhung der Übersicht und

[2] Der Begriff Setup soll im Unterschied zu dem Begriff Experimentname deutlich machen, daß es sich bei der Konfiguration nicht zwangsläufig um ein komplettes Experiment handeln muß, sondern vielmehr auch in sich abgeschlossene Teile davon, die die Grundlage der Datenaufnahme innerhalb eines verteilten Systems bilden.

für ihre spätere Benutzung bei der Auswertung der Daten ist es evtl. sinnvoll, diese Definitionen auf mehrere Dateien zu verteilen. Zu diesem Zweck kann ohne Einschränkungen von den Möglichkeiten des C-Präprozessors Gebrauch gemacht werden. Aus softwaretechnischen Gründen sollte hier weder Variablendeklarationen noch C-Anweisung enthalten sein.

Log-Datei
dient der Spezifizierung der logischen Beschreibung und muß unter dem Namen *setup*`.log` zur Verfügung gestellt werden. Sie kann entweder die dazu notwendigen Definitionen selbst enthalten oder dazu mithilfe entsprechender C-Präprozessor-Anweisungen die zugehörige Header-Datei benutzen. Diese Information wird komplettiert durch die Deklaration einer Variablen mit dem Setup-Namen und vom Typ der Datenstruktur, die den experimentellen Aufbau bzw. Teilaufbau komplett beschreibt.

Phys-Datei
hat die Aufgabe, die physikalische Beschreibung wie in Abschnitt 6.2.3 dargestellt, aufzunehmen. Der Name der Datei ist *setup*`.phys`.

Att-Datei
dient der Zuordnung zwischen den einzelnen Elementen der logischen und physikalischen Beschreibung und sollte den Namen *setup*`.att` besitzen.

Rdt-Datei
enhält C-Unterprogramme mit Bearbeitungsvorschriften für die verschiedenen Phasen der Datenaufnahme. Dazu gehören u. a. Mechanismen zur selektiven Auslese, die bedingte Bearbeitung von einzelnen Detektorteilen oder auch einfach nur die sequentielle Abfolge von Anweisungen, mit denen die Auslese vorgenommen werden soll. Solche Anweisung können ergänzt werden durch beliebigen Programmcode für weitergehende Aufgaben zur Datenreduktion oder Vorauswertung. Dieser Programmcode muß unter dem Dateinamen *setup*`.rdt` gespeichert sein.

Diese Dateien bilden die vollständige Grundlage zur Generierung von experimentspezifischen Tabellen und Programmcode. Das Programm `configure` benutzt dazu den beschriebenen C-Parser `mcc`.

6.7.3 mcc-Backends

Um aus der Experimentkonfiguration konkrete Information für die Datenaufnahme zu gewinnen, stehen mehrere Programme zur Verfügung, die als Back-Ends des C-Parsers mcc arbeiten. Sie werten die aus der Syntax-Analyse gewonnene Information aus und generieren im wesentlichen Code für verschiedene Aufgabenstellungen der Datenaufnahme (s. Abb. 6.12):

1. Mithilfe des Programmes `devconf` wird für die in der physikalischen Experimentbeschreibung aufgeführten Geräte relevante Information aus der Berätebibliothek extrahiert.

2. Das Programm `physconf` erzeugt unter Zuhilfenahme der zuvor extrahierten Geräteinformation C-Quellcode für Tabellen, die den experimentellen Aufbau physikalisch in allen Details beschreiben.

3. Das Programm `logconf` generiert den vollständigen logischen Konfigurationsbaum in Form von C-Quellcode. Dieser ist die Basis für viele Aufgaben bei der Datenaufnahme und insbesondere für die Formatierung der Meßdaten. Er enthält für jeden einzelnen Datenkanal Verweise auf entsprechende Tabelleneinträge in der physikalischen Beschreibung.

4. Die statischen Tabellen mit logischer und physikalischer Beschreibung werden schließlich mithilfe des Programmes `progconf` durch umfangreiche Programmteile ergänzt, die für die Bearbeitung experiment- bzw. hardwarespezifischer Aufgaben in den unterschiedlichen Phasen der Datenaufnahme zuständig sind. Sie decken die in Abschnitt 5.8.2 definierten Schnittstellen zwischen allgemeiner Kernsoftware und experimentspezifischer Software ab.

Viele Teile der generierten Software ersetzen i. w. allgemeine Bibliotheksfunktionen, die die experimentspezifische Tabelleninformation auch interpretativ abarbeiten könnten. Um jedoch unnötigen Rechenaufwand unter Echtzeitbedingungen zu vermeiden, werden diese Aufgaben durch jeweils individuell auf die speziellen Gegebenheiten angepaßte Software erledigt. Hierfür findet die Interpretation der Konfigurationstabellen einmalig und außerhalb der Datenaufnahme statt, also unter Bedingungen, unter denen ein Mehraufwand unkritisch ist. Der dabei generierte Code ist zwar nicht immer so effizient wie unmittelbar vom Experten für das spezielle Problem implementiert, doch stellt er gegenüber einer interpretativen Tabellenabarbeitung in den meisten Fällen einen deutlichen Gewinn dar.

Auch aus softwaretechnischer Hinsicht bietet dieses Verfahren Vorzüge. Dazu gehört z. B. die Möglichkeit der frühzeitigen Erkennung von Inkonsistenzen in der Konfigurationsphase, deren Behandlung erst zum Zeitpunkt der Ausführung der Anwendung wesentlich kritischer ist. Durch die Verwendung fester Algorithmen bei der Code-Generierung reduziert sich auch die Fehleranfälligkeit des generierten und letztendlich bei der Datenaufnahme ausgeführten Codes. Gleichzeitig zeigen sich auch Vorteile bei Debugging. Kleine übersichtliche, auf den Speziallfall abgestimmte Programmteile lassen sich in ihrem Laufzeitverhalten wesentliche besser analysieren, als komplexe, für den Allgemeinfall konzipierte Software, die noch dazu effizient sein soll.

Bei der Generierung von experimentspezifischem Code wird im wesentlichen alle Information der statischen Experimentbeschreibung ausgenutzt. Bisher war es nicht notwendig, auch die Bearbeitungsvorschriften mithilfe des C-Parsers zu verarbeiten. Die Funktionalität des C-Präprozessors reichte bei den bis jetzt vorkommenden Anwendungen aus, um die Formulierung der Bearbeitungsvorschriften für den Benutzer bei minimalen Kenntnissen der Programmiersprache C noch gut handhabbar zu halten. Die rdt-Information ergänzt daher unmittelbar den generierten Code.

6.7.4 Ergänzende Software

Alle bei der Experimentbeschreibung verwendeten Dateien werden mithilfe eines Standard-Archivierungsprogrammes zusammengepackt und in Form eines initialisierten Datenbereichs als C-Quellcode zur Verfügung gestellt. Dieses Information wird dann im Rahmen der Datenerfassung für Zwecke der Selbstbeschreibung zu den Meßdaten hinzugefügt.

Mithilfe der im beschriebenen Werkzeuge wird nach und nach Programmcode generiert, der alle für die Datenerfassung notwendige experimentspezifische Information enthält. Er wird mithilfe eines herkömmlichen C-Compilers übersetzt und mit Software aus Bibliotheken

- zur Formatierung der Daten
- zur Verwaltung und Nutzung der Beschreibungsbäume
- zur Gerätebeschreibung
- für die experimentunabhängigen Kernaufgaben
- für allgemeine Routineaufgaben
- bei Bedarf Slave-Systemsoftware

zusammengebunden.

6.7.5 Make als grundlegendes Werkzeug

Sowohl bei der Code-Generierung als auch beim späteren Übersetzen und Binden der generierten bzw. in Quellform vorliegenden Programmteile macht `configure` intensiv von dem UNIX-Werkzeug `make` Gebrauch [Feld79]. Es erlaubt die einfache Spezifizierung von Abhängigkeiten und Vorschriften zum Generieren und Weiterverarbeiten von Code. Darüberhinaus bietet es dem Experimentator zusätzliche Konfigurations- und Eingriffsmöglichkeiten bei der Generierung eines lauffähigen Datenerfassungsprogrammes. Dazu gehört in erster Linie die Möglichkeit, noch zusätzliche Software aus anderen Quellen dazuzubinden. Unter anderem kommen hierfür Gerätebeschreibungen in Frage, die noch nicht in der Standard-Bibliothek berücksichtigt wurden oder gegenüber dieser spezielle Erweiterungen oder Änderungen besitzten. Es ist aber auch sehr einfach, bei Bedarf für den Einzelfall nicht ausreichende Standard-Komponenten der anderen Bibliotheken durch spezielle, stärker auf experiment- oder hardwarebedingte Eigenheiten eingehende Realisierungen zu ersetzen. Auch softwaretechnische Gesichtspunkte wie die Wahl des Compilers und dessen Betriebsmodi sind leicht zu berücksichtigen. Abbildung 6.13 zeigt einen Ausschnitt aus einem für die Verwendung von `make` notwendigen Makefiles. Auf dieser Grundlage wird schließlich ein lauffähiges Programm generiert, das alle Aufgaben der Datenaufnahme erfüllen kann.

```
SUBSYSTEM = simple
SRC = .
DEVSC =
DEVSO =
LCLSRC =
EXTDIR =
EXTSRC =
OBJECTS =

...

all: $(LBIN)/$(SUBSYSTEM)

devices: $(LBIN)/devices
        @:

$(LBIN)/devices: $(DEVICES) $(LIBDEV) $(LIBMASTER) $(LIBGEN)
        $(CC) $(LDFLAGS) $(DEVICES) -ldev -lmaster -lgen -o $@

$(SUBSYSTEM): $(LBIN)/$(SUBSYSTEM)
        @:

$(LBIN)/$(SUBSYSTEM): $(ACQ) $(LIBDEV) $(LIBDESC) $(LIBACQ) \
        $(LIBGEN) $(LIBFMT) $(LIBRDT) $(LIBMASTER)
        $(CC) $(LDFLAGS) $(ACQ) -lacq -lrdt -lfmt -ldev -lmaster -lgen -o $@

$(SUBSYSTEM).unix: $(LBIN)/$(SUBSYSTEM).unix
        @:

$(LBIN)/$(SUBSYSTEM).unix: $(ACQ) $(LIBDEV) $(LIBDESC) $(LIBACQ) \
        $(LIBGEN) $(LIBFMT) $(LIBRDT) $(LIBMASTER)
        $(CC) $(LDFLAGS) $(ACQ) -lacq -lrdt -lfmt -ldev -lslave -lgen -o $@

$(SUBSYSTEM).stand: $(LETC)/$(SUBSYSTEM).stand
        @:

$(LETC)/$(SUBSYSTEM).stand: $(LIBSTAND) $(LIBSLAVE) \
        $(ACQ) $(LIBDEV) $(LIBDESC) $(LIBACQ) $(LIBGEN) $(LIBFMT) $(LIBRDT)
        $(LD) -T $(BRELOC) $(START) $(MACHDEP) -e _strt $(LDFLAGS) -o $@ \
        $(ACQ) -lacq -lrdt -lfmt -ldev -lslave -lstand -lgen -lclib

reset: $(LETC)/reset

$(LETC)/reset: $(LIBSTAND) reset.o
        $(LD) -T 300000 $(START) reset.o -e _strt $(LDFLAGS) -o $@ -lstand

$(LBIN)/$(SUBSYSTEM).pixie: $(LBIN)/$(SUBSYSTEM)
        pixie $(LBIN)/$(SUBSYSTEM)

...
```

Abb. 6.13: Typisches Konfigurations-Makefile

6.8 Formatierung und Dekodierung der Meßdaten

6.8.1 Problemstellung

Eine sehr wichtige Aufgabe der Datenerfassungs-Software ist neben der Auslese der Daten auch ihre korrekte Umsetzung in eine Form, die ihr effizientes Archivieren und Weiterverarbeiten erlaubt, ohne daß dabei ein Informationsverlust auftritt. Wie bereits an früherer Stelle geschildert, ist die Datenaufnahme für eine effiziente Abwicklung so organisiert, daß nur relevante Information aufgenommen wird. So werden z. B. die digitalisierten Meßwerte nur der Detektoren bzw. Detektorteile ausgelesen, die Teilchen in geeigneter Form nachgewiesen haben; alle anderen — und das ist in vielen Fällen der bei weitem überwiegende Teil — bleiben unbeachtet und müssen daher auch nicht archiviert werden.

Um die vollständige Information über ein Ereignis festzuhalten, ist es also nicht sinnvoll, pro Ereignis einen festen Datensatz abzuspeichern und evtl. weiterzuverarbeiten, der die Werte aller möglichen Datenkanäle inklusive der, die keine gültige Information besitzten, in bestimmter Reihenfolge enthält. Speichert man jedoch nur die gültigen Meßwerte ohne Zusatzinformation ab, geht die implizite Zuordnung zwischen Position innerhalb des Datensatzes und der Bedeutung des Meßwertes verloren. Um das zu vermeiden und stets eine exakte Beschreibung eines physikalischen Ereignisses zu gewährleisten, muß neben der Menge der erfaßten Meßwerte auch die Information, welchen physikalischen Kanälen sie jeweils entsprechen, unbedingt mit festgehalten werden. Diese Adreßinformation muß daher bei der Datenaufnahme, wo sie oftmals nur implizit verfügbar ist, festgehalten und in geeigneter Form zusammen mit den Meßwerten archiviert werden.

6.8.2 Lösungsmöglichkeiten

Für diese Problemstellung gibt es nun sehr unterschiedliche Lösungsmöglichkeiten, die von vielen Faktoren abhängig sind. Bei Verwendung von Hardware mit automatischer Offsetkorrektur und Nullenunterdrückung liefert diese oftmals schon die mit Adreßinformation versehenen Meßwerte. Das ist auch bei einem Teil der an MAMI eingesetzten Datenerfassungs-Hardware der Fall. Leider wird dabei kein einheitliches Format verwendet, so daß die spätere Dekodierung der Daten aufwendig und extrem abhängig von der bei der Datenerfassung verwendeten Hardware ist. Das Problem wird noch verschärft, denn sogar bei identischer Hardware können Unterschiede im Datenformat in Abhängigkeit von benutzten Betriebsmodus auftreten.

Ein zweites Problem taucht auf, wenn mehrere gleichartige solcher Geräte unabhängig voneinander benutzt werden. Dann ist die Information, die die Geräte liefern nicht mehr eindeutig, und sie muß im Rahmen der Datenaufnahme softwaremäßig ergänzt werden. Bei einfacher Hardware wie herkömmlichen CAMAC-Modulen muß schließlich die komplette Adreßinformation durch die Software generiert werden. Das bedeutet also, bis auf den Fall, daß ausschließ-

lich Hardware verwendet wird, die die Meßdaten selbsttätig einheitlich formatiert, ist immer Software notwendig, um die erfaßten Meßdaten entsprechend aufzubereiten. Für die Auswertung von archivierten Meßdaten ist schließlich Dekodierungs-Software stets unumgänglich, um an die eigentlichen Meßwerte zu gelangen. Mechanismen und Aufwand sind dabei jedoch naturgemäß stark abhängig von den Formatierungsmethoden bei der Datenaufnahme.

In Analogie zu den zu Beginn dieses Kapitels gemachten Überlegungen über die logische und physikalische Sichtweise bei der Beschreibung eines Experiments bieten sich auch bei der Formatierung der Meßdaten grundsätzlich zwei Verfahren an:

1. Formatierung nach Hardware-Gesichtspunkten
2. Formatierung auf der Basis der logischen Experimentstruktur

6.8.2.1 Formatierung nach Hardware-Gesichtspunkten

Das erste Verfahren würde einerseits die oben beschriebenen Eigenschaften intelligenter Hardware voll ausnutzen und würde sich andererseits für die noch verbleibende Arbeit stark an dem physikalischen Aufbau des Experiments orientieren und die Formatierungsinformation bei Bedarf durch Hardware-Adressen ergänzen. Damit könnte für die Datenaufnahme auf den ersten Blick wohl die höchste Effizienz erreicht werden, da sich hier die neben der eigentlichen Datenauslese anfallende Arbeit auf ein Minimum zu beschränken scheint. Das ist jedoch nicht immer der Fall. Dieses Verfahren funktioniert dann sehr gut, wenn die intelligente Hardware bei weitem überwiegt oder die Datenaufnahme sich stark an dem Aufbau der Meßelektronik orientiert.

Das ist jedoch in vielen Fällen nicht möglich. Hier entscheiden logische Zusammenhänge, ob und wann welche Hardware anzusprechen ist. Ein Beispiel dafür ist die selektive Bearbeitung von Detektoren aufgrund eines Hitpattern oder ganz allgemein die bedingte Bearbeitung von Detektorteilen auf der Basis vorher eingelesener Daten. Auch kann durch Detektorstruktur und -eigenschaften eine bestimmte, evtl. von Ereignis zu Ereignis variierende Bearbeitungsreihenfolge vorgegeben werden. In diesen Fällen muß sich die Datenaufnahme-Software die Adreßinformation stets mit Zusatzaufwand verschaffen.

Richtet sich die Formatierung ausschließlich nach der Struktur der Meßelektronik, kann die zu den Meßwerten hinzuzufügende Information bei Verwendung der für die Auslese benötigten absoluten Adreßinformation zum Teil recht umfangreich werden und deutlich mehr Speicherplatz als der eigentliche Meßwert in Anspruch nehmen. Eine relative Adressierung kann nur dann unmittelbar genutzt werden, wenn die Bearbeitungsreihenfolge durch die Struktur der Meßelektronik vorgegeben ist.

Mit zunehmender Rechenleistung ist es schließlich möglich, während der Datenerfassung bereits Vorauswertungen durchzuführen. Die dabei softwaremäßig

erzeugten und ebenfalls zu archivierenden Ergebnisse besitzen keine direkte Entsprechung in der Meßelektronik, deren Adreßinformation verwendet werden könnte. Sie lassen sich nur künstlich in eine Hardware-Ordnung einfügen.

Alles in allem sind die bisher genannten Probleme in der Regel noch mit geringem Aufwand zu lösen. Gravierender sind jedoch die Schwierigkeiten beim Dekodieren der Daten in Hinblick auf eine möglichst gute Selbstbeschreibung der Meßdaten. Zur Dekodierung der Daten ist die Kenntnis der Formatierungsvorschriften notwendig, die abhängig von der Hardware, die bei der Datenaufnahme verwendetet wurde, sehr unterschiedlich sein können. Daher müssen Programme zur Auswertung der Daten entsprechende hardwarespezifische Dekodier-Software enthalten. Diese kann zur Reduzierung des Aufwands durch den Benutzer ein fester Bestandteil der Analyse-Software sein. Die damit verbundene Beschränkung auf nur bestimmte Hardware läßt sich aber auch durch Neuübersetzen bzw. Neubinden des Auswerteprogramms reduzieren, wobei die benötigte Dekodier-Software individuell hinzugefügt wird. Damit lassen sich sehr einfach neue Formatierverfahren berücksichtigen, was jedoch stets mit einem gewissen, nicht zu automatisierenden Arbeitsaufwand verbunden ist und die Fehleranfälligkeit erhöht.

Bei diesem Verfahren besteht ein grundsätzliches Konsistenzproblem zwischen Formatierung bei der Datenaufnahme und Dekodierung der Meßdaten bei der Auswertung. Es ist nämlich, insbesondere bei einer Vielfalt an möglichen Formaten, nicht einfach sicherzustellen, daß die Daten nach den gleichen Vorschriften dekodiert werden, mit denen sie (irgend wann einmal) formatiert wurden. Besonders problematisch wird es, wenn in der Zwischenzeit Modifikationen der Dekodier-Software wegen Änderungen in der Meßelektronik notwendig wurden. Diese Probleme ließen sich nur dann sauber umgehen, wenn die Meßdaten eine Experimentbeschreibung mit genügend hardwarespezifischen Informationen enthielten, die die Dekodierung der Daten ohne weitere Hilfsmittel erlaubten.

6.8.2.2 Formatierung auf der Basis der logischen Experimentstruktur

Die Hardware-Abhängigkeit von Formatierung, Dekodierung und Repräsentation der Daten läßt sich vermeiden, wenn für diese Aufgaben die logische Experimentstruktur als Grundlage verwendet wird. Im Rahmen der Datenaufnahme ist — insbesondere dort, wo bereits die Hardware eine gewisse Vorformatierung der Daten vornimmt — eine einheitliche Formatierung, die mithilfe der logischen Experimentbeschreibung erreicht werden kann, zwar mit Mehraufwand verbunden, doch läßt sich dieser bei geeigneter Programmierung ohne stärkere Beeinflussung des Totzeitverhaltens abwickeln.

Für die spätere Verarbeitung der Daten im Rahmen von Eventbuilding und Analyse ist hier kein Zusatzaufwand zu erwarten, da die archivierten Daten in allen Fällen erst dekodiert werden müssen, bevor die eigentlichen Meßwerte verfügbar sind. Ein neu entwickeltes, einheitliches Format könnte sogar dazu beitragen, den Aufwand eher zu reduzieren, wenn anstelle vielfältiger, evtl. notdürftig implementierter Dekodier-Software für unterschiedliche Hardware ein auf das

einheitliche Format optimiertes Softwarepaket eingesetzt werden kann. Damit wird nicht nur die Datensicherheit verbessert, sondern es erlaubt auch eine sehr gute Automatisierung der Datendekodierung, wobei sich die dazu nötige Arbeit auf die einmalige Implementation von entsprechender Software beschränkt.

Die durch Verwendung der logischen Experimentstruktur erreichte Hardware-Unabhängigkeit bedeutet nicht nur praktisch die Erleichterung der Dekodierung, sondern bietet grundsätzlich eine im Vergleich zur hardwareorientierten Lösung wesentlich allgemeinere Beschreibung der Meßdaten. Der Einsatz unterschiedlicher Hardware wird in den Meßdaten nicht mehr erkennbar. Diese repräsentieren nur noch unmittelbar den physikalischen Meßwert. Kenntnisse über Hardware-Details bleiben damit nur noch auf die Datenaufnahme beschränkt, wo sie ohnehin schon benötigt werden.

Bis auf grundsätzlich bedingte Offset- und Eicheffekte lassen sich so bei der Auswertung der Daten mit unterschiedlicher Hardware gemessene Werte auf Anhieb besser vergleichen oder Daten aus verschiedenen Messungen bzw. Teilmessungen kombinieren. Der Austausch von Meßelektronik hat für die Darstellung der Daten keine grundlegenden Konsequenzen. Daten, die im Rahmen einer Vor- oder Zwischenauswertung erzeugt werden und kein Hardware-Gegenstück besitzen, fügen sich nahtlos in die Meßdaten ein. Damit läßt sich auch weiterführende Software zur Auswertung der Meßdaten besser automatisieren, und der Benutzer hat sich bei der Analyse weniger mit technischen Details auseinanderzusetzen.

Den Vorzügen der Formatierung auf der Basis der logischen Experimentbeschreibung steht ein gravierender Nachteil gegenüber. Sie ist in aller Regel mit zusätzlichem Aufwand verbunden, der on-line von einem oder mehreren der an der Datenaufnahme beteiligten Rechnern geleistet werden muß. Ein asynchrones Datenaufnahmeverfahren, wie es in MECDAS Verwendung findet (s. Abschnitt 5.2.3.3), erlaubt jedoch, Auslese und Formatierung so weit zu trennen, daß dieser Mehraufwand weder direkt noch indirekt zur Totzeit beiträgt. So bietet sich einerseits die Möglichkeit an, die Formatierung auf einen solchen Rechner zu verlagern, der nicht an den Echtzeitaktivitäten der Auslese beteiligt ist und in der Lage ist, die Daten parallel zu Auslese im zeitlichen Mittel schnell genug zu formatieren. Andererseits kann aber auch die Tatsache ausgenutzt werden, daß der Datenaufnahmerechner in den Zeiten zwischen der Auslese von Ereignissen freie Rechenkapazitäten besitzt. Denn aufgrund der charakteristischen Eigenschaften der Zeitintervallverteilung, mit der die zu untersuchenden kernphysikalischen Ereignisse auftreten und die kleine Zeiten bevorzugt, und der Berücksichtigung der Anforderung, hohe, nicht mehr kontrollierbare Totzeitverluste zu vermeiden, kann der Rechner im zeitlichen Mittel von der Auslese nur zu einem kleinen Teil ausgelastet werden. Es verbleibt in aller Regel noch genügend Zeit zur Bearbeitung von Aufgaben, die nicht mehr Echtzeitanforderungen unterliegen. Dazu gehört auch die Formatierung der Meßdaten. Da diese Aktivitäten auf dem Datenaufnahmerechner mit einer niedrigen Priorität ablaufen, beeinflussen sie die Auslese nicht und erhöhen lediglich die mittlere Auslastung der Frontendrechner, was schließlich zu einer Verbesserung deren Rentabilität beiträgt und Zusatzinvestitionen in anderen Bereichen vermeidet.

6.8.3 Das MECDAS-Datenformat

Aufgrund der damit verbundenen Vorzüge wurde als Standard-Realisierung innerhalb von MECDAS die Formatierung der Daten auf der Basis der logischen Experimentkonfiguration gewählt. Die implementierte Software stellt den Versuch dar, eine möglichst allgemeine Lösung zur Verfügung zu stellen, mit der auf Anhieb ein weiter Bereich auch sehr unterschiedlicher Experimente abgedeckt werden kann. Im Einzelfall wird sich u.U. eine auf das Problem besser angepaßte Alternative finden lassen, die bei Bedarf die Standard-Lösung ersetzen kann. Das System stellt im Rahmen der Trennung zwischen experimentunabhängiger Kernsoftware und experimentspezifischen Softwareteilen hierfür explizit Schnittstellen zur Verfügung, von denen auch die implementierte Lösung ausschließlich Gebrauch macht (s. Abschnitt 5.8.2).

Es wurde Software zur Formatierung und Dekodierung der Meßdaten und all der Zusatzinformation, die zu einer möglichst umfassenden Beschreibung der Daten notwendig ist, entwickelt und i.w. in Form von Unterprogrammbibliotheken zur Verfügung gestellt. Innerhalb des Datenaufnahmeprogrammes deckt sie die bis dahin experimentspezifischen Aufgaben der Formatierung vollständig ab, die wie in Abschnitt 5.2.3.3 beschrieben asynchron zur Auslese auf den Frontend-Rechnern bearbeitet werden, bei Bedarf aber auch von einem übergeordneten Rechner übernommen werden können. Die Software wird ergänzt durch ein abgeschlossenes Programm, das unter Ausnutzung dieser Bibliotheken in der Lage ist, die Daten jedes beliebigen mit MECDAS durchgeführten Experiments vollständig zu dekodieren (s. Kap. 8).

6.8.3.1 Datensegmente

Das MECDAS-Datenformat definiert die von MECDAS-Software im Rahmen der Datenaufnahme, des Eventbuilding oder der Analyse erzeugten Daten als Strom von einzelnen, in der Regel voneinander unabhängigen Datensegmenten, deren Struktur festen Regeln unterworfen ist. Solche Datensegmente können sein:

- die eigentlichen Meßdaten, die ein Ereignis bzw. Teilereignis beschreiben
- Kennblöcke mit Experimentname und Zeitstempel
- die Experimentbeschreibung
- ergänzende Konfigurationsinformation für die Einzelmessung
- Name und Nummer der Einzelmessung
- ergänzende, nicht durch die Datenerfassung aufgenommene Daten, die über den aktuellen Zustand der Meßapparatur informiert
- Benutzerkommentare

Jedes Datensegment setzt sich aus zwei Teilen zusammen, dem Datensegment-Kopf und dem Datensegment-Körper, die jeweils selbst wieder aus einer mehr oder weniger großen Anzahl von 16-Bit-Worten bestehen.

Elementare Datenworte

Die Wortbreite von 16 Bit spielt im MECDAS-Format eine zentrale Rolle. Sie stellt die kleinste Dateneinheit innerhalb von Kopf und Körper dar. Sie wurde teils aus historischen Gründen[3], teils aus praktischen Erwägungen gewählt, denn 16 Bit stellen in vielerlei Hinsicht einen sehr guten Kompromiß dar.

Zum einen ist eine durchgängige einheitliche Wortbreite hilfreich, um die systematische Bearbeitung der Daten zu vereinfachen (siehe auch unten). Dabei ist es sinnvoll, sich an den elementaren Wortgrößen der eingesetzten bzw. in Frage kommenden Rechnern zu orientieren. Hier ist es inzwischen so, daß nicht nur auf die Anwendung der Datenerfassung bezogen sondern generell alle Rechner bis auf exotische Ausnahmen Worte aus 8, 16 und 32 Bit verarbeiten können. Andererseits zeigt sich, daß bei der Mehrheit der verwendeten Datenerfassungs-Hardware im CAMAC-, Fastbus- und VMEbus-Bereich, Daten üblicherweise nicht mehr als 16 Bit umfassen und damit 16 Bit eine natürliche Wortbreite darstellen. In den wenigen Fällen, in denen die Wortbreite größer ist, lassen sich die Daten mit wenig Aufwand in 16-Bit-Worte zerlegen; im anderen Fall, also bei Abspeicherung von einzelnen Bytes, wird etwas Speicherplatz verschenkt. Erfahrungsgemäß sind diese Probleme in der Praxis jedoch nicht von allzu großer Bedeutung.

Die Verwendung einer Einheitswortgröße von 32 Bit würde demgegenüber in der überwiegenden Mehrzahl der Fälle unnötig Speicher verschwenden. Bei Verwendung von 8-Bit-Worten, der üblichen Einheit bei der Speicherplatzverwaltung in UNIX, könnte zwar der Speicherplatz bestmöglich ausgenutzt werden, doch gäbe es hier prozessorspezifische Probleme bei der Aufeinanderfolge von 8-Bit und 16-Bit-Worte. Viele Prozessoren sind nicht in der Lage, 16-Bit-Worte auf ungerade Adressen abzulegen. In diesem Fall müßte jedes Datenwort, auch wenn es nur zum Umkopieren ist, explizit in seine Bytes zerlegt bzw. wieder zusammengesetzt werden. Das würde die Software an einer besonders kritischen Stelle, die für jedes Datum durchlaufen wird, unnötig aufblähen. Werden ausschließlich 16-Bit-Worte verwendet, können bei korrekter Programmierung nie ungerade Adressen auftreten, wodurch hier das Problem entfällt.

Der Datensegment-Kopf

Der Datensegment-Kopf besitzt die in Abbildung 6.14 dargestellte Struktur. Er besteht aus sechs nicht vorzeichenbehafteten 16-Bit-Worten, die einerseits formatspezifische Information und andererseits übergeordnete Nutzinformation enthalten. Das erste Wort gibt die **Größe des Datensegment-Körpers** in Einheiten von 16-Bit-Worten an, die zwischen 0 und $2^{16} - 1 = 65535$ Worten bzw. 131070 Bytes betragen kann. Das zweite Worte gibt die sog. **Byte-Orientierung der Daten** an, eine wichtige Eigenschaft, in der sich trotz vieler Übereinstimmungen bei der Definition von elementaren Datenworten von Prozessoren

[3] Der Einsatz von MECDAS zusammen mit GOOSY erforderte die Berücksichtigung der von diesem System vorgegebenen Randbedingungen. Dazu gehörte u. a. die elementare Wortbreite von 16 Bit. Aber auch der Datensegment-Kopf selbst ist GOOSY-kompatibel.

Länge des Datensegment-Körpers	
Byte-Orientierung	
Format-Version	
Datensegment-Typ	formatspezifische Flaggen
laufende Nummer des Datensegments von diesem Typ	
typspezifischer Code	

Abb. 6.14: Die Struktur des Datensegment-Kopfs.

die Rechner unterscheiden können (s. Anhang A.11). Wenn binäre Daten zwischen Rechnern ausgetauscht werden, macht eine unterschiedliche Byte-Orientierung eine entsprechende Anpassung notwendig. Das heißt, bei Worten, die mehr als ein Byte enthalten, müssen die Bytes entsprechend ihrer Wertigkeiten vertauscht werden, allgemein unter dem Begriff „Byte-Swapping“ bekannt. Der Einsatz eines verteilten, heterogenen Systems bei der Datenerfassung macht nun den ständigen Datenaustausch zwischen Rechnern unterschiedlicher Byte-Orientierung notwendig. Um eine korrekte Datenübertragung zu gewährleisten, müssen die Daten entweder immer einheitlich abgespeichert werden, oder es wird Information über die Byte-Orientierung der abgespeicherten und übertragenen Meßdaten benötigt. Eine einheitliche Abspeicherung — ein Verfahren, wie es bei dem unter TCP/IP arbeitenden Kommunikationsprotokoll XDR u. a. auch im Steuerungssystem der A1-Experimentierapparatur benutzt wird [Kram95] — hat zur Folge, daß auch bei der Datenübertragung zwischen zwei Rechnern mit identischer Byte-Orientierung stets eine Hin- und Rückwandlung erfolgen muß, obwohl es in diesem Fall gar nicht nötig wäre. Der damit verbundene Aufwand kann sich insbesondere bei der Datenaufnahme nachteilig bemerkbar machen. Diese Nachteile entfallen bei der zweiten Methode.

Mit der Kenntnis der Byte-Orientierung der Meßdaten ist das Byte-Swapping nur im Bedarfsfall nötig. Da der Datenaustausch nicht nur explizit über eine direkte Rechnerkommunikation, sondern auch implizit durch Bewegen von Datenträgern, auf denen die Daten abgespeichert sind, erfolgen kann, muß die Information über die Byte-Orientierung in den Daten selbst enthalten sein. Da außerdem einzelne Segmente innerhalb eines Datenstroms von unterschiedlichen Rechnern stammen können, muß die Byte-Orientierung individuell für jedes Datensegment definierbar sein. Dazu dient das zweite Wort im Datensegment-Kopf. Es wird bei Generierung der Daten entsprechend der lokalen Byte-Orientierung gesetzt und beim Lesen der Daten überprüft. Falls die Byte-Orientierung der abgespeicherten Daten nicht mit der des Rechners übereinstimmt, wird Byte-Swapping für dieses Datensegment notwendig. In diesem Fall erlaubt die einheitliche 16-Bit-Struktur der Daten eine sehr effiziente, vom Inhalt der Daten unabhängige Ausführung.

Die dritte Komponente des Datensegment-Kopfes enthält die **Versionsnummer des Formats**, mit dem die Daten abgespeichert sind. Sie erlaubt das gleichzeitige Nebeneinanderbenutzen von verschiedenen Formatierungsverfahren. Die derzeitige Versionsnummer ist 30000, entsprechend der Format-Version 3.0. Im Gegensatz zur Byte-Orientierung muß innerhalb eines Meßdatenstroms eine ein-

heitliche Versionsnummer verwendet werden. Sie dient zusätzlich bei der Dekodierung der Daten Konsistenzüberprüfungen und wird als Synchronisierungshilfsmittel verwendet, wenn bei der Dekodierung verstümmelte Daten festgestellt werden und übersprungen werden müssen.

Der Datensegment-Kopf enthält in seiner vierten Komponente Informationen, die die Bedeutung und damit auch die Formatierungs- bzw. Dekodierungsvorschriften der nachfolgenden Daten festlegen. Das höherwertige (!) Byte definiert den **Typ des Datensegments**, das niederwertige enthält **zusätzliche Formatierungsangaben**, mit deren Hilfe z.B. das Byte-Swapping für dieses Segment generell unterbunden werden kann.

Die letzten beiden Worte enthalten, abhängig von Segment-Typ, bereits Nutzinformation, die sich z.B. aus der Datenaufnahme ergeben kann, aber globale Bedeutung hat. In der Regel wird in der fünften Komponenten des Segment-Kopfes die **laufende Nummer des Segments** von diesem Typ modulo 2^{16} angegeben. Infolge der begrenzten Wortbreite ist dieser Wert nur in einem lokalen Bereich eindeutig, sollte aber dort streng monoton steigend sein, wenn er überhaupt gültig ist. Im Falle von regulären Meßdaten-Segmenten entspricht dieser Wert der laufenden Ereignisnummer. Die sechste und letzte Komponente des Datensegment-Kopfes ist eine **typspezifische Kennzeichnung des Segment-Körpers**. Bei Meßdatensegmenten wird damit der Ereignistyp spezifiziert, also ob die Daten zu einem Einzel-, Koinzidenz- oder Diagnoseereignis gehören. Diese Information bietet in Kombination mit der Ereignisnummer eine sehr gute, experimentunabhängige Grundlage für die Selektion bestimmter Ereignisse bis hin zu den Aufgaben des Eventbuilding, ohne daß dazu die experimentspezifischen Daten des Segment-Körpers interpretiert werden müßten.

Der Datensegment-Körper

Der Datensegment-Körper enthält in Abhängigkeit von dem im Kopf spezifizierten Typ die eigentlichen **Nutzdaten**. Tabelle 6.3 zeigt eine kurze Übersicht der verschiedenen, bisher unterstützten Datensegmenttypen. Von besonderem Interesse sind reguläre Ereignisdaten, die Experimentbeschreibung und -Kennzeichnung und Informationen über den Zustand der Meßappartur. Sie sollen im folgenden näher beschrieben werden.

6.8.3.2 Das Ereignisformat

Bei der Formatierung der Meßdaten, die jeweils ein nachgewiesenes physikalisches Ereignis beschreiben, wird ausschließlich von der hierarchisch strukturierten logischen Experimentkonfiguration Gebrauch gemacht. Ziel der Formatierung ist die Abspeicherung der Meßwerte und ausreichend Information zu deren eindeutigen Identifikation. Dazu bietet der aus der logischen Beschreibung konstruierbare Baum (s. Abschnitt 6.4) alle benötigten Informationen. Jeder Datenkanal ist darin eindeutig durch den Adressierungspfad des ihm entsprechenden Blattes identifiziert. Wie bei jedem anderen Baumknoten setzt sich dessen

Typ	Inhalt der Datensegmente
0	reguläre Ereignisdaten
1	Verwaltungsinformation
2	Experimentkennzeichnung
3	Experimentbeschreibung
4	Informationen über den Zustand der Meßappartur
5	nicht weiter spezifizierte Daten
6	Zusatzinformationen über die aktuelle Einzelmessung
7	Kennzeichnung eines Experimentteils
8	fehlerhafte Ereignisdaten
9	rudimentär verpackte Ereignisdaten
10	benutzerspezifische Informationen und Kommentare
11	Name und Nummer der Einzelmessung
12	Fehlermeldungen oder andere Hinweise für den Benutzer, die im Laufe der Datenaufnahme von der Software erzeugt wurden

Tab. 6.3: Bisher unterstützte Datensegmenttypen.

Adressierungspfad aus dem Pfad des übergeordneten Knotens (Vater), falls dieser vorhanden ist, und der relativen Adresse des betreffenden Knoten innerhalb des durch den Vater definierten Teilbaums zusammen. Die Software stellt diese Adreßinformation sowohl in symbolischer als auch numerischer Form zur Verfügung, d. h., der Adressierungspfad besteht entweder aus einem n-Tupel aus symbolischen Namen oder aus einem n-Tupel aus Zahlen. Für die Formatierung bietet sich die Verwendung der Zahlen an.

Infolge der hierarchischen Struktur kann innerhalb eines Teilbaums auf die volle Angabe des n-Tupels verzichtet werden. Hier reicht die relative Adresse, im einfachsten Fall lediglich eine Zahl, zur eindeutigen Identifizierung aus. Von dieser Tatsache wird bei der Formatierung Gebrauch gemacht. Die Daten werden der logischen Struktur entsprechend so verpackt, daß zu einem Teilbaum gehörige Daten in der Regel zusammen abgespeichert werden. In diesem Fall reicht es aus, die einzelnen Daten lediglich durch ihre relativen Adressen zu kennzeichnen, nachdem die logische Adresse des Teilbaums, die wiederum selbst relativ sein kann, nur einmal angegeben werden mußte.

Dieses Verfahren hat den Vorteil, daß einerseits sich in den formatierten Daten die logische Struktur wiederfindet, andererseits die Adreßinformation dabei auf ein Minimum beschränkt ist. Damit ist es bei der Datenerfassung möglich, die Daten einzelner Teilbäume unabhängig voneinander zu verpacken, was u. a. die Verteilung ihrer Bearbeitung auf mehrere Rechner erleichtert bzw. beim Eventbuilding eines aus mehreren Subsystemen bestehenden Experiments sehr hilfreich ist (siehe Abschnitt 7.3). Auch bei der Auswertung der Meßdaten wird so eine unabhängige Bearbeitung der Teilbäume vereinfacht. Es besteht die Möglichkeit, Daten einzelner Teilbäume selektiv auszuwerten, ohne dafür alle Daten vollständig dekodieren zu müssen. Andererseits wird es damit leichter, die Auswerte-Software auch gegen Änderungen der Experimentkonfiguration unempfindlicher zu machen.

Das Format, mit dem die Meßdaten in MECDAS verpackt werden, versucht diesen Möglichkeiten weitestgehend Rechnung zu tragen. Daten, die zu einem Teilbaum gehören, werden zusammen abgespeichert und durch relative Adressen und Längeninformationen gekennzeichnet. Sie bilden jeweils Datenblöcke, die selbst wiederum Bestandteil eines übergeordneten Teilbaums sein können, und entsprechend mit Adresse und Länge gekennzeichnet und zusammengefaßt werden, bis schließlich die Wurzel des Konfigurationsbaums erreicht ist. Das Format besitzt damit eine rekursive Struktur.

Adreß- und Längeninformation werden den Daten bzw. Datenblöcken in Form vorzeichenbehafteter 16-Bit-Worte vorangestellt. Die eigentliche Information steckt dabei in den Absolutwerten; die Vorzeichen dienen u. a. zur Unterscheidung zwischen elementaren Meßdaten und aus weiterer Formatierungsinformation und Daten zusammengesetzten Datenblöcken. Die exakte Definiton des Ereignisformats geht aus folgendem BNF-Diagramm hervor:

```
Ereignisdaten:   Datenblock
               | Datenblock  Ereignisdaten

Datenblock:      POSITIVE ADRESSE  POSITIVE LAENGE  DATUM
               | NEGATIVE ADRESSE  POSITIVE LAENGE  Ereignisdaten
               | NEGATIVE ADRESSE  NEGATIVE LAENGE  Datenvektor
               | POSITIVE ADRESSE  0
               | NEGATIVE ADRESSE  0
               |

Datenvektor:     POSITIVE ADRESSE  DATUM
               | POSITIVE ADRESSE  DATUM  Datenvektor
```

Die terminalen Symbole `POSITIVE ADRESSE`, `NEGATIVE ADRESSE`, `DATUM` und `0` stellen 16-Bit-Worte dar, für die gilt:

```
0 < POSITIVE ADRESSE < 32768
0 < POSITIVE LAENGE  < 32768

-32769 < NEGATIVE ADRESSE < 0
-32769 < NEGATIVE LAENGE  < 0

-32769 < DATUM < 32768
```

Durch die ausschließliche Verwendung relativer logischer Adressen und die konsequente Benutzung der Längeninformation konnte erreicht werden, daß die nach diesem Format verpackten Daten prinzipiell auch ohne Kenntnis der konkreten logischen Konfiguration dekodiert und weiterverarbeitet werden können. Damit ist eine noch bessere Modularisierung der Analyse-Software möglich. Sowohl zur Formatierung als auch zur Dekodierung konnte so die Implementierung auf jeweils eine kompakte Routine beschränkt werden.

6.8.3.3 Die Experimentbeschreibung

Um zumindest für die allgemeinen und gut automatisierbaren Aufgaben bei der Verarbeitung archivierter Meßdaten möglichst wenig oder sogar keine zusätzliche Information zu benötigen, war ein wesentliches Ziel, die auf Magnetband, Platte oder einem anderen Medium archivierten Daten so gut es geht selbstbeschreibend zu machen. Das konnte bis zu einem gewissen Punkt durch die Wahl des Formats für die Meßdaten erreicht werden, wurde jedoch vervollständigt durch das Hinzufügen der Experimentbeschreibung und zusätzlicher Konfigurationsinformation zu den Meßdaten.

Dabei wurde von der Fähigkeit des MECDAS-Datenformats Gebrauch gemacht, nicht nur Meßdaten, die auf eine bestimmte Art und Weise verpackt wurden, abzuspeichern, sondern bis zu 256 verschiedene Datensegment-Typen und damit auch entsprechend viele unterschiedliche Formatierungsvorschriften zu definieren. Mit der Definition entsprechender Datensegment-Typen war formal die Möglichkeit gegeben, nun auch die Experimentbeschreibung und andere Information im Datenstrom unterzubringen und von den regulären Meßdaten zu unterscheiden. Die dazugehörigen Formatierungs- bzw. Dekodierungsvorschriften reduzieren sich in den meisten Fällen auf dasselbe Prinzip:

Bei der Formatierung wird die in der Regel in Form von einer oder mehrern Dateinen vorliegenden Information mithilfe von Standard-UNIX-Werkzeugen wie **tar** oder **shar** zu einem sog. Archiv verpackt, das Dateiinhalte neben zusätzlicher Information zum späteren Wiederherstellen der Daten enthält. Dieses Archiv bildet schließlich das neue Datensegment. Bei der Dekodierung ist nichts anderes zu tun, als den Inhalt eines solchen Datensegments durch das entsprechende Auspackprogramm extrahieren zu lassen. Die Verwendung von Shell-Archiven hat dabei sogar den Vorteil, daß der Inhalt des Datensegments in Kombination mit der Shell bereits ein lauffähiges Programm ist und das Auspackprogramm selbst darstellt.

Im Falle der Experimentbeschreibung, die ja aus einer Reihe von durch den Benutzer bereitzustellender Dateien besteht, sorgt bereits die Konfigurations-Software automatisch dafür, daß alle diese Dateien und solche, von denen sie zusätzlich abhängig sind, zu einem Archiv verpackt werden. Dabei werden die Dateien exakt so, wie sie sind, in das Archiv übernommen. Insbesondere bleibt damit ihr Original-Layout inklusive der vom Benutzer angegebenen Kommentare als ergänzende Dokumentationsmöglichkeit erhalten.

6.8.3.4 Die Experimentkennzeichnung

Ergänzend zu der im vorangegangenen Abschnitt diskutierten Experimentbeschreibung wurde ein Datensegment definiert, das die eindeutige, zu einer bestimmten Experimentbeschreibung gehörende Experimentkennzeichnung enthält. Sie besteht aus einem einzelnen Textstring, der sich in einer wohldefinierten Weise aus dem Namen der bei dieser Messung verwendeten Experimentkonfiguration und Datum und Uhrzeit, wann die Experimentbeschreibung erstellt wur-

de, zusammensetzt. Diese Experimentkennzeichnung dient bei der Dekodierung im Bedarfsfall zum automatischen Verarbeitung der richtigen Experimentbeschreibung. Die Datenaufnahme-Software streut die Experimentkennzeichnung, versehen mit einem Zeitstempel, von Zeit zu Zeit in die Meßdaten ein und bietet so die Möglichkeit weitergehender Konsistenzchecks der Daten bei der Auswertung.

6.8.3.5 Zustand der Meßapparatur

Neben den ereignisweise zu erfassenden Meßdaten werden noch weitere, der Datenaufnahme nicht direkt zugängliche Daten benötigt, um alle das Experiment beschreibende physikalische Parameter zu erfassen. Solche Daten sind im Falle der Drei-Spektrometer-Anlage u. a.:

- Strahlenergie und -strom
- Magnetfeldeinstellungen der Spektrometer
- Winkel der Spektrometer
- Detektorspannungen und Diskriminatorschwellen
- Informationen über die Koinzidenz- und Trigger-Logik
- Informationen über Gasfluß, Druck und Temperatur verschiedener Komponenten der Apperatur

Diese Inforamtion wird von der Software zur Steuerung der Experimentierapparatur zur Verfügung gestellt [Kram95] und muß zu den regulären Meßdaten hinzugefügt werden. Auch hierfür sind Vorkehrungen in der Software und im MECDAS-Datenformat getroffen. Ein spezielles Datensegment dient zur Aufnahme dieser Apparaturstatusinformation. Bisher wurden diese Daten in einem experiment- bzw. hardwarespezifischen Format von externer Software zur Verfügung gestellt und ohne weitere Bearbeitung übernommen. Es ist aber sinnvoll, in Zukunft auch hier auf die für Meßdaten verwendete und inzwischen bewährte Formatierungsmethode überzugehen, die der Dekodier-Software das automatisierte Auspacken auch dieser Daten erlaubt.

Kapitel 7

Die Software zur Realisierung von komplexen Systemen

7.1 Verteilte Datenerfassung

In der bisherigen Beschreibung von MECDAS orientierte sich die Darstellung an den grundlegenden Problemen der Datenerfassung, die bereits bei einfachen Experimenten auftreten. Die an MAMI durchzuführenden Experimente zeigen jedoch eine deutlich größere Komplexität. Ein sehr gutes Beispiel dafür stellt die Drei-Spektrometer-Anlage dar. Hier genügt nicht mehr der Einsatz eines einzigen Frontend-Systems, sondern es ist aus vielerlei, bereits diskutierten Gründen notwendig, ein aufwendiges, verteiltes Rechnersystem einzusetzen. Es wird je nach Komplexität des Experiments eine mehr oder weniger große Anzahl von Rechnern benötigt, die jeweils unabhängig voneinander Teilaufgaben übernehmen. Eine natürlich Aufteilung ist dabei durch die räumliche Trennung der Detektorsysteme oder die Art der anfallenden Aufgaben vorgegeben.

In vielen Fällen ist es sinnvoll, zur individuellen Behandlung einzelner, in sich abgeschlossener Detektorsysteme wie der Arme einer Koinzidenzanordung bzw. größere Detektoreinheiten, aus denen eine komplexe Experimentieranlage bestehen kann, jeweils eigene Frontend-Rechner einzusetzen. Diese nehmen dann die lokale Ansteuerung der jeweiligen Detektorelektronik vor und helfen damit nicht nur bei der Vermeidung unnötigen Verkabelungsaufwands. Sie können vielmehr durch den mit dem Einsatz mehrerer Rechner verbundenen Parallelisierungseffekt einerseits und durch die Möglichkeit einer besseren an spezielle Probleme angepaßten Bearbeitung ihrer Aufgaben andererseits deutlich zur Steigerung der Leistung des Gesamtsystems beitragen. Bei der Drei-Spektrometer-Anlage (Abb. 7.1) bietet sich eine Aufteilung an, bei der jedem Spektrometer mindestens ein Frontend-System zugeordnet wird. Jedes Frontend-System kann dann unabhängig von den anderen ausschließlich die auf dem jeweiligen Spektrometer anfallenden Aufgaben individuell bearbeiten.

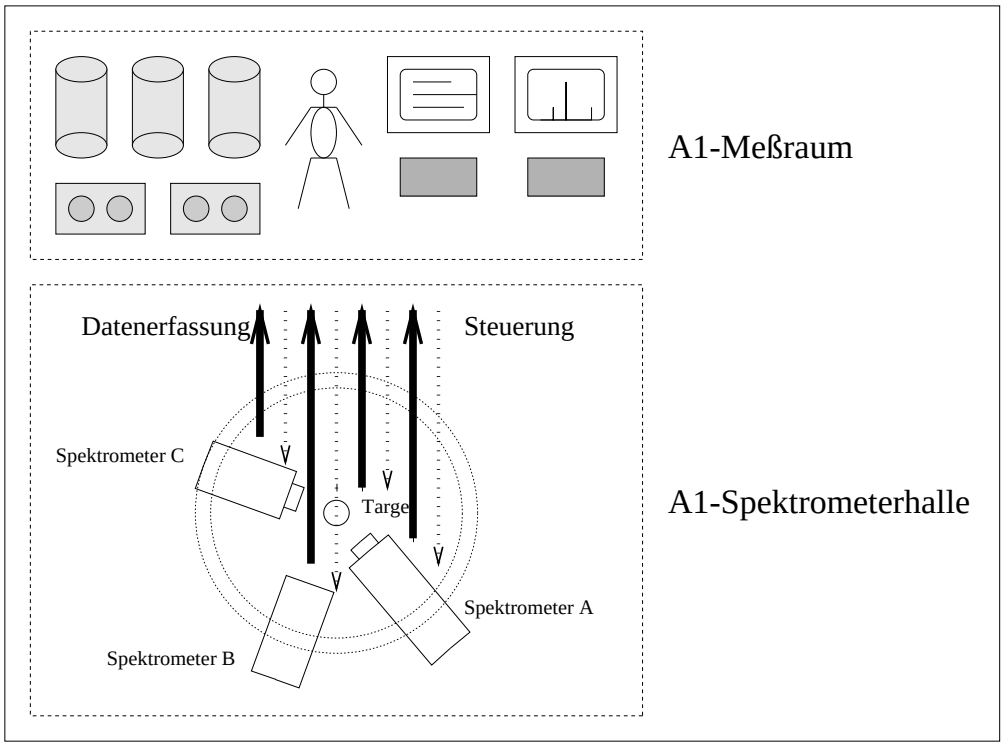

Abb. 7.1: Die räumliche Verteilung der verschiedenen Komponenten der Drei-Spektrometer-Anlage und ihre Relevanz für das Rechnersystem.

Mit dem Einsatz eines derartigen verteilten Systems treten nun ganz neue Aufgabenstellungen auf, die nicht weniger wichtig sind wie die Aufgaben, die das Basissystem bereits abdeckt. Im wesentlichen sind diese Aufgaben:

- Unterstützung ein weitergehende Verteilung der Datenerfassungsaufgaben auf verschiedne Rechner
- Synchronisierung des Systems
- Steuerung des verteilten Systems
- Übermittlung der Meßdaten
- Eventbuilding bei Mehrarmexperimenten
- Hilfen bei der Durchführung und Verwaltung von Messungen
- Sicherstellen, daß die Daten alle benötigten Informationen enthalten
- Archivierung der Meßdaten
- Bereitstellung einer Bedienoberfläche

7.2 MECDAS als verteiltes Datenerfassungssystem

Aufbauend auf der bereits beschriebenen Basissoftware von MECDAS konnten auch die neuen Aufgabenstellungen mit wenig Aufwand abgedeckt werden. Durch einige wenige Zusätze wurde so ein vollständiges, in sich abgeschlosses Datenerfassungssystem realisiert, das sowohl die Durchführung einfacher Experimente gestattet, bei der der Einsatz eines einzigen Rechners ausreicht, als auch für sehr komplexe Experimente geeignet ist, die die Verwendung eines verteilten Rechnersystems erfordern. Basis ist ein einheitliches Grundkonzept, das beide Extremfälle, die so auch in der Realität vorkommen, letztendlich identisch behandelt. Es nutzt die hierarchische Strukturierbarkeit eines Experiments aus, das als Gesamtsystem nach denselben Methoden arbeitet wie dessen Teilsysteme.

7.2.1 Systemtopologien

Bei der näheren Analyse von verteilten Systemen, die für die Datenerfassung in Frage kommen, zeigt sich, daß man zwischen vier verschiedenen Systemtopologien mit praktischer Bedeutung unterscheiden kann (Abb. 7.2):

1. Ein Rechner bzw. ein Master-Slave-System übernimmt alle mit der Datenerfassung zusammenhängenden Aufgaben entsprechend dem ausführlich besprochenen Basismodell.

2. Der Datenaufnahmerechner aus dem Basismodell gibt Teile seiner Aufgaben an Rechner ab, die als intelligente Subsysteme arbeiten und dedizierte Aufgaben bei der Datenaufnahme übernehmen. Diese Subsysteme sind dazu in der Regel mit spezieller Software ausgestattet, um bestimmte Teile der Meßelektronik individuell zu bearbeiten. Dabei werden Daten aufgenommen und Ereignis für Ereignis — also in Echtzeit — dem übergeordneten Rechner zur Verfügung gestellt.

3. Ein Rechner arbeitet als Master und kontrolliert mehrere Slaves. Wie beim einfachen Master-Slave-Modell wird hier eine strikte Aufgabenteilung vorgenommen: ausschließlich die Slaves erledigen die Aufgaben der Datenaufnahme unter Echtzeitbedingungen; der Master übernimmt übergeordnete Aufgaben, steuert die Slaves und sorgt für die Datenarchivierung. Die Kommunikation zwischen Master und Slave ist gepuffert und stellt keine Echtzeitanforderungen.

4. Mehrere in sich abgeschlossene und mehr oder weniger komplexe Frontend-Rechnersysteme teilen sich die Aufgaben der Datenerfassung. Dieser Fall unterscheidet sich vom dritten nur dadurch, daß die einzelnen Rechnersysteme durch unabhängige Rechner kontrolliert werden, die unter einem Multitaskingsystem und mit geringen Echtzeitanforderungen arbeiten. Wie an früherer Stelle (Kapitel 3) diskutiert werden in der Regel die Frontend-Rechner durch einen oder mehrere Rechner ergänzt, die übergeordnete und koordinierende Aufgaben wahrnehmen. Im konkreten Fall werden hier Workstations eingesetzt.

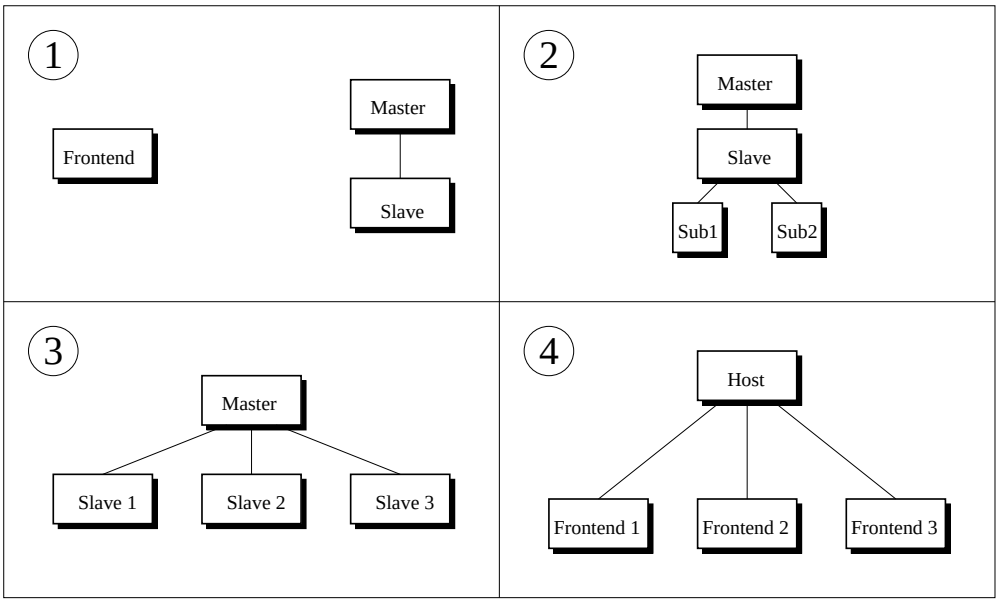

Abb. 7.2: Die vier verschiedenen, im Rahmen der Datenerfassung interessanten Systemtopologien. Mischungen sind natürlich ebenfalls möglich.

7.2.2 Lösungsansätze

MECDAS wurde so konzipiert, daß es neben dem ersten auch alle anderen der im vorangegangenen Abschnitt beschreibenen Systemtopologien abgedeckt.

7.2.2.1 Intelligente Subsysteme

Für MECDAS unterscheidet sich der zweite Fall vom ersten nicht. Ein intelligentes Subsystem stellt aus der Sicht der Software als herkömmliche Datenerfassungs-Hardware dar. Damit diese von MECDAS genutzt werden kann, sind lediglich geeignete Gerätebeschreibungen (s. Abschn. 6.3) erforderlich, die je nach Hardware-Schnittstelle, über die ein Subsystem an den Datenaufnahmerechner angeschlossen ist, wie z.B. VMEbus, CAMAC, Fastbus, SCSI, V.24 usw. die korrekte Kommunikation bzw. den Datenaustausch zwischen Datenaufnahmerechner und Subsystem sicherstellt. Damit können in weitem Rahmen intelligente Subsysteme in das Gesamtsystem integriert werden. Für die Software auf derartigen Subsystemen existieren von Seiten der Datenerfassung a priori keine Vorgaben. Sie muß neben der Datenerfassungsaufgaben lediglich die korrekte Kommunikation mit dem übergeordneten Rechner ermöglichen.

7.2.2.2 Verteilung auf gleichberechtigte Systeme

Zwischen dem dritten und vierten Fall zeigen sich für MECDAS grundsätzlich keine Unterschiede. In der Praxis machen sich lediglich die verschiedenen Kommunikationsmechanismen zwischen Frontends und übergeordneten Rechnern be-

1. Ein Master mit n Slaves:

```
start -i slave1
start -i slave2
start -i slave3

archive -i slave1 -o file1 &
archive -i slave2 -o file2 &
archive -i slave3 -o file3 &
```

2. n Master-Slave-Systeme:

```
rsh frontend1 start
rsh frontend2 start
rsh frontend3 start

rsh frontend1 archive -o - > file1 &
rsh frontend2 archive -o - > file2 &
rsh frontend3 archive -o - > file3 &
```

3. Hierarchisches Client-Server-Modell:

```
start

archive -o file &
```

Abb. 7.3: Beispiele für Kommandos zur Steuerung der Datenerfassung in einem verteilten Systems bei unterschiedlicher Realisierung der Verteilung.

merkbar. In beiden Fällen findet eine Verteilung der Aufgaben auf grundsätzlich gleichberechtigte, parallel arbeitende Rechner statt. Für beide Fälle läßt sich in erster Näherung ein trivialer Lösungsansatz finden. Das Master/Slave-Modell von MECDAS erlaubt es ohne weiteres, mehrere Slaves durch einen Master zu kontrollieren, soweit die Slaves eine korrekte Kommunikation erlauben und aufgrund unterschiedlicher Kommunikationsadressen eindeutig zu identifizieren sind. In diesem Fall ist bei n Slaves jedes Steuerkommando n-mal unter Angabe der verschiedenen Adressen zu wiederholen (siehe auch Abbildung 7.3). Dasselbe gilt analog auch bei einem System, bei dem bereits die Master-Funktionalität auf mehrere Rechner verteilt ist. Für eine zentrale Steuerung sind jedoch rechnerübergreifende Kommunikationsmechanismen zu nutzen, die das Ausführen von Kommandos auf anderen, entfernten Rechnern erlauben.

In beiden Fällen bietet der UNIX-Kommandointerpreter als unmittelbare Benutzerschnittstelle die Möglichkeit, über die Definition von Kommandoprozeduren jeweils n sich entsprechende Steuerkommandos zu einem zusammenzufassen. Sie können helfen, die Bedienung auch eines komplexen Systems einfach zu halten. Bei dieser trivialen Lösung bleiben jedoch zwei grundlegende Probleme:

1. Es ist schwer, insbesondere beim Auftreten von Fehlern, das Gesamtsystem immer in einem konsistenten Zustand zu halten. Hierfür ist eine aufwendige Programmierung notwendig, die nicht mehr durch Shell-Programmierung erreicht werden kann.

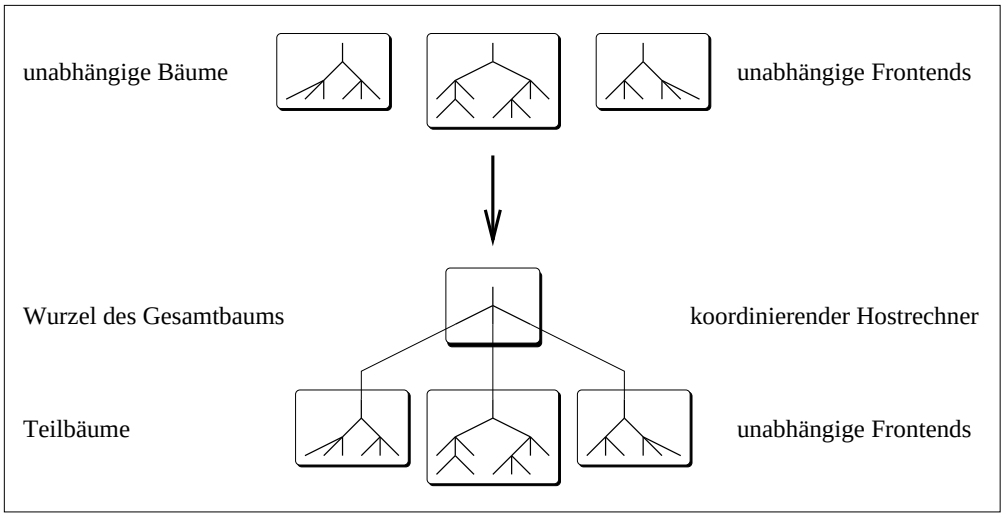

Abb. 7.4: Gegenüberstellung von n-Einzelmessungen und Gesamtkonfiguration mithilfe der Bäume und Teilbäume

2. Jedes Teilsystem liefert einen unabhängigen Meßdatenstrom, der immer nur bruchstückhaft Ereignisinformationen besitzt. Erst in Kombination mit den Daten der anderen Frontends ergibt sich die vollständige physikalische Information. Um sie abzuspeichern, müssen also entweder alle Datenströme getrennt archiviert und bei der späteren Auswertung zusammengefaßt werden, oder die Zusammenfassung geschieht on-line und erlaubt so die Beschräunkung der Archivierung auf einen einzigen Datenstrom.

7.2.3 Hierarchisches Client-Server-Modell

Beide Probleme lassen sich durch konsequente Anwendung der in Kapitel 5 und 6 beschriebenen Verfahren auf das Gesamtexperiment lösen, was jedoch in den genannten Lösungsansätzen noch nicht getan wurde. Die triviale Lösung entspricht nämlich n unabhängigen Einzelmessungen, d. h., die n Slaves bzw. Frontend-Systeme bearbeiten unabhängig voneinander n einzelne logische Konfigurationsbäume. Die Tatsache, daß diese Bäume in Wirklichkeit Teile eines einzelnen Baumes und damit zu einem einzigen Experiment gehören, wurde bisher nicht berücksichtigt.

Die Information über die Zusammensetzung eines Experiments aus verschiedenen Armen — beschrieben durch seinen logischen Konfigurationsbaum, der aus einzelnen Teilbäumen besteht — kann jedoch ohne weiteres als abstrakte Beschreibung des Gesamtexperiments von MECDAS benutzt werden, in der die einzelnen Datenkanäle nicht mehr herkömmlicher Meßelektronik, sondern Rechnern bzw. der auf diesen Rechnern laufenden Datenerfasssungsprogrammen entsprechen. Diese können als virtuelle Geräte angesehen werden, für die bereits MECDAS-Gerätebeschreibungen existieren bzw. mit wenig Aufwand realisiert werden können. In Analogie zur herkömmlichen Meßelektronik enthalten die Beschreibungen Informationen, wie die virtuellen Geräte zu steuern sind, und wie der Datentransport zur Auslese zu bewerkstelligen ist.

```
/* ------------------------ logische Beschreibung ------------------------ */

struct abc {
        struct a        a;                          /* 1. Teilbaum/Exp.-Arm */
        struct b        b;                          /* 2. Teilbaum/Exp.-Arm */
        struct c        c;                          /* 3. Teilbaum/Exp.-Arm */
} abc;                                              /* Gesamtexperiment     */

/* --------------------- physikalische Beschreibung ---------------------- */

extern Device   rsh;             /* virt. Geraet (Frontend-Ansteuerung)   */
extern Device   eventbuilder;    /* virt. Geraet (Eventbuilding auf Host) */

Group speka, spekb, spekc, host;

Computer computer = {
        FRONTEND(eventbuilder, host),          /* uebergeordneter Host     */
        FRONTEND(rsh, speka),                  /* Frontend mit Namen speka */
        FRONTEND(rsh, spekb),                  /* Frontend mit Namen spekb */
        FRONTEND(rsh, spekc),                  /* Frontend mit Namen spekc */
};

/* ------------------ physikalisch/logische Zuordnung ------------------- */

abc.a = speka[0];             /* 1. Teilbaum bearbeitet von Frontend speka */
abc.b = spekb[0];             /* 2. Teilbaum bearbeitet von Frontend spekb */
abc.c = spekc[0];             /* 3. Teilbaum bearbeitet von Frontend spekc */
```

Abb. 7.5: Beispiel für die Experimentbeschreibung eines Gesamtexperiments

Experiment- und Gerätebeschreibungen zusammen bilden nach den bereits erläuterten Regeln schließlich die Grundlage für ein übergeordnetes Datenaufnahmeprogramm, mit dessen Hilfe die Steuerung der einzelnen Arme der Experimentieranlage koordiniert und durchgeführt wird. In aller Regel sorgt dieses Programm dafür, daß auch die Daten, die es von den einzelnen Armen geschickt bekommt, zu einem einzigen konsistenten Datenstrom zusammengefaßt werden. Für den Benutzer stellt das Programm die zentrale Schnittstelle zum Gesamtexperiment dar, das wie jedes andere Datenaufnahmeprogramm mit herkömmlichen Kommandos gesteuert werden kann, ohne daß sich der Benutzer um Details kümmern müßte. Auch Archivierung und On-line-Analyse der Meßdaten erfolgt nach den üblichen, oben beschriebenen Methoden. Für den Benutzer ist nicht erkennbar, ob das Datenaufnahmeprogramm unmittelbar auf Hardware zugreift oder dazu mit untergeordneter Software kommuniziert.

Die beschriebene Funktionalität läßt sich zusammenfassend mit dem Begriff „hierarchisches Client-Server-Modell“ bezeichnen. Damit soll zum Ausdruck gebracht werden, daß die Aufgaben der Datenerfassung auf ein hierarchisch strukturiertes System von Datenaufnahme-Servern verteilt werden, deren Beziehungen zueinander über das bereits beschriebene Client-Server-Konzept von MECDAS geregelt sind. Dabei kann ein Server entweder direkt durch den Benutzer gesteuert werden, oder aber er wird mit weiteren Servern zu einer logischen Einheit zusammengefaßt und durch einen übergeordneten Server kontrolliert.

Dieser stellt für die anderen effektiv einen Klienten dar und kann mit dieser Eigenschaft ihre Dienste in Anspruch nehmen. Damit ist es generell möglich, die Datenerfassung für ein beliebig komplexes Experiment mithilfe von MECDAS auf der Basis eines geeignet verteilten Rechnersystems durchzuführen. Welche Möglichkeiten dabei geboten werden, wird im wesentlichen durch die verfügbare Software zur Ansteuerung der Frontends bestimmt, die die von MECDAS kontrollierten Teilarme des Experiments repräsentieren

Der übergeordnete Datenerfassungs-Server kann nun auf einem Rechner gestartet werden, der übergeordnete Aufgaben bei der Durchführung eines Experiments wahrnehmen soll — im folgenden unter dem Begriff „Host“ bzw. „Hostrechner“ bezeichnet. Ein gesonderter Rechner ist jedoch nicht zwangsläufig notwendig. Diese Funktion kann ebenso einer der beteiligten Frontend-Master-Rechner übernehmen, falls dessen CPU-Leistung nicht durch die unmittelbaren Aufgaben der Datenerfassung ausgeschöpft werden. Im allgemeinen jedoch sollen die Begriffe „Host“ bzw. „Frontend“ die Funktion der entsprechenden Rechner innerhalb eines verteilten Systems zur Datenerfassung deutlich machen. Während der Host einen Rechner mit vorwiegend übergeordneten Aufgaben repräsentiert, ist der Frontend für die hardwarenahen Aufgaben zuständig und hat sich bei Bedarf einem Host unterzuordnen.

7.2.4 Steuerung des verteilten Systems

7.2.4.1 Grundprinzip

Eine der wichtigsten Aufgaben, die das hierarchische Client-Server-Konzept leistet, ist die zentrale Steuerung der auf der Basis eines verteilten Rechnersystems arbeitenden Datenerfassungs-Software. Dabei wird ausgenutzt, daß in der herkömmlichen Anwendung ein MECDAS-Datenaufnahmeprogramm nicht nur die Ansteuerung der Meßelektronik unmittelbar bei der Auslese vornimmt, sondern bei Bedarf auch deren, im Rahmen der Datenerfassung notwendige Steuerung aufgrund von Zustandsübergängen des Systems vornehmen kann. Eine Verallgemeinerung dieser Aufgaben schließt auch die Ansteuerung virtueller Geräte, also u. a. die auf den Frontend-Rechnern laufende Datenerfassungs-Software, ein. Ein aus solchen Einzelkomponenten bestehendes verteiltes System kann mit wenig Aufwand gesteuert werden. Dabei wird lediglich anstelle der Software zur Ansteuerung der Meßelektronik solche verwendet, die die Übermittlung von Steuerkommandos und Statusinformationen zwischen Host und Frontends ausführt.

Dazu wird die bei herkömmlichen Geräten nicht oder nur selten benutzte `control`-Komponente der Device-Datenstruktur verwendet, die zur Beschreibung der Datenerfassungs-Hardware dient (Abschn. 6.3). Bei virtuellen Geräten, die für die jeweiligen Frontends zuständig sind, zeigt sie auf eine Routine, die dafür zu sorgen hat, daß Steueranweisungen, die das Datenaufnahmeprogramm erhalten hat, an die eigentlich ausführenden Frontend-Server weitergeleitet werden. Nach Ausführung der Anweisungen wird das Ergebnis an die aufrufende Software zurückgeliefert, die dann ihrerseits den wartenden Klienten über Erfolg oder Mißerfolg der Anweisung benachrichtigt.

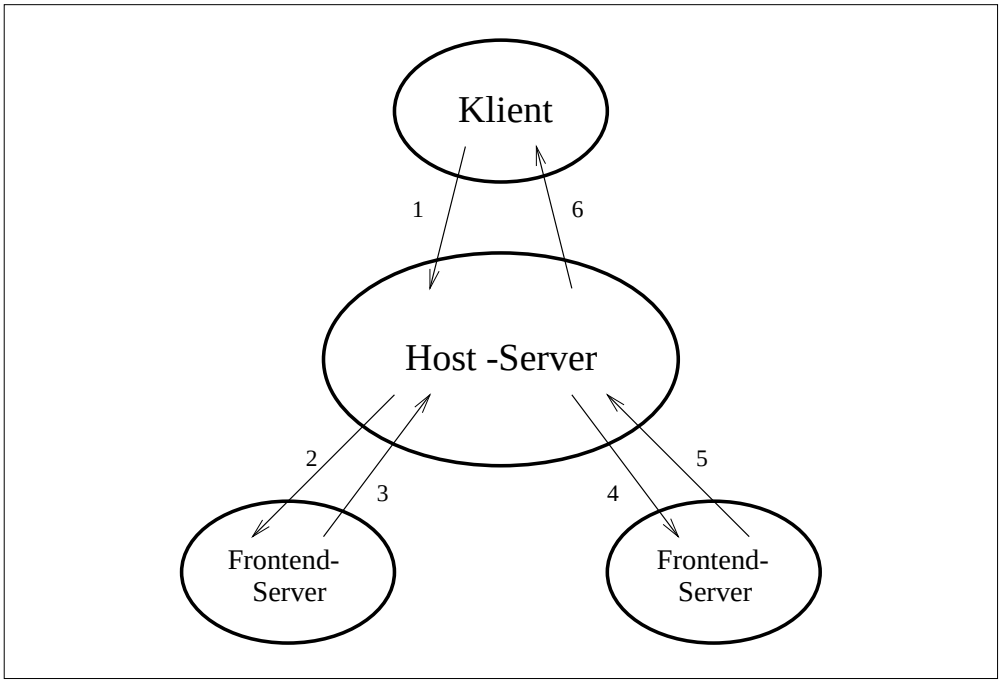

Abb. 7.6: Schematische Darstellung des Nachrichtenaustauschs zur Übermittlung von Steuerungsinformationen in einem einfachen hierarchischen Client-Server-System.

Alle Aktionen erfolgen dabei unter Kontrolle der Zustandsmaschine des übergeordneten Servers, die bestrebt ist, den Zustand der Frontends konsistent mit dem ihren zu halten. Das geschieht dadurch, daß einerseits die mit ihrem aktuellen Zustand nicht verträglichen Steueranweisungen abgeblockt und gar nicht an die Frontends weitergegeben werden; nur erlaubte Steueranweisungen werden weitergeleitet. Andererseits werden die Antworten der Frontends auf die Steueranweisungen überprüft. Ist bei einem Frontend ein Fehler aufgetreten, wird der Steuervorgang abgebrochen und der Versuch unternommen, das System wieder in den Ausgangszustand zu bringen. Entsprechende Fehlermeldungen werden an den Klienten zurückgeschickt.

Aufgrund der Unabhängigkeit der verschiedenen parallel arbeitenden Rechner ist es in einigen Fällen jedoch nicht trivial, hier eine saubere Fehlerbehandlung durchzuführen. Besonders in diesem Bereich machen sich die prinzipiellen Probleme eines verteilten Systems deutlich bemerkbar. Das war unter anderem mit ein Grund, die Rechnerverteilung unter MECDAS streng hierarchisch zu organisieren. Die hierarchische Struktur definiert eine feste und gleichzeitig einfache Beziehung zwischen den einzelnen Komponenten des Systems. Jeder Rechner bzw. die darauf laufenden Programme kennen genau ihre Kommunikationspartner. Der Datenerfassungs-Server auf einem Host hat über die Experimentbeschreibung die eindeutige Information, mit welchen Frontend-Rechnern er es zu tun hat. Die Frontend-Rechner wiederum kommunizieren grundsätzlich nur mit genau einem, dem ihnen unmittelbar übergeordneten Host bzw. dem darauf aktiven Server-Programm, von dem sie Anweisungen erhalten und an den sie die bei der Ausführung erhaltenen Ergebnisse zurückmelden müssen.

Soweit diese Beziehnungen korrekt in der Software umgesetzt sind, können Inkonsistenzen aufgrund von Fehlbedienungen einerseits und Fehlern bei der Datenaufnahme andererseits weitestgehend ausgeschlossen werden, auch wenn ihre Vermeidung bisweilen einen größeren Aufwand verursacht. So gibt es Fälle, in denen es prinzipiell nicht möglich ist, beim fehlerhaften Versuch, einen Zustandsübergang herbeizuführen, wieder unmittelbar in den Ausgangszustand zurückzukehren, da Teile des Systems einen irreversiblen Zustandsübergang bereits durchgeführt haben können, während andere noch daran arbeiten und erst später dabei Fehler feststellen. Dieses prinzipielle Problem hat seine Ursache darin, daß die beteiligten Rechner einzelne Teilaufgaben unabhängig voneinander bearbeiten, und zu ihrer Synchronisierung Kommunikationsaufwand notwendig ist, der sich nicht nur in Form von zusätzlichem Rechenaufwand bemerkbar macht, sondern auch durch das Verstreichen der dazu benötigten Zeit, innerhalb der auf den beteiligten Rechnern wiederum unabhängig voneinander Ereignisse eintreten können, die zu einer Inkonsistenz führen. Problematisch bleiben schließlich auch Effekte aufgrund von Fehlersituationen wie der Absturz von einzelnen Programmen oder kompletten Rechnern oder Störungen in der Kommunikation innerhalb des verteilten Systems.

Während viele Probleme zu einem großen Teil bereits durch die zentrale Zustandmaschine abgehandelt werden, bleiben einige kommunikationsspezifische Aspekte, die nur in Zusammenhang mit konkreten Mechanismen zur Übermittlung der Steuerinformation zu lösen sind. Für einige Verfahren wurde Software implementiert. Sie manifestiert sich in entsprechenden Gerätetreibern für virtuelle Geräte unterschiedlichen Typs.

7.2.4.2 Remote-Shell-Mechanismus

Das erste Verfahren nutzt den Remote-Shell-Mechanismus. Remote-Shell (`rsh`) ist ein Programm, mit dessen Hilfe herkömmliche UNIX-Kommandos unter wesentlicher Beibehaltung von Standard-Ein/Ausgabe-Eigenschaften auf einem entfernten, über TCP/IP ansprechbaren Rechner zur Ausführung gebracht werden können [Rsh]. Mit seiner Hilfe wird das der Steueranweisung entsprechende Kommando als herkömmliches UNIX-Kommando auf dem Frontend ausgeführt. Dabei gemachte Ausgabe oder auftretende Fehler werden an den Aufrufenden zurückgeliefert und können wie bei einem lokalen Kommando ausgewertet und weiterverarbeitet werden.

Diese Variante hat den Vorteil, daß sie sich auf ein Minimum an Implementationsaufwand beschränkt, da sie intensiv von existierender Systemsoftware Gebrauch macht. Während in der Gerätesofteware nur einfache Standard-Bibliotheksfunktionen korrekt aufgerufen werden müssen, ist für den Frontend sogar überhaupt keine Programmierung erforderlich. Hier sorgt das System automatisch für die Bearbeitung entsprechender Anfragen von anderen Rechnern. Das hat auch den Vorzug, daß zum Funktionieren dieses Verfahrens auf den Frontend-Rechnern keine spezielle Software permanent aktiv sein muß. Das erübrigt das Starten dieser Software und vermeidet Fehler durch dessen unkontrollierten Absturz oder auftretender Fehlfunktionen.

Ein Nachteil dieses Verfahrens in extremen Situationen ist sein relativ großer Aufwand bei der Ausführung der notwendigen System- und Benutzer-Software auf dem Frontend. Er ist jedoch im Regelfall nicht weiter von Bedeutung, da Steuerungsaktivitäten im Vergleich zu den regulären Aktivitäten der Datenerfassung sehr selten sind.

Das Verfahren nutzt implizit das verbindungsorientierte Protokoll TCP, das für eine fehlergesicherte Informationsübermittlung und eine gute Diagnosemöglichkeit bei Problemen in der Kommunikation selbst bzw. verursacht durch den Kommunikationspartner sorgt. Die Verwendung dieses Protokolls besitzt nebenbei den Vorzug, daß dieses Verfahren ohne Änderungen sowohl rechnerübergreifend als auch lokal auf ein- und demselben Rechner anwendbar ist.

7.2.4.3 Control-Server

Als Alternative zum rsh-Mechanismus besteht die Möglichkeit, spezielle Server-Programme auf den Frontend-Rechnern einzusetzen, mit denen das übergeordnete Datenerfassungsprogramm oder eines seiner dafür zuständigen Klienten eine je nach Bedarf feste oder lose Kommunikationsverbindung aufbaut und Steuerungsinformation austauscht. Bei Verfügbarkeit von TCP/IP kann dazu wiederum auf das TCP-Protokoll zurückgegriffen werden; ebenso ist die Nutzung des Message-Systems MUPIX möglich. Werden andere Kommunikationsmechanismen verwendet, ist spezielle Software notwendig. Unabhängig davon arbeitet dieses Verfahren nach einem einheitlichen Prinzip: Steueranweisungen der übergeordneten Software werden in Form von speziellen Nachrichten dem Control-Server zugestellt. Dieser gibt sie auf dem Frontend-Rechner unmittelbar nach Erhalt direkt an den lokalen Datenerfassungs-Server weiter. Nach Ausführung der Anweisungen wird das Ergebnis an den Control-Server zurückgegeben, der dann eine entsprechende Antwortnachrichten an den Aufrufer übermittelt.

Dieses Verfahren besitzt trotz oder gerade wegen des damit verbundenen Implementationsaufwands einen deutlichen Vorteil gegenüber der ersten Alternative, denn es erlaubt die Ausführung der Aufgaben mit wesentlich geringerem Aufwand in bezug auf Rechenzeit und Systemressourcen. Andererseits funktioniert es nur dann korrekt, wenn der Control-Server ständig aktiv ist und vor der ersten Benutzung gestartet wird. Zumindest hier muß auf ein Standard-Verfahren wie den rsh-Mechanismus zurückgegriffen werden.

7.2.5 Datentransfer

Im Vergleich zur Steuerung der Datenerfassung kaum weniger wichtig ist die Übermittlung und zentrale Verwaltung der Meßdaten innerhalb des verteilten Systems. Die virtuellen Geräte zur Ansteuerung der für einzelne Teile des Experiments zuständigen Frontends stellen für die restliche Datenerfassungs-Software prinzipiell ebenso Geräte dar wie einzelne CAMAC- oder Fastbus-Module. Es existiert jedoch ein gravierender Unterschied: Die Daten, für die ein virtuelles

Gerät zuständig ist, werden nicht ereignisweise ausgelesen, sondern in Form eines einlaufenden Datenstroms entgegengenommen, der von dem entsprechenden Frontendrechner übermittelt wird.

Um diesen Datentransport möglichst effizient abwickeln zu können, besitzt die Kernsoftware die Möglichkeit, neben den bereits in Abschnitt 5.2.4 beschriebenen Datenkanälen zum Abtransport von Meßdaten auch Datenkanäle zur Entgegennahme von Daten einzurichten. Die Mechanismen zur Datenübertragung sind dabei identisch. Sie bestehen aus dem Austausch von Nachrichten über die Client-Server-Schnittstelle und nutzen die Übertragung der Daten über Shared Memory. Die Aufgabe der Geräte-Software beschränkt sich nur noch auf das Aufsetzen der Kommunikationsverbindung für jede Einzelmessung — die eigentliche Kommunikation läuft später selbsttätig ab.

Grundlage dafür bildet in der Regel ein Paar aus Prozessen, die auf den beteiligten Rechnern zu Beginn jeder Messung gestartet werden und in der Lage sind, die Daten auf geeignete Weise vom Frontend zum Hostrechner zu transportieren. Beide stellen jeweils einen Klienten für den ihm zugeordneten Datenerfassungs-Server dar. Im Frontendbereich arbeitet der entsprechende Klient als Archivierungsprogramm, das die Daten nicht lokal abspeichert, sondern über den gewählten Kommunikationsmechanismus seinem Partner auf dem Hostrechner übermittelt. Dieser nimmt die Daten in Empfang und leitet sie an den übergeordneten Datenerfassungs-Server weiter (Abb. 7.7).

Die Verwendung externer Kommunikationsprozesse anstelle der unmittelbaren Kommunikation durch das Datenerfassungsprogramm selbst hat mehrere Gründe:

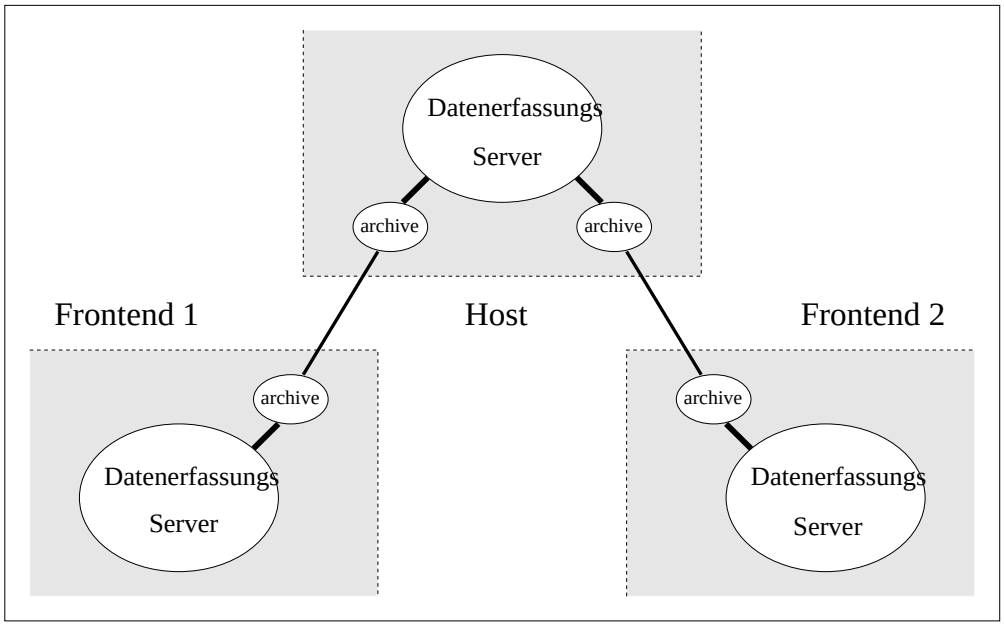

Abb. 7.7: Hierarchisches Client-Server-System mit Kommunikationsprozessen.

1. Der Datenerfassungs-Server besitzt bereits ein einheitliches Kommunikationsinterface für ähnliche Aufgaben, das eine größere Anzahl von leistungsfähigen Kommunikationskanälen gleichzeitig zur Verfügung stellt. Diese Funktionalität ist unabhängig davon, ob der Server auf einem Rechner mit oder ohne Betriebssystem läuft. Im Frontendbereich existiert zudem bereits die hier verwendete Archivierungsschnittstelle.

2. Direkter I/O innerhalb des Servers könnte zu dessen zeitweiser Blockierung führen. Damit könnte der Server seine anderen Aufgaben, insbesondere im Rahmen der Experimentsteuerung, nicht mehr zu jedem Zeitpunkt ausführen. Für Standard-Ein/Ausgabe-Methoden gibt es zwar Möglichkeiten, das zu vermeiden, doch schränkt das die I/O-Möglichkeiten ein und macht den I/O schlechter handhabbar.

3. Mehrere Prozesse sind wesentlich besser in der Lage, gleichzeitigen I/O über mehrere Datenkanäle durchzuführen, ohne daß es zu einer unnötigen Synchronisierung und damit Sequentialisierung der Ein/Ausgabe kommt.

4. Insbesondere vom Standard abweichende Ein/Ausgabe, wie z.B. die Kommunikation über optische Verbindungen, läßt sich in einem gesonderten Programm individuell gestalten und an Problemstellung und Hardware gut anpassen. Dabei sind wesentlich weniger Randbedingungen zu berücksichtigen als innerhalb eines Programmes mit der Vielfalt an Aufgaben und der dadurch bedingten Komplexität des Servers.

5. Bei Änderung des Kommunikationsmechanismus ist lediglich das Programmpaar auszutauschen. Der Rest der Software bleibt davon unbeeinflußt. Software-Entwicklung und Debugging gestalten sich in der Regel einfacher.

Der Typ des virtuellen Gerätes definiert das Kommunikationsverfahren und

- welche Programme zur Datenübertragung
- auf welchen der beteiligten Rechnern
- auf welche Art und Weise

zu starten sind. Er muß je nach Experimentkonfiguration und zwischen Host und Frontend verfügbaren Kommunikationsmöglichkeiten gewählt werden.

In der Regel wird hier von der Möglichkeit Gebrauch gemacht, eine verbindungsorientierte Kommunikation zwischen den beiden Prozessen zu benutzen. Sie sorgt neben der Fehlersicherung für eine unmittelbare Reaktion auf Verbindungsabbrüche. Damit ist die Konsistenz des Zustands der Kommunikationspartner gewährleistet. Bisher erwies es sich als ausreichend, hierfür herkömmliche TCP-Verbindungen über Ethernet zu verwenden. Bei Bedarf können aber auch andere Kommunikationsmechanismen, die als Medium z.B. Lichtleiter verwenden, eingesetzt werden.

7.3 Eventbuilding

7.3.1 Problemstellung

Der Einsatz mehrerer Frontend-Rechner erlaubt eine sehr gute Parallelisierung der Datenaufnahme, indem verschiedenen Teilen des Detektorsystems jeweils eigene Rechner zugeordnet werden, die unabhängig voneinander für die ereignisweise Bearbeitung der Datenaufnahmeaufgaben dieser Teilsysteme zu sorgen haben. Damit läßt sich eine Leistungssteigerung des Systems erreichen, die sich in Form niedriger Totzeiten bzw. in einer Steigerung der maximal möglichen Ereignisraten niederschlägt — im Idealfall um einen Faktor, der der Anzahl der eingesetzten und unabhängig arbeitenden Datenerfassungsrechner entspricht. Grundvoraussetzung dabei ist, daß die einzelnen Datenaufnahmerechner wirklich unabhängig arbeiten können und zur Abarbeitung ihrer Aufgaben annähernd die gleiche Zeit benötigen. Es sollte daher angestrebt werden, die Aufgaben möglichst gleichmäßig zu verteilen. Das läßt sich in gewissem Maße durch die Zuordnung der Frontend-Rechner zu jeweils ähnlich komplexen, eigenständigen Untereinheiten erreichen. Im Falle der Drei-Spektrometeranlage boten sich hierfür die einzelnen Spektrometer sehr gut an.

Durch die Verteilung der Datenaufnahme auf mehrere unabhängige Rechner werden aber zusätzliche Maßnahmen notwendig, denn jeder Frontend nimmt immer nur einen Teil der Information auf, die das jeweilige physikalische Ereignis charakterisiert. Zu seiner vollständigen Beschreibung sind die Informationen aller Frontend-Systeme notwendig. Sie müssen daher zu einem geeigneten Zeitpunkt wieder zusammengeführt werden. Wenn dieser Vorgang — üblicherweise mit dem Begriff „Eventbuilding“ bezeichnet — jeweils zum Zeitpunkt der Aufnahme eines Ereignisses stattfindet, ist er mit einem erheblichen Zusatzaufwand verbunden, denn er erfordert nicht nur zusätzlichen Rechenaufwand, sondern macht außerdem noch eine Kommunikation der Frontends untereinander bzw. mit einem übergeordneten Rechner notwendig. In dieser Phase ist Parallelisierung nur noch sehr begrenzt möglich. Dem Leistungsgewinn durch die verteilte Datenaufnahme steht damit umso drastischer der Mehraufwand durch das Eventbuilding gegenüber.

Bei den in Frage kommenden Experimenten ist es daher nur dann sinnvoll, das Eventbuilding bereits synchron zur Datenaufnahme zu machen, wenn die dafür nötige Zeit klein gegenüber der eigentlichen Auslesezeit ist. In allen anderen Fällen würde der Parallelisierungseffekt wieder verloren gehen und im ungünstigsten Fall sogar zu einer Vergrößerung der Totzeit führen. Es besteht jedoch auch die Möglichkeit, das Eventbuilding asynchron zur Aufnahme eines Ereignisses abzuwickeln, d. h., die Daten nach ihrer Erfassung erst einmal zu puffern, um sie dann in einem zweiten Schritt zusammenzufügen. Durch geeignete Programmierung lassen sich so die Aktivitäten, die zur Rechnertotzeit beitragen im wesentlichen wieder auf die Auslese beschränken. Dabei ist es jedoch notwendig, zusätzliche Synchronisierungsinformation aufzunehmen und mit den Meßdaten abzuspeichern, die der Eventbuilding-Software das spätere Identifizieren und eindeutige Zuordnen der Teilinformationen erlaubt. Die zeitliche Abfolge

der Ereignisdaten, die in den von den Frontends erzeugten Datenströmen erhalten bleibt, reicht dafür nicht aus. In der Realität muß man immer damit rechnen, daß die unabhängig erzeugten Datenströme durch fehlende oder zusätzlich erzeugte Daten asynchron werden können und nicht mehr korrekt den Ereignissen zugeordnet werden können. Solche Effekte können erfahrungsgemäß aufgrund defekter Hardware, fehlerhaften Timings in der Triggerelektronik oder auch nur herkömmlicher, elektrischer Vörgänge auf den Leitungen, wie z. B. Reflexionen an Störstellen, auftreten. Auch wenn sie noch so selten vorkämen, bestünde dennoch eine endliche Wahrscheinlichkeit dafür. Ihr Auftreten würde beim Eventbuilding zu einer nicht unmittelbar erkennbaren, permanent fehlerhaften Ereignisrekonstruktion führen. Ohne Synchronisierungsinformation könnte keine Aussage darüber gemacht werden, ob und wo so ein Fehler passiert ist, und damit grundsätzlich über die Konsistenz der Daten.

Mithilfe entsprechender Zusatzinformation ist es jedoch möglich, die erfaßten Ereignisse eindeutig zu kennzeichnen. Diese Information stellt dann die Grundlage für das Eventbuilding dar, das on-line asynchron zur Datenaufnahme oder sogar off-line vorgenommen werden kann. Für die konkrete Wahl des Verfahrens ist der Aufwand des Eventbuildings mitentscheidend, denn wenn er größer als der Ausleseaufwand ist, läßt er sich on-line nicht ohne nachteilige Beeinflussung der Totzeit abwicklen. Umgekehrt hat das Verfahren Rückwirkungen auf die Datenerfassungsmethode, die benötigte Hardware und die erforderlichen Rechner.

Um die Vorzüge von MAMI als Elektronenbeschleuniger mit bis zu 100 μA Strahlstrom im Dauerstrichbetrieb nicht unnötig einzuschränken, war es zur Totzeitminimierung notwendig, die ereignissynchronen Aktivitäten auf ein Mindestmaß zu beschränken. Gleichzeitig sollte das Eventbuilding so frühzeitig wie möglich gemacht werden, um unmittelbar während der Messung bereits Inkonsistenzen der Daten und darauf zurückzuführende Synchronisierungsprobleme bei der Datenerfassung erkennen und beseitigen zu können. Damit einhergehend lassen sich so Fehler bei der späteren Ereignisrekonstruktion im Rahmen der Datenanalyse auf ein Minimum reduzieren, insbesondere wenn alle Daten einer Einzelmessung konsequent in einer sich selbstbeschreibenden Datei abgespeichert sind, für deren Auswertung weder Zusatzinformation noch ein fehleranfälliges Hantieren mit mehreren Informationsquellen notwendig wird.

Ein zur Datenaufnahme asynchrones Eventbuilding hat zusätzlich den Vorteil, daß es weitergehende Möglichkeiten der Konsistenzüberprüfung bzw. der Untersuchung und Beseitigung von aufgetretenen Inkonsistenzen in den Daten bietet. Denn außer den Daten eines Ereignisses selbst können auch die Daten aus der unmittelbaren Umgebung, also die vorhergehenden und nachfolgenden Ereignisse, untersucht und gegeneinander geprüft werden. Auch für die Datenerfassung selbst hat das den Vorteil, auf triviale Art und Weise Deadlocks bei Asynchronitäten der einzelnen Detektorsysteme zu vermeiden. Gleichzeitig ist es jedoch für die Eventbuilder-Software nicht mehr möglich, beim Auftreten von Inkonsistenzen unmittelbar in den Meßablauf einzugreifen und bei Bedarf zusätzliche Information zur Klärung der Situation anzufordern.

Das Konzept von MECDAS sieht dementsprechend vor, die Datenerfassung

für ein komplexes Experiment durch mehrere Frontend-Rechner vornehmen zu lassen, die neben der eigentlichen Meßdaten auch Synchronisationsinformation aufnehmen. Die Daten werden zwischengespeichert und einem übergeordneten Rechner übermittelt, der der Eventbuilding übernimmt und für die endgültige Archivierung der Daten sorgt. Die zentrale Steuerung des Experiments wird in der Regel ebenfalls von diesem Rechner übernommen.

7.3.2 Grundprinzip

Das durch die Verteilung der Datenerfassung auf mehrere, unabhängig arbeitende Rechner notwendige Eventbuilding in MECDAS basiert auf zwei Elementen:

1. eine zentrale Hardwarekomponente zur eindeutigen Kennzeichnung der zu einem registrierten Ereignis gehörenden Daten für alle Teilsysteme der Experimentieranordnung

2. ein Eventbuilder-Programm, das die von den Frontends kommenden Datenströme aufgrund der eindeutigen Kennzeichnung ereignisweise zusammenfaßt

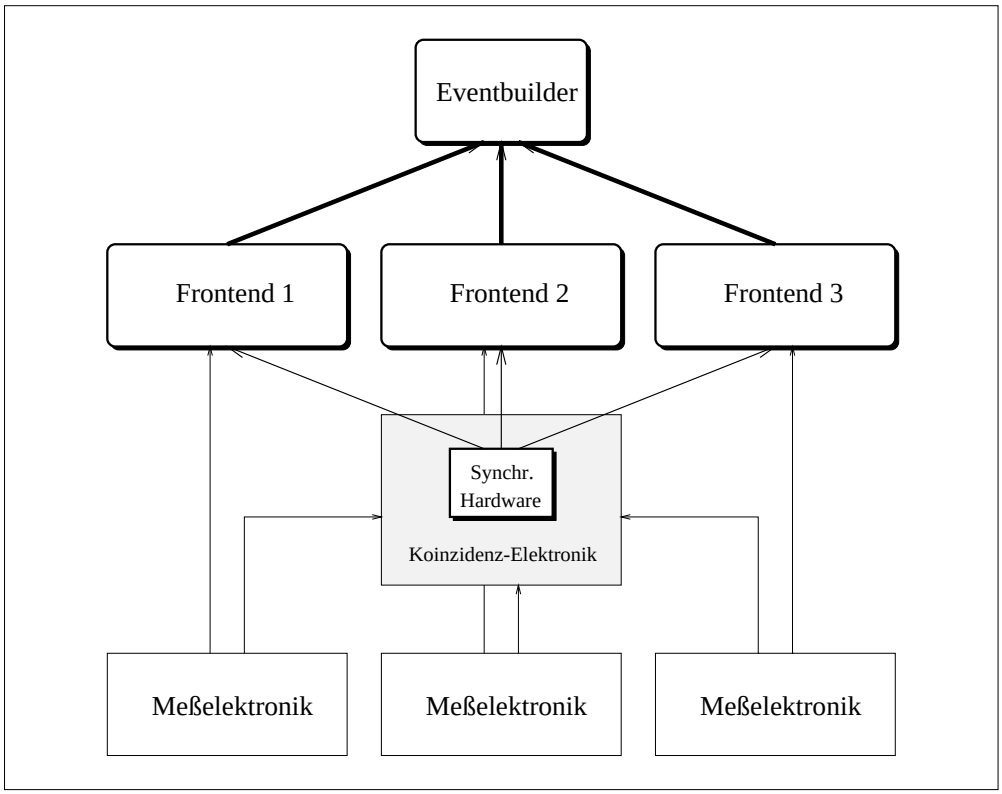

Abb. 7.8: Grundprinzip des Eventbuilding in MECDAS.

7.3.3 Hardware

Für eine eindeutige Ereignisidentifizierung muß schon zum Zeitpunkt der Triggerentscheidung entsprechende Information generiert werden. Hardware, die aufgrund der hohen Geschwindigkeitsanforderungen dazu notwendig ist, wird in aller Regel Bestandteil der Triggerelektronik eines Experiments sein. Für ein besseres Verständnis der Problematik soll daher kurz die Funktionsweise dieser Hardware innerhalb der Triggerelektronik am Beispiel der Drei-Spektrometer-Anlage beschrieben werden.

Die Triggerelektronik der Gesamtanlage besteht aus individueller Triggerelektronik der einzelnen Spektrometer, die jeweils gleichartig aufgebaut ist, und einer zentralen Koinzidenzelektronik [Rich95]. Für jedes Spektrometer wird als Triggerbedingung im allgemeinen das Ansprechen von mindestens einem Segment der beiden Szintillationsdetektorebenen (dE und ToF) gefordert. Je nach zu untersuchender Teilchenart kann auch ein Vetosignal der Cherenkov-Detektoren oder dessen Nichtvorhandensein berücksichtigt werden.

Die Identifikationssignale der einzelnen Spektrometern werden einer zentralen Logik (Koinzidenz-PLU) zugeführt, die die Möglichkeit bietet, für jede Kombination der acht Eingangssignale beliebige Kombinationen von acht Ausgangssignalen zu generieren. Damit ist es möglich, beliebige Koinzidenz/Antikoinzidenz-Bedingungen zu realisieren, die durch einfaches Umprogrammieren leicht geändert werden können. Ebenfalls berücksichtigt werden hierbei frei mitlaufende Einarmereignisse, die mithilfe eines programmierbaren Untersetzers in jedem gewünschten Verhältnis den Koinzidenzereignissen begemischt werden können.

Das von der zentralen Koinzidenzlogik erzeugte Triggermuster dient der Kennzeichnung und Nummerierung der Ereignisse für das spätere Zusammenführen der zunächst getrennt erfaßten Daten. Diese Aufgabe wird von einem speziellen CAMAC-Modul übernommen, das eigens für die Drei-Spektrometer-Anlage konzipiert und im Hause gebaut wurde [Geig93c]. Alle Ereignisse, bei denen die Koinzidenz-PLU für mindestens ein Spektrometer ein Auslesesignal generiert hat, werden mit einer fortlaufenden Nummer gekennzeichnet. Sie wird allen drei Spektrometern mittels eines Flachbandkabels als 13-Bit-Information zugeführt und enthält darüberhinaus in den höchstwertigen Bits die Information über die Art des jeweiligen Ereignisses. Diese Information wird von jedem Frontend bei der Datenaufnahme individuell ausgelesen und zur eindeutigen Kennzeichnung des aktuellen Ereignisses verwendet.

7.3.4 Software

7.3.4.1 Maßnahmen bei Datenaufnahme

Die im Rahmen des Eventbuilding notwendige Software besteht aus zwei Teilen. Der erste Teil dient zur eindeutigen Kennzeichnung der Daten im Rahmen der Datenaufnahme. Dazu wird die von der entprechenden Hardware zur Verfügung gestellte Identifikationsinformation zum einen wie jedes andere Meßdatum

```
Data *
rdtMain(Data *buf)
{
        short   eventNumber, eventType;

        ...
        pread(syncinfo);
        eventNumber = a.x.y.z & 0x1fff;
        eventType = (a.x.y.z >> 13) & 0x7;
        pack(a, &a, buf, eventNumber, eventType);

        return (buf);
}
```

Abb. 7.9: Beispiel für den Programmteil, der bei der Datenaufnahme das Auslesen der Synchronisationsinformation und entsprechendes Kennzeichnen der Daten macht

ausgelesen und abgespeichert, aber auch als globale, nicht experimentspezifische Ereignisinformation festgehalten, um bei der Formatierung der Daten verwendet zu werden. Dabei wird Ereignistyp und -nummer in die dafür vorgesehenen Felder des Event-Datensegmentkopfes `code` und `number` (s. Abschnitt 6.8.3.2) eingetragen. Damit werden die Ereignisdaten unabhängig von einer speziellen Experimentkonfiguration einheitlich gekennzeichnet. Beim späteren Eventbuilding ist somit die Identifizierung der Daten möglich, ohne deren Struktur zu kennen und sie selbst zu untersuchen.

Sowohl die Aufnahme der Ereignisidentifikation als auch ihr Eintragen in den Datensegmentkopf sind Bestandteil der bereits beschriebenen Standard-Software. Das Einlesen dieser Information erfolgt so wie die Aufnahme aller anderen Meßdaten; für die Eintragungen in den Datensegmentkopf sorgt eine spezielle Routine, die die experimentspezifische Ausleseanweisung üblicherweise abschließt (s. Abb. 7.9).

7.3.4.2 Der Eventbuilder

Speziell für das Eventbuilding implementiert ist der zweite Teil. Er besteht im wesentlichen aus einem eigenständigen Programm, das in der Lage ist, die Frontend-Datenströme ereignisweise zusammenzufügen. Es stellt für die Datenerfassungs-Software ein virtuelles Gerät dar, das analog zu den Frontends in der übergeordneten Experimentbeschreibung spezifiziert sein muß. Das zuständige Datenaufnahmeprogramm aktiviert mit jedem Start einer Einzelmessung nicht nur Software zur Datenübertragung von den Frontends sondern auch das Eventbuilder-Programm. Dieses arbeitet als Klient des Datenaufnahmeprogramms und kommuniziert mit ihm über dessen Client-Server-Schnittstelle. Auf diesem Weg erhält das Eventbuilder-Programm alle zu den Experiment gehörigen Meßdaten und gibt sie — entsprechend neu formatiert — wieder an den Server zurück, der sie schließlich zur Archivierung oder On-line-Analyse zur Verfügung stellt.

Bei der Implementierung des Eventbuilders wurde besonders auf hohe Effizienz geachtet, um Beinträchtigungen der Datenaufnahme zu vermeiden. Dabei wurde von der hierarchischen Organisation des Datenformats Gebrauch gemacht, in dem die Daten vor bzw. nach dem Eventbuilding verpackt sind. Da die Experimentkonfiguration eines Frontends stets Teilbaum der Gesamtkonfiguration ist, setzen sich auch die formatierten Daten der Gesamtkonfiguration für ein Ereignis unmittelbar aus den bereits durch die Frontends verpackten Ereignisdaten zusammen. Eine aufwendige und vollständige Dekodierung und Neuformatierung der Einzeldaten ist nicht notwendig. Die Frontend-Datenströme werden lediglich in die einzelnen Datensegmente aufgeteilt. Die Datensegmente, die die zu einem Ereignis gehörenden Meßdaten enthalten, werden zum Eventbuilding herangezogen. Dazu wird ausschließlich die Information aus dem Datensegmentkopf benutzt. Sie enthält Ereignistyp und -nummer, mit deren Hilfe eine eindeutige Zuordnung korrespondierender Datensegmente aus den einzelnen Armen möglich ist. Die Software erkennt anhand des Ereignistyps, ob es sich um Daten von Einzel- oder Koinzidenzereignissen handelt, und, von welchen Armen es registriert wurde. Sie trägt damit der Möglichkeit Rechnung, daß bei der Auslese nicht nur Koinzidenzereignisse, sondern auch Einzelereignisse bzw. Koinzidenzen zwischen einzelnen Teilen der Apparatur aufgenommen werden können.

Die Ereignisnummer entscheidet über die Reihenfolge der Bearbeitung von Einzel- und Koinzidenzereignissen und identifiziert zusammengehörige Datensegmente aus den einzelnen Datenströmen bei Koinzidenzereignissen. Dabei wird ausgenuzt, daß die Ereignisnummer die zeitliche Abfolge der Ereignisse kennzeichnet, da sie während des Experiments als fortlaufende, streng monoton steigende Nummer generiert wurde. Daneben sorgt die Software in allen Bereichen von der Datenaufnahme über den Datentransport bis zur weiteren Verarbeitung dafür, daß die zeitliche Abfolge, mit der die Daten erfaßt wurden, stets erhalten bleibt. Damit definieren, auch bei starker Mischung von Einzel- und Koinzidenzdaten, jeweils die ersten, noch nicht abgearbeiteten Datensegmente der Datenströme zwangsläufig die Zusammensetzung des aktuellen Ereignisses. Es wird bestimmt durch das oder die Datensegmente mit der kleinsten Ereignisnummer. Zusätzlich wird die in diesem Fall redundante Information des Ereignistyps ausgewertet und mit der tatsächlichen Zusammensetzung der vorliegenden Daten verglichen. Damit ist es möglich, Inkonsistenzen aufgrund von Synchronisationsfehlern in der Hardware, Fehlern in der Datenaufnahme oder Übertragungsfehler zu erkennen. Redundanzen werden auch dazu verwendet, um nach Auftreten eines solchen Fehlers die verlorengegangene Synchronisierung soweit möglich wiederherzustellen und die korrekte Zuordnung der nächsten Ereignisdaten wieder zu gewährleisten.

Bei erfolgreicher Identifikation eines Ereignisses werden die dazugehörigen Daten zusammengepackt. Dazu werden die Datenblöcke der infrage kommenden Datensegmente mit zusätzlicher Adreß- und Längeninformation versehen und durch einfaches Kopieren zu einem neuen Datensegment zusammengefaßt. Die Adreßinformation wird für jedes Datensegment durch den zugehörigen Frontend-Datenstrom bestimmt, und stellt die relative Adresse des entsprechenden Teilbaums innerhalb der Gesamtkonfiguration dar. Auf diese Weise wird ein Da-

tensegment erzeugt, das in konsistenter Form entsprechend der Gesamtkonfiguration alle aufgenommenen Daten enthält, die ein Ereignis vollständig beschreiben. Die so erzeugten Daten sind von ihrer Struktur her vollkommen identisch denen, die bei unmittelbarer Auslese des kompletten Experiments durch ein einzigen Frontend-Rechner erhalten worden wären.

Die beschriebenen Vorgänge werden Ereignis für Ereignis wiederholt. Die dabei generierten Datensegmente bilden schließlich den Ausgangsdatenstrom für das gesamte Experiment. Datensegmente, die keine Ereignisinformation enthalten, sondern z. B. die Experimentbeschreibung, Statusinformation usw., werden im wesentlichen transparent weitergeleitet.

Der Ausgangsdatenstrom wird ergänzt durch die Beschreibungsinformation für das Gesamtexperiment, die die bereits aus den Frontend-Datenströmen vorhandene Teilexperimentbeschreibungen vervollständigt. Zur Vermeidung von Inkonsistenzen enthält die Beschreibung für das Gesamtexperiment keine Detailinformation über die Frontends, sondern gibt nur die grobe Struktur wieder und verweist ansonsten auf die Teilbeschreibungen. Somit ist in dem generierten Datenstrom wieder alle Information enthalten, um die darin enthaltenen Daten selbstbeschreibend zu machen.

Das Eventbuilder-Programm konnte aufgrund des hierarchisch orientierten Datenformats sehr allgemein gehalten werden. Es benötigt keine Kenntnisse über die Eingangsdaten und nur minimale Information über die Struktur der Gesamtkonfiguration. Diese erhält es bei seinem Start durch das übergeordnete Datenaufnahmeprogramm in Form von Namen und relativen Adressen der Konfigurationsteilbäume mitgeteilt, deren Daten zusammengefaßt werden sollen.

Das Eventbuilder-Programm wurde aus mehreren Gründen als ein separates Programm realisiert und nicht zu einem unmittelbaren Bestandteil des Datenaufnahmeprogrammes gemacht. Einerseits sollten, ähnlich wie schon beim Datenaustausch mit den Frontends, Blockierungen des Datenerfassungs-Servers vermieden werden, andererseits aber auch eine längerfristige Belegung von Systemressourcen durch die Eventbuilder-Software reduziert werden. Bei sehr unterschiedlichen Datenraten auf den einzelnen Armen des Experiments ist es für eine korrekte Ausführung des Eventbuilding nämlich notwendig, größere Arbeitsspeicherbereiche zu allokieren, die jedoch erst nach Beenden des entsprechenden Programmes wieder an das System zurückgegeben werden können. Die dadurch bedingte Blockierung von Systemspeicher über einen gewissen Zeitraum ist aber bei einem Programm, das nur für eine Einzelmessung aktiv ist, deutlich unkritischer, als bei dem über eine ganze Meßreihe, also in der Realität über mehrere Tage, laufenden zentralen Datenaufnahme-Server.

Nicht zuletzt waren für die Aufteilung der Software wie an anderer Stelle auch softwaretechnische Aspekte wie die Erzielung einer möglichst guten Modularität mitentscheidend, denn ein separates, auf eine bestimmte Funktionalität beschränktes Programm läßt sich nicht nur einfacher entwickeln und warten, sondern es bietet auch die Möglichkeit, es bei Bedarf sehr einfach durch ein anderes zu ersetzen.

7.4 Datenarchivierung

7.4.1 Problemstellung

Zur permanenten Archivierung der Meßdaten werden üblicherweise zwei Methoden eingesetzt, die jedoch alle mehr oder weniger ausgeprägte Schwächen zeigen. Die Archivierung auf eine **Plattendatei** stellt zwar eine sehr schnelle und flexible Form der Archivierung dar und bietet auch für den späteren Zugriff große Vorteile, doch hat sie — auch bei den inzwischen verfügbaren großen Plattenkapazitäten — den prinzipiellen Nachteil, daß der Speicherplatz beschränkt ist und zur permanenten Aufbewahrung mit unverhältnismäßig hohen Kosten verbunden wäre. Disketten, die hier nur der Vollständigkeit halber erwähnt werden sollen, kommen wegen ihrer niedrigen Geschwindigkeit und geringen Kapazität überhaupt nicht in Frage; in Zukunft könnten evtl. optische Speichermedien eine Rolle spielen, doch auch hier waren die Kosten bislang zu hoch.

Eine sehr kostengünstige Alternative ist die Verwendung von **Magnetbändern**, insbesondere seitdem sie als kompakte, einfach zu handhabende Kassetten mit Kapazitäten weit im Gigabyte-Bereich verfügbar sind. Ihr besonderer Vorteil gegenüber herkömmlichen Festplatten ist die Trennung zwischen Laufwerk und Medium, was prinzipiell eine unbeschränkte Speicherkapazität ermöglicht. Gleichzeitig ist damit aber auch ein gravierender Nachteil verbunden. Denn gereade die Auswechselbarkeit des Mediums erfordert eine höhere mechanische Komplexität. Diese Tatsache und das Funktionsprinzip von Magnetbändern führt grundsätzlich zu einer größeren Fehleranfälligkeit infolge von Verschmutzungen, Dejustierungen der Mechanik oder Defekten am Medium. Ein weiterer Nachteil von Magnetbändern ist je nach Aufzeichnungsverfahren und verwendeter Mechnanik ihre im Vergleich zu Platten deutlich niedrigere Geschwindigkeit beim Schreiben oder Lesen der Daten und noch viel ausgeprägter bei Positionieren des Mediums. Während die Platte wahlfreie Zugriffe erlaubt, stellt das Magnetband ein typisches sequentielles Medium dar, das daher auch nicht durch mehrere Programme zur gleichen Zeit benutzt werden kann. Ebenfalls unangenehm ist im Gegensatz zur Platte die in der Regel ungenügende Information über die freie Speicherkapazität eines Bandes.

Die Vor- und Nachteile der Verfahren zeigen, daß keines der beiden die optimale Archivierungsmethode darstellt. Durch ihre Kombination ist es jedoch möglich, ihre Vorteile zu nutzen, ohne groß von den Nachteilen beeinträchtigt zu werden.

7.4.2 Spooling

Das in MECDAS bevorzugte Verfahren ist eine Kombination von Platten- und Bandarchivierung. Es nutzt die von anderen Anwendungen bekannten Vorteile eines Spooling-Systems: die Daten werden nicht vom Anwenderprogramm selbst auf ein Ausgabegerät geschrieben, sondern erst auf eine Plattendatei, von der nach Abschluß der Schreibaktion ein spezielles Programm den Transfer zum Aus-

gabegerät asynchron vornimmt. Im allgemeinen ist auch der umgekehrte Vorgang denkbar, bei dem die Daten, die ein Gerät liefert durch spezielle Software in einer Plattendatei gesammelt werden, aus der dann später ein Anwenderprogramm die Daten auf herkömmliche Art und Weise lesen kann.

Für die Datenerfassung ist vorerst einmal nur das Schreiben von Daten über ein solches Spooling-System von Interesse, das ein Magnetbandgerät als Ausgabegerät bedient. Ein spezielles Programm sorgt dafür, daß die unmittelbar erfaßten Daten zuerst als herkömmliche Dateien in einem bestimmten Verzeichnis abgespeichert werden. Nach Beendigung jeder Einzelmessung, mit der auch die entsprechende Datei abgeschlossen wird, wird die Spooling-Software beauftragt, für den Transfer dieser Datei auf das Magnetband zu sorgen. Dazu wird die Datei in eine Queue eingehängt, die von der Spooling-Software verwaltet wird. Sie sorgt dafür, daß die in der Queue vorliegenden Dateien sukzessive auf das Magnetband geschrieben werden.

Die Asynchronität dieses Verfahrens bietet eine ganze Reihe von Vorzügen. Sie realisiert eine Entkopplung zwischen der Datenerfassung und dem endgültigen Abspeicherungsvorgang, die sich in beiden Bereichen vorteilhaft bemerkbar macht. Einerseits erlaubt die Trennung, die Zugriffe auf das Magnetband zu optimieren und besser auf spezielle Eigenschaften der Hardware abzustimmen. Damit kann nicht nur eine bestmögliche Ausnutzung der zur Verfügung stehenden Speicherkapazität erreicht werden, sondern insbesondere auch bei einer im Vergleich zur Platte langsameren Speichermedium die höchstmögliche Geschwindigkeit ausgenutzt werden, damit die Archivierung für die Datenerfassung nicht zum Engpaß wird. Andererseits ist mit der Asynchronität eine Pufferwirkung verbunden, die Geschwindigkeitsschwankungen ausgleicht. Die Daten können kurzzeitig mit deutlich höherer Geschwindigkeit angeliefert werden, als vom Magnetbandgerät normalerweise erlaubt, solange sie im Mittel die Geschwindigkeit des Bandgeräts nicht übersteigt. Auch Meßpausen können dann sinnvoll genutzt werden. Sollten dennoch höhere Raten notwendig werden, bietet dieses Verfahren aber auch die besten Voraussetzungen, um die Archivierung durch Einsatz mehrerer Bandgeräte zu parallelisieren. Eine triviale Möglichkeit bietet sich dabei durch die Verteilung unterschiedlicher Dateien auf verschiede Bandgeräte. Die Datenerfassungs-Software selbst muß sich nicht darum oder andere Details kümmern. Vielmehr ist dafür spezielle Software innerhalb des Spooling-Systems zuständig. Nur diese besitzt Zugriff auf das Magnetband und übt darüber die volle Kontrolle aus. Kollisionen durch unkoordinierten Zugriff von herkömmlichen Anwenderprogrammen werden damit vermieden, Bedienungsfehler reduziert und so die Betriebssicherheit des Archivierungssystems erhöht. Nicht zuletzt bietet das zweistufige Verfahren wesentlich bessere Möglichkeiten, auf Fehler beim Mangentband-I/O zu reagieren. Während die Platte eine sehr hohe Fehlersicherheit aufweist, kann es beim Magnetband aufgrund mechanischer Effekte, Mediumfehlern, Dejustierungen oder Verschmutztungen der Schreib/Leseköpfe häufiger zu Fehlern kommen, die im Normalfall zum Datenverlust führen würden. Da jedoch die Daten beim Spooling-Verfahren auf Platte zwischengespeichert sind, kann bei Band-I/O-Fehlern der Archivierungsvorgang leicht wiederholt werden. Zur Erhöhung der Datensicherheit ist sogar

eine Verifizierung möglich: Die Daten bleiben solange auf Platte zwischengespeichert, bis durch explizites Zurücklesen und Vergleichen mit der Kopie auf Band die korrekte Abspeicherung sichergestellt ist.

Das Spooling-Verfahren bietet eine wesentlich bessere Kontrolle über den noch zur Archivierung verfügbaren Speicherplatz. Einerseits läßt sich dieser für den Spooling-Bereich auf der Platte mit Standard-Hilfsmitteln des Betriebssystems sehr einfach bestimmen, andererseits kann das Band ohne Rücksicht auf dessen Füllungsgrad beschrieben werden. Ähnlich wie bei herkömmlichen Fehlern ist das Erreichen des Bandendes nicht mit Datenverlust verbunden. Da die Daten auf Platte zwischengespeichert sind, kann die Archivierung der Datei, die nicht mehr vollständig auf ein bereits gefülltes Band paßte, ohne Aufwand mit einem neuen Band wiederholt werden. Die fehleranfällige Fortsetzung dieser Datei auf dem neuen Band ist nicht notwendig. Bei drohendem Überlauf der Platte läßt sich, wenn das nicht bereits automatisch geschieht, durch Löschen oder Auslagern der ältesten Dateien ohne Beeinträchtigung einer laufenden Messung wieder Platz schaffen. Sowohl im Fehlerfall als auch bei Erreichen des Bandendes kann das notwendige Wechseln des Mediums ohne Beeinflussung der Messung vorgenommen werden. Schließlich hat das Verfahren noch den Vorzug, daß die auf der Platte abgespeicherten Daten noch während des Archivierungsvorgangs bereits durch Auswerte-Software zur Off-line-Analyse verwendet werden können.

7.4.3 Realisierung

7.4.3.1 Das Programm `tape`

Als eine erste, einfache Variante eines Spooling-Systems zur Datenarchivierung stand das Programm `tape` in Form einer Kommandoprozedur zur Verfügung, das jedoch keinen unmittelbaren Bestandteil der MECDAS-Software bildet [Dist93c]. Es nutzt die Möglichkeit einer halbautomatischen Datenarchivierung. Mit seiner Hilfe kann der Benutzer Meßdateien, die durch die MECDAS-Archivierungs-Software in ein spezielles Spooling-Verzeichnis abgelegt wurden, geordnet auf Magnetband abgespeichert, ohne sich dabei um nähere Details kümmern zu müssen. Es stellt Hilfsmittel, für eine einfache Verwaltung der Archivierung zur Verfügung. Dazu gehört das kontrollierte Löschen, Komprimieren oder Verlagern von bereits archivierten Meßdateien und eine vereinfachte Verifizierung des Bandinhalts. Alle Aktivitäten laufen dabei mit strenger Benutzerinteraktion ab. Das ermöglicht zwar eine individuelle Kontrolle aller Vorgänge, erfordert jedoch auch eine gewisse Kenntnis und Aufmerksamkeit des Benutzers, da er jeden Vorgang explizit anstoßen und überwachen muß. Das Programm sieht keine weitergehende Automatisierung vor, die aber auch im Rahmen einer Kommandoprozedur nur schlecht zu realisieren wäre und in diesem Zusammenhang eher zu einer größeren Fehleranfälligkeit führen könnten. In Hinblick auf eine bessere Fehlerüberprüfung einzelner Vorgänge und einer deutlichen Entlastung des Benutzers bei Routineaufgaben, aus denen sich die Archivierung im wesentlichen zusammensetzt, ist jedoch leistungsfähigere Spooling-Software unabdingbar.

7.4.3.2 Das Tape-Archiving-System

Zur Ablösung des Programmes `tape` wurde auf der Grundlage etablierter Software das **Tape Archiving System (TAS)** realisiert. Es stellt ein vollwertiges Spooling-System dar und orientiert sich an dem LPR-System von Berkeley-UNIX [Lpr]. Diese Software dient zwar dem Ausdrucken von Texten, also der Ausgabe von Dateien über Drucker als Ausgabegeräte, doch deckt sie im wesentlichen alle Aufgaben ab, die auch von einem Spooling-System zur Bandarchivierung gefordert werden:

1. Entgegennahme von auszugebenden Dateien
2. Einhängen dieser Dateien in eine Queue
3. Verwaltung der Archivierungs-Queue
4. Sukzessive Bearbeitung der Queue
5. Automatische Ausgabe der Dateien

Druckerausgabe und Bandarchivierung unterscheiden sich lediglich im Ausgabegerät und dessen Ansteuerung und in aller Regel in der Größe der zu bearbeitenden Dateien. Beide Problematiken werden aber bereits im Konzept des LPR-Systems berücksichtigt, so daß keine tiefergehenden Eingriffe in die Software notwendig waren, um sie als Bandarchivierungssystem zu benutzen. Änderungen der frei verfügbaren Software betrafen lediglich Namen von Programmen, Konfigurations- und Spooldateien bzw. Verzeichnissen und Ausgabetexte von Kontrollprogrammen. Die Software wurde ergänzt durch ein spezielles Ausgabeprogramm, das wie in dem System vorgesehen, für die Ansteuerung der speziellen Hardware — in diesem Fall also Bandgeräte — zuständig ist.

TAS besteht aus sechs einzelnen Programmen. Das Programm `tapecopy` stellt den wichtigsten Teil der Benutzerschnittstelle von TAS dar. Das Programm wird durch den Benutzer oder ein Teil der Datenerfassungs-Software aufgerufen und sorgt dafür, daß ein oder mehrere Meßdateien in die Archivierungs-Queue übernommen werden. Im Gegensatz zu den ansonsten analogen Kommando `lpr` kopiert es jedoch nicht die Datei selbst in das Spooling-Verzeichnis, sondern legt darin lediglich einen symbolischen Verweis auf die Datei an. Dadurch wird in Anbetracht der Größe einer Meßdatei — typischerweise in der Größenordnung von 100 MB — eine unnötige und aufwendige Belastung des Rechners vermieden. Mithilfe des Programmes `taq` (tape archive queue) kann der Zustand der Archivierungs-Queue angezeigt werden. Das Programm `tarm` (tape archive remove) dient zum Entfernen von Queue-Einträgen. Damit können ursprünglich zur Archivierung bestimmte Dateien wieder von der Archivierung ausgenommen werden. Das Programm `tac` (tape archive control) steht für allgemeinen Verwaltungsaufgaben des Archivierungssystems wie Blockieren oder Freigabe des Queue-Mechanismus, Starten und Anhalten der Bandarchivierung, Löschen oder Umorganisieren der Queues zur Verfügung. Im Gengesatz zu den anderen

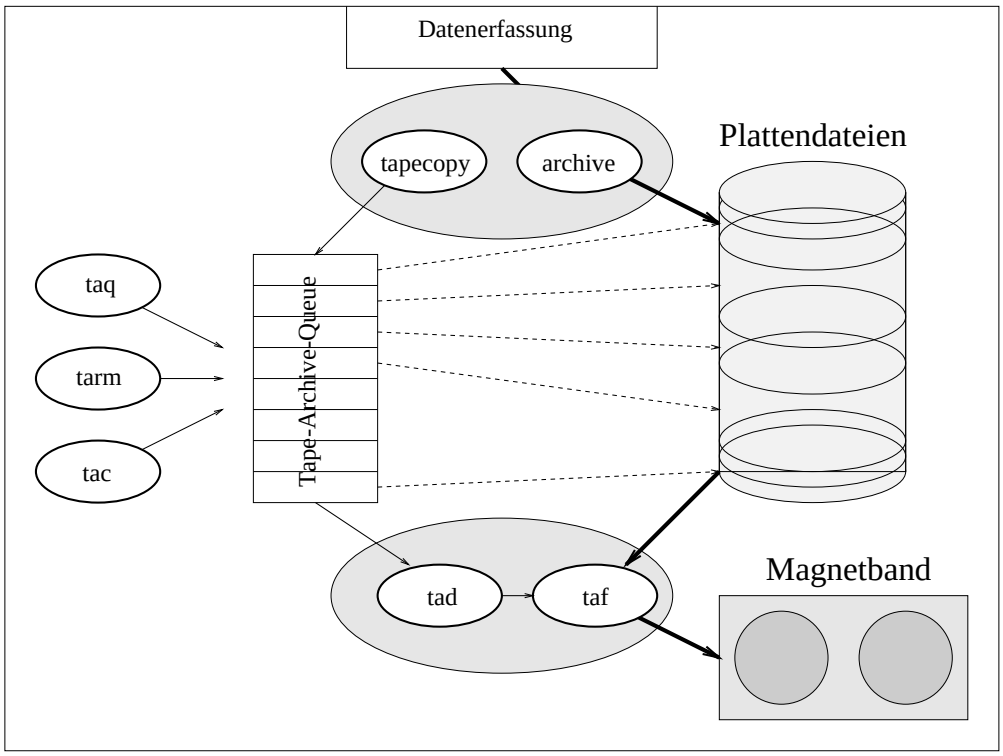

Abb. 7.10: Struktur des Tape Archiving Systems.

Programmen zur Bedienung des Systems erfordert dieses Programm spezielle Systemprivilegien und kann daher in der Regel nicht von den regulären Benutzern des Systems in Anspruch genommen werden.

Das einzige spezielle Programm ist `taf` (tape archive filter). Es dient dem direkten Ansteuern des Bandgeräts. Es wird durch den Tape-Archive-Daemon gestartet und erhält von ihm alle Informationen wie Name des Bandgerätes oder zu verwendende Blockgröße. Damit sorgt es für die korrekte Ansteuerung eines oder sogar mehrerer Bandgeräte und führt den Kopiervorgang der Daten von der Plattendatei auf Band aus.

Das Herzstück des Systems ist das Programm `tad` (tape archive daemon). Es steuert alle Aktivitäten des Systems. Anfragen von der Benutzerschnittstelle (ta, taq, tarm, tac) werden von ihm bearbeitet und es startet bei Bedarf die Programme, die die Daten endgültig auf Band schreiben (`taf`). Das Programm benutzt dabei Informationen aus einer Konfigurationsdatei, durch die das System flexibel an sehr unterschiedliche Randbedingungen angepaßt werden kann. Viele der bei der Druckerausgabe hilfreichen Möglichkeiten machen jedoch hier keinen Sinn. Die Datei bietet aber sehr einfach die Möglichkeit, Namen der zu benutzenden Bandgeräte, Blockgröße, Name des Programmes zur Bandansteuerung und ähnliche Information zu konfigurieren. Bei Bedarf können mehrere Queues für unterschiedliche Zwecke definiert werden.

7.5 Ergänzende Software

Bisher wurden alle wesentlichen Einzelkomponenten vorgestellt, die im Rahmen von MECDAS realisiert wurden, um damit Experimente unterschiedlichster Komplexität durchzuführen. Kernstück stellt das für das jeweilige Experiment speziell konfigurierte Datenaufnahmeprogramm dar. Zur Steuerung dieses Programmes und damit des Meßablaufs steht eine auf etwa ein Dutzend beschränkte Zahl von elementaren Kommandos zur Verfügung. Sie erlauben ohne weitere Hilfsmittel die vollständige Kontrolle über das Experiment. Obwohl diese bewußt einfach gehalten und daher auch einfach zu handhaben sind, bieten sie im Regelfall nicht die geeignete Arbeitsumgebung, mit der der Experimentator unmittelbar zu tun haben sollte, da er sich hier um zu viele Details kümmern und Routineaufgaben ausführen müßte. Gleichzeitig fallen im Rahmen der Datenerfassung eine Reihe von Aufgabenstellungen an, die nicht unmittelbar durch die elementaren Kommandos abgedeckt werden; dazu gehören

1. eine automatisierte Datenarchivierung
2. synchron zum Meßablauf automatisch ablaufende Vorgänge
3. die Protokollierung des Experimentablaufs
4. das Anzeigen und Festhalten von Fehlern
5. das Anzeigen des Systemzustands
6. Hilfestellung bei der Überwachung des Experiments

Solche Aufgaben wurden in der bisher beschriebenen MECDAS-Software ausgeklammert, da ihr Umfang und die damit verbundenen Anforderungen im allgemeinen Fall sehr weitreichend sein können, andererseits ihre Bearbeitung im konkreten Fall an spezielle Randbedingungen geknüpft ist. Für den Einzelfall lassen sich jedoch Aufgaben dieser Art in der Regel recht einfach unter Ausnutzung der elementaren Kommandos und standardmäßig verfügbarer UNIX-Werkzeuge lösen. Sie stellen zusammen ein Baukastensystem dar, das die Möglichkeit bietet, mithilfe einfacher Bausteine komplexere Funktionalitäten zu erzielen. Da hierzu einige Kenntnisses des Betriebssystems und seiner Möglichkeiten notwendig sind, wurde für die genannten Aufgaben Lösungen implementiert, die für viele der durchzuführenden Experimente an MAMI ausreichend sind. Sie stellen Prototypen dar, die bei Bedarf an spezielle Bedürfnisse leicht angepaßt werden können, oder auch nur als Beispiel für völlig neue Lösungen dienen können. Sie bestehen fast ausschließlich aus Kommandoprozeduren, da die meisten der oben genannten Aufgaben weder einer speziellen Programmierung bedürfen, noch besonders zeitkritisch sind. Bei Bedarf könnten sie jedoch als herkömmliches, in C oder einer anderen Hochsprache geschriebenes Programm implementiert werden.

7.5.1 `run`

Ein zentrales Programm, das einige der genannten Aufgabenbereiche berührt, ist die Kommandoprozedur `run`. Sie verbindet die elementaren Steueranweisungen mit der Ausführung automatisch ablaufender Vorgänge

- zur Verwaltung von Messungen
- zur Protokollierung des Experimentablaufs
- zur automatisierten Datenarchivierung
- für spezielle, vom Benutzer definierbare Aktionen
- zur automatischen Ausgabe von Statusinformation

und nimmt dabei umfangreiche Fehlerüberprüfungen vor, um unkontrollierte Programmaktivitäten zu vermeiden.

Ähnlich wie das auch bei dem Datenaufnahme-Server in Kombination mit seinen Klienten der Fall ist, realisiert `run` ein persistentes Objekt, das nun das Gesamtexperiment repräsentiert. Es erbt dabei alle Eigenschaften des Datenaufnahmeobjekts. Im Gegensatz zu den herkömmlichen Verfahren bei der Objektorientierten Programmierung ist dieses Objekt nicht Bestandteil eines abgeschlossenen Programmes, sondern es verteilt sich auf mehrere Programme, Prozesse und Dateien. Es stellt eine Reihe von Methoden in Form von Kommandos der Art „`run` *command*“ zur Verfügung, mit denen der Meßablauf gesteuert werden kann. Die Kommandoprozedur bleibt dabei nie permanent aktiv, sondern wird stets neu aufgerufen. Sie stößt lediglich die jeweils notwendigen Aktivitäten an. Die über den Kommandoparameter *command* spezifizierte reguläre Steueranweisung wird zuerst einmal an den Datenaufnahme-Server weitergegeben. Nachdem dieser sie bearbeitet hat, werden je nach Anweisung Programme gestartet oder angehalten, Dateien gelesen oder modifiziert oder wiederum mit anderen Objekten Nachrichten ausgetauscht. Ähnlich wie bei den anderen Aktivitäten der Datenerfassung spielt auch hier die Zustandsmaschine des Datenaufnahme-Servers eine zentrale Rolle. Bei Bedarf werden Dateien zur Aufnahme von Statusinformation oder zur Dokumentation des Experiments angelegt und in bestimmten Fällen auch auf der Standard-Ausgabe ausgegeben. Neben den elementaren Steuerkommandos benutzt sie dabei auch einige der anderen Hilfsprogramme.

7.5.2 `archiver`

Zur Automatisierung der Datenarchivierung steht eine Reihe von Kommandoprozeduren zur Verfügung, die hier einheitlich mit dem Namen `archiver` bezeichnet werden sollen. Dieser Name wird auch für das Kommando benutzt, das z. B. innerhalb der `run`-Kommandoprozedur zur Steuerung von Archivierungsaktivitäten Verwendung findet. In der objektorientierten Sprechweise stellt

`archiver` analog zu `run` eine abstrakte Klasse für persistente Objekte dar, die die Aufgabe haben, für die korrekte Archivierung der während einer Einzelmessung anfallenden Daten zu sorgen. Sie stellt eine Reihe von Methoden zur Verfügung, mit deren Hilfe die Archivierung zu steuern ist. Verschiedene Instanzen dieser Objektklasse dienen der Unterstützung unterschiedlicher Ausgabegeräte wie z. B. Plattendatei, Magnetband, Netzwerk oder auch Software mit individueller und an spezielle Randbedingungen abgestimmter Behandlung. In ihrer Realisierung als Kommandoprozeduren besitzten sie ein identisches Interface zum Benutzer bzw. zur aufrufenden Software hin in Form von Kommandos wie `archiver init`, oder `archiver start`. Dabei werden je nach Anwendung Vor- und Nachbereitungsaufgaben zur Archivierung vorgenommen, in Dateien Informationen abgelegt, Programme gestartet oder angehalten. Die Kommandoprozedur `archiver` ist dazu nicht permanent aktiv, sondern dient ausschließlich dazu, den Zustand des Archiver-Objekts zu ändern.

Grundlage ist dabei in allen Fällen das Programm `archive`, das lediglich in unterschiedlichen, an die spezielle Problemstellung angepaßten Betriebsmodi benutzt wird. Es übernimmt die Meßdaten von der Datenerfassungs-Software und leitet sie an das vorgegebene Ziel weiter. Es wird ergänzt durch mehr oder weniger gerätespezifische Software. Im Falle von Dateien wird z. B. für eine eindeutige Namensvergabe für unterschiedliche Einzelmessungen gesorgt, bei der Archivierung auf Magnetband wird ebenfalls der Name des zu verwendeten Geräts festgelegt und vor bzw. nach der Archivierung Statusüberprüfungen vorgenommen.

Anstelle eines herkömmlichen Gerätes bietet UNIX mithilfe von Pipes auch die Möglichkeit, die Daten unmittelbar an spezielle Software weiterzuleiten, die unterschiedlichste Aufgaben übernehmen kann. Dazu gehört z. B. die Datenarchivierung über ein Netzwerk oder ein anderes spezielles Medium, aber auch beliebige Filterfunktionalität, die von einfacher Datenkompression bis hin zu experimentspezifischer Auswertung oder Manipulation der Daten reicht.

7.5.3 `logger`

Die Kommandoprozedur `logger` dient dazu den Meßablauf zu protokollieren. Dazu hält sie in bestimmten Phasen der Messung Statusinformation fest und sorgt dafür, daß diese zum Ende jeder Messung zur Kontrolle durch den Benutzer auf der Bedienkonsole ausgegeben und als Grundlage eines herkömmlichen Protokollbuchs auf einen Drucker ausgedruckt wird. Die Statusinformation umfaßt dabei die Angaben, welche die Datenaufnahme-Software über den Zustand von Experiment und Datenerfassung zur Verfügung stellt und mit entsprechenden Steueranweisungen angefordert werden können. Sie wird ergänzt durch Informationen des Experimentkontrollsystems [Kram95] über den Zustand der Meßapparatur, also Feldstärken der Spektrometermagnete, Hochspannungen von Detektoren, Strahlstrom usw. Daneben kann sie erweitert werden durch Einträge in spezielle Dateien, die vom Benutzer per Hand oder mithilfe anderer Software angelegt und aktualisiert werden können. Sowohl `archiver` als auch `logger` werden von `run` zu geeigneten Zeitpunkten aufgerufen, können aber auch eigenständig verwendet werden.

7.5.4 `analyser`

Ähnlich wie `archiver` und `logger` arbeitet die Kommandoprozedur `analyser`. Sie realisiert ein On-line-Analyse-Objekt, das durch `run` zu Beginn und am Ende einer Einzelmessung aktiviert bzw. desaktiviert wird. Je nach Realisierung durch den Anwender wird dabei zu seiner Aktivierung ein On-line-Analyseprogramm, wie es in Abschnitt 8 beschrieben ist, gestartet und über die nicht-synchronisierte Datenschnittstelle des Datenaufnahmeprogrammes mit den aktuellen Meßdaten versorgt. Es bearbeitet diese solange, bis das Ende der Daten einer Einzelmessung erreicht ist oder seine explizite Desaktivierung erfolgt. Art und Umfang der dazu notwendigen Aktivitäten werden ebenso wie das On-line-Analyse-Programm selbst durch den Anwender bestimmt. Wegen der Abhängigkeit des Analyseprogramms vom Experiment stehen hierfür lediglich einfache Prototypen zur Verfügung, die bei Bedarf durch experimentspezifische Software ersetzt werden müssen.

7.5.5 `watcher`

`watcher` ist ein einfaches Beispiel für eine Kommandoprozedur, die den Benutzer bei der Überwachung des Experiments unterstützt. Sie fordert regelmäßig den Status der Messung an, speichert diese Information zwischen und verarbeitet diese in ihrer zeitlichen Entwicklung nach bestimmten Kriterien. Sie nimmt dabei einfache Konsistenzchecks vor und sorgt für die Bildschirmausgabe des aktuellen Experimentzustands. Bei Bedarf werden dabei Meldungen ausgegeben, wenn Inkonsistenzen festgestellt werden bzw. eine voreingestellte Zeit abgelaufen ist, um den Experimentator z. B. zum Eingreifen in den Meßablauf aufzufordern.

7.6 Bedienoberfläche

7.6.1 Problemstellung

Die Bedienoberfläche (auch Benutzeroberfläche) einer Anwendung stellt die Schnittstelle zwischen dem Benutzer und der eigentlichen Anwendung dar. Sie hat im Rahmen des Dialogs zwischen Benuzter und Rechner für die Abwicklung der Ein- und Ausgabe zu sorgen, sollte den jeweiligen Betriebszustand präsentieren und muß Eingriffe des Benutzers für Interaktionen, Auskünfte, Hilfestellungen und Erklärungen zulassen. Um den Benutzer bei der Bedienung zu entlasten, sollte die Oberfläche innerhalb der kompletten Anwendung ein einheitliches Erscheinungsbild und einen homogenen Präsentations- und Interaktionsstil verwenden. Die Erfahrung zeigt, das das bei fast jedem Anwendungsprogramm anders ist, und Benutzer viel Zeit damit verlieren, sich ständig auf andere Darstellungen einstellen zu müssen. Aber auch bei der Erstellung von Software für eine Bedienoberfläche wird oftmals viel Zeit verschwendet, da diese einen erheblichen Anteil der Implementationsarbeit kostet. Andererseits zeigt die Entwicklung der letzten Jahre im Bereich der Personal Computer und Workstations, daß die Hersteller

von Computern und Systemsoftware ihre Produkte verstärkt mit einheitlichen Fenstersystemen als Benutzungsoberfläche ausstatten. Bei Betriebssystemen wie UNIX oder VMS hat sich inzwischen das am MIT entwickelte X Window System als Standard etabliert [X11doc]. Aber auch dessen Nutzung ist mit hohem Implementationsaufwand verbunden und stellt gleichzeitig zusätzliche Anforderungen an die Rechner-Hardware.

Eine wichtige Grundregel bei der Konzeption von MECDAS war daher, Anwendung und Benutzungsoberfläche klar zu trennen. Damit konnte auch der Forderung Rechnung getragen werden, daß viele Komponenten der Software nicht nur interaktiv durch den Benutzer, sondern auch durch andere Software nutzbar sein mußten. Hilfreich bei der Umsetzung dieser Grundregel waren die Eigenschaften von UNIX und insbesondere die der Kommandosprache dieses Betriebssystems. Sie ermöglichten es unter anderem, vorerst ganz ohne die Implementation einer speziellen Bedienoberfläche auszukommen.

7.6.2 Die Shell als einheitliche Benutzerschnittstelle

Bei entsprechender, konsequenter Programmierung der Anwendungs-Software bilden die unter UNIX verfügbaren Kommandointerpreter (Shells) selbst eine ausreichende Grundlage für eine einfache Bedienoberfläche. Sie wurden genutzt, um mit wenig Aufwand eine allgemeine, leistungsfähige und flexible Benutzungsschnittstelle zur Bedienung des Systems zu realisieren. Insbesondere in neueren Ausführungen erlaubt die Standard-Shell nicht nur eine verhältnismäßig komfortable Bedienung im Dialog-Betrieb, sondern stellt aufgrund ihrer leistungsfähigen Kommandosprache weitreichende Programmiermöglichkeiten zur Verfügung, mit denen auch komplexe Aufgaben gelöst werden können. Damit bildet sie eine einheitliche Grundlage sowohl für die interaktive Bedienung als auch bei der Formulierung nicht-interaktiver Systemaufträge in Form von Kommandoprozeduren.

7.6.3 Die MECDAS-Shell

Speziell zur Unterstützung des Benutzers bei der Durchführung der Datenerfassung wurde eine einfache Bedienoberfläche auf der Basis des Bash-Kommandointerpreters [Bash] realisiert. Dieser Kommandointerpreter bietet neben den Eigenschaften der Standard-UNIX-Shell insbesondere bei der interaktiven Bedienung zusätzliche Hilfsmittel wie Command Completion, History-Mechanismen oder bequeme Editier-Möglichkeiten. Durch die konsequente Ausnutzung des Kommandokonzepts von UNIX innerhalb von MECDAS reichte die Funktionalität dieses Standardwerkzeuges voll und ganz aus; es war nicht notwendig, spezielle Software wie z. B. einen eigenen Kommandointerpreter als Benutzeroberfläche zu implementieren, um die Datenerfassungs-Software bedienen zu können.

Wie die bereits beschriebenen Hilfsprogramme wird auch zur Aktivierung der Bedienoberfläche eine Kommandoprozedur verwendet. Im Gegensatz zu den

anderen Programmen geht sie nach Durchlaufen einer Initialisierungsphase in einen interaktiven Betriebsmodus über, in dem dem Benutzer alle MECDAS-, UNIX- und selbstdefinierten Kommandos zur Verfügung stehen. Das unter dem Namen `mecdas` zu startende Programm bereitet zu Beginn eine an das durchzuführende Experiment angepaßte Umgebung vor. Es definiert und initialisiert dazu eine Reihe innerhalb des Kommandointerpreters verfügbarer sogenannter Shell-Variablen, die einerseits experimentspezifische Voreinstellungen für Parameter von Programmen darstellen, die in der Folge benutzt werden, andererseits variable Parameter zur Beieinflussung des Experimentablaufs. Es werden Dateien und Verzeichnisse angelegt, die im Laufe der späteren Aktivitäten von der Benutzeroberfläche selbst oder anderen Programmen benötigt werden.

Für eine effizientere Bearbeitung auch komplexerer Aufgaben definiert es über sogenannte Shell-Funktionen neue Kommandos. Dabei werden auch die in Abschnitt 7.5 beschriebenen Kommandoprozeduren genutzt. Insbesondere die Funktionalität des `run`-Kommandos wird damit intrinsischer Bestandteil der MECDAS-Shell. In Kombination mit dem X Window System sorgt es schließlich für das Darstellen von Fenstern zur Ausgabe von Status- und Fehlermeldungen.

Mit dem Abschluß der Initialisierungsphase geht die MECDAS-Shell in ihren interaktiven Betriebszustand über, in dem sie nun wie jeder andere UNIX-Kommandointerpreter jedes beliebige UNIX-Kommando ausführen kann, daneben aber auch experimentspezifische Kommandos zur Steuerung der Datenerfassung und zur Beeinflussung von deren Ablauf. Zentrale Rolle spielt dabei das `run`-Kommando, mit dessen Hilfe die Datenerfassung vollständig gesteuert werden kann, wobei je nach Bedarf ergänzende Aktivitäten angestoßen werden. Es orientiert sich dabei jedoch vollständig am Zustand der Datenaufnahme. Damit ist die gesamte MECDAS-Shell in Bezug auf die Datenerfassung zustandslos. Daher kann sie zu einem beliebigen Zeitpunkt verlassen und neu gestartet werden — unabhängig vom Zustand der Messung. Auch der unter Umständen notwendige mehrfache Start der MECDAS-Shell kann nicht zu Inkonsistenzen in der Datenerfassung führen; statt dessen sorgt eine solche Vorgehensweise eher für eine Unübersichtlichkeit und sollte daher vermieden werden.

Ebenso wie die elementaren Kommandos können auch die Kommandos der MECDAS-Shell zu komplexeren Kontrollstrukturen zusammengefaßt werden (s. Abschn. 5.7), um so unter Zuhilfenahme der Shell-Programmierung und in Kombination mit Standard-Betriebssystemwerkzeugen auf einfache Art und Weise automatische Meßabläufe zu realisieren. Auch die Verwendung von Kommandoprozeduren sind möglich. Im Gegensatz zu den elementaren Kommandos beeinflussen diese jedoch ebenso wie die interaktiven Kommandos der MECDAS-Shell nicht nur den Zustand des Datenaufnahmeprogrammes sondern alle anderen an der Datenerfassung beteiligten Software-Objekte zur Datenarchivierung, Überwachung und On-line-Analyse.

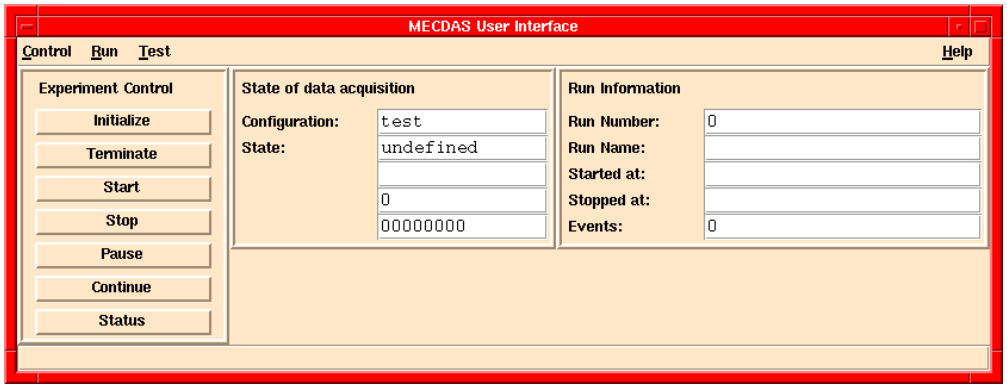

Abb. 7.11: Xmecdas

7.6.4 `xmecdas`

Die MECDAS-Shell ist so konzipiert, daß sie an jeder Art von Datensichtgerät benutzt werden kann, da sie für ihre Grundfunktionalität lediglich von Text-Ein- und -Ausgabe über eine herkömmliche Terminal-Schnittstelle Gebrauch macht. Das X-Window-System wird nur dann genutzt, wenn es verfügbar ist und dann nur für einfache Status- und Fehlerprotokollfunktionen.

Die MECDAS-Shell bietet eine sehr große Flexibilität und große Freiheiten bei ihrer Bedienung, die jedoch in der Regel nur von erfahreren Anwendern nutzbrigend eingesetzt werden können. Für weniger Versierte können diese Eigenschaften schnell zu einer Fehlerquelle werden. Als Ergänzung zur MECDAS-Shell bietet es sich daher an, eine alternative Bedienoberfläche zu realisieren, die sich im wesentlichen auf die zur Steuerung notwendigen Aktivitäten beschränkt, dafür aber komfortablere Mechanismen unter stärkerer Berücksichtigung von ergonomischen Gesichtspunkten verwendet.

Hierfür bietet sich der Einsatz von X an, das im wesentlichen auf allen bei der Datenerfassung eingesetzten Rechnern verfügbar ist und für die genannten Anforderungen eine sehr gute Grundlage bildet. Der damit verbundene Implementationsaufwand ist jedoch so hoch, daß er im Rahmen dieser Arbeit nicht zu leisten war. Unter Ausnutzung neuartiger, leistungsfähiger Software-Werkzeuge [Oust94] wurde jedoch ein erster, noch rudimentärer Prototyp implementiert, der als ein erfolgsversprechender Grundstock für eine vollständige Bedienoberfläche verwendet werden kann, in der die Vorzüge des X-Window-Systems voll genutzt werden können. Abbildung 7.11 demonstriert das Aussehen des entwickelten Prototyps.

Die X-Bedienoberfläche setzt ebenso wie die MECDAS-Shell auf den `run`-Kommandos auf und nutzt damit exakt dieselbe Schnittstelle zur Datenerfassungs-Software wie diese. Beide Bedienoberflächen können daher vollkommen gleichberechtigt nebeneinander verwendet werden.

Kapitel 8

Software zur On-line-Analyse

In Ergänzung zur eigentlichen Datenerfassungs-Software wurde im Rahmen von MECDAS auch Software implementiert, die den Experimentator unterstützt, online, also bereits während der laufenden Messung, die aktuellen Meßdaten auszuwerten, daraus Histogramme zu generieren und diese graphisch darzustellen. Die Software setzt sich aus drei grundlegenden Paketen zusammen:

1. Unterprogrammbibliotheken und Hilfsprogramme für eine allgemeine, experiment- oder benutzerspezifische On-line-Analyse
2. Histogramm-Paket
3. Programme zur graphischen Darstellung von Meßdaten und Histogrammen

8.1 On-line-Auswertung von Meßdaten

8.1.1 Problemstellung

Die bisher beschriebene Datenerfassungs-Software stellt alle Hilfsmittel zur Verfügung, um ein Experiment vollständig durchzuführen — von der Auslese der Meßelektronik über das Formatieren der Daten, den Datentransport, das Eventbuilding bis hin zur Archivierung der Daten, deren eigentlicher physikalischer Inhalt in der Regel erst im Anschluß an die Messung (also off-line) in einer aufwendigen Auswertung bestimmt werden muß. In den verschiedenen Phasen der Datenerfassung sorgt die Software aus Effizienzgründen im wesentlichen ohne weitgehende inhaltliche Überprüfung der Daten für die Erfüllung ihrer Aufgaben. Dabei werden lediglich funktionelle Störungen und Fehlfunktionen erkannt und an den Experimentator gemeldet. Auch wenn die Möglichkeiten der Software genutzt werden, mehr oder weniger umfangreiche, experimentspezifische Statusinformation anzufordern, reicht damit die dem Experimentator zur Verfügung stehende Information nicht aus, um einen ausreichenden Überblick über den Ablauf der Messung zu erhalten.

Es ist vielmehr notwendig, unmittelbar während der Messung auch die Meßdaten zu untersuchen, um einerseits die korrekte Funktion der Detektoren und der Datenerfassungs-Software zu kontrollieren, andererseits auch Fehleinstellungen von Experimentparametern wie Energie, Magnetfeldeflüsse, Diskriminatorschwellen, Hochspannungen oder Winkel zu erkennen. Dazu reichen oftmals einfache Konsistenzüberprüfungen auf einer relativ niedrigen Ebene aus; unter Umständen kann es aber auch notwendig werden, eine Auswertung mit aufwendigeren Maßnahmen vorzunehmen. Dazu gehört z. B. die ereignisweise Berechnung der physikalisch relevanten Information aus den Rohdaten, wie im Falle der Drei-Spektrometer-Anlage die Berechnung der Teilchenkoordinaten im Detektorsystem, ihre Rückrechnung auf Targetkoordinaten, die Teilchenidentifikation und schließlich die Bestimmung von Impuls und Energie der untersuchten Teilchen. Auch statistische Auswertungen können von Interesse sein. Mithilfe ein- oder mehrdimensionaler Schnitte (sogenannter Cuts) oder auch komplexer Auswahlbedingungen müssen echte Ereignisse von Untergrundereignissen getrennt werden bzw. — etwas allgemeiner formuliert — Ereignisse nach verschiedenen Kriterien in unterschiedliche Gruppen sortiert werden. Auf dieser Grundlage sind Histogramme zu generieren, die die Häufigkeitsverteilung von interessierenden Rohdaten oder der daraus resultierenden physikalischen Information in verschiedenen Stufen der Analyse wiedergeben. In den seltensten Fällen werden bereits Endergebnisse wie Wirkungsquerschnitte zu erwarten sein, da eine vollständige Auswertung der Daten indirekte, mehrstufige Arbeitsmethoden erfordert.

Zur Erledigung dieser Aufgaben ist es in der Regel nicht sinnvoll, dazu notwendige Software unmittelbar in die Datenaufnahme-Software zu integrieren, da zum einen der erforderliche Rechenmehraufwand das zeitliche Verhalten der Datenaufnahme verschlechtern würde, zum anderen die besonderen Anforderungen im Rahmen der Datenaufnahme evtl. Einschränkungen bei der Realisierung der Auswerteaufgaben mit sich bringen könnten. Nicht zuletzt lassen sich auch hier mit einer klaren Aufgabenabgrenzung die Vorzüge eines modularen Systems nutzen, was gerade in Hinblick auf die unterschiedlichen Anforderungen von Datenerfassung und On-line-Analyse sowohl in Bezug auf die auszuführenden Aufgaben als auch ihre Bedienung und Interaktion mit dem Benutzer wichtig ist. Dennoch ist eine möglichst enge Ankopplung an die Datenerfassung notwendig, um möglichst frühzeitig bei der Datenaufnahme auftauchende Probleme erkennen und beheben zu können. Gerade diese Tatsache macht herkömmliche, etablierte Auswerte-Software, die in der Regel off-line eingesetzt wird, zur On-line-Analyse schlecht oder überhaupt nicht geeignet. Off-line liegen die Daten in Form von abgeschlossenen Dateien auf Band oder Platte vor, die bei Bedarf auch mehrfach bearbeitet werden können. On-line dagegen bilden die Meßdaten einen fortlaufenden Datenstrom, dessen Anfang und Ende für das Auswerteprogramm nicht definiert ist.

Eine besonders wichtige Anforderung an Software zur On-line-Analyse ist eine möglichst gute Synchronität zur Datenaufnahme, d. h., gerade aufgenommene Meßdaten sollten stets innerhalb kürzester Zeit ausgewertet werden, um dem Experimentator ständig einen aktuellen Überblick über die Messung zu gewährleisten. Dabei muß zwangsläufig dem Unterschied der Geschwindigkeiten

Rechnung getragen werden, mit denen der Datenstrom einerseits einläuft und archiviert wird und andererseits ausgewertet werden kann. Wenn das Auswerteprogramm die einlaufenden Daten nicht schnell genug bearbeiten kann, dennoch aber stets die aktuellen Daten bearbeiten soll, müssen Daten ausgelassen werden. Auch für den Fall, daß die Auswerteprogramme schneller als der einlaufenden Datenstrom ist, muß es immer wieder mit dem Datenstrom synchronisiert werden. Falls die Daten nicht auf ein sequentielles Medium wie Magnetband sondern auf Plattendateien abgespeichert werden, kann zwar unter Umständen auch ein Off-line-Analyseprogramm auf die kurz zuvor gemessenen Daten zugreifen, doch bleibt das beschriebene Synchronitätsproblem ungelöst. Hierfür ist eine spezielle Kommunikation zwischen Datenerfassungs-Software und On-line-Analyseprogramm notwendig.

Um das Hauptziel der On-line-Analyse zu erreichen, den Experimentator stets einen aktuellen Überblick über das Experiment zu geben, muß die Analyse-Software nicht nur für die Synchronität mit der Datenaufnahme sorgen, sondern ebenso dem Benutzer stets die aktuellen Ergebnisse auch zur Verfügung stellen. Der Benutzer sollte auf diese Ergebnisse mit minimalem Aufwand und ohne Beeinflussung von Datenaufnahme und On-line-Analyse Zugriff haben.

8.1.2 Grundprinzip der MECDAS-On-line-Analyse

Wie bereits an früherer Stelle beschrieben enthält schon die Kernsoftware von MECDAS Vorkehrungen für die On-line-Analyse. Der Datenaufnahme-Server stellt sowohl auf Frontend-Ebene als auch beim Eventbuilding neben den Schnittstellen zur synchronen Datenarchivierung auch solche zur Verfügung, die für Zwecke der On-line-Analyse verwendet werden können und genau den im vorangegangenen Abschnitt beschriebenen Anforderungen entsprechen, um geeigneten Analyseprogrammen unabhängig von den verwendeten Archivierungsmethoden und, ohne die Datenerfassung zu beeinflussen, unmittelbaren Zugang zu den Meßdaten zu gewähren. Jedes MECDAS-Analyseprogramm zur On-line-Analyse stellt daher grundsätzlich einen eigenständigen Klienten des Datenaufnahme-Servers dar. Neben dem hierfür notwendigen Programmcode, der für eine Kommunikation mit dem Server und eine Übermittlung der aktuellen Meßdaten sorgt, enthält das Programm schließlich Software, die zur Bearbeitung der für die Auswertung der Daten anfallenden Aufgaben benötigt wird.

Diese Software setzt sich aus zwei grundlegenden Teilen zusammen. Der erste umfaßt wesentliche Bestandteile der MECDAS-Software und dient allgemeinen Routineaufgaben, die unabhängig vom aktuellen Experiment ausgeführt werden können. Als eine der wichtigsten Aufgaben gehört dazu die Dekodierung der Meßdaten. Diese stellen zwar das Ergebnis eines individuellen Experiments dar, sind jedoch mit den in Kapitel 6.8.3 beschriebenen Methoden so formatiert, daß weder zusätzliche Information zu ihrer Rekonstruktion notwendig ist, noch spezielle Software benötigt wird, um dabei experimentspezifische Eigenheiten zu berücksichtigen. Allein die mit den Meßdaten abgespeicherte Information reicht auf der Basis allgemeiner Software für die vollständige Ereignisrekonstruktion

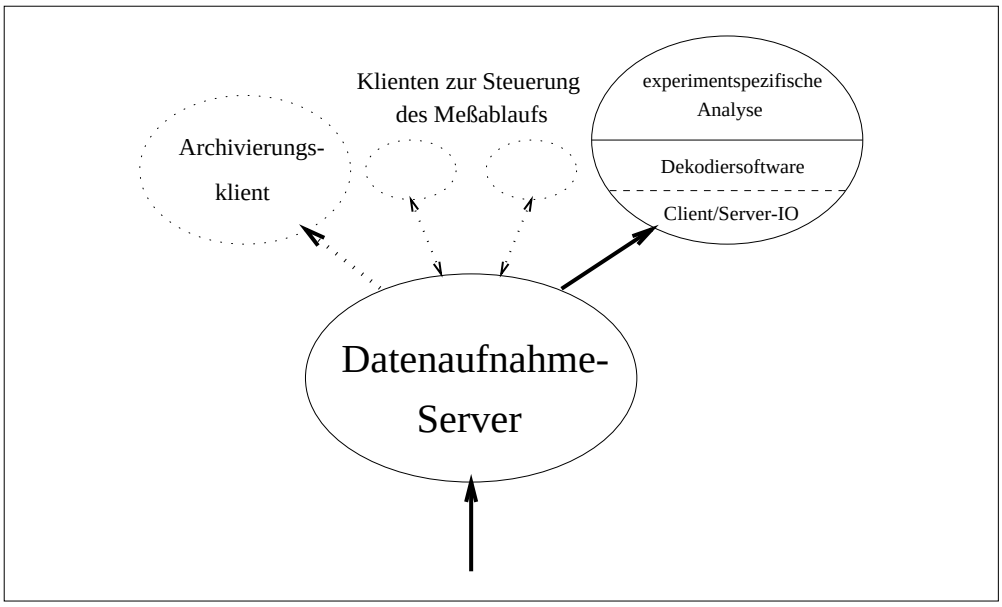

Abb. 8.1: Das Analyseprogramm stellt einen Klienten des Datenaufnahme-Servers dar. Es hat damit einerseits direkten Zugriff auf die Daten, ist aber andererseits nur lose an die Datenerfassung gekoppelt.

aus. Die so wiederhergestellten Meßdaten werden dann dem zweiten Teil des Analyseprogrammes zur Verfügung gestellt. Dieser Teil repräsentiert den eigentlich experimentspezifischen Anteil der Analyse-Software. Sein Programmcode muß vollständig vom Experimentator bereitgestellt werden. Abbildung 8.1 gibt schematisch den Aufbau eines On-line-Analyseprogramms wieder und versucht dessen Rolle innerhalb des Gesamtsystems zu veranschaulichen.

Die bisher realisierte erste Ausbaustufe der On-line-Analyse sieht für den experimentspezifischen Teil ein reguläres C- oder C++-Programm vor, das zwar in letzter Konsequenz vom Experimentator erstellt oder anhand existierender Beispiel-Software angepaßt werden muß, in dem aber von einer Reihe von Bibliotheksfunktionen der Standard-MECDAS-Software Gebrauch gemacht werden kann. Die zentrale Rolle spielt dabei die logische Experimentkonfiguration. Ihre in den Daten enthaltene Beschreibung wird von der Dekodier-Software zum Auspacken der Daten verwendet und stellt gleichzeitig einen wesentlichen Bestandteil der Schnittstelle zwischen der allgemeinen Software und dem Anwenderteil des Analysepogrammes dar. Sie bietet dem Benutzer die Möglichkeit, ohne Detailkenntnisse mit der in C verwendeten symbolischen Adressierung über Variablennamen unmittelbar auf die Meßdaten zugreifen zu können.

Prinzipiell kann zur Implementierung des Anwenderteils auch eine andere Programmiersprache wie z. B. Fortran verwendet werden, doch schränkt das die Nutzung der allgemeinen MECDAS-Bibliotheken stark ein. Auch auf die an vielen Stellen hilfreiche Verwendung des C-Präprozessors muß dann verzichtet werden. Für die Neuerstellung von Analyse-Software sollte daher ausschließlich auf C/C++ zurückgegriffen werden, insbesondere da C die Implementationssprache der restlichen Software ist und so neue Software auch in Hinblick auf die

Übernahme in Bibliotheken sich besser ins Gesamtsystem einfügt. Gerade auch bei Ausnutzung der leistungsfähigen Mechanismen von C++ ist im Vergleich zu Fortran keinerlei Verzicht auf Funktionalität verbunden. Ganz im Gegenteil bietet sich damit die Möglichkeit, dem Benutzer, der den experimentspezifischen Teil der Analyse-Software im wesentlichen eigenständig kontrolliert, mit dem Einsatz zeitgemäßer Methoden eine verbesserte Arbeitsgrundlage zur Verfügung zu stellen. Dennoch wurde großen Wert darauf gelegt, daß bereits existierende Fortran-Software mit MECDAS-On-line-Analyse-Software mit wenig Aufwand kombiniert werden kann. Somit ist die Nutzung des umfangreichen Software-Angebots aus dem Fortran-Bereich ohne weiteres möglich (s. Abschnitt 9.1.4.2).

Auch der Einsatz anderer Programmiersprachen ist denkbar, sie spielen jedoch zur Zeit keine bedeutende Rolle. Ebenso wurde nach umfangreichen Untersuchungen vorerst auf die Verwendung einer speziellen auf die Analyse kernphysikalischer Daten angepaßten Kommandosprache verzichtet, die interpretativ abgearbeitet wird und den Experimentator die Möglichkeit bietet, z. B. Analysebedingungen dynamisch zu definieren und zu verändern. Sowohl die eigene Implementation eines entsprechenden Interpreters als auch die Anpassung bzw. Integration existierender Software war nicht unmittelbar Ziel dieser Arbeit und hätte diese bei weitem gesprengt. Statt dessen wurde mit der unmittelbaren Verwendung von zu compilierendem Quellcode ein möglichst einfaches Konzept verfolgt, das einerseits den Implementationsaufwand auf ein Minimum beschränkte, andererseits die Möglichkeiten, beliebige Auswertealgorithmen zu formulieren, nicht beschneidet und damit die Basis für ein leistungsfähiges On-line-Analysesystem darstellt. Ähnlich wie bei der Datenaufnahme werden diese Möglichkeiten ergänzt durch die Fähigkeiten der UNIX-Shells in Kombination mit vielen anderen Betriebssystemwerkzeugen.

Die Verwendung von herkömmlichem Quellcode hat zwar den Nachteil, daß in der Regel auch bei der kleinsten Änderung der Analyseanweisungen eine Neuübersetzung des Programmes notwendig wird, doch bewegt sich der damit verbundene Mehraufwand unter Verwendung geeigneter Betriebssystemwerkzeuge und aufgrund der hohen Verarbeitungsgeschwindigkeit der modernen Rechner sowohl in Bezug auf die Bedienung als auch auf den Zeitaufwand in vertretbarem Rahmen. Auch die Tatsache, keine einfache und problemorientierte Sprache benutzen zu können, ist bei Verwendung geeigneter Präprozessoren und insbesondere bei der konsequenten Ausnutzung der von der Sprache C++ zur Verfügung gestellten Möglichkeiten wie Operator-Overloading in Kombination mit der Nutzung von entsprechenden Klassenbibliotheken für spezielle Aufgabenbereiche weniger von Bedeutung. Mit diesen Hilfsmitteln läßt sich vielmehr bei beschränktem Aufwand eine leicht zu handhabende und gleichzeitig flexible Möglichkeit schaffen, mehr oder weniger komplexe Analysevorschriften in ein leistungsfähiges Analyseprogramm umzusetzen. Im einfachsten Fall können für feste Experimentieranordnung entsprechend kodierte Programme verfügbar gemacht werden, die alle für diesen Fall notwendigen On-line-Analyse-Aufgaben abdecken, ohne daß eine ständige Anpassung des Programmes notwendig wird (s. Abschnitt 9.1.4.2). Darüber hinaus stellt dieses Verfahren einen sehr guten Ausgangspunkt für zukünftige Entwicklungen dar.

8.1.3 Realisierung

8.1.3.1 Basissoftware

Grundlage der On-line-Analyse-Software von MECDAS bildet einerseits der auch bei der Datenaufnahme eingesetzte Programmcode zur Ein- und Ausgabe und andererseits die Software zur Dekodierung der Meßdaten, die nach dem in Abschnitt 6.8.3 beschriebenen Format verpackt sind. Beide Bereiche sind realisiert als ein Satz von Unterprogrammen, die drei jeweils in sich abgeschlossene Objektklassen darstellen. Ihre Methoden umfassen neben Konstruktoren und Destruktoren in der Hauptsache jeweils eine Routine, die alle wesentlichen Aufgaben

- zum Einlesen der Meßdaten
- bei der Dekodierung der Meßdaten
- zur Ausgabe eventuell modifizierter Meßdaten

übernimmt. Sie können in einem beliebigen Anwenderprogramm, das für die Analyse der Meßdaten bestimmt ist, verwendet werden. Abbildung 8.2 gibt die grundlegende Struktur eines typischen Analyseprogrammes auf dieser Basis wieder. Es durchläuft drei Phasen: jeweils eine, in der Initialisierungs- und Aufräumarbeiten zu Beginn bzw. am Ende des Programmes ausgeführt werden, und eine zentrale Schleife, in der die Hauptdekodierroutine immer wieder aufgerufen wird. Diese zerlegt bei ihren wiederholten Aufrufen einen einlaufenden Datenstrom in die einzelnen Datensegmente, aus denen er bei der Datenaufnahme bzw. beim Eventbuilding zusammengesetzt wurde (Abb. 8.3). Sie nimmt dabei abhängig vom Typ des jeweiligen Datensegments seine weitere Dekodierung vor und führt bei Bedarf dafür notwendige zusätzliche Aktivitäten aus. Dazu gehört unter anderem das Auspacken und Interpretieren der Experimentbeschreibung oder die Dekodierung der Meßdaten — also die Ereignisrekonstruktion.

Ein- und Ausgabe laufen asynchron dazu ab. Das Einlesen des Datenstroms wird indirekt durch die Dekodier-Software gesteuert, die eventuelle Ausgabe von der experimentspezifischen Analyse-Software. Die Eingabe beschränkt sich dabei ebensowenig wie die Ausgabe nicht nur auf den in erster Linie wichtigen Datenaustausch über die Client-Server-Schnittstelle, sondern umfaßt ebenso Standard-I/O, Datei- oder Band-Ein/Ausgabe und die Kommunikation über Netzwerk. Das erlaubt die Verwendung der Analyse-Software auch in Off-line-Betrieb und bietet eine sehr gute Grundlage, sie auf mehrere Rechner zu verteilen.

8.1.3.2 Die Verarbeitung regulärer Ereignisdaten

Datensegmente, die reguläre Meßdaten eines Ereignisses enthalten, werden auf der Grundlage der zuvor ausgepackten bzw. von außen vorgegebenen Experimentbeschreibung weiter dekodiert. Wie bereits bei der Datenaufnahme spielt auch hier der logische Baum, der sich aus der Experimentbeschreibung ergibt und mit entsprechender Software für das Analyseprogramm verfügbar gemacht wird,

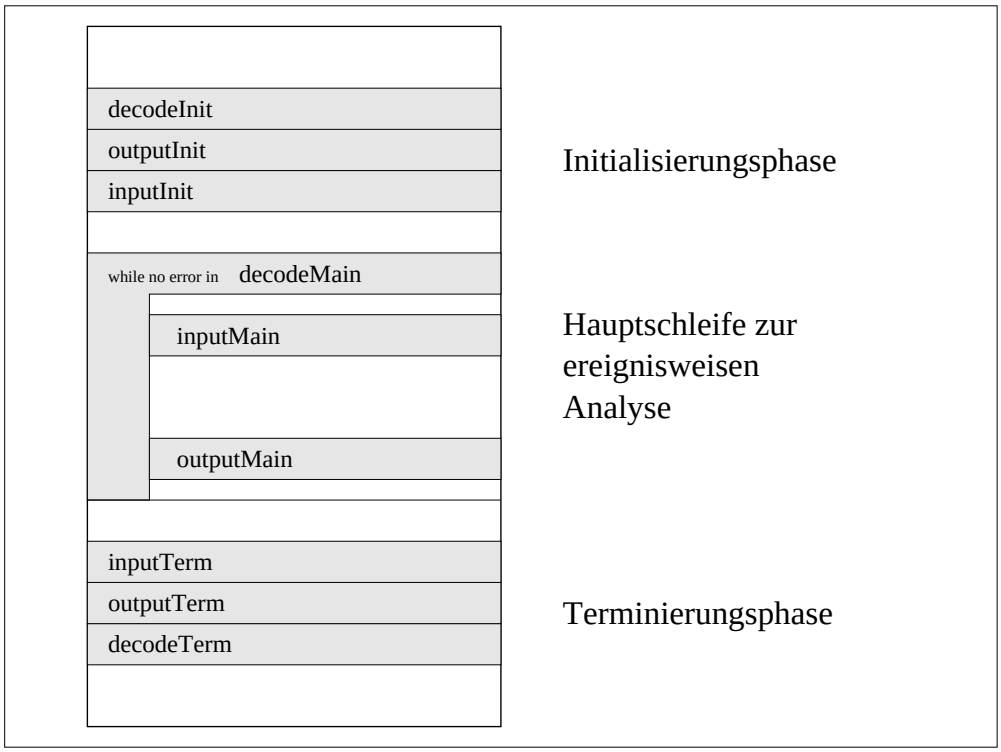

Abb. 8.2: Grundlegende Struktur eines Analyseprogramms unter Verwendung der **decode**-, **input**- und **output**-Objekte

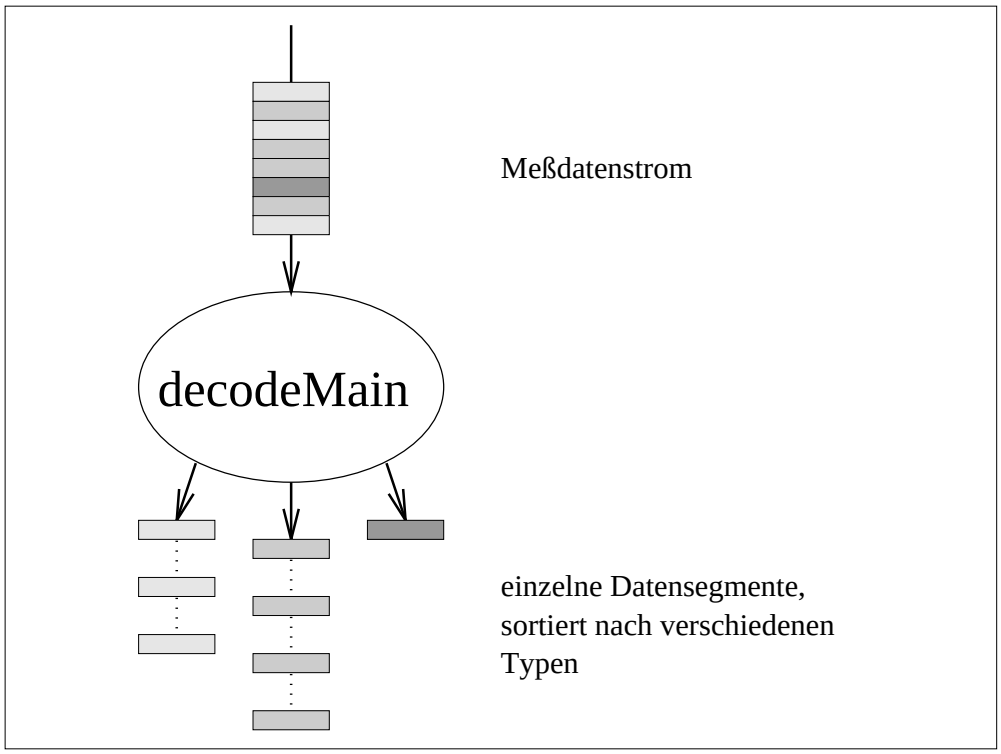

Abb. 8.3: Zerlegung eines Datenstroms in seine Segmente durch `decodeMain`

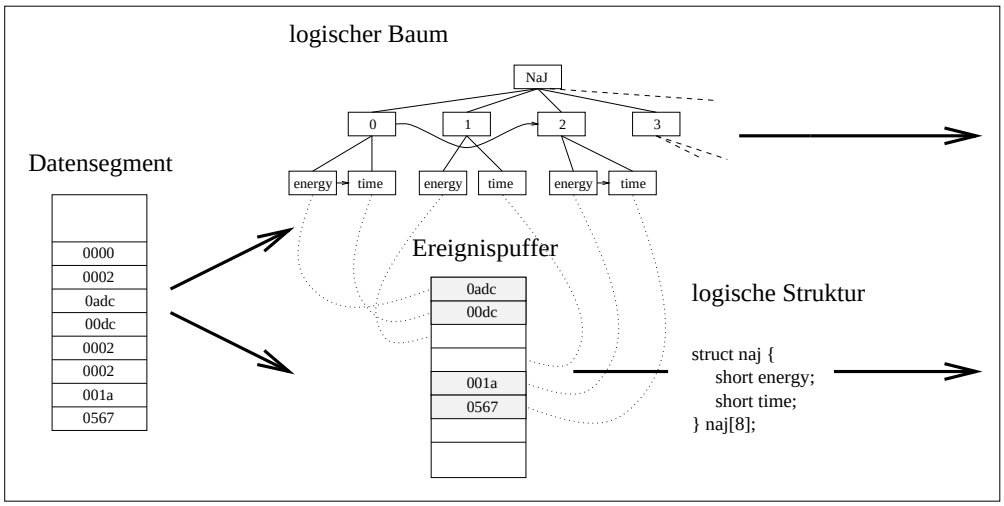

Abb. 8.4: Dekodierung von regulären Meßdaten. Die Informationen eines Datensegments werden dabei mithilfe des logischen Experimentbaumes dekodiert und sowohl darüber als auch einen Ereignispuffer für die weitere Benutzung verfügbar gemacht.

die zentrale Rolle. Er ist ein unmittelbares Abbild der grundlegenden C-Datenstruktur, die den individuellen logischen Aufbau eines Experiments beschreibt und jeder zu erfassenden Meßgröße einen Speicherplatz definierter Größe und einen bestimmten Datentyp (also damit bestimmte, für die Bearbeitung durch den Rechner wichtige Eigenschaften) zuordnet. Im folgenden wird diese Datenstruktur auch mit dem Begriff Maximalevent bezeichnet — also die größtmögliche Menge der Daten, die zu einem Ereignis gehörenen können. Der logische Baum macht den internen Aufbau der Datenstruktur und die Eigenschaften seiner Komponenten für das Anwenderprogramm zugänglich, was ansonsten innerhalb der Programmiersprache C/C++ selbst nicht unmittelbar möglich ist.

Der Beschreibungsbaum wird für allgemeine Aufgaben wie die Dekodierung der Daten verwendet, steht aber auch direkt für die Erledigung von speziellen Auswerteaufgaben zur Verfügung. Seine programminterne Realisierung besitzt die Möglichkeit, jeder logischen Einheit des Maximalevents ein Unterprogramm zu deren individuellen Bearbeitung zuzuordnen, das im Rahmen der Dekodierung aufgerufen wird, wenn das entsprechende Meßdatum bzw. die Datengruppe auch tatsächlich in den Daten eines Ereignisses vorkommt. Damit wird es möglich, zumindest einen Teil der Analyse vollkommen datengesteuert ablaufen zu lassen und so den Aufwand für das ständige, unnütze Abfragen von Meßwerten, die immer nur sporadisch auftreten, zu reduzieren.

Bei der Dekodierung werden alle zu einem Ereignis gehörenden Meßdaten über die Adreßinformation, die mit in den Rohdaten abgespeichert ist, individuell identifiziert und in einem die logische Experimentstruktur widerspiegelnden Datenpuffer so abgelegt, daß jedes einzelne Meßdatum von der aufrufenden Software über dessen logische Adresse angesprochen werden kann. Im einfachsten Fall kann das direkt mithilfe der C-Datenstruktur geschehen, die das Maximalevent repräsentiert. Sie erlaubt aus einem herkömmlichen C-Programm heraus

den wahlfreien Zugriff auf die Meßdaten eines Ereignisses mithilfe einfacher Variablenadressierung. Sie bildet grundsätzlich bei allen Aktivitäten, bei denen auf die Daten zugegriffen wird, eine einheitliche Adressierungsgrundlage. Das gilt sowohl für die On-line-Analyse als auch, wie schon früher beschrieben, für die Datenaufnahme. In beiden Fällen werden daher auch ähnliche, wenn nicht sogar identische Konstrukte verwendet. Sie bieten die Möglichkeit, sehr einfach Aktivitäten z. B. der Vorauswertung aus dem Bereich der Datenaufnahme in die On-line-Analyse und umgekehrt zu verlagern. Neben der unmittelbaren C-Struktur-Adressierung unter Ausnutzung der Compilation des Analyseprogrammes bietet die MECDAS-Software noch ein weiteres Verfahren, um über eine symbolische Adressierung auf die Daten eines Ereignisses zuzugreifen. Dabei wird ausschließlich von der im Beschreibungsbaum abgelegten Namensinformation Gebrauch gemacht.

Im Laufe des Dekodiervorgangs werden nicht nur die Meßwerte selbst sondern auch die dazugehörige Adreßinformationen mithilfe des Beschreibungsbaums in verketteten Listen festgehalten. Sie vervollständigen einerseits die zur Auswertung der Daten benötigte Information, stellen andererseits aber auch eine alternative Möglichkeit zur Verfügung, mit wenig Aufwand auf die tatsächlich erfaßten Meßdaten zuzugreifen. Insbesondere beim systematischen Bearbeiten der Meßdaten eines Ereignisses kann dieses Verfahren vorteilhaft genutzt werden, wenn die Zahl der gemessenen Werte klein im Vergleich zur Größe des Maximalevents ist. Das ist z. B. bei der Drei-Spektrometer-Anlage der Fall. Hier beträgt der Umfang der Meßwerte in der Regel etwa 1% der Größe des Maximalevents. Aber auch bei Adressierung der Meßdaten über die Datenstruktur ist die im Beschreibungsbaum abgelegte Adreßinformation wichtig, denn sie kann dazu genutzt werden, die Gültigkeit eines Datums zu überprüfen. Entsprechende Software steht dafür zur Verfügung.

Es ist nun Aufgabe des Anwenders, ein geeignetes Programm zu schreiben, in dem die Dekodierroutine gefolgt von Programmcode zur weiteren Verarbeitung der rekonstruierten Ereignisdaten in einer Schleife aufgerufen wird. Da dabei die C-Datenstruktur des Maximalevents als Schnittstelle fungiert, können die Meßdaten innerhalb des Analysecodes ohne zusätzliche Aktionen unmittelbar und mit einfachen C-Strukturzugriffen angesprochen werden. Damit ist ohne jegliche Detailkenntnisse der Interna von MECDAS und mit nur minimalen C-Programmierkenntnissen bereits ein einfaches Analyseprogramm realisierbar. Im Gegensatz zu einer speziellen Kommandosprache mit eingeschränkter Funktionalität läßt sich jedoch mit demselben Verfahren auch ein Analyseprogramm beliebiger Komplexität realisieren. Abbildung 8.5 zeigt ein einfaches Programmbeispiel. Es enthält i. w. die Ereignis für Ereignis durchlaufene Hauptschleife mit Aufruf der Dekodierroutine und dem anschließenden Verarbeiten der Meßdaten.

Um den Benutzer bei der Erstellung eines Analyseprogrammes zu entlasten und ihn von der Bearbeitung von bisher noch gar nicht genannten Routineaufgaben weitestgehend zu befreien, wurde die Software für Standardanwendungen erweitert. Sie umfaßt nicht nur einfache Bibliotheksfunktionen sondern bildet einen kompletten Programmrahmen, der nur noch durch drei experimentspe-

```
...
events = 0;
inputMain(...);
while ((stype = decodeMain(...)) > 0) {
        int     energy = 0, n = 0;

        switch (stype) {
        case FMTevent:
                break;
        case FMTendbuf:
                inputMain(...);
                continue;
                ...
        }
        for (i = 0; i < 8; ++i) {
                if (eventOk(naj[i].energy) &&
                    naj[i].energy > emin &&
                    naj[i].time > tmin[i] && naj[i].time < tmax[i])) {
                        energy += naj[i].energy;
                        n += 1;
                }
        }
        printf("%d: %d %d\n", ++events, energy, n);
}
...
```

Abb. 8.5: Ausschnitt aus dem vom Anwender zu erstellenden Programmcode eines Analyseprogrammes. Er enthält ein einfaches Beispiel für die Hauptschleife, die Ereignis für Ereignis abgearbeitet wird, und demonstriert den einfachen Zugriff auf die Meßdaten. Konkret wird hier die Summe der Energieinformation der angesprochenen Detektoren gebildet, falls gewisse Schwellen- bzw. Zeitbedingungen erfüllt sind, und das Ergebnis ereignisweise zusammen mit ergänzender Information ausgedruckt.

zifische Routinen zu ergänzen ist. Damit ist es für den Anwender nicht mehr nötig, ein von Grund auf neues Programm zu schreiben, in dem er sich je nach Komplexität der Anwendung auch um eine Reihe allgemeiner Verwaltungsaufgaben, die nichts mit dem Experiment zu tun haben, kümmern muß, wie z. B. die Ein- und Ausgabe, die Verwaltung von Speicher, die Initialisierung von Variablen und das korrekte Aktivieren von Software zum richtigen Zeitpunkt. Er kann sich statt dessen voll und ganz auf die experimentspezifischen Aufgabestellungen beschränken. Gleichzeitig reduziert sich damit auch die Fehleranfälligkeit des gesamten Programmes, da nun die Schnittstelle zwischen allgemeinem und experimentspezifischem Teil deutlich kompakter ist (vgl. Abb. 8.5 und 8.6). Dieses Programm ist dann vollständig datengesteuert. Die drei bereitzustellenden Routinen (`eventInit()`, `eventMain()`, `eventTerm()`) werden ausschließlich in Abhängigkeit vom Typ der Datensegmente des untersuchten Datenstroms aufgerufen. Der Benutzer kann nur innerhalb seiner drei Routinen den Programmablauf bestimmen. Trotz solcher Einschränkungen reicht dieser Mechanismus in vielen Fällen aus. Da jedoch nicht immer die dabei verwendeten Automatismen angebracht sind, kann dann doch auf die allgemeinere Lösung zurückgegriffen werden. Sie erlaubt es, Programmstruktur oder -ablauf frei zu bestimmen, die

```
...
#define cut(min, val, max) ((val) > (min) && (val) < (max))
...

void
eventInit()
{
        events = 0;
        ...
}

int
eventMain()
{
        int     energy = 0, n = 0, i;

        for (i = 0; i < 8; ++i) {
                if (eventOk(naj[i].energy) &&
                    cut(emin, naj[i].energy, INFINITE) &&
                    cut(tmin[i], naj[i].time, tmax[i])) {
                        energy += naj[i].energy;
                        n += 1;
                }
        }
        printf("%d: %d %d\n", ++events, energy, n);

        return (0);
}

void
eventTerm()
{
        ...
}
```

Abb. 8.6: Das vorangegangene Beispiel in alternativer Form unter Verwendung des **event**-Objekts und vereinfachender C-Präprozessor-Makros.

Ein/Ausgabe oder andere Routineaufgaben auf besondere Art vorzunehmen und bietet dem Benutzer insgesamt größere Freiheiten. Insbesondere die Einbindung in andere Software unterliegt dadurch keinen Einschränkungen.

8.1.3.3 Andere Datensegmenttypen

Neben den bisher besprochenen Datensegmenten, die reguläre Ereignisdaten enthalten, können im Datenstrom in der Regel auch Datensegmente anderer Art vorkommen. Diese enthalten neben der Experimentbeschreibung nähere Informationen über die Einzelmessung allgemeinerer Art wie symbolischer Name der Messung oder Zeitstempel, aber auch spezielle Angaben über das Experiment, die nicht im Rahmen der ereignisweisen Datenaufnahme erfaßt wurden. Das sind einerseits vom Benutzer gemachte Angaben i. w. im Klartext und andererseits Informationen über den Zustand der Meßapparatur, die die entsprechende

Steuerungs-Software gesammelt hat und während der Datenaufnahme in den Meßdatenstrom eingefügt wurde. Nicht weniger wichtig sind schließlich die Texte von Fehlermeldungen und Warnungen, die in den verschiedenen Phasen der Datenaufnahme ebenfalls in den Datenstrom eingefügt werden konnten.

All diese Informationen werden im Rahmen der Dekodierung analysiert und in geeigneter Form den weiteren Teilen des Analyseprogramms zur Verfügung gestellt. Die für die weitere Dekodierung bestimmter Datensegmenttypen wichtige Experimentbeschreibung wird, wie schon erwähnt, stets ausgewertet und steht für die verschiedensten Aufgaben der Analyse-Software zur Verfügung. Dabei wird sie oftmals indirekt über die Funktionalität ergänzender Unterprogrammbibliotheken genutzt. Die Dekodier-Software ermöglicht jedoch auch den direkten Zugriff auf die extrahierte Information, die sich im wesentlichen im Beschreibungsbaum niederschlägt. Der größte Teil der verbleibenden Datensegmenttypen enthält Information in Textform. Bis auf die systematisch erzeugten Fehlermeldungen und Warnhinweise entziehen diese sich jedoch einer allgemeinen, automatisierten Verarbeitung. Sie werden daher in der Regel lediglich zwischengespeichert und können bei Bedarf durch spezielle, evtl. für das jeweilige Experiment implementierte Algorithmen im experimentspezifischen Auswerteteil des Analyseprogramms weiterverarbeitet werden. Entsprechende Zugriffsmechanismen wurden dafür implementiert.

Eine Besonderheit stellen die Daten dar, die den Status der Experimentierapparatur enthalten. Im Gegensatz zu den Informationen aus den anderen, eben angesprochenen Datensegmenten enthalten sie ergänzende Meßdaten, die die bei der ereignisweisen Datenaufnahme erhaltenen Daten vervollständigen und für die Auswertung des Experiments unerläßlich sind. Für ihre Kodierung stehen zwei Alternativen zur Verfügung. Die erste sieht eine beliebige Verpackung dieser Daten vor. In diesem Fall kann die allgemeine Dekodier-Software nichts weiter tun, als diese Daten unverändert an spezielle Software weiterzureichen, die in der Lage ist, die Daten endgültig zu bearbeiten. Von dieser Möglichkeit wurde im Rahmen der Experimente an der Drei-Spektrometer-Anlage Gebrauch gemacht [Kram95]. Sie erlaubte eine sehr gute Unabhängigkeit bei der Entwicklung der Software für die verschiedenen Aufgabenbereiche. Die zweite Alternative nutzt die bereits bei der Verarbeitung der Ereignisdaten eingesetzten Methoden aus. Hier werden auch die Statusdaten in einer selbstbeschreibenden Form abgespeichert. Die Daten, die nach denselben Vorschriften wie die Ereignisdaten kodiert werden, werden dabei durch eine Beschreibung der Experimentierapparatur ergänzt.

8.1.4 Analyse-Filter

Programme zur Auswertung von Meßdaten lassen sich in zwei Gruppen aufteilen: bei der ersten erzeugt das Analyseprogramm aus dem eingelesenen Meßdatenstrom auch einen Datenstrom in der Ausgabe, bei der zweiten werden aus den Meßdaten unmittelbar Histogramme generiert. Die erste Gruppe repräsentiert typische Filterprogramme, die einen einlaufenden Datenstrom nach

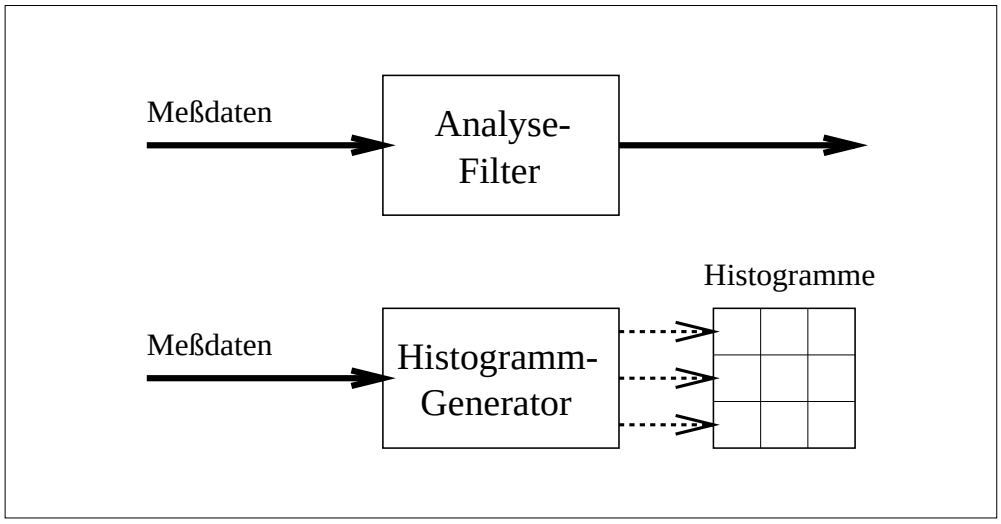

Abb. 8.7: Gegenüberstellung der beiden Grundtypen von Analyseprogrammen, die in der MECDAS-On-line-Analyse Verwendung finden.

bestimmten Kriterien bearbeitet und ihn dabei in einen neuen, auslaufenden Datenstrom umwandelt (Abb. 8.7). Dieser kann in Inhalt, Umfang oder Struktur sehr unterschiedlich ausgeprägt sein. Er wird entweder direkt für den Benutzer ausgegeben oder dient als Eingabe eines weiteren Programmes. Hier bietet der Pipe-Mechanismus von UNIX sehr einfach zu handhabende Möglichkeiten, eine beliebige Kette dearartiger Filterprogramme zu generieren, die jeweils eine Teilaufgabe der Analyse realisieren. Besonders interessant ist dabei, daß hier nicht nur speziell implementierte Programme Verwendung finden können, sondern auch allgemeine Werkzeuge oder Standard-Auswerte-Software.

Im Bereich der On-line-Analyse von Meßdaten finden sich eine Reihe von Aufgabenstellungen, die sich sehr gut für eine Filteranwendung eignen. Sie erlauben eine Verteilung der Analyse in unterschiedliche Programme, um z. B. allgemeine Aufgaben von experiment- oder benutzerspezifischen zu trennen. Im folgenden werden typische derartige Aufgaben beschrieben, für die in der MECDAS-Analyse-Software Vorkehrungen getroffen wurden.

8.1.4.1 Vorauswertung

Bei der Auswertung von unterschiedlichen Experimenten, die an ein und derselben Apparatur durchgeführt werden, gibt es oft experimentübergreifende Aufgaben, die ausschließlich durch die Apparatur definiert werden. Oftmals könnten derartige Aufgaben bereits im Rahmen der Datenaufnahme ausgeführt werden, wenn ihr Rechenaufwand das zuließe. Zu diesen Aufgaben gehören z. B. Eichkorrekturen von Meßwerten, Koordinatenbe- und -rückrechnung aus den gemessenen Zeiten einer Driftkammer oder eine Teilchenidentifikation. In vielen Fällen ist mit einer derartigen Auswertung auch eine Datenreduktion verbunden, d. h., es werden entweder unnütze Daten vollständig eliminiert (z. B. Nullenunterdrückung) oder die Information, die in einer großeren Anzahl von

Meßdaten steckt, auf das Wesentliche reduziert (z. B. Koordinatenberechnung). Gerade weil eine derartige Vorauswertung prinzipiell schon bei der Datenaufnahme vorgenommen werden kann, bietet es sich an, die resultierenden Daten nach demselben Verfahren wie die Eingangsdaten zu formatieren. Damit kann ein nachfolgendes Programm unabhängig davon, ob die Vorauswertung bereits unmittelbar bei der Datenaufnahme oder erst später vorgenommen wurde, die Daten einheitlich verarbeiten.

Während die Algorithmen zur speziellen Vorauswertung der Daten durch zusätzliche Software realisiert werden müssen, enthält die MECDAS-Analyse-Software neben dem Programmcode zur Dekodierung der Meßdaten auch solchen zur erneuten Formatierung auf der Basis einer Experimentbeschreibung, die von der für die Eingangsdaten abweichen kann.

8.1.4.2 Datenselektion

Hier werden komplette Ereignisse aufgrund vorgegebener Kriterien aus dem Datenstrom aussortiert. Solche Kriterien können beispielsweise ein- oder mehrdimensionale Schnitte (sogenannte Cuts) oder auch komplexere Auswahlbedingungen sein, die sowohl auf die Eingangsdaten als auch auf Ergebnisse weiterer Berechnungen angewendet werden können. Ein nach außen hin typisches Kennzeichen dieser Aufgabenstellung ist, daß Ein- und Ausgangsformat identisch sind. Aus diesem Grund ist die erneute, mit zusätzlichem Rechenaufwand verbundene Formatierung der Daten im Idealfall nicht notwendig. Die Filterfunktion beschränkt sich dann auf das bedingte Kopieren der Daten aus dem Eingangsdatenstrom in die Ausgabe. Auch dieser Mechanismus wird von der Analyse-Software unterstützt. Er wird automatisch aktiviert, wenn die Experimentbeschreibung für Ein- und Ausgabe identisch ist und die Ereignisdaten nicht manipuliert wurden.

8.1.4.3 Formatkonversion

Unabhängig von speziellen Auswerteaktivitäten können die Ergebnisse nicht nur im MECDAS-Format ausgegeben werden sondern auf beliebige Art und Weise. Das kann sowohl unter Zuhilfenahme von anderer Software zur Verarbeitung von kernphysikalischen Meßdaten wie z. B. der CERN-Bibliothek [Brun94] geschehen als auch über einfachen Mechanismen der Standard-Ein/Ausgabe wie mithilfe der `printf()`-Funktion in der Programmiersprache C. Im einfachsten Fall ist auch eine ausschließliche Formatumwandlung ohne Datenmanipulation möglich. Damit kann auf einfache Weise beliebige Auswerte-Software — insbesondere auch fremde, fest kodierte und in sich abgeschlossene Programme — in die Online-Analyse mit einbezogen werden.

MECDAS stellt standardmäßig bereits ein Programm zur Verfügung, das beliebige im MECDAS-Format verpackte Daten dekodiert und in einem alternativen Format ausgibt. Es kann unter dem Namen `decode` aufgerufen werden und bietet einige einfache Ausgabe-Formate zur Auswahl, die über Programmparameter

beim Start selektiert werden können. `decode` bietet einerseits die Möglichkeit, wie erwähnt, sehr einfach Fremd-Software zur Analyse von MECDAS-Daten zu verwenden, stellt aber andererseits auch ein interaktives Werkzeug dar, das dem Benutzer die Gelegenheit gibt, die Meßdaten ohne jegliche Analyse-Software zu untersuchen. Dabei können die Daten zusammen mit numerischer oder symbolischer Adreßinformation im Klartext ausgegeben werden und unmittelbar durch den Benutzer inspiziert werden oder durch Standard-UNIX-Werkzeuge wie `grep`, `sed`, `awk` oder `wc` bearbeitet, gefiltert und auf einfache Art ausgewertet werden. Daneben stellt `decode` selbst noch einige primitive, experimentunabhängige Auswertemöglichkeiten zur Verfügung.

8.1.4.4 Kombination mehrerer Verfahren

Programme, die die Filterverfahren wie in den vorhergehenden Abschnitten beschreiben verwenden, können relativ frei miteinander kombiniert werden. In der Regel werden die einzelnen Programme dabei bereits Mischungen einzelner Mechanismen untereinander in Kombination mit Software für bestimmte Auswerteaufgaben sein. Schließlich können sie noch durch die im folgenden Abschnitt beschriebenen Methoden zur Histogrammierung der Meßdaten ergänzt werden (s. Abb. 8.8).

8.1.5 Histogrammierung

Ein wichtiger Mechanismus bei der Auswertung von Meßdaten ist die Generierung von Histogrammen aus Rohdaten, Zwischen- oder Endergebnissen, die aus Berechnungen, durch Cuts und Datenreduktionen aus den Meßwerten erzeugt wurden. Histogramme sind eine spezielle Diagrammform die der Darstellung von Häufigkeitsverteilungen von Meßergebnissen (sog. Variablen) dient. Sie setzen sich aus einer bestimmten Anzahl einzelner Kanäle zusammen, die bei diskreten Variablen die möglichen bzw. interessierenden Werte, die die darzustellende Variable annehmen kann, repräsentieren und bei kontinuierlichen Variablen entsprechende Werteintervalle von definierter Breite darstellen. Sie können ein- oder auch mehrdimensional organisiert sein. In der Regel werden Histogramme

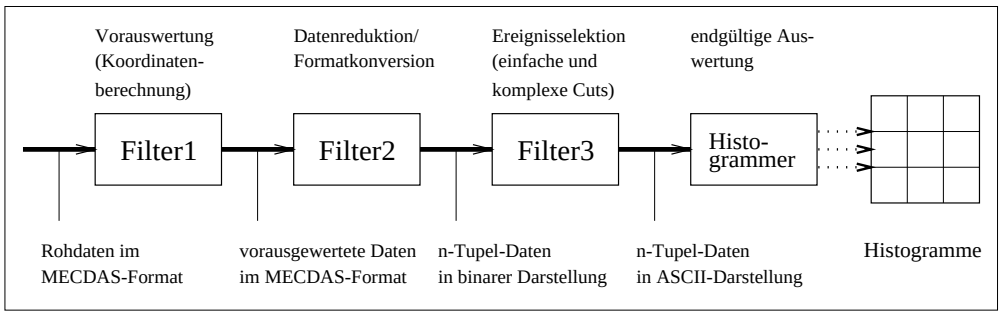

Abb. 8.8: Ausnutzung der Filterfunktionalität durch Kombination mehrerer, elementarer Filterprogramme zu einer leistungsfähigen Analyse.

mithilfe entsprechender Software als Balkendiagramme graphisch dargestellt. Bei der Generierung von Histogrammen lassen sich grundsätzlich drei Phasen unterscheiden:

1. Erzeugen eines Histogramms mit definierten Eigenschaften für eine zu untersuchende Variable
2. Manipulieren der einzelnen Kanalinhalte — in der Regel ihr allmähliches Aufakkumulieren anhand der Werte der zu untersuchenden Variablen
3. Vernichten eines Histogramms

Während des Auswertevorgangs ist ausschließlich die zweite Phase von Bedeutung. Damit unterscheidet sich die Histogrammierung technisch von Filterverfahren, wie sie in Abschnitt 8.1.4 beschrieben sind. Anstatt die laufenden Ergebnisse der Analyse auszugeben und daraus einen wachsenden Datenstrom zu generieren, wird lediglich der Zustand der als definierte Objekte existierenden Histogramme modifiziert. Es wird also nichts Neues erzeugt, sondern nur Existierendes verändert. Das hat nun weitreichende Konsequenzen für die Analyse-Software, insbesondere wenn diese im On-line-Betrieb eingesetzt werden soll. Denn während ein Filterprogramm seine Ergebnisse per definitionem explizit ausgibt, müssen bei der Histogrammierung zusätzliche Mechanismen zur Verfügung gestellt werden, um dem Benutzer die Histogramminhalte zugänglich zu machen; im On-line-Betrieb sollte das sogar während des Auswertevorgangs möglich sein.

Im Rahmen von MECDAS wurde nun ein Histogramm-Paket entwickelt, das dem Benutzer für die On-line-Analyse einerseits ausreichende Hilfsmittel zur Erzeugung und Verwaltung von geeigneten Histogrammen mitgibt und andererseits ihre einfache Benutzung innerhalb der bisher erläuterten Analyse-Software erlaubt. Es trägt besonders der Tatsache Rechnung, dem Benutzer mit minimalem Aufwand gerade im On-line-Betrieb vollen Zugriff auf die in Bearbeitung befindlichen Histogramme zu gewähren. Die Software erlaubt die Verteilung der anfallenden Aufgaben auf mehrere Programme, so daß ohne gegenseitige Beeinflussung parallel zur Akkumulierung der Histogramme in einem On-line-Analyseprogramm eine graphische Darstellung der Histogramme oder eine Bearbeitung ihrer Inhalte vorgenommen werden kann. Damit war eine klare Trennung der Aufgaben zur eigentlichen Analyse der Meßdaten und der Verwaltung der Histogramme möglich. Insbesondere erlaubte dieses Konzept eine sehr gute Absonderung von den in der Regel sehr komplexen Aufgaben, die mehr in den Bereich der Benutzeroberfläche fallen. So konnte das Analyseprogramm auf die ausschließliche Bearbeitung der Auswertung beschränkt werden. Alles andere geschieht außerhalb dieses Programmes und wird durch in sich abgeschlossene Software unterstützt. Diese Maßnahmen trugen erheblich mit dazu bei, die Erstellung von On-line-Analyse-Software in MECDAS einfach zu halten und auf ein Minimum an Aufwand durch den Anwender zu reduzieren, der sich im wesentlichen auf die experimentspezifische Aufgaben beschränken kann. Das Histogramm-Paket wird ergänzt durch Software zur graphischen Darstellung der Histogramme. Beide Pakete werden in den folgenden Abschnitt beschrieben.

8.2 Das Histogramm-Paket

8.2.1 MECDAS-Histogramme

8.2.1.1 Grundprinzip

Wie bereits im vorhergehenden Abschnitt erläutert war für die On-line-Analyse einfach zu handhabende Software zur Erzeugung, Verwaltung und Benutzung von Histogrammen erforderlich, die einerseits ein effizientes Bearbeiten von Histogrammen gestattet andererseits die Möglichkeit der Entkopplung der verschiedenen Aufgaben untereinander und insbesondere von der eigentlichen Auswertung der Meßdaten bot. Die daraufhin im Rahmen von MECDAS entwickelte Software [Heil91a] löst diese Problemstellungen durch die Verteilung unterschiedlicher Aufgaben auf verschiedene Programme (Abb. 8.9) zur

- Verwaltung von Histogrammen
- Analyse der Meßdaten und Generierung von Histogrammen
- Bearbeitung der Histogramme
- graphischen Darstellung der Histogramme

unter Ausnutzung der unter UNIX aber auch vielen anderen Betriebssystemen zur Verfügung stehenden Möglichkeit des **Shared Memory**, also Arbeitsspeicher, der von mehreren unabhängigen Programmen gleichzeitig angesprochen werden kann. In solchen Shared-Memory-Bereichen werden die Histogramm-Daten abgelegt und können von verschiedenen Programmen über herkömmliche Speicherzugriffsmechanismen ebenso wie programminterner Speicher adressiert und benutzt werden.

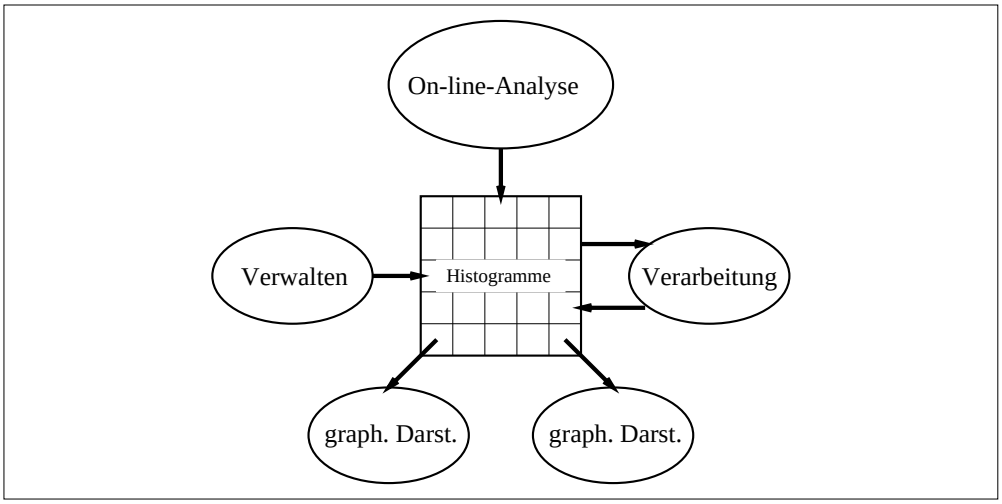

Abb. 8.9: MECDAS-Histogramme als programmübergreifende Objekte.

8.2.1.2 Alternativen

Alternative Lösungsmöglichkeiten besitzen im Vergleich zu diesem Verfahren deutliche Nachteile. Die zwei wichtigsten sind:

- Vereinigung aller Aufgaben inklusive Analyse in einem einzigen Programm
- Aufgabenverteilung unter Verwendung von Client-Server-Mechanismen

Bei der ersten Alternative entfällt zwar das Problem, die in der Regel sehr umfangreiche Histogramm-Information mehreren unabhängigen Programmen gleichzeitig und in konsistenter Form zur Verfügung zu stellen, doch erfordert die Umsetzung der im On-line-Bereich vorliegenden Anforderungen, die die gleichzeitige Berarbeitung verschiedener Aufgaben notwendig machen, mit herkömmlichen Programmiermechanismen einen sehr hohen Implementationsaufwand. Eine in der Off-line-Analyse [Brun91] [Offe93] häufig eingesetzte und dort auch ausreichende sequentielle Abarbeitung der einzelnen Aufgaben genügt den Anforderungen im On-line-Fall nicht, da hier z.B. ständig zwischen Auswertung und graphischer Darstellung der Meßergebnisse hin- und hergeschaltet werden müßte, um den aktuellen Verlauf der Messung zu verfolgen. Das führt aber zwangsläufig zu einer sehr schlechten Ausnutzung der Rechenleistung des Analyserechners und ist damit sehr ineffizient.

Eine derartige Sequentialisierung ist lediglich dann nicht mehr zwingend, wenn Mechanismen zur quasiparallelen (oder sogar echt parallelen) Ausführung von Teilen eines abgeschlossenen Programms verfügbar sind, wie z.B. bei der Nutzung von sog. Threads in modernen Betriebssystemvarianten. Ihre Verwendung beschränkt den Programmieraufwand wieder auf ein vertretbares Maß. Zum Zeitpunkt der Entwicklung der Histogramm-Software standen jedoch solche Verfahren überhaupt noch nicht zur Verfügung, und auch inzwischen können sie noch nicht auf allen der bei der Datenerfassung eingesetzten Rechner, die für die On-line-Analyse in Frage kommen, verwendet werden.

Unabhängig davon macht zudem die Konzentration aller Aufgaben auf ein einziges Programm die Software unübersichtlich und fehleranfällig, insbesondere wenn dieses Programm keinen eigenen Interpreter zur Bearbeitung der experiment- und benutzerspezifischen Analysevorschriften besitzt, sondern dafür zu einem wesentlich Teil Programmcode des Benutzers enthält, der immer wieder modifiziert und angepaßt werden muß. Dieses Verfahren wurde daher für die Implementation der Histogramm-Software nicht verwendet.

Für die zweite Alternative ist eine Realisierung denkbar, in der ein zentrales Programm die komplette Verwaltung der Histogramme vornimmt. Andere Programme können nur über dieses dann als Server fungierende Programm auf die Histogramme zugreifen. Dabei müssen IPC und bei Bedarf auch rechnerübergreifende Kommunikationsmechanismen eingesetzt werden. Diese vergrößern den Aufwand beim Zugriff auf die Histogramm-Daten in der Regel deutlich gegenüber programminternen Zugriffen, da hier die Daten nicht nur mehrfach umkopiert werden müssen, sondern auch der Kommunikations-Overhead nicht unerheblich beiträgt.

Bei relativ seltenen Zugriffen typischerweise im Zusammenhang mit Benutzerinteraktionen bzw. bei der Verarbeitung vollständiger Histogramme ist dieser Aufwand im Vergleich zu den restlichen Aktivitäten der Software oftmals vernachlässigbar, auch wenn dabei die Daten kompletter Histogramme zu übertragen sind. Problematisch sind jedoch Zugriffe auf einzelne Kanäle, die extrem häufig im Zusammenhang mit dem akkumulierenden Aufbau der Histogramme im Ablauf der Analyse vorkommen. Auch bei sehr aufwendiger Optimierung des dazu notwendigen Informationsaustauschs, der evtl. das hier sehr ungünstige Verhältnis von Verwaltungs- zu Nutzaufwand verbessern kann, jedoch andererseits das Laufzeitverhalten verschlechtert, ist der damit verbundene Aufwand immer noch ein Vielfaches dessen, was die entsprechenden programminterne Zugriffe benötigen. Dieser zeit- und rechenintensive Aufwand läßt sich nur vermeiden, wenn das Akkumulieren der Histogramme nicht in einem Klienten sondern durch den Server selbst vorgenommen wird. D. h., dieser sorgt nicht nur für die Verwaltung der Histogramme, sondern muß auch die komplette Analyse übernehmen. Damit nähert man sich wieder einer eher monolithischen Lösung mit den bereits angesprochenen Nachteilen.

8.2.1.3 Histogramm-Objekte

Die letzendlich realisierte Lösung nutzt die Vorzüge beider Alternativen ohne deren Nachteile in Kauf nehmen zu müssen. Darüberhinaus enthält sie noch einige Besonderheiten, die im Rahmen der On-line-Analyse sehr hilfreich sein können. Insbesondere bei der Verwaltung der Histogramme verfolgt sie ein sehr flexibles Konzept.

MECDAS-Histogramme stellen eigenständige Objekte dar, deren Daten im Gegensatz zu den beschriebenen Verfahren nicht im Arbeitsspeicher eines Programmes abgelegt sind. Damit ist die Existenz eines Histogramms auch nicht an das Vorhandensein bestimmter Programme gebunden. Sie nutzen vielmehr einfache Betriebssystemsmechanismen, die es erlauben, elementare Systemobjekte wie Dateien oder Shared-Memory-Segmente weitgehend permanent anzulegen. Lediglich für ihre Erzeugung, ihre spätere Manipulation und das Vernichten wird Software benötigt, die jedoch nicht zwangsläufig Bestandteil eines einzigen Programmes sein muß, sondern sich jeweils auf eigenständige, voneinander unabhängige Programme verteilen kann.

Die Grundlage von MECDAS-Histogrammen bilden herkömmliche Plattendateien. Sie bieten neben ihrer Persistenz eine Reihe von Eigenschaften und Methoden, die sie an die Histogramme vererben. Dazu gehören einerseits der Name, unter dem eine Datei angesprochen werden kann, deren Größe und Zugriffsberechtigungen. Auch Zeitinformationen wie Erzeugungs-, Modifikations- und Zugriffsdatum sind Bestandteil der Eigenschaften. Andererseits stehen eine Reihe von Verwaltungsmechanismen wie Erzeugung und Löschen von Dateien, Umbenennen, Ändern von Größe und Zugriffsberechtigungen usw. zur Verfügung. Diese Eigenschaften und Methoden werden ergänzt durch die in den Dateien enthaltenen Daten und die Mechanismen, auf diese zuzugreifen bzw. sie zu modifizieren, also Lese- und Schreiboperationen.

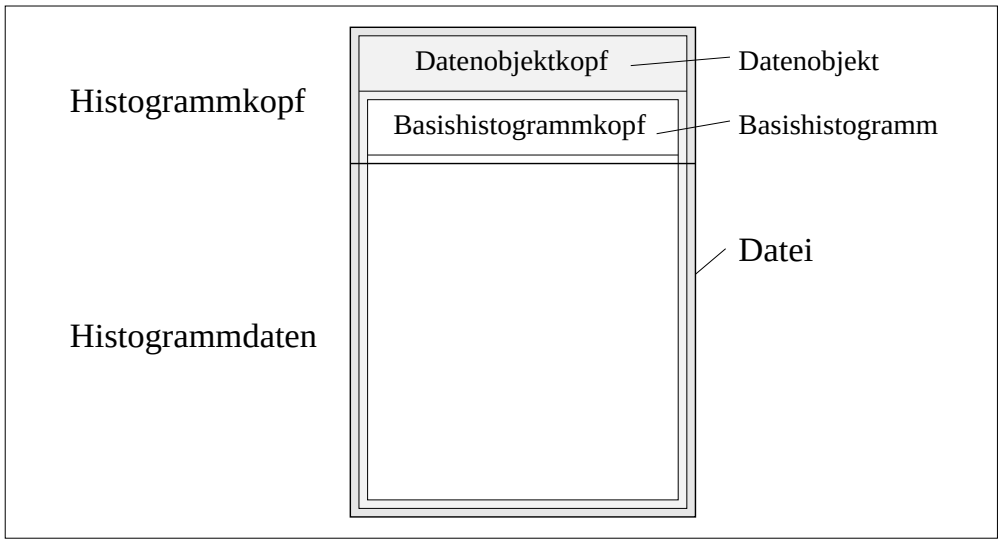

Abb. 8.10: Grobstruktur eines Histogramms, das aus einem Histogramm-Kopf und dem Datenbereich besteht. Der Kopf ist selbst noch einmal aus zwei Telen zusammengesetzt. Er besteht aus einem allgemeinen Anteil, der beliebige Datenobjekte, die nach einem einheitlichen Mechanismus verwaltet werden sollen, beschreibt, und einem zweiten Teil, der speziell die Eigenschaften eines Histogramms abdeckt.

Bei der Histogrammierung in MECDAS wird nun diese Basisfunktionalität genutzt. Sie wird ergänzt durch neue Funktionalität für die spezielle Aufgabenstellung unter Zuhilfenahme des Dateiinhalts in Kombination mit entsprechender Software. Der Inhalt einer Histogramm-Datei teilt sich in zwei Bereiche auf:

1. Histogramm-Kopf
2. Histogramm-Daten

Der Histogramm-Kopf ist durch eine C-Datenstruktur definiert, mit deren Hilfe Histogramm-Objekte grundsätzlich charakterisiert und die speziellen Eigenschaften einzelner Histogramme bestimmt werden. Die Histogrammier-Software nutzt die in dieser Struktur abgelegte Information für die korrekte Ausführung ihrer Aufgaben und ist in der Lage, einzelne Komponenten und damit die Eigenschaften eines Histogramms zu verändern. Auf diese Weise wird eine Objektklasse für Histogramme nach den Prinzipien der objektorientierten Programmierung definiert, deren Instanzen die eigentlichen Histogramme mit ihren speziellen Eigenschaften sind, und für die es eine Reihe von Methoden gibt.

Im zweiten Bereich innerhalb der Histogramm-Datei sind die eigentlichen Histogramm-Daten als Menge aller Kanalinhalte abgelegt, die in ihrer Gesamtheit das Histogramm repräsentieren. Das Histogramm-Objekt bietet nun spezielle Mechanismen, mit deren Hilfe effiziente und wahlfreie Zugriffe auf diese Daten vorgenommen werden können. Dabei wird ein besonderer Mechanismus des UNIX-Betriebssystems ausgenutzt, der es erlaubt, reguläre Dateiein in den Arbeitsspeicher eines Programmes einzublenden und Zugriffe auf die einzelnen

darin enthaltenen Daten über herkömmliche, entsprechend effiziente Speicherzugriffe auszuführen. In älteren UNIX-Versionen, in denen dieser Mechanismus noch nicht zur Verfügung steht, mußte er mit Hilfe gesonderter Shared-Memory-Segmente simuliert werden. In diesem Fall befindet sich eine Arbeitskopie der Histogramm-Daten im Arbeitsspeicher. Die Software sorgt dafür, daß stets mit dieser Kopie gearbeitet wird; die Datei selbst wird in der Regel nicht manipuliert. Lediglich auf explizite Anweisung wird die Arbeitskopie auf die Plattendatei zurückgeschrieben. Dieses Verfahren verhält sich im wesentlichen identisch zu dem ursprünglichen Verfahren. Es müssen lediglich Einschränkungen in der Flexibilität und der Größe des für Histogramme verfügbaren Speicherbereichs hingenommen werden, die jedoch in der Praxis selten von Belang sind.

8.2.1.4 Eigenschaften der Histogramme

Jedes Histogramm besitzt einen Namen, mit dem es symbolisch adressiert und eindeutig identifiziert werden kann. Da es als Datei innerhalb des UNIX-Filesystems abgelegt ist, gelten für den Histogramm-Namen dieselben Regeln wie für Dateinamen. Die Ordnungsmechanismen des Filesystems erlauben es, verschiedene Histogramme in unterschiedlichen Verzeichnissen unterzubringen und so leicht hierarchisch zu organisieren. Zum Ansprechen der Histogramme können dabei alle erlaubten Varianten für eine absolute oder relative Adressierung verwendet werden. Neben dem Namen besitzen Histogramme auch alle anderen Eigenschaften einer regulären Datei; von besonderer Bedeutung sind dabei die Zugriffsberechtigungen.

Aus dem Histogramm-Kopf ergeben sich einige für die Anwendung wichtige Eigenschaften. Dazu gehört z. B. eine Kennzeichnung, die die Histogramm-Datei von herkömmlichen Dateien einerseits und anderen MECDAS-Datenobjekten andererseits unterscheidet. Sie selektiert die für die Bearbeitung solcher Dateien zur Verfügung stehenden Methoden. Der Name des Rechners, auf dem die Histogramme angelegt wurden, ist in einem verteilten System ebenso wie eine spezielle Identifikationsnummer für korrekte Funktion notwendig. Die Kennzeichnung der Byte-Orientierung, mit der die Daten abgelegt wurden, erlaubt den Austausch der Histogramm-Dateien zwischen Rechnern verschiedener Architekturen.

Für unterschiedlichste Zwecke stehen Software-Versionsnummer, Zeitstempel, Größe des Datenbereichs und weitere allgemeine Informationen zur Verfügung. Speziell zur Beschreibung der Histogramme dient ein Histogramm-Typ, mit dem z. B. festgelegt wird, ob die Kanalinhalte ganze Zahlen oder Fließkommazahlen sind, damit die Software die korrekten Operationen ausführen kann. Von ganz besonderer Wichtigkeit sind schließlich Informationen, die unmittelbar das Histogramm beschreiben. Das sind im wesentlichen die Anzahl der Kanäle und der Wertebereich, der durch die Grenzen eines halboffenen Intervalls $[min, max[$ spezifiziert wird. Die eigentlichen Daten sind im Anschluß an den Histogramm-Kopf abgelegt. Sie sind als ein- oder zweidimensionale Datenfelder organisiert (s. [Kryg95c]).

`hcreate()`	Histogramm-Konstruktor
`hdelete()`	Histogramm-Destruktor
`hopen()`	Öffnen eines existierenden Histogramms
`hclose()`	Schließen eines existierenden Histogramms
`halloc()`*	Erzeugen von Shared Memory für Histogramme
`hrealloc()`*	Veränderung der Größe von Histogramm-Shared Memory
`hfree()`*	Vernichten von Histogramm-Shared Memory
`hcheck()`	Konsistenzüberprüfung
`hmem()`	Information über Speicherverbrauch eines Histogrammes
`hsize()`	Information über verwendete Wortgröße
`hlen()`	Information über Kanalanzahl
`hname()`	Information über Histogramm-Name
`hcomment()`	Zugriff auf für den Benutzer frei verfügbaren Speicherbereich im Histogramm-Kopf
`hhost()`	Information über Hostnamen des Rechners, auf dem die Histogramme residieren
`hdata()`	unmittelbarer Zugriff auf Histogramm-Daten
`hxmin()`	Information über untere Kanalgrenze der ersten Dimension
`hxmax()`	Information über untere Kanalgrenze der ersten Dimension
`hxlen()`	Information über Kanalzahl der ersten Dimension
`hymin()`	Information über untere Kanalgrenze der zweiten Dimension
`hymax()`	Information über untere Kanalgrenze der zweiten Dimension
`hylen()`	Information über Kanalzahl der zweiten Dimension
`hsave()`*	Retten der Arbeitskopie eines Histogramms aus dem Shared Memory in eine Datei

Tab. 8.1: Die wichtigsten Methoden zur Verwaltung von MECDAS-Histogrammen.

8.2.2 Die Histogramm-Bibliothek

Um MECDAS-Histogramme einfach und effizient benutzen zu können, wurde ein Satz von Unterprogrammen implementiert, die in einer Bibliothek zusammengefaßt sind. Sie werden ergänzt durch eine Reihe von Präprozessor-Makros, die insbesondere die Zugriffe auf die Histogramm-Daten einerseits effizient halten andererseits aber soweit kapseln, daß dafür keine Detailkenntnisse notwendig sind. Die Software realisiert damit eine nach OOP-Prinzipien organisierte Histogramm-Objektklasse. Ihre Methoden lassen sich grundsätzlich in drei Gruppen unterteilen, die im folgenden kurz erläutert werden sollen.

8.2.2.1 Verwaltung von Histogrammen

Die erste Gruppe dient den elementaren Aufgaben zur Nutzung von Histogrammen. Sie umfaßt Software zum Anlegen und Löschen von Histogrammen, also den Konstruktur und Destruktor für Histogramm-Objekte, und Routinen und Makros, die einerseits Programmen den Zugriff auf komplette Histogramme gewähren, die bereits existieren und von anderen Programmen erzeugt wurden, andererseits den Zugriff auf Histogramm-Kopf und -daten regeln und da-

`hread()`	Lesen der Histogramm-Daten
`hwrite()`	Schreiben der Histogramm-Daten
`hchan()`	Adressierung des Inhalts eines definierten Histogramm-Kanals
`hinc()`	Inkrementieren eines Kanalinhalts um 1
`hadd()`	Erhöhen eines Kanalinhalts um einen spezifizierten Wert
`hset()`	Setzten eines Kanalinhalts auf einen bestimmten Wert
`hget()`	Lesen des Inhalts eines bestimmten Histogramm-Kanals

Tab. 8.2: Methoden für Zugriff und Manipulation einzelner Histogramm-Inhalte

mit die Methoden zum Bestimmen und zur Manipulation der Histogramm-Eigenschaften darstellen. Tabelle 8.1 gibt einen Überblick über die verfügbare Software. Die markierten Funktionen nehmen darin eine Sonderstellung ein. Sie repräsentieren nicht unmittelbar Histogramm-Methoden, sondern dienen der Verwaltung von Shared Memory, das unter bestimmten Umständen für eine korrekte Funktion notwendig ist.

8.2.2.2 Benutzung der Histogramme

Der kontrollierte Zugriff auf die Histogramm-Inhalte wird im wesentlichen durch drei Funktionen realisiert. Die ersten beiden dienen dem effizienten Zugriff auf ein komplettes Histogramm, während die dritte die Möglichkeit bietet, einzelne Histogramm-Kanäle zu addressieren. Sie wird in einer Reihe von Methoden verwendet, um die entsprechenden Kanalinhalte zu lesen oder zu modifizieren. Diese Methoden sind in der bisherigne C-Implementation der Bibliothek aus Effizienzgründen als Makros realisiert und stellen vorerst nur einfache Prototypen für die endgültige Lösung als allgemeine Zugriffsmechanismen für die Inhalte von Histogrammen dar. Tabelle 8.2 gibt einen Überblick über die wesentlichen Bestandteile dieser Software.

8.2.2.3 Bearbeitung von Histogrammen

Die letzte, bisher nur rudimentär implementierte Gruppe umfaßt Software zur Bearbeitung kompletter Histogramme. Sie beschränkt sich im Gegensatz zur zweiten Gruppe nicht auf Zugriffe auf einzelne Datenkanäle, sondern betrachtet ein Histogramm als abgeschlossenes Objekt, das in seiner Gesamtheit eine Reihe

`hall()`	Anwendung eines spezifizierten Algorithmus auf alle Histogramm-Kanäle
`hbox()`	Anwendung eines spezifizierten Algorithmus auf einen ausgewählten Histogramm-Bereich
`hint()`	Integrieren eines ausgewählten Histogramm-Bereichs
`hplus()`	Addieren zweier Histogramme
`hminus()`	Subtrahieren zweier Histogramme

Tab. 8.3: Methoden zur Bearbeitung von Histogrammen als abgeschlossene Objekte

`hcreate`	Histogramm-Konstruktor
`hdelete`	Histogramm-Destruktor
`halloc`*	Erzeugen von Shared Memory für Histogramme
`hrealloc`*	Veränderung der Größe von Histogramm-Shared Memory
`hfree`*	Vernichten von Histogramm-Shared Memory
`hcheck`	Konsistenzüberprüfung
`hfind`	Auffinden von Histogrammen in einem Verzeichnisbaum
`hread`	Lesen der Histogramm-Daten
`hwrite`	Schreiben der Histogramm-Daten
`hcopy`	Kopieren des Inhalts eines Histogramms auf ein zweites
`hclear`	Löschen der Histogramm-Daten
`hmem`	Information über Speicherverbrauch eines Histogrammes
`hsize`	Information über verwendete Wortgröße
`hint`	Integrieren eines ausgewählten Histogramm-Bereichs
`hinfo`	Information über die Eigenschaften eines Histogramms
`hparam`	Information über die Eigenschaften eines Histogramms
`hval`	Ausgabe der Kanalinhalte eines ausgewählten Histogramm-Bereichs
`hsave`*	Retten der Arbeitskopie eines Histogramms aus dem Shared Memory in eine Datei

Tab. 8.4: Kommandos zur Verwaltung, Manipulation und Bearbeitung von MECDAS-Histogrammen

von Eigenschaften besitzt, die bestimmt bzw. manipuliert werden können. Die Software definiert einfache Operationen auf diesen Histogrammen, die die Basis für eine eigene Histogramm-Arithmetik bilden.

8.2.3 Werkzeuge zur Histogramm-Verwaltung

Ein Grundprinzip der On-line-Analyse im Rahmen von MECDAS ist die Verteilung der Anwendung auf mehrere Programme. Sie verleiht der Software die besonders im On-line-Bereich wichtige Fähigkeit, mehrere Aufgaben parallel und so weit wie möglich unabhängig voneinander zu bearbeiten. Das ist in erster Linie von Bedeutung, um während der Ausführung des Analyseprogrammes stets Zugriff auf dessen Auswerteergebnisse wie Histogramme zu gewährleisten, kann aber daneben noch für weitere Aufgabenstellungen genutzt werden. Wie bereits bei der Datenerfassung konnte auch im Rahmen der On-line-Analyse eine konsequente Aufteilung vieler elementarer Aufgaben auf einzelne Programme erreicht werden mit dem Ziel, Implementationsaufwand zu reduzieren, indem von den bereits vom Betriebssystem zur Verfügung gestellten Hilfsmitteln Gebrauch gemacht wird. Zentrale Rolle spielt auch hier die UNIX-Shell, deren Nutzung es vorerst erübrigte, einen speziellen Kommandointerpreter für die Steuerung der On-line-Analyse zu programmieren. Sie bot vielmehr eine leistungsfähige Grundlage zur interaktiven Bedienung des Systems als auch zur Bereitstellung einfacher, durch den Anwender benutzbarer Programmiermechanismen zur Erledigung von Routineaufgaben und zur Realisierung von Automatismen.

Somit wurde es möglich, bei Bedarf das eigentliche, experimentspezifische Analyseprogramm, das auf herkömmliche Art programmiert werden muß, auf ein Minimum zu reduzieren und damit auch übersichtlich zu halten. An die Stelle eines einzigen, monolithisch aufgebauten Programms tritt damit ein verteiltes Analysesystem, das auf der Basis eines modularen Baukastensystems mit wenig Aufwand unterschiedlichen Anforderungen angepaßt werden kann. In diesem Zusammenhang bot sich an, auch die im Rahmen der On-line-Analyse anfallenden Verwaltungsaufgaben entsprechend über Kommandos verfügbar zu machen. Hierfür erlaubte das der Histogramm-Software zugrundeliegende Konzept eine einfache Aufteilung der Aufgaben auf einzelne Programme. Fast jeder Funktion aus der Histogramm-Bibliothek konnte dabei, soweit sinnvoll, unmittelbar ein entsprechendes Kommando gegenübergestellt werden. Tabelle 8.4 zeigt eine Überblick über die wichtigsten Kommandos, die zur Histogramm-Verwaltung realisiert wurden. Einige von ihnen vereinen die Funktionalität mehrerer Bibliotheksfunktionen, andere stellen bereits zusammengesetzte Kommandos dar.

Jedes dieser Kommandos kann grundsätzlich zu jedem Zeitpunkt aufgerufen werden. Bei einigen, insbesondere solchen, die Manipulationen an den Histogrammen vornehmen, macht jedoch ihre Benutzung nur vor bzw. nach einem Analyselauf Sinn. Bei diesen kann es unter Unständen zu Seiteneffekten kommen, da bisher auf eine Synchronisierung der Zugriffe verschiedener Programme auf ein- und dasselbe Histogramm verzichtet wurde.

8.2.4 Das Programm `display`

Die im vorangegangenen Abschnitt beschriebenen Kommandos zu Verwaltung und Bearbeitung von Histogrammen werden durch ein wichtiges Programm ergänzt. Es dient der graphischen Darstellung von Histogrammen und kann unter dem Namen `display` als herkömmliches Shell-Kommando aufgerufen werden. Das Programm nutzt einerseits Unterprogramme aus der Histogramm-Bibliothek, die den Zugriff auf die Histogramme gewähren, und andererseits die in Abschnitt 8.3 beschriebene Software zur graphischen Darstellung.

`display` bietet die Möglichkeit, mit jedem Aufruf jeweis ein ein- oder zweidimensionales Histogramm als Balkendiagramm bzw. Scatter-Plot darzustellen. Es erlaubt mithilfe seiner Kommandozeilenparameter neben der Auswahl des Histogramms über dessen Namen die Selektion des anzuzeigenden Kanalbereichs und die zusätzliche Zusammenfassung von Histogramm-Kanälen in der graphischen Darstellung. Das Programm beschafft sich alle weiteren Informationen, die es sonst noch benötigt, aus dem Histogramm selbst. Auf diese Art wird auch der Typ des Histogramms festgestellt und in Abhängigkeit davon schließlich die korrekte Software zur graphischen Darstellung aktiviert.

Nach seinem Aufruf arbeitet `display`, dessen Aufgabe in erster Linie die Histogramm-Darstellung im interaktiven Betrieb auf entsprechenden Bildschirmgeräten ist, in der Regel so lange, bis es durch den Benutzer terminiert wird. Es überprüft dabei in einer Schleife, die alle zehn Sekunden durchlaufen wird, den Zustand des angewählten Histogramms und übermittelt der Graphik-

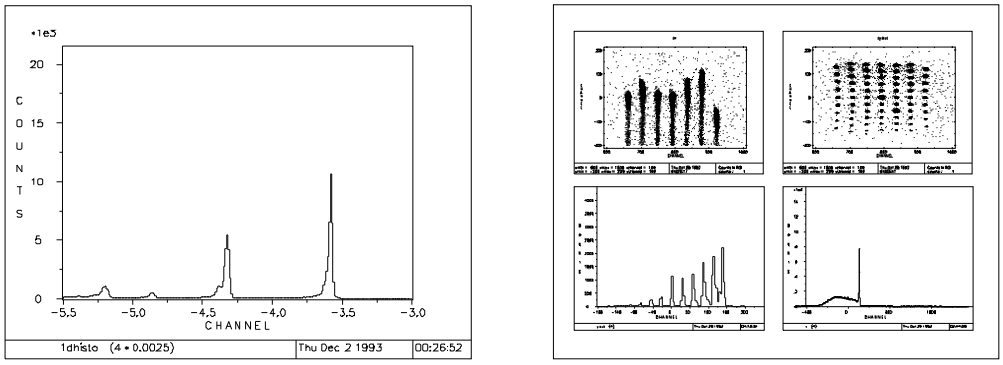

Abb. 8.11: Beispiele für die graphische Darstellung eines Histogrammes mithilfe des Programmes `display` bzw. `hplot` und mehrer Histogramme mithilfe des Programmes `picture`

Software bei Änderungen die neu darzustellenden Daten. Damit sorgt `display` ohne weitere Benutzerinteraktion für eine stets aktuelle Darstellung von Histogrammen.

Als nicht interaktive Variante existiert das Programm ein zweites Mal unter dem Namen `hplot`. Bei dessen Aufruf wird der Histogramm-Inhalt lediglich einmal ausgelesen und an die Graphik-Software weitergeben. Dabei arbeitet das Programm ansonsten auf vollkommen identische Art und Weise wie `display`, terminiert jedoch unmittelbar nach der Graphikausgabe des aktuellen Histogramm-Zustands. `hplot` wird in der Regel zur Ausgabe von Histogramm-Darstellungen auf nicht interaktive Geräte wie Drucker benutzt, kann aber auch diese Information auf Plattendateien ablegen oder an andere Software zur graphischen Weiterverarbeitung übermitteln. Abbildung 8.11a zeigt eine auf diese Art und Weise entstandene Darstellung eines Histogramms.

Eine mit `hplot` erzeugte Druckerausgabe unterscheidet sich bis auf die Auflösung prinzipiell nicht von der Bildschirmausgabe. `hplot` kann sogar dazu verwendet werden, um Histogramme auf einem Bildschirmgerät darzustellen. Diese Tatsache nutzt das Hilfsprogramm `picture`. Es ist als eine Kommandoprozedur realisiert, die in der Lage ist, im Gegensatz zu `display` mehrere Histogramme zur gleichen Zeit auf einem Bildschirmfenster in übersichtlicher Weise darzustellen. Es benutzt ausschließlich einfache Grundbausteine der Graphik-Software und des UNIX-Kommandointerpreters und demonstriert sehr deutlich die einfache aber effiziente Möglichkeit, wie mit der Kombination simpler Grundelemente auch komplexe Aufgabenstellungen zu bewältigen sind (Abb. 8.11b).

8.3 Die graphische Darstellung von Meßergebnissen

8.3.1 Problemstellung

Um die im Rahmen der On-line-Analyse aus den Meßdaten berechneten Ergebnisse dem Benutzer einfach zugänglich zu machen, ist ihre graphische Darstellung sehr hilfreich. Die Verschiedenheit der Meßergebnisse und die Variationsbreite ihrer Darstellungsformen, die für eine effektive Überwachung des Meßbetriebs denkbar sind, erfordern dabei eine geeignete, an unterschiedliche Problemstellungen angepaßte Behandlung. Von besonderem Interesse hierbei ist die Darstellung von ein- und zweidimensionalen Histogrammen in Form von Balkendiagrammen, Scatter- oder sog. Lego-Plots und funktionale Zusammenhänge oder zeitliche Entwicklungen von einfachen Zahlentupeln als mehrparametrige zwei- oder dreidimensionale Diagramme.

Neben der Vielzahl der Möglichkeiten zur graphischen Darstellung tritt gleichzeitig die Vielfalt der Geräte, auf die die graphische Ausgabe erfolgen soll. Dabei sind nicht nur die Ausgabegeräte im Zusammenhang mit der interaktiven Bedienung des Systems wie Konsolen oder Bildschrimgeräte wichtig, sondern ebenso Drucker oder Plotter zur permanenten Dokumentation der Ergebnisse. Im Laufe der Entwicklung haben sich hier die Schwergewichte zwar etwas verschoben, doch die grundsätzliche Problematik blieb unverändert. Ursprünglich wurden interaktiv fast ausschließlich graphikfähige Textterminals eingesetzt und zur Druckausgabe Zeilen-, Nadel- und Tintenstrahldrucker bis hin zu einfachen Laserdruckern. Inzwischen hat sich das Bild gewandelt. Neben diesen Geräten, die jedoch immer weniger zum Einsatz kommen, werden in erster Linie Workstation-Konsolen und leistungsfähige X-Terminals eingesetzt, die unter dem X-Window-System arbeiten, und Laserdrucker, die in der Lage sind, in der Seitenbeschreibungssprache Postscript formulierte Text- oder Graphikausgaben unmittelbar auszudrucken.

Sowohl die unterschiedliche Umsetzung der Daten in Graphiken als auch die individuelle Ansteuerung der Ausgabegeräte erforderte die Realisierung spezieller Software, die den besonderen Aufgabenstellungen gerecht werden mußten. Die Anforderungen an diese Software wurden noch verstärkt durch die Tatsache, daß sich ihr Einsatz nicht nur auf die Rechner beschränkt, die im Rahmen des letzendlich zu realisierenden verteilten Systems dafür vorgesehenen sind, sondern gerade in der Anfangsphase bei der Hard- und Software-Entwicklung aber auch später für ähnliche Aufgaben auch auf den Frontend-Rechenern im stand-alone-Betrieb notwendig wurde. Erschwert wurde die Sache noch dadurch, daß diese Rechner mit minimalen Ressourcen ausgestattet sind und insbesondere bei der urprünglichen Verwendung des Betriebssystems OS-9 nur sehr eingeschränkte Systemunterstützung erhielten.

Gerade aus diesen Gründen konnte auf etablierte Graphik-Software wie GKS[1], Erlgraph[2] oder entsprechende Teile der CERN-Bibliothek nicht zurück-

[1] Graphics Kernel System [DIN86]
[2] Erlanger Graphik-System [Erlg84]

`arc()`	Generierung eines Kreisbogens
`circle()`	Generierung eines Kreises
`closepl()`	Beenden der Graphikausgabe
`color()`	Setzen der aktuellen Farbe
`cont()`	Fortsetzung einer Linie
`fontname()`	Setzen des aktuellen Schrifttyps
`fontsize()`	Setzen der aktuellen Schriftgröße
`label()`	Ausgabe einer Textzeile
`line()`	Ausgabe einer Linie
`linemod()`	Setzen der aktuellen Linienform
`move()`	Ändern der aktuellen Plotposition
`openpl()`	Vorbereitung der Graphikausgabe
`point()`	Setzen eines Punktes
`rotate()`	Setzen des aktuellen Winkels
`space()`	Setzen des aktuellen Zeichenbereichs

Tab. 8.5: Überblick über die in den PLOT-Bibliotheken verfügbaren Funktionen für elementare graphische Operationen.

gegriffen werden. Diese war für die meisten der eingesetzten Rechner bzw. Betriebssysteme nicht unmittelbar verfügbar. Auch ihre Portierung war infolge fehlender Compiler und zu geringer Systemressourcen meist nicht möglich bzw. wäre wegen der hohen Komplexität der Pakete zu aufwendig geworden, während die dann verfügbare Funktionalität in diesem Umfang gar nicht benötigt worden wäre. Mit der Verfügbarkeit des Programmes `paw` auf den Workstations bot sich jedoch die Gelegenheit, mit wenig zusätzlicher Software MECDAS-Histogramme auch damit graphisch darzustellen oder gar weiterzuverarbeiten [Kram94, Heil91b].

8.3.2 Ein modulares Graphiksystem

Zur Erledigung der genannten Aufgabenstellungen wurde ein modulares Software-System (MOPS[3]) erstellt, das einfach zu handhabende Mechanismen zur Verfügung stellt und sich bestmöglich in das bisher beschriebene On-line-Analysekonzept von MECDAS einfügt. Es basiert auf dem PLOT-Graphikpaket von UNIX, das bei einigen UNIX-Varianten zum Standardumfang des Betriebssystems gehört [Plot] aber auch als PD-Software [Murp89] und damit betriebssystemübergreifend verfügbar ist.

Die PLOT-Software stellt einen einfachen Graphikstandard dar, der von einer Reihe von Anwendungen innerhalb des Betriebssystems genutzt wird, aber auch von anderen Graphikpaketen mit unterstützt wird. Sie setzt sich aus mehreren Unterprogrammbibliotheken zusammen, die durch eine Reihe abgeschlossener Programme ergänzt werden. Jede dieser Bibliotheken besteht aus einem übersichtlichen Satz einfacher Bibliotheksfunktionen für elementare graphische

[3] Modular On-line Plot System

Operationen (s. Tab. 8.5). Die verschiedenen Bibliotheken sind identisch aufgebaut und dienen der Ansteuerung jeweils unterschiedlicher Ausgabegeräte. Damit steht eine einfache und einheitliche Schnittstelle auf Quellcodeebene zur Verfügung, die es erlaubt, eine graphische Anwendung weitestgehend unabhängig von den evtl. einzusetzenden Ausgabegeräten zu implementieren.

In der Regel stehen Bibliotheken für die Ausgabe auf graphikfähigen Bildschirmgeräten unterschiedlichen Typs zur Verfügung; auch spezielle Drucker werden unterstützt. Eine der verfügbaren Bibliotheken nimmt jedoch eine Sonderstellung ein. Sie unterstützt nicht direkt einen konkreten Gerätetyp, sondern dient dazu, eine geräteunabhängige Ausgabe zu erzeugen, die dann mithilfe getrennter Treiberprogramme zur Ansteuerung der verschiedenen Geräte verwendet werden kann. Auch solche Treiberprogramme stehen standardmäßig für eine Reihe von Ausgabegeräten zur Verfügung.

Die geräteunabhängige Bibliothek definiert das sog. PLOT-Format, das eine eineindeutige Abbildung der PLOT-Bibliotheksfunktionen darstellt. Es realisiert eine Graphikausgabe, die mithilfe eines geeigneten Interpreters stets wieder in eine Sequenz entsprechender Funktionsaufrufe umgewandelt werden kann. Es erlaubt so die klare Trennung zwischen Anwendungs-Software und geräteabhängigen Treiberprogrammen ohne Einbußen in Funktionalität, Information und Qualität. Das ist besonders dann von Interesse, wenn für eine Anwendung die Ausgabe auf mehrere Geräte notwendig ist. Während bei der Verwendung der Gerätebibliotheken für jede Anwendung mehrere gerätespezifische Programme notwendig werden, erlaubt die Verwendung des PLOT-Formats die Beschränkung auf jeweils ein einziges, anwendungsspezifisches Programm, das von einer mehr oder weniger großen Anzahl von Treiberprogrammen ergänzt wird, die auch allen anderen Anwendungen zur Verfügung stehen.

Das PLOT-Format stellt nun die Grundlage der im Rahmen von MECDAS entwickelten und eingesetzten Graphik-Software dar. Es bot die Möglichkeit, die benötigte Funktionalität mit minimalem Implementationsaufwand durch die Nutzung eines etablierten Standards wie auch in anderen Bereichen des Datenerfassungssystems zu erreichen. Dort, wo die existierende Software nicht aussreichte, wurde sie durch eigene Entwicklungen ergänzt. Auch wenn sich die Grundfunktionalität der PLOT-Software nur auf das Notwendigste beschränkt und keine höheren Graphikprimitiven direkt zur Verfügung stehen, konnte gerade ihre Einfachheit und Modularität sehr gut zur Realisierung eines flexiblen Graphiksystems genutzt werden, das den Anforderungen der On-line-Analyse genügt.

Es setzt sich zusammen aus Treiberprogrammen für einzelne Ausgabegeräte, die auch für die graphische Ausgabe bei ganz anderen Anwendungen verwendet werden können, und anwendungsorientierter Software, die konkret auf die Anforderungen im Rahmen der On-line-Analyse eingeht. Diese ist selbst noch einmal in zwei Teile aufgeteilt, wobei zwischen speziellen Aufgabenstellungen und immer wieder auftretenden allgemeinen Routineaufgaben unterschieden wird. Ihre differenzierte Berücksichtigung im Software-Konzept erlaubte eine möglichst gute Wiederverwendbarkeit vieler Softwaremodulen und gleichzeitig eine Vereinfachung der Schnittstellen für den Benutzer.

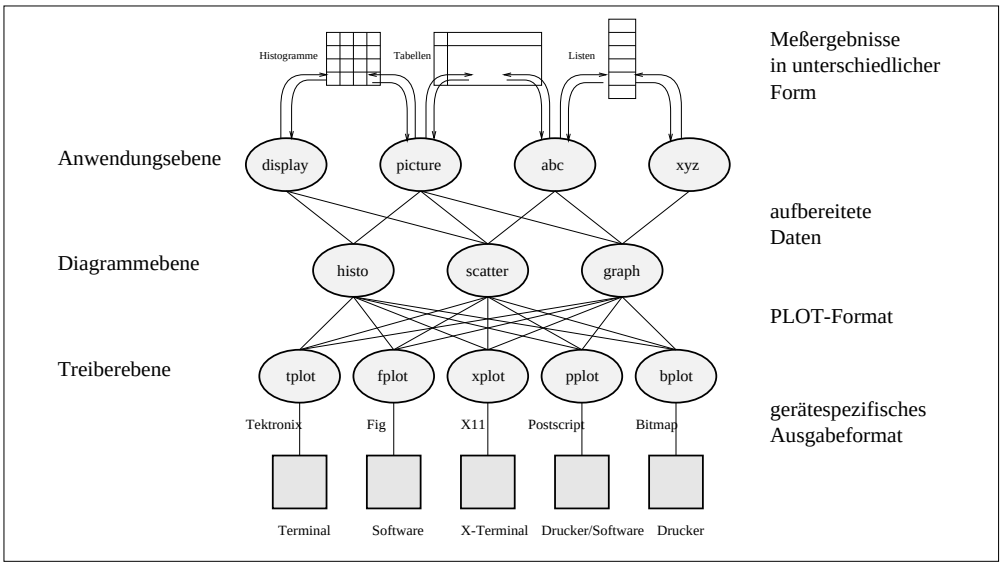

Abb. 8.12: Schematische Darstellung der Struktur von MOPS.

Abbildung 8.12 gibt einen Überblick über die Struktur des Graphiksystems. Es zeigt deutlich die drei sich letzendlich ergebenden Software-Ebenen, die sowohl horizontal als auch vertikal aus deutlich voneinander abgegrenzten Programmen mit definierten Schnittstellen bestehen. Zwischen ihnen herrscht eine klare hierarchische Beziehung: Programme aus einer höheren Ebene benutzen die aus der darunterliegenden, um die graphische Ausgabe auf geeignete Weise durchzuführen. Die Abbildung macht deutlich, daß infolge des modularen Aufbaus, auch bei einer nur kleinen Anzahl von Softwaremodule eine sehr umfangreiche Zahl von sinnvollen Kombinationsmöglichkeiten erreicht werden kann.

Durch die Beschränkung der Anwendungs-Software auf eine einzige und im Vergleich zu anderen Graphikpaketen sehr einfache und kompakte Bibliothek konnte zum einen die Größe der Programme auf ein Minimum reduziert werden, was bei den moderneren Rechnern zwar kaum noch von Bedeutung ist, jedoch bei den eingesetzten Frontend-Rechnern notwendig war. Viel wichtiger jedoch ist, daß damit die Komplexität der Software und schließlich die Fehleranfälligkeit insbesondere bei der Software-Entwicklung klein gehalten werden konnte. Andererseits erlaubte und erlaubt auch weiterhin die Abtrennung der gerätespezifischen Software bei Bedarf eine sehr einfache Erweiterung des Graphiksystems auf neue Ausgabegeräte, ohne daß dazu die Anwendungsprogramme modifiziert oder auch nur neu gebunden werden müßten. Hier ist lediglich die Implementation eines neuen, auf das Gerät abgestimmten Treiberprogramms nötig.

8.3.3 Die realisierte Software

8.3.3.1 Anwendungsebene

Unmittelbar in die On-line-Analyse integriert ist die Anwendungsebene von MOPS. Sie stellt eine Reihe von Programmen zur Verfügung, die unter Zuhilfe-

nahme der Mechanismen, die die eigentliche Analyse-Software bereitstellt, die darzustellenden Meßergebnisse einliest und in ein geeignetes Diagrammformat umwandelt. Das entsprechende Programm aus der zweiten Ebene, das in der Lage ist, die graphische Umsetzung des Diagramms vorzunehmen, wird ebenso wie das Ausgabegerät entweder automatisch oder aufgrund von Benutzeranweisungen ausgewählt. Schließlich wird die entsprechende Software aktiviert.

Die Programme der Anwendungsebene stellen im wesentlichen die Kommandos zur graphischen Darstellung dar. Sie können entweder unmittelbar durch den Benutzer über eine einfache kommandoorientierte oder aber auch graphische Benutzeroberfläche aktiviert werden, oder sie werden im Rahmen von Kommandoprozeduren oder sogar direkt durch Analyseprogramme aufgerufen. Standardmäßig können auf diese Weise die on-line erzeugten Histogramme, die mithilfe der Histogramm-Bibliothek angesprochen werden können, dargestellt werden. Dazu stehen die bereits in Abschnitt 8.2.4 beschriebenen Programme `display` und `picture` zur Verfügung. Neben der Umwandlung der speziellen Histogramm-Information in eine allgemeinere Darstellung, die durch zusätzliche, für die graphische Ausgabe wichtige Information ergänzt wird, sorgen beide Programme in Abhängigkeit vom bearbeitenden Histogramm-Typ und Kommandozeilenparametern für die Auswahl der richtigen Programme aus der Diagrammebene und der Treiberebene und aktivieren diese. Dabei wird z. B. automatisch unterschieden zwischen ein- und zweidimensionalen Histogrammen oder der Ausgabe auf ein Tektronix- oder X-Terminal.

Auch andere Ergebnisse der On-line-Analyse, die etwa in tabellarischer Form in Dateien abgespeichert sind oder unmittelbar von einem als Filter arbeitenden Analyseprogramm geliefert werden, können graphisch dargestellt werden. In diesem Fall können die Daten ein beliebiges Format besitzen. Je nach Auswertemethoden kommt dabei eine unterschiedliche Darstellung in Frage; dementsprechend unterscheiden sich dann auch die Programme in der Anwendungsebene. Da jedoch für diese Analysemethode standardmäßig keine Software im Rahmen des MECDAS-Analysepakets vorgesehen ist, existieren lediglich Prototypen für die graphische Darstellung damit gewonnener Daten.

8.3.3.2 Diagrammebene

Die Software in der Diagrammebene sorgt für die Umsetzung der Diagramm-Information in konkrete Graphikanweisungen im PLOT-Format. Hier wird das Zeichnen der Koordinatenachsen, die Skalierung und Achsenbeschriftung bis hin zur Darstellung der Kurven oder Datenpunkte vorgenommen. Zur Zeit besteht diese Ebene i. w. aus drei Programmen für unterschiedliche Aufgabenstellungen. Sie werden in der Regel nicht direkt vom Benutzer aufgerufen sondern mit geeigneten Parametern über eines der Programme aus der Anwenderebene.

Das Programm `histo`

Das Programm `histo` dient der Generierung von eindimensionalen Balkendiagrammen, die in erster Linie zur Histogramm-Darstellung Verwendung finden

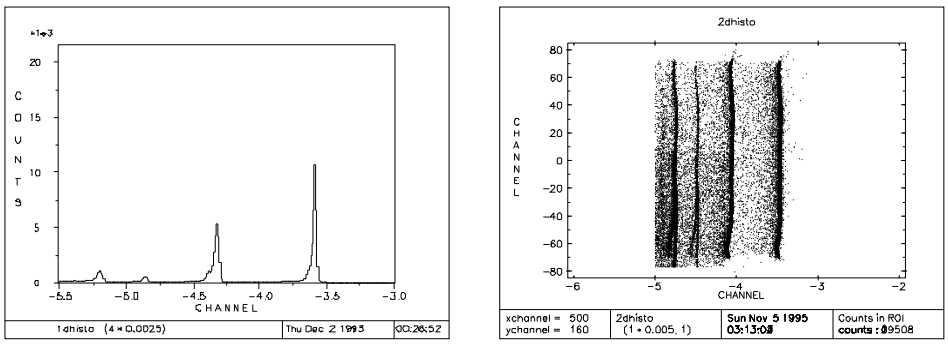

Abb. 8.13: Beispiele für die Ausgabe der Programme `histo` und `scatter`

(Abb. 8.13a). Es läßt sich in unterschiedlichen Modi betreiben, die bei seinem Aufruf über die Programmparameter bestimmt werden können. Darüber wird auch die Achsenskalierung und -beschriftung vorgegeben. `histo` nimmt nach seinem Start über seinen Standard-Eingabekanal sukzessive normalisierte Wertepaare, die Nummer und Inhalt von Histogramm-Kanälen spezifizieren, entgegen und erzeugt mit dieser Information die nötigen PLOT-Anweisungen, um in der nachfolgenden Ausgabe vertikale Balken an den entsprechenden Stelle innerhalb des Gesamtdiagramms zu generieren. In der Regel werden die PLOT-Anweisungen synchron mit der einlaufenden Information über die Standard-Ausgabeschnittstelle des Programmes einem Treiberprogramm zugeführt, das letzendlich die Graphikausgabe auf einem konkreten Gerät vornimmt.

Im Standard-Modus nimmt das Programm so lange Daten entgegen, bis durch eine explizite End-Of-File-Markierung das Ende des Datenstroms gekennzeichnet wird. Solange das nicht der Fall ist, interpretiert es ständig die einlaufende Information. Stellt es dabei Änderungen der Inhalte von bereits dargestellten Kanälen fest, erzeugt es die nötigen PLOT-Anweisungen zur Korrektur der entsprechenden Diagrammbalken. Bei Verwendung von geeigneter Ausgabe-Hardware, die in der Lage ist, das dafür notwendige Löschen von vorher gezeichneten graphischen Elementen vorzunehmen, sorgt `histo` so für eine dynamische Anpassung der Histogramm-Darstellung. Somit bietet `histo` die Möglichkeit, die zeitliche Entwicklung eines Histogrammes ohne ständige Benutzerinteraktion graphisch wiederzugeben.

Das Programm `scatter`

Zur Darstellung von zweidimensionalen Histogrammen als Scatter-Plots wurde das Programm `scatter` entwickelt [Schr93] (Abb. 8.13b). Analog zu dem Programm `histo` wird es in der Regel über das Kommando `display` aktiviert. Es wird ebenso über Kommandozeilenparamter gesteuert und arbeitet nach seinem Start als Filterprogramm, daß die Histogramm-Information über seinen Standard-Eingabekanal einliest, nun aber die PLOT-Anweisungen zur Darstellung von Punkten innerhalb des zweidimensionalen Diagramm generiert. Ansonsten wird die Ausgabe wieder an ein Treiberprogramm zur entgültigen graphi-

schen Darstellung weitergereicht. Ebenso wie `histo` ermöglicht auch `scatter` die Wiedergabe der zeitlichen Entwicklung des Histogramms, die sich in einer mit der Zeit zunehmenden Dichte der Punkte innerhalb des Scatter-Plots zeigt.

Das Programm `graph`

Das Programm `graph` stellt einen Standardbestandteil des PLOT-Pakets dar. Es ist in der Lage, Wertepaarreihen graphisch darzustellen. Es liest Wertepaare, die den x- und y-Koordinaten von Punkten entsprechen, ein und erzeugt die PLOT-Anweisung für deren Darstellung. Je nach Angabe von Kommandozeilenparametern erzeugt es Koordinatenachsen, deren Skalierung und Beschriftung. Die Punkte können z. B. als isolierte Symbole dargestellt werden oder durch Linien unterschiedlichen Aussehens verbunden werden. Mehrere Diagramme können innerhalb einer Ausgabe gleichzeitig dargestellt werden.

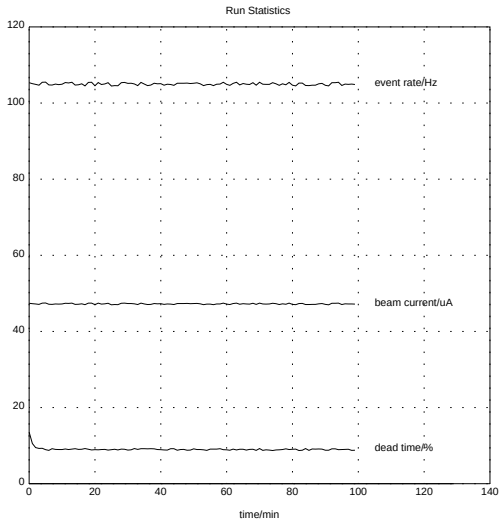

Abb. 8.14: Beispiel für die Ausgabe des Programmes `graph`

Im Gegensatz zu `histo` und `scatter` arbeitet `graph` nicht synchron zur Eingabe, sondern es kann prinzipbedingt seine Ausgabe erst dann vornehmen, wenn es die komplette Eingabe-Information verarbeitet hat. Daher ist es für die Darstellung dynamischer Vorgänge nicht geeignet, kann aber jegliche Art an statischer Information graphisch wiedergeben.

8.3.3.3 Treiberebene

Bei den Treiberprogrammen konnte zu einem Großteil auf Standard-Software zurückgegriffen werden. Es wurden jedoch, wie schon bei der PLOT-Bibliothek, auch hier größere Teile der fast vollständig in Quellcode vorliegenden Software umfassend überarbeitet. Das diente einerseits der Beseitigung von vorhandenen Fehlern und Inkompatibilitäten, andererseits aber insbesondere der Vereinheitlichung der aus unterschiedlichen Quellen stammenden Software inklusive Neuimplementierungen für noch nicht unterstützte Ausgabegeräte.

Tabelle 8.6 zeigt eine Übersicht über die im Rahmen von MECDAS unmittelbar unterstützen PLOT-Treiberprogramme. Sie werden ergänzt durch eine mehr oder weniger große Zahl von Programmen aus dem PD-Bereich oder anderen Softwarepaketen und solchen, die das Betriebssystem eines Rechners bereits standardmäßig zur Verfügung stellt, Unter anderem besteht unter UNIX die Möglichkeit, PLOT-Dateien unmittelbar mithilfe des Standard-Kommandos `lpr` auf Druckern auszugeben.

Treiberprogramm	Ausgabegerät
`tplot`	Tektronix 4010/4014
`xplot`	X-Window-System
`pplot`	Postscript (Drucker bzw. weiterverarbeitende Software insbesondere zum Einbinden in Textdokumenten)
`fplot`	Software zur Verarbeitung von Fig-Graphikdateien (z. B. das Programm `xfig`)
`bplot`	Software zur Verarbeitung von Bitmaps (insbesondere Druckertreiber)

Tab. 8.6: Die von MECDAS unterstützten PLOT-Ausgabegeräte

Alle Treiberprogramme besitzen einen ähnlichen Aufbau. Sie setzen sich aus zwei Grundelementen zusammen. Kernstück bildet bei allen ein einheitlicher Interpreter für das PLOT-Format, der jeweils durch die gerätespezifische PLOT-Bibliothek ergänzt wird. Infolge der einfachen Struktur des PLOT-Formats konnte ein effizienter Interpreter mit wenig Aufwand vollständig neu implementiert werden. Er war notwendig, um tatsächlich alle der eingesetzten Ausgabegeräte einheitlich zu unterstützen.

Eine Besonderheit in jeder Hinsicht bildetet der PLOT-Treiber für X11. Die ereignisorientierte Arbeitsweise des X-Window-Systems macht eine ganz spezielle Struktur und damit auch Programmierung von X-Anwendungen erforderlich. Kernstück eines solchen Programmes ist eine zentrale Schleife (sog. *Event Loop*), in der ständig auf Ereignisse wie Tastatureingaben, Fensteraktivitäten oder Mausmanipulationen gewartet wird und aus der heraus bei Eintreten eines bestimmten Ereignisses die dazugehörige Software aktiviert werden muß. Die konkrete Anwendung muß sich dieser Programmstruktur unterordnenen. Im Gegensatz zu herkömmlichen Geräten reicht zur Graphikansteuerung unter X11 daher nicht mehr die Einbindung einer geeigneten Bibliothek in ein existierendes, sequentiell ablaufendes Programm. Im Normalfall ist statt dessen ein Eingriff in die Programmstruktur notwendig, der den sequentiellen Ablauf in einen periodischen überführt. Die in MOPS vorgenommene Trennung von Anwendung und graphischer Ausgabe erübrigt jedoch derartige Maßnahmen vollständig. Während die Anwendungs-Software i. w. ohne Einschränkungen ausschließlich von der Lösung problemorientierter Aufgaben bestimmt ist, muß sich lediglich der Gerätetreiber den besonderen Bedingungen von X unterordnen. Die zentrale Funktion des Treibers, die in der Interpretation von PLOT-Anweisungen und Aktivierung der entsprechenden Graphikausgabe besteht, läßt sich damit jedoch sehr gut vereinbaren.

Im Rahmen einer ergänzenden Diplomarbeit [Schr93] wurde der Prototyp eines X-Gerätetreibers entwickelt, der inzwischen durch das in Funktionalität und Programmierung verbesserte Programm `xplot` ersetzt werden konnte. `xplot` stellt einen zentralen Bestandteil einer graphischen Benutzungsoberfläche für die MECDAS-On-line-Analyse dar. Es wird ergänzt durch eine Reihe weitere Programme für verschiedene Aufgaben.

8.4 Voruntersuchungen für zukünftige Entwicklungen

Im Rahmen dieser Arbeit mußte sich die Realisierung von Software zur On-line-Analyse wegen des Umfangs dieser Aufgabenstellung auf das Nötigste beschränken, und so konnten viele Probleme, wenn überhaupt, lediglich rudimentär und in Form von Prototypen, wie in den vorangegangnen Abschnitten erläutert, gelöst werden. Für andere wurden Vorarbeiten geleistet, die jedoch noch nicht in allgemein nutzbare Software umgesetzt werden konnten. Ein besonderes Bestreben war es, von starrer Analyse-Software mit fest programmierten Analysevorschriften wegzukommen. Unter anderem wurde ein Konzept für eine dynamische Organisation der On-line-Analyse entwickelt.

8.4.1 Modularisierung

Das bisher verwendete Verfahren, On-line-Analyseprogramme auf Quellcode-Ebene zu modifizieren, um sie an spezielle Randbedingungen anzupassen, ist eine sehr allgemeine Lösung, die alle Sonderfälle abdeckt, aber im alltäglichen Betrieb oftmals recht unhandlich ist, wenn nur kleine Anpassungen das wiederholte Editieren und Neuübersetzen des Programmes notwendig machen. Insbesondere bei ungeeigneter Organisation des Analyseprogrammes kann der Übersetzungsvorgang größere, nicht mehr zu vernachlässigende Zeiten beanspruchen[4]. Hier hilft bereits eine klare Modularisierung des Programmes mit deutlicher Abgrenzung einzelner Teilaufgaben. Die damit mögliche Aufteilung verschiedener Aufgaben in unterschiedliche Quelldateien erlaubt nicht nur eine erhebliche Reduktion der Übersetzungszeiten, sondern sorgt auch für eine bessere Übersichtlichkeit und damit auch eine Verminderung der Fehleranfälligkeit, die beim ständigen Verändern von Quellcode grundsätzlich hoch ist. Dabei hat das Verfahren den Vorzug, daß keine tiefergehenden Eingriffe in das Analysekonzept mit einer mehr oder weniger aufwendigen Programmierung notwendig werden. Es nutzt dabei weitestgehend bereits implementierte detektor- und experimentspezifische Software.

Zur Modularisierung des On-line-Analyseprogrammes bieten sich zwei Verfahren an:

- Aufteilung in verschiedene, gut gekapselte Programm-Module innerhalb weiterhin eines einzigen Analyseprogrammes
- Aufteilung in mehrere, unabhängige Programme, etwa in Form von hintereinandergeschalteten Analyse-Filtern

Beide Verfahren wurden bereits in Form von Prototypen softwaremäßig umgesetzt, wobei die Existenz von Software genutzt werden konnte, die im Rahmen

[4] An dieser Stelle zeigt sich ein gravierender Nachteil der Verwendung von C++ im Vergleich zu C zumindest in Zusammenhang mit den bisher verfügbaren Compilern und Linkern: das Generieren eines läuffähigen C++-Programmes kostet ein Vielfaches von der Zeit, die für ein entsprechendes C-Programm benötigt wird. Auch die erhöhte Rechenleistung neuer Rechner macht diesen Effekt bisher noch nicht wett.

des allmählich einsetzenden Experimentierbetriebs erstellt wurde. Dabei zeigte sich, daß insbesondere bei Experimenten, die mit einer festen Experimentieranordnung arbeiten, wie sie die Drei-Spektrometer-Anlage darstellt, eine sehr gute Kapselung von allgemeinen und detektorspezifischen Aufgaben möglich ist, so daß der Benutzer nur noch mit experimentspezifischen Programmteilen konfrontiert wird. Bei dem zweiten Verfahren ist diese Kapselung besonders ausgeprägt. Hier werden die experimentübergreifenden Aufgaben, wie etwa Dekodierung der Daten, Koordinatenbe- und -rückrechnung, Aufakkumulieren von bestimmten Rohspektren usw., von festen, abgeschlossenen Programmen übernommen, die nur noch sehr selten modifiziert werden müssen. Lediglich experimentspezifische Programme müssen geeignet manipuliert werden. Seiteneffekte aufgrund von Fehlern im diesem Bereich die auf den allgemeinen zurückwirken, reduzieren sich dabei auf ein Minimun. Damit verbessert sich die Benutzbarkeit auch für Nicht-Experten deutlich. Ohne weitere Vorkehrungen hat jedoch dieses Verfahren einerseits den Nachteil einer geringeren Flexibilität und andererseits infolge des notwendigen Datenaustauschs zwischen verschiedenen Programmen einer niedrigeren Effizienz. Erste Tests haben jedoch gezeigt, das beides mit geeigneten Maßnahmen in den Griff zu bekommen ist.

8.4.2 Konfigurierbare Paramter

In Bezug auf die Anpassung der On-line-Analyse an experimentspezifische Randbedingungen kann man unterscheiden zwischen:

1. der Änderung komplexer Analysevorschriften
2. der Änderung einzelner Parameter, die bei der Analyse benötigt werden

Während der erste Fall — ausgehend von dem bisherigen Konzept eines direkt programmierten Analyseprogramms — Änderungen in der Struktur des Analyseprogramms bedeutet und damit eine experimentspezifische Umprogrammierung erforderlich macht, kann der zweite Fall durch spezielle programmtechnische Verfahren abgefangen werden, indem variable Parameter nicht mehr fest in die Software einprogrammiert werden.

Der einfachste Ansatz ist hier die Ausnutzung des Präprozessors, der bei der Compilation von C/C++-Programmen stets aktiviert wird. Er erlaubt z. B. die Definition von Parametern außerhalb der Programmdateien in sog. Header-Dateien auf sehr einfache und übersichtliche Art. Diese Vorgehensweise kann bei geeigneter Programmierung die Bedienbarkeit verbessern und die Fehleranfälligkeit reduzieren, doch erfordert sie weiterhin die Neucompilation des Analyseprogrammes bei Änderungen der Parameter. Das läßt sich mit dem zweiten Verfahren umgehen, in dem alle veränderlichen Parameter als echte Variablen innerhalb des Programmcodes spezifiziert sind und ein Mechanismus zur Verfügung steht, diese Variablen von außen zu manipulieren. Die einfachste Alternative, die Initiaisierung dieser Variablen beim Programmstart aufgrund der Angaben in einer Parameterdatei, konnte im Rahmen der A1-Kollaboration bereits auf verschiedenen Wegen mit beschränktem Aufwand realisiert werden [Dist93a, Kram95].

Eine wesentlich komfortablere Möglichkeit, die Parameter mithilfe eines speziellen Kommandointerpreters evtl. sogar während des Programmablaufs verändern zu können, erfordert jedoch deutlich mehr Aufwand. In Hinblick auf die damit nur zu einem Teil abgedeckten Anforderungen an ein flexibles On-line-Analysesystem wurde diese Möglichkeit bisher nicht weiter verfolgt. Ebenso wie für den ersten Fall in der obigen Aufzählung sind hier wesentlich weitergehende Maßnahmen notwendig, um den Benutzer in die Lage zu versetzen, flexibel, aber gleichzeitig ohne direkte Programmierung, Analysebedingungen zu verändern.

8.4.3 Dynamische Analyse

8.4.3.1 Problemstellung

Aufbauend auf den Entwicklungen für experimentspezifische Standard-Analyse-Software ist es dringend notwendig, auch ein allgemeineres Software-System zu realisieren, daß es erlaubt, einerseits mit wenig Aufwand für jedes Experiment, das mithilfe von MECDAS durchgeführt wird, auch eine adäquate Analyse bereitzustellen, andererseits aber auch komfortabel bedient zu werden, so daß auf möglichst einfache Art auch komplexe Analysevorschriften definiert und geändert werden können.

Ähnlich wie bei der Datenerfassung bietet sich auch hier die in den Daten enthaltene Experimentbeschreibung als Grundlage an, jedoch mit dem gravierenden Unterschied, daß nun die Software in der Lage sein muß, anstelle der festen Bearbeitungsvorschriften zur Datenaufnahme nun dynamische Analysevorschriften zu bearbeiten. Die Software muß u.a. die Möglichkeit bieten,

- Auswertebedingungen wie
  - einfache Überprüfungen von Schwellen
  - einfache Schnitte
  - mehrdimensionale Schnitte
  - Schnitte aufgrund komplexer Kriterien
  - Kombinationen aus diesen Auswertebedingungen

  zu definieren, zu ändern und zu entfernen,
- solche Auswertebedingungen auf die Daten anzuwenden,
- Auswerteaktionen zu definieren, die aufgrund der Auswertebedingungen oder auch unbedingt ausgeführt werden sollen,
- ein- oder mehrdimensionale Histogramme zu definieren, sie zu ändern oder zu entfernen,
- die Histogramme aufgrund von Auswertebedingungen bzw. auf der Basis der Auswerteaktionen zu manipulieren,

- unmittelbare oder aus Auswerteaktionen gewonnene Meßergebnisse auszugeben.

Für die On-line-Analyse besonders wichtig ist, daß das alles während einer laufenden Messung möglich sein sollte und jede Änderung sich mit geringst möglicher Verzögerung auswirken sollte. Hier kommt also der Formulierung, der Organisation und der Art der Verarbeitung der Analysevorschriften ein hoher Stellenwert zu.

8.4.3.2 Formulierung der Analysevorschriften

Wie bereits bei der Datenerfassung zeigt sich auch hier, daß für eine ausreichende Formulierung der erfahrungsgemäß benötigten experimentspezifischen Analysevorschriften eine vollständige Sprache, wie sie eine höhere Programmiersprache darstellt, notwendig ist. Aus diesem Grund verwenden die herkömmlichen Ansätze zur Datenanalyse [Offe93, Brun91] auch in der Tat die Programmierung in einer solchen Hochsprache, insbesondere Fortran. Auch bei Einsatz einer einfacheren Kommandosprache zur Definition und Manipulation bestimmter Parameter kommt man an einer detaillierteren Programmierung in einer Hochsprache nicht vorbei, es sei denn, die Kommandosprache hätte einen vergleichbaren Sprachumfang. Beide Lösungen finden sich in der zitierten Software wieder.

Dementsprechend ist der triviale Ansatz die ausschließliche Programmierung des Auswerteprogramms inklusive seiner experimentspezifischen Teile in einer Hochsprache. Dieser Ansatz wurde bisher verfolgt, besitzt jedoch in Bezug auf die einfache Veränderbarkeit der Analysevorschriften die schon erwähnten Nachteile, da jede Änderung das Editieren von Quellcode bedeutet und eine Neuübersetzung zumindest einzelner Teile des experimentspezifischen Programmcodes erforderlich macht. Der in [Schr93] beschriebene Ansatz versucht diesen Nachteilen durch eine geeignete Benutzeroberfläche und im Hintergrund ablaufende Automatismen Rechnung zu tragen. Die Software konnte jedoch bisher noch nicht in eine allgemein nutzbare Form gebracht werden.

Die Alternative dazu ist die Verwendung eines Kommandointerpreters, der in der Lage ist, experimentspezifische Analysevorschriften, die durch den Anwender interaktiv angegeben wurden, unmittelbar in entsprechende Aktionen innerhalb der Analyse-Software umzusetzen. Der Sprachumfang eines solchen Kommandointerpreter kann von einfachen Anweisungen zur Definition von Schnitten und simplen Bedingungen bis hin zur einer komplexen Sprache reichen.

Zur Vermeidung zweier unterschiedlicher Sprachen bietet sich letztendlich an, als Kommandosprache auch die Hochsprache zu verwenden, in der der nichtdynamische Teil des Auswerteprogramms formuliert ist. Das hätte den Vorteil, daß der Benutzer einerseits nur eine einzige Sprache lernen muß, und es andererseits möglich ist, Programmcode zwischen interpretiertem und compiliertem Programmbereich auszuwechseln. Aus i. w. denselben Gründen, wie sie bereits im Rahmen der Datenerfassung diskutiert wurden (siehe Abschnitt 6.6.1), bietet sich auch hier die Verwendung der Sprache C oder evtl. sogar C++ an.

8.4.3.3 C-Interpreter und virtueller Prozessor

Ausgehend von den in den vorangegangenen Abschnitten angestellten Überlegungen wurde die Möglichkeit untersucht, einen Interpreter auf der Basis der Sprache C zu implementieren, mit dessen Hilfe die oben genannten Aufgaben einer flexiblen Analyse möglichst einfach, aber auch effizient abgewickelt werden können. Ein solcher Interpreter sollte dann in der Lage sein, einerseits in der Sprache C formulierten Programmcode auszuführen und damit sehr komplexe Analysevorschriften zu bearbeiten, andererseits auf deren Änderungen mit wenig Aufwand zu reagieren, ohne daß dazu immer wieder ein Analyseprogramm evtl. unter Informationsverlust angehalten, neu compiliert und anschließend wieder gestartet werden muß.

Es wurde in konsequenter Weiterverfolgung des bereits in [Bohn90] verwirklichten Ansatzes ein Konzept entwickelt, bei dem in Kombination mit dem in Kapitel 6 beschriebenen C-Parser ein in Software realisierter virtueller Prozessor der zentrale Bestandteil ist. Wie jeder herkömmliche Prozessor ist dieser in der Lage, speziellen Programmcode auszuführen und damit aktiv zu werden. Er ist jedoch in seinen Eigenschaften und Fähigkeiten auf die Aufgaben der Auswertung von Meßdaten optimiert und nutzt höhere Software-Schnitstellen. Ein virtueller Prozessor bietet sehr einfach die Möglichkeit, dynamisch veränderbaren Code auszuführen, was bei Verwendung eines hardwaremäßig realsierten Prozessors, der unter der Kontrolle eines herkömmlichen Betriebssystems läuft, in der Regel nur mit großem Aufwand und Eingriffen in das Betriebssystem möglich ist, solange das System nicht die Fähigkeit besitzt, Code dymanisch nachzuladen. Keines der im Institut für Kernphysik eingesetzen Betriebssysteme war bzw. ist dazu in der Lage. Die Realisierung eines virtuellen Prozessors auf der Basis eines herkömmlichen C-Programmes macht dagegen den Gebrauch dynamisch veränderbaren Codes vollkommen maschinen- und betriebssystemunabhängig.

Um eine möglichst große Flexibilität zu erhalten, wurde der Befehlssatz des virtuellen Prozessors so allgemein gehalten, daß er in der Lage ist, im wesentlichen jedes Sprachkonstrukt von C zu bearbeiten. Der C-Parser `mcc` wird wie bereits im Rahmen der Experimentbeschreibung dazu verwendet, C-Code, der in diesem Fall die Analysevorschriften enthält, zu analysieren und in Information umzusetzen, die diesem Programmcode äquivalent ist. Diese Information wird schließlich genutzt, um ein dem ursprünglichen C-Code entsprechendes Maschinenprogramm für den virtuellen Prozessor zu generien. Dieser läuft dann als wesentlicher Bestandteil des Analyseprogrammes mit direktem Zugriff auf Meßdaten, Programmvariablen und Histogramme. Der virtuelle Prozessor arbeitet den ihm zugewiesenen Programmcode interpretativ ab, wobei er von ähnlichen Mechanismen wie ein herkömmlicher Prozessor Gebrauch macht. Um den Eigenschaften seiner Implementationssprache C am besten gerecht zu werden, wurde der virtuelle Prozessor als Drei-Adreß-Maschine angelegt.

Die Software zur Realisierung dieses Konzepts wurde in wesentlichen Teilen bereits implementiert und nach verschiedenen Aspekten qualitativ untersucht. Dabei wurde festgestellt:

- die Umsetzung dieses Konzepts ist mit vertretbarem Aufwand möglich
- um ein flexibles, dynamisch arbeitendes Analysesystem zu erhalten, das funktionsfähig und allgemein nutzbar ist, ist jedoch noch erheblicher Aufwand zu investieren
- die oben genannten Anforderungen in Bezug auf Funktionalität sollten erfüllt werden
- die Abarbeitung von Programmcode durch den virtuellen Prozessor ist infolge seiner interpretativen Arbeitsweise deutlich langsamer als bei direkt compiliertem Code; gemessen wurden Faktoren zwischen drei und zehn
- der Geschwindigkeitsunterschied ist jedoch stark von benutztem Rechner und verwendetem Compiler/Optimierer aber auch von der Einbeziehung von compiliertem Analyse-Code abhängig

Kapitel 9

Inbetriebnahme und Einsatz des Datenerfassungssystems

9.1 Datenerfassung der A1-Kollaboration

MECDAS findet seine Hauptanwendung bei Experimenten der A1-Kollaboration. Hier stellt es das Standard-Datenerfassungssystem dar, das zur Durchführung einfacher Detektortests bis hin zu Dreifach-Koinzidenzexperimenten an der Drei-Spektrometer-Anlage dient. Erste Versionen wurden bereits sehr frühzeitig zu Beginn der Detektorentwicklung eingesetzt. Mit dem Start des Experimentierbetriebs im Jahre 1992 wurde es in vollem Umfang in Betrieb genommen. Seitdem befindet es sich regulär im Einsatz und hat alle im Rahmen der A1-Kollaboration durchgeführten Experimente abgewickelt. Dabei wurden bisher etwa eine Milliarde Ereignisse in Form von ca. 200 GB an Meßdaten aufgenommen.

9.1.1 Das Rechnersystem von A1

9.1.1.1 Frontend-Rechner

Zur Datenaufnahme und zur Steuerung von Experiment und Apparatur ist jedes Spektrometer vor Ort mit einem VMEbus-System ausgestattet, das die Grundlage für das Prozeßrechnersystem im Frontend-Bereich bildet. Es besitzt eine Reihe von Schnittstellen zur Ansteuerung der Experimentelektronik und wird jeweils durch ein CAMAC-System und in der Regel auch Fastbus ergänzt. Ein weiteres VMEbus-System steht für zentrale Aufgaben zur Verfügung. Elektronik und Rechner stehen in unmittelbarer Nähe zu den Detektorsystemen, wodurch eine ausgedehnte, fehleranfällige Verkabelung vermieden werden konnte. Jedes VMEbus-System ist als Master-Slave-Mehrprozessorsystem organisiert. Es verfügt über zwei CPU-Karten — z. Z. Karten der Firma Eltec, die in Zukunft auch durch andere ersetzt werden können. Beide CPU-Karten bilden jeweils im wesentlichen eigenständige Rechner. Als Master werden durchweg Eurocom 6-Karten (E6) [Elte90] eingesetzt; die Slave-Funktion übernimmt in der Regel eine Eurocom 5-Karte (E5) [Elte87].

Eine E6-Karte besitzt als Zentraleinheit einen MC 68030-Prozessor mit 2–3 MIPS Rechenleistung, dessen integrierte Memory Management Unit (MMU) sie auch für anspruchsvolle Betriebssysteme geeignet macht. Eine optionale Fließkommaeinheit (FPU[1]) wird nur bei Bedarf eingesetzt. Die Karte ist standardmäßig mit 4 MB Arbeitsspeicher ausgestattet und besitzt elementare Ein/Ausgabe-Möglichkeiten über serielle und parallele Schnittstellen. Sie verfügt über ein standardisiertes Massenspeicher-Interface (SCSI[2]), über das preisgünstige Festplatten und Bandgeräte angeschlossen werden können. Alle E6-Karten wurden mit einem Ethernet-Interface ausgerüstet, das ihnen Zugang zum Institutsnetz und damit auf einfache Art Kommunikationsmöglichkeiten mit anderen Rechnern gewährleistet. Diese werden bereits durch die in einem EPROM untergebrachte Boot-Software zum komfortablen Laden von Betriebsystem und Anwender-Software über Ethernet ausgenutzt. Damit ist die Grundvoraussetzung gegeben, beim Betrieb dieser CPU-Karten auf lokalen Massenspeicher — einer möglichen zusätzlichen Fehlerquelle durch die hier notwendige Mechanik — zu verzichten.

Die als Slaves eingesetzten Karten vom Typ Eurocom 5 haben ganz ähnliche Eigenschaften wie die E6-Karten. Sie unterscheiden sich im wesentlichen durch ihre CPUs und Schnittstellen-Hardware. Anstelle der MC 68030-Prozessoren werden hier die bis auf minimale Details kompatiblen MC 68020-CPUs verwendet. Lediglich das Fehlen einer korrekt funktionierenden MMU empfiehlt die E5-Karten für etwas einfachere Anwendungen. Auf Anwenderebene verhalten sich die beiden Rechner identisch; die etwas höher getakteten E6-Karten sind lediglich etwas schneller.

Jedes VMEbus-System ist mit einem CAMAC-Interface (CBD 8210) [CES86] der Firma CES in Form einer Einschubkarte versehen, das CAMAC-Zugriffe auf einem komplexen Branch-System von herkömmlichen VMEbus-CPUs über sog. memory mapped I/O erlaubt. In diesem Fall werden die zum Ansprechen einer CAMAC-Untereinheit notwendigen, um die Branch-Nummer erweiterten CNAFs[3] auf herkömmliche Speicheradressen des VMEbus abgebildet. Beim Zugriff auf eine solche Adresse wird die entsprechende Untereinheit adressiert, die Funktion ausgeführt und in Abhängigkeit des Funktionswerts ein Datenwort mit einem Lese- oder Schreibzugriff transferiert. Damit läßt sich der CAMAC-I/O programmtechnisch auch auf Anwenderebene sehr einfach und effizient handhaben. Die Zugriffsgeschwindigkeit wird i. w. von der CAMAC-Zykluszeit von 1 μs und der Zeit bestimmt, die der Mikroprozessor für einen Speicherzugriff benötigt und die in derselben Größenordnung liegt. In diesem Fall verlieren Mechanismen wie Blocktransfer oder Auto-CNA[4] ihre Bedeutung. Außer ihnen unterstützt das Interface ansonsten alle wesentlichen Betriebsmodi von CAMAC. Auch LAM-Handling ist unterstützt. Es bietet die Möglichkeit LAMs u. a. zur Generierung von Interrupts zu verwenden.

[1] Floating Point Unit

[2] Small Computer Systems Interface [SCSI]

[3] Tupel aus **C**rate-Nummer, Einschub-**N**ummer, Sub**A**dresse und **F**unktions-Code

[4] automatisches Durchlaufen einer Modulgruppe durch sukzessives Erhöhen von Subaddressen, Stationsnummern und Crate-Adressen

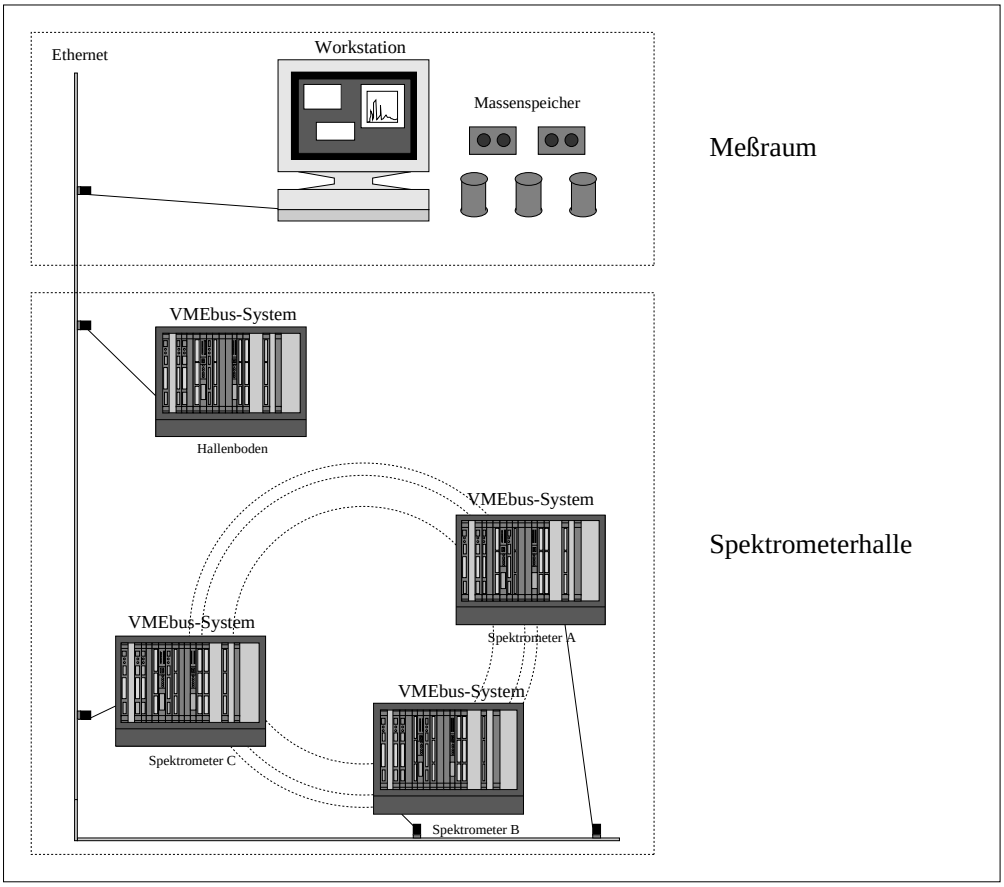

Abb. 9.1: Die Struktur des verteilten Rechnersystems, das zur Datenerfassung und Experimentsteuerung an der Drei-Spektrometer-Anlage eingesetzt wird.

Die zum Ansprechen von Fastbus eingesetzte Interface-Karte ist eine Eigenentwicklung des Instituts [Merl88a] und basiert auf einem entsprechenden CAMAC-Modul der Firma LeCroy [Lecr1], dessen Funktionalität für den VMEbus nachgebildet wurde. Es stellt die Verbindung zu dem Segment Manager SMI 1820 von LeCroy dar, der die Kontrolle über das jeweils eingesetzte Fastbus-Segment ausübt. Infolge der höheren Funktionalität von Fastbus ist auch dessen Programmierung etwas aufwendiger als bei CAMAC, doch läßt sich auch das ohne Einschränkungen und effizient von Anwenderebene erledigen.

9.1.1.2 Host-Rechner

Die Wahl der übergeordneten Rechner war weit weniger an spezielle Anforderungen im Rahmen der Experimentdatenverarbeitung gebunden, als das im Frontend-Bereich der Fall war. Vielmehr deckten die auf dem Markt verfügbaren Workstations diese bereits von vornherein schon gut ab. Aus größtenteils historischen Gründen finden bei A1 z. Z. noch ausschießlich Workstations der Firma Digital ihren Einsatz. Bis vor kurzem waren es ausschließlich DECstations auf der Basis des RISC-Prozessors R 3000 der Firma MIPS, der je nach Ausfüh-

rung eine Integer-Rechenleistung von etwa 20–30 MIPS erreicht. Die eingesetzten Workstations besitzten Arbeitsspeicher zwischen 32 und 64 MB, mindestens ein Ethernet-Interface und in der Regel zwei serielle V.24-Schnittstellen. Über ein SCSI-Interface, das bei Bedarf auch um ein weiteres ergänzt werden kann, sind an jeder Workstation Festplatten und Bandlaufwerke angeschlossen, deren Größe und Anzahl stark differieren und davon abhängig sind, für welche Zwecke die jeweiligen Rechner konkret eingesetzt werden.

Seit kurzer Zeit werden auch Workstations anderer Bauart mit deutlich höherer Leistung als bei DECstations verwendet. Insbesondere die neuen Workstations der Firma DEC auf der Basis des Alpha-Prozessors, die eine 64-Bit-Architektur besitzen, werden in Zukunft die älteren Modelle ergänzen und schließlich ablösen. Bisher genügten diese jedoch noch vollständig den an sie gestellten Anforderungen.

9.1.2 Die verwendete Systemsoftware

Auf Betriebssystemebene wird durchgängig UNIX eingesetzt. Die DECstations arbeiten unter Ultrix, dem Standardbetriebssystem der mit MIPS-Prozessoren arbeitenden Workstations von Digital, das bereits für VAX-Rechner verfügbar war und auch im Institut lange Zeit eingesetzt wurde. Ultrix ist weitestgehend zu BSD 4.3 kompatibel, auf dem auch die UNIX-Variante VMEbsd basiert, mit dem die VMEbus-Rechner arbeiten.

Alle beteiligten Rechner sind an Ethernet angeschlossen. Die Fähigkeit von VMEbsd, vollständig ohne lokalen Massenspeicher zu arbeiten (sog. Diskless-Betrieb), ohne in seiner Funktionalität größere Einschränkungen hinnehmen zu müssen, wurde konsequent ausgenutzt und half, einerseits Komplexität und Fehleranfälligkeit in diesem Bereich zu reduzieren und andererseits die Verwaltung des aus einer Vielzahl von Rechnern bestehenden verteilten Systems zu erleichtern. Ähnliches gilt auch für die Workstations, für die sich zwar der Diskless-Betrieb nicht als sinnvoll erwies, die jedoch in Form eines lose gekoppelten Rechner-Clusters so weit wie möglich Ressourcen gemeinsam nutzen. In beiden Fällen wird intensiv von den Eigenschaften von NFS Gebrauch gemacht, das es erlaubt, Programme und Daten sehr einfach und transparent zu verteilen. Workstations und VMEbus-Rechner unterliegen einer gemeinsamen Benutzerverwaltung und nutzen gleichermaßen netzwerkweit verfügbare Dienste. Am Netzwerk angeschlossene X-Terminals bzw. viele der im Institut eingesetzten Drucker können von allen UNIX-Rechnern gleichberechtigt genutzt werden. Damit steht auf allen Ebenen eine identische Benutzer- und Entwicklungsumgebung zur Verfügung.

Auch neue Workstation-Architekturen wie SPARCstations von SUN oder Alpha-Workstations von Digital fügen sich sehr gut in das bestehende Rechnersystem ein. Die UNIX-Varianten Solaris und OSF/1, mit denen diese Maschinen arbeiten, stammen zwar nicht mehr direkt von BSD 4.x ab, doch besitzten sie i. w. voll dessen Funktionalität, und sind zumindest auf Anwenderebene nicht zuletzt wegen sich immer stärker etablierender Standards gut kompatibel.

9.1.3 Kommunikation

Zur rechnerübergreifenden Kommunikation ist jeder Rechner mit einem und in besonderen Fällen auch mit mehreren Ethernet-Interfaces ausgestattet und darüber mit dem Institutsnetzwerk verbunden. Damit können alle Frontend-Systeme und Workstations untereinander und auch mit anderen Rechnern innerhalb des Instituts kommunizieren. Mit dem Einsatz von UNIX stehen hierbei standardmäßig die Protokolle von TCP/IP zur Verfügung, die für verschiedenste Zwecke der Kommunikation eine gute Grundlage bilden.

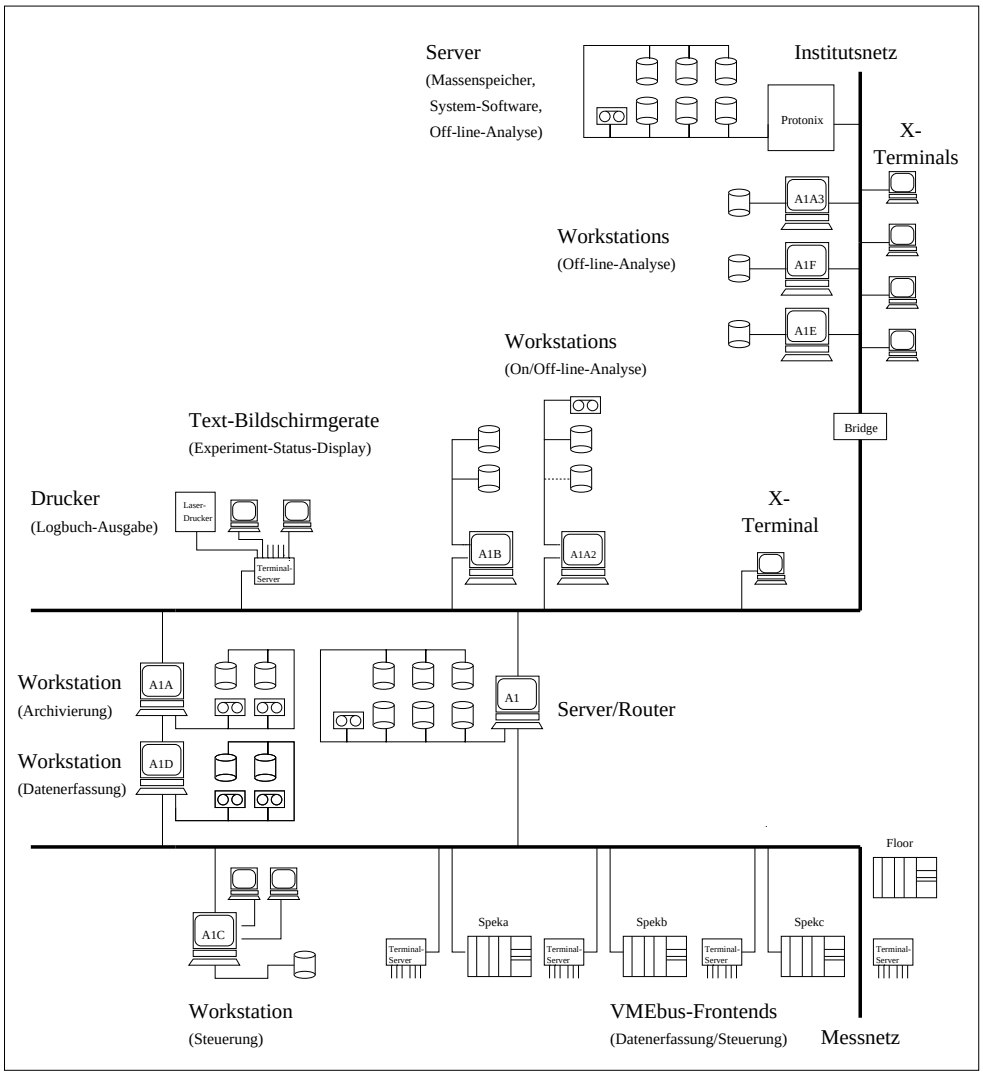

Abb. 9.2: Schematische Darstellung des A1-Rechnernetzes Es besteht aus zwei Segmenten, die über einen zentralen Server (A1) mit Routing-Funktion gekoppelt sind. Die im Meßnetz installierten DECstations werden als Hostrechner für die Datenerfassung (A1d) und Steuerungsrechner (A1c) eingesetzt und sind direkt mit den VMEbus-Rechnern (Speka, Spekb, Spekc, Floor) verbunden. Außerhalb stehen weitere DECstations und Alpha-Workstations für die Off-line-Analyse zur Verfügung.

Neben den Aufgaben zur Steuerung von Datenerfassung und Apparatur decken sie insbesondere auch die Anforderungen für den Transport der Meßdaten ab. Beides konnte bisher noch sehr gut auf der Basis von Ethernet und mithilfe der Standard-Protokolle abgewickelt werden. Um jedoch einerseits eine unnötige Belastung des Institutsnetzes zu vermeiden, andererseits unempfindlich gegenüber äußeren Einflüssen zu werden, wurde zusätzlich zu den bereits existierenden Segmentierungen des Institutsnetzes eine weitere Aufteilung des Netzes vorgenommen. Das den A1-Rechnern zugängliche Netz wurde aufgespalten in ein Off-line-Netz, das unmittelbar mit dem Institutsnetz verbunden ist, und ein On-line-Netz, an das nur die an der Datenerfassung und Experimentsteuerung unmittelbar beteiligten Rechner angeschlossen sind (s. Abb. 9.2). Lediglich mithilfe eines Rechners, der zwei Netzwerk-Interfaces besitzt und damit an beide Netzsegmente angeschlossen werden konnte, ist ein gezielter Datenaustausch zwischen Meßnetz und Außenwelt möglich. Dieser Rechner stellt neben seiner Funktion als zentraler Server in beiden Netzen einen TCP/IP-Router dar, der in der Lage ist, Netzwerkpakete, die für das jeweils andere Segment bestimmt sind, weiterzureichen. Er gewährleistet eine kontrollierte und auch sehr gute Entkopplung der beiden Teilnetze bei geringstmöglicher Funktionsbeeinträchtigung der beteiligten Rechner.

Trotz der Segmentierung konnte daher auch von den bereits angesprochenen Vorzügen einer gemeinsamen Ressourcennutzung und Verwaltung der Rechner Gebrauch gemacht werden. Es konnte die Möglichkeit genutzt werden, einerseits die Workstations andererseits aber auch die VMEbus-Rechner zu einem Rechner-Cluster zusammenzufassen. Der Workstation-Cluster, der z. Z. aus sechs DECstations und vier Alpha-Rechnern besteht, ist eingebettet in den Instituts-Cluster, auf den insbesondere im Off-line-Betrieb zurückgegriffen wird, und der allgemeine Service-Leistungen zur Verfügung stellt. Er ist funktional unterteilt in Rechner für Off-line- und On-line-Aufgaben und allgemeine Dienstleistungen und entsprechend auf die Netzsegmente verteilt. Im On-line-Bereich findet eine Unterscheidung statt zwischen Rechnern, die in erster Linie für übergeordnete Aufgaben der Datenerfassung, Archivierung der Meßdaten, On-line-Analyse und Überwachung zuständig sind, und andere, die für den Betrieb von Datenbank und Programmen zur Steuerung der Apparatur eingesetzt werden. Sie haben alle eine direkte Kommunikationsmöglichkeit mit den VMEbus-Rechnern.

9.1.4 Zusätzliche Software

Die Software zur Datenerfassung ist Bestandteil eines komplexen Gesamtsystems. Es enthält daneben weitere Softwarepakete, die einerseits der Steuerung und Überwachung der Experimentierapparatur dienen und andererseits zur On- und Off-line-Auswertung der Meßdaten Verwendung finden.

9.1.4.1 Das Steuerungssystem

Infolge der Komplexität der Drei-Spektrometer-Anlage war auch zur Steuerung und Überwachung der Apparatur ein leistungsfähiges Software-System notwen-

dig, das ebenso wie die Datenerfassungs-Software dezentral organisiert ist und auf dem verteilten Rechnersystem von A1 arbeitet. Es wurde im Rahmen mehrer Doktor- und Diplomarbeiten realisiert [Kram95, Kund95, Stef93, Hake93, Mart93].

Ebenso wie das Datenerfassungssystem wurden auch hier die unterschiedlichen Aufgaben auf einzelne Prozesse aufgeteilt, die parallel auf den Meßrechnern laufen und über das Message-Passing-System MUPIX miteinander kommunizieren. Das komplette System ist in mehrere Schichten verschiedener Funktionalität aufgeteilt. Die untersten, hardwarenahen Schichten bestehen aus Gerätetreibern für verschiedenste elementare I/O-Interfaces (ADCs, DACs, Digital-I/O, IEC-bus, CAMAC, V24) aber auch komplexe Geräte, die die Bestandteile des Meßapparatur bilden. Die Treiber definieren standardisierte Schnittstellen für die übergeordnete Software und sind über eine Konfigurationsdatenbank leicht an unterschiedliche Bedinungen anpaßbar. Sie stellen die Ansteuerung von Geräten netzwerkweit als Dienste zur verfügung.

Das Laufzeitsystem besteht neben den Treibern auch aus zentralen Diensten. Eine besondere Rolle spielt dabei das Monitorsystem, über das der Zustand der Anlage automatisch überprüft und eine eventuelle Fehlfunktion an den Experimentator weitergeleitet wird. Im Extremfall kann diese Information auch zur automatischen Beeinflussung des Meßablaufs genutzt werden. Der Zustand der Apparatur wird mit den im Rahmen der Datenerfassung aufgenommenen Meßdaten auf Band geschrieben.

9.1.4.2 Software zur On- und Off-line-Analyse

Aufbauend auf dem bereits vorgestellten Konzept der MECDAS-On-line-Analyse wurde im Rahmen des A1-Projekts weiterführende Software in Form von Unterprogrammbibliotheken und abgeschlossenen Programmen realisiert. Es steht zur Zeit Software für die Aufgabenbereiche

- Berechnung der Driftkammerkoordinaten [Dist95]
- Rückrechnung der Driftkammerkoordinaten auf Targetkoordinaten [Kram95]
- Überwachung der on-line aus den Meßdaten gewonnen Information im Rahmen des oben genannten Monitorsystems [Kram95]
- Graphische Bedienoberfläche für die On-line-Analyse [Schr93]

zur Verfügung. Sie wird ergänzt durch Software wie Cindy++ [Dist93a] und Espace [Offe93], die mehr im Rahmen der Off-line-Analyse eingesetzt wird.

9.2 Einsatz bei der Detektorentwicklung

Der erste Einsatz der im Rahmen dieser Arbeit entwickelten Datenerfassungs-Software fand nun nicht unter den im vorhergehenden Abschnitt beschriebenen Umständen statt, die die endgültigen Situation beschreiben, sondern wurde schon sehr frühzeitig in der Entwicklungsphase der Drei-Spektrometer-Anlage nötig. Wenn auch nicht in der endgültigen Funktionalität, so wurde doch bereits zur Entwicklung und dem Test der einzelnen Komponenten des Detektorsystems funktionsfähige Software zur Datenerfassung benötigt. Zu diesem Zeitpunkt konnte weder auf ein leistungsfähiges Workstation-Cluster zurückgegriffen werden, noch war die Infrastrukur für Software-Entwicklung und Rechnerbetrieb bereits ausreichend ausgebaut. Gleichzeitig waren auch die Arbeiten an der Datenerfassungs-Software erst in der Anfangsphase, so daß in erster Linie auf erste Prototypen zurückgegriffen werden mußte.

Auf der anderen Seite stellten Detektortests auch ganz andere Anforderungen an die Software zur Datenerfassung. So war dafür in der Regel der lokale Betrieb des Datenerfassungssystems ausschließlich auf einzelnen VMEbus-Rechnern ausreichend; ein umfangreiches und gekoppeltes Rechnersystem war gar nicht nötig. Vielmehr mußte es möglich sein, die in verschiedenen Bereichen parallel stattfindende Detektorentwicklung unabhängig voneinander und ohne gegenseitige Beeinflussung durchzuführen. In vielen Fällen erübrigte sich auch die Archivierung der Meßdaten in größerem Umfang; statt dessen war es notwendig, auf den VMEbus-Rechnern neben der Datenerfassung auch eine rudimentäre Online-Analyse und die graphische Darstellung der erzielten Meßergebnisse vorzunehmen. In der Regel wurden die Rechner im Labor betrieben und über direkt angeschlossene, einfache Bildschirmgeräte bedient.

Zu Beginn, im Jahre 1988, standen für diese Zwecke eine Reihe von VMEbus-Rechnern zur Verfügung, die lediglich unter dem Betriebssystem OS-9 arbeiteten, doch einen sehr guten Stand-alone-Betrieb gewährleisteten und für die ersten Versionen von MECDAS eine ausreichende Plattform darstellten. Das Datenerfassungssystem stellte zu diesem Zeitpunkt die geforderte Grundfunktionalität bereit, war jedoch in vielen Bereichen noch rudimentär implementiert. Viele durch das Betriebssystem OS-9 auferlegte Beschränkungen konnten mit der Verfügbarkeit von UNIX im VMEbus-Bereich im Jahre 1991 bald beseitigt werden. Nach und nach konnte auf dieser Basis das System mit verbesserter Funktionalität genutzt werden.

Einhergehend mit den Detektortests bei in der A1-Kollaboration wurden die ersten Versionen von MECDAS auch im Rahmen der A2-Kollaboration eingesetzt. Ursprünglich fand die Software auch hier bei der Entwicklung der Detektoren Verwendung. Noch unter OS-9 wurde sie bereits im Jahre 1988 bei Experimenten mit energiemarkierten Photonen am Beschleuniger MAMI A [Kund88] und seit 1989 bei regulären Experimenten mit reellen Photonen an MAMI B eingesetzt[Peis89]. Ursprünglich wurde nach ihrer Anpassung dabei noch auf existierende Analyse-Software aus dem CAROLA-Paket [Klei87a] zurückgegrif-

fen; später wurde MECDAS zusammen mit dem Datenauswertesystem GOOSY[5] [Esse88] betrieben, das unter dem Betriebssystem VMS auf VAX-Rechnern lief und eine komfortable On- und Off-line-Analyse von Meßdaten aus kernphysikalischen Experimenten erlaubte. Infolge der deutlich geringeren Anforderungen der damit durchgeführten A2-Experimente und eine Beschränkung auf minimale, aber ausreichende Funktionalität gewährleistete die Software bereits in dieser Version und in Kombination mit GOOSY einen stabilen Experimentierbetrieb und wurde bis in die nähere Vergangenheit regelmäßig eingesetzt. Es fand im Rahmen der A2-Kollaboration sogar außerhalb der Universität Mainz am Bonner Beschleuniger ELSA Verwendung [Kall93].

9.3 Inbetriebnahme der Drei-Spektrometer-Anlage

Zur Inbetriebnahme von Spektrometer A und dessen Detektorsystems wurde das Datenerfassungssystem MECDAS erstmals im Sommer 1992 regulär eingestzt und ist seitdem bei allen Messungen an der Drei-Spektrometer-Anlage in Verwendung. Die ersten Testmessungen dienten zur Untersuchung der Eigenschaften des Spektrometers und des Detektorsystems und wurden zum großen Teil mit elastischer Elektronenstreuung an dünnen ^{181}Ta- und ^{12}C-Targets durchgeführt. Sie umfaßten neben Funktionstests des Detektorsystems die Bestimmung der Rückrechnungsmatrizen zur Durchführung von Software-Korrekturen und des Auflösungsvermögens. Abbildung 9.3 zeigt das erste mit Spektrometer A aufgenommene Driftkammerspektrum. Es demonstriert neben korrekter Funktion von Spektrometer und Detektorsystem auch die hohe Auflösung und die guten Abbildungseigenschaften der Anordnung. Die elastische und die ersten inelastischen Linien aus der Reaktion ^{12}C(e,e′), E = 495 MeV, Θ = 49,2° lassen sich leicht identifizieren, ebenso wie der breite quasielastische Peak zu großen Drahtnummern hin.

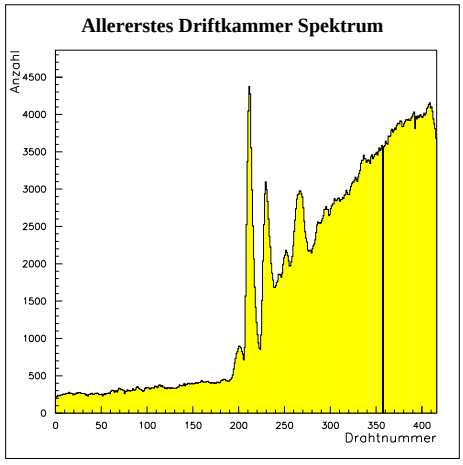

Abb. 9.3: Erstes Drahtspektrum an der Drei-Spektrometer-Anlage. Die Drahtnummer entspricht in erster Näherung dem Impuls des gestreuten Elektrons [SFBa93].

Bei der Inbetriebnahme von Spektrometer A befanden sich die beiden anderen Spektrometer noch in der Fertigstellung. Die ersten Experimente beschränkten sich daher zwangsläufig auf Einarmmessungen. Für die Datenerfassung bedeutete das eine Reduzierung der Komplexität von Rechner-Hardware und der Software. Es wurde lediglich ein einziger VMEbus-Rechner im Frontend-Bereich eingesetzt, der für die Datenerfassung an dem Sepktrometer sorgte und die aufgenommenen Daten an einen übergeordneten Rechner zur Archivierung weitergab, wo sie auf

[5] GSI On-line Off-line SYstem

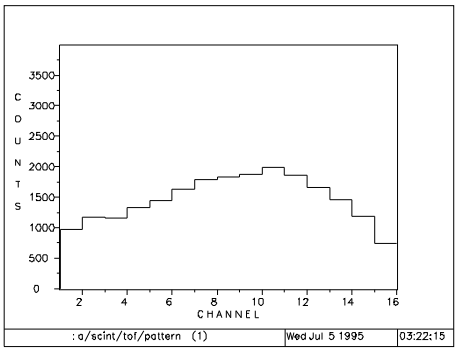

Abb. 9.4: Die Segmente der Time-of-Flight-Szitillatorebene.

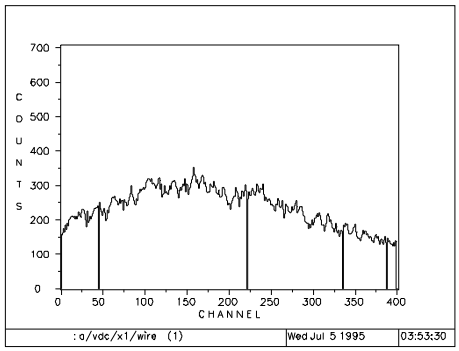

Abb. 9.5: Drahtspektrum der x1-Ebene der Driftkammer von Spektrometer A. Es sind deutlich defekte Drähte zu erkennen.

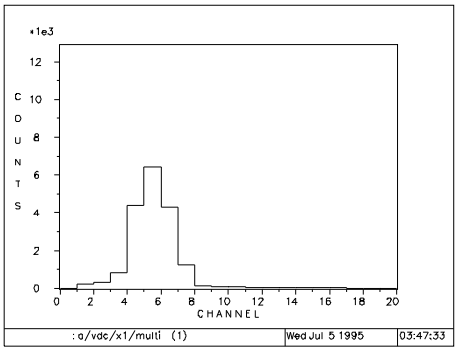

Abb. 9.6: Multipolitätsverteilung der x1-Ebene von VDC A. Im Mittel stehen etwa 5 Stützstellen für die Spurberechnung zur Verfügung.

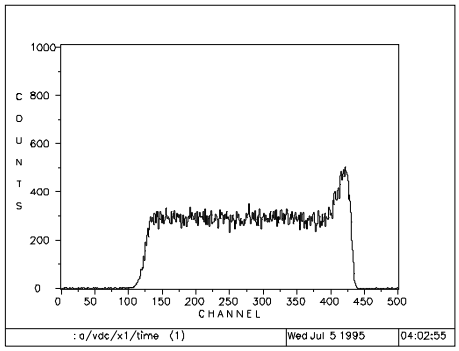

Abb. 9.7: Driftzeitspektrum der x1-Ebene von VDC A. Die kirchturmartige Überhöhung wird durch inhomogenen Feldverlauf in der Nähe der Signaldrähte verursacht.

Platte oder Magnetband abgespeichert wurden. Ein Eventbuilding war natürlich nicht notwendig. Die Software auf dem Hostrechner beschränkte sich daher ausschließlich auf das Übermitteln von Information: einerseits Steueranweisungen des Benutzer an den Frontend und andererseits Transport der Meßdaten vom Frontend auf das Archivierungsmedium. Die Software auf dem Frontend dagegen ist unabhängig davon, ob das Experiment ein Ein- oder Mehrarmexperiment darstellt. In Anhang B.2.1 ist die Experimentbeschreibung für Spektrometer A wiedergegeben, aus der letztendlich die Frontend-Software generiert wird.

9.3.1 Funktionstest des Detektorsystems

Erstes Ziel der Testmessungen war die Überprüfung der Funktionalität der einzelnen Komponenten vom Spektrometer über das Detektorsystem bis hin zur Datenerfassung. Das Detektorsystem bestehend aus zwei Doppeldriftkammern zur Spurrekonstruktion, zwei segmentierten Szintillationsdetektoren als Trigger und zur Teilchendiskriminierung und ein Gas-Čerenkov-Vetodetektor zur Unterscheidung von Pionen und Elektronen bzw. Positronen. Ihre Funktion läßt sich

bereits anhand der Rohspektren gut beurteilen. Die Abbildungen 9.4 bis 9.7 zeigen das Aussehen einiger typischer Spektren, wie sie bereits on-line, unmittelbar aus den Meßdaten generiert werden können. Während diese Abbildungen durch die beschriebene MECDAS-Software zur graphischen Ausgabe von Histogrammen mithilfe des Programmes `display` generiert wurden, demonstrieren die Abbildungen 9.8 und 9.9 die Graphikausgabe auf der Basis des Programmes `paw` [Brun91], mit dessen Hilfe entweder unmittelbar On-line-Histogramme ausgegeben werden können, das aber auch eigenständig zur Analyse der Meßdaten bentuzt werden kann.

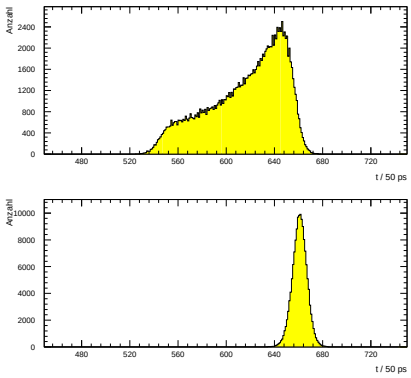

Abb. 9.8: Zeitauflösung der Triggerdetektoren von Spektrometer A; Rohspektrum (oben) und laufzeitkorrigiertes Zeitspektrum (unten) [SFB95]

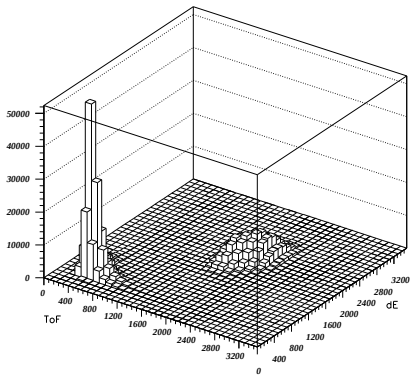

Abb. 9.9: Pulshöhenverteilung des aus den beiden Szintillatorebenen dE und ToF bestehenden Triggerdetektorsystems. Zu erkennen sind minimal-ionisierende Teilchen (links) und Protonen (rechts) [SFB95]

9.3.2 Experimentelles Raytracing

Die Berechnung der Targetkoordinaten aus den vom Detektorsystem gemessenen Bahndaten am Austritt des Spektrometers verlangt die Kenntnis der Transfermatrizen, die die Abbildungseigenschaften des Spekrometers beschreiben. Die Targetkoordinaten sind die Impulsablage δ, der Teilchenort x_0 und der Teilchenwinkel Θ_0 in der dispersiven Ebene, der Teilchenort y_0 un der Teilchenwinkel Φ_0 in der nicht-dispersiven Ebene sowie die Tiefe z_0. Die Teilchenbahn am Austritt des Spektrometers wird durch die Ortskoordinate x und den Winkel Θ in der dispersiven Ebene und durch die Ortskoordinate y und den Winkel Φ in der nicht-dispersiven Ebene beschrieben. Aus den vier gemessenen Bildebenenkoordinaten können jedoch nur ebenfalls vier Targetkoordinaten eindeutig bestimmt werden. Daher wird bei der Bestimmung der Transfermatrizen von einem in x_0- und z_0-Richtung punktförmigen Target ausgegangen, d. h., es gilt $x_0 = 0$ und $z_0 = 0$. Da trotz sorgfältiger magnetischer Vermessungen der Spektrometermagnete die so bestimmten Transfermatrizen noch Abweichungen von der Realität aufweisen und da trotz Optimierung von Quadrupol- und Sextu-

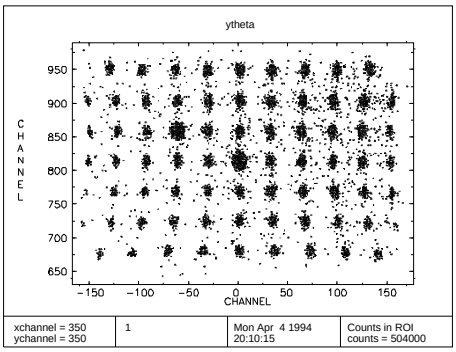

Abb. 9.10: Abbild des Loch-Kollimators von Spektrometer A in der y-Θ-Ebene der Detektorkoordinaten. Es gibt deutlich die Struktur der $7 \cdot 11$ Löcher wieder, zeigt jedoch auch die durch Abbildungsfehler des Spektrometers verursachte Verzerrung.

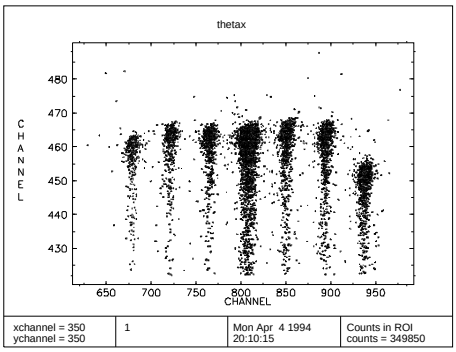

Abb. 9.11: Θ-x-Abbild der elastischen Linie von ^{181}Ta bei einer Messung mit dem 77-Loch-Kollimator. Dargestellt sind die Ereignisse der in der der Mittelebene liegenden Lochreihe.

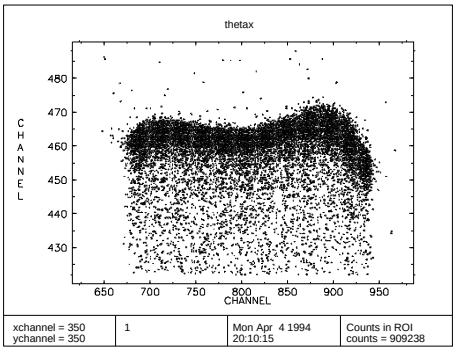

Abb. 9.12: Θ-x-Abbild der elastischen Linie von ^{181}Ta bei einer Messung mit dem 28-msr-Kollimator.

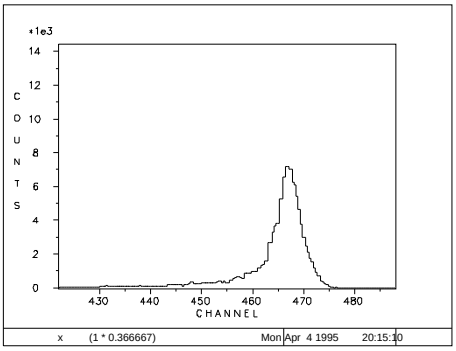

Abb. 9.13: Spektrum der elastischen Linie von ^{181}Ta in der x-Koordinaten bei vollem Raumwinkel von 28 msr.

polfeldeinstellung bei extremen Winkeln Abweichungen von der idealen Punkt-zu-Punkt-Abbildung auftreten, ist die experimentelle Bestimmung der Transfermatrizen notwendig. Zu diesem Zweck wurden umfangreiche Testmessungen mit elastisch an ^{12}C und ^{181}Ta gestreuten Elektronen bei verschiedenen Elektronenenergien zur Untersuchung der optischen Eigenschaften des Spektrometers durchgeführt [Korn95, Scha95].

Bei diesen Messungen müssen die Targetkoordinaten zunächst bekannt sein, um aus der Verknüpfung von Bildebenenkoordinaten und Targetkoordinaten die Abbildungseigenschaften ermitteln zu können. Mithilfe von Lochkollimatoren wurden bestimmte Startwinkel Θ_0 und Φ_0 herauspräpariert. Dabei wurde u. a. ein Loch-Kollimator eingesetzt, der aus einem Blech einer speziellen Legierung mit hoher Dichte besteht und 77 Bohrungen besitzt. Die durch die Löcher hindurchtretenden gestreuten Elektronen erzeugen wegen der Punkt-zu-Punkt-Abbildung des Spektrometers ein entsprechendes Bild, das durch das Detektorsystem unmittelbar nachgewiesen werden kann (Abb. 9.10). Aufgrund der guten

optischen Eigenschaften des Spektrometers sind die einzelnen Löcher hierbei sehr gut trennbar, und eine Zuordnung von Detektorkoordinaten und Targetwinkeln somit leicht möglich.

So wie in Abbildung 9.10 gezeigt, kann das Bild unmittelbar während der Messung mithilfe eines On-line-Analyseprogrammes dargestellt werden. Dabei wird von einer im Rahmen der A1-Kollaboration entwickelten Unterprogrammbibliothek Gebrauch gemacht, mit deren Hilfe die von den vier Driftkammerebenen gemessenen Zeiten in die magnetoptischen Koordinaten (x, Θ, y, Φ) umgerechnet werden können [Dist93b]. Ähnliches gilt auch für Abbildung 9.11. Sie stellt für die gleiche Messung das Abbild der elastischen Linie in der Θ-x-Ebene dar, wobei hier durch einen Schnitt auf einen bestimmten y- und Φ-Bereich nur die Ereignisse der in der Mittelebene liegenden Lochreihe ausgewählt wurden. Abb. 9.12 zeigt, wie das Spektrum unter identischen Bedingungen jedoch ohne Verwendung des Loch-Kollimators aussieht. Für die gleiche Messung zeigt Abb. 9.13 das Spektrum der elastischen Linie in der berechneten Koordinaten x.

9.4 Koinzidenzexperimente

9.4.1 Das BGO-Ball-Experiment

Ende 1992 wurde als erstes Koinzidenzexperiment ein mehrwöchiges Experiment zur Mehrhadronen-Elektroproduktion an ^{12}C durchgeführt. Dabei diente Spektrometer A dem Nachweis des gestreuten Elektrons, während die geladenen Hadronen (Protonen und Pionen) durch einen BGO-Ball-Detektor in Koinzidenz nachgewiesen wurden. Dieses Experiment zeigte unter anderem, daß aufgrund der hohen Qualität des MAMI-Elektronenstrahls ein untergrundempfindliches Detektorsystem wie der BGO-Ball in unmittelbarer Nähe zum Elektronenstrahl betrieben werden kann. Weiterhin lieferte es neue physikalische Erkenntnisse zur Multiplizität und Energieabhängigkeit der Hadronenemission im quasielastischen, „dip“- und Δ-Resonanz-Gebiet [Kund95, Edel95].

Für die Datenerfassung war dieses Experiment lediglich eine Erweiterung der zuvor durchgeführten Einarmexperimente. Aufgrund technischer Schwierigkeiten war nicht, wie vom Konzept von MECDAS her vorgesehen, für den BGO-Ball als zweites Detektorsystem ein eigenes Frontend-System zur Verfügung gestellt worden, sondern die hinzukommende Experimentelektronik wurde in das Meßsystem von Spektrometer A integriert. Die einfache Konfigurierbarkeit der Datenerfassungs-Software unterstützte diese Vorgehensweise, ohne daß Eingriffe in die Software notwendig gewesen wären. Jedoch traten dabei zwei Probleme auf, die bei einer konsequenten Aufgabenteilung hätten vermieden werden können. Einerseits verschlechterte sich nun durch die erheblich stärkere Belastung des Frontend-Rechners dessen Datenaufnahmegeschwindigkeit deutlich, andererseits zeigte die nun wesentlich komplexer gewordene Meßelektronik eine deutlich höhere Fehleranfälligkeit bzw. Unüberschaubarkeit bei der Fehlersuche. Bei einer Fortsetzung des Experiments zu einem späteren Zeitpunkt wurde daher zur individuellen Bearbeitung der Datenerfassungsaufgaben für das BGO-Ball-Detektorsystem ein separates Frontend-System mit Erfolg verwendet.

9.4.2 Messungen zur (e,e'p)-Reaktion an ^{12}C und ^{16}O

Nach der Inbetriebnahme von Spektrometer B im Sommer 1993 wurden erstmals Koinzidenzexperimente durchgeführt, die dem ursprünglich ausgearbeiteten Datenerfassungskonzept entsprechen, das jedem Detektorsystem ein eigenes, unabhängig arbeitendes Frontend-System zuordnet und für das Synchronisieren und Zusammenfügen der Daten entsprechende Maßnahmen vorsieht. Diese ersten Koinzidenzexperimente mit zwei Spektrometern zum Nachweis zweier geladener Teilchen in den Ausgangskanälen einer Reaktion wurden im Herbst 1993 durchgeführt. Sie dienten in erster Linie dazu, durch die Reproduktion bereits an anderer Stelle gemessener Daten die Funktionsfähigkeit der Apparatur nachzuweisen. Es wurden $(e, e'p)$-Messungen an ^{12}C und ^{16}O durchgeführt [Saue95, Wolf95]. Für die Messungen an ^{16}O wurde ein am Institut entwickeltes Wasserfalltarget [Voeg82] verwendet, das bereits bei früheren Untersuchungen zum Einsatz kam.

Die Koinzidenz-Ansprechwahrscheinlichkeit zwischen den beiden Spektrometern A und B wurde mit einer ^{1}H(e,e'p)-Reaktion am Wasserfall-Target gemessen. Durch Auswahl der kinematischen Parameter wurde sichergestellt, daß zu jedem in Spektrometer B, das einen im Vergleich zu Spektrometer A kleineren Raumwinkel besitzt, nachgewiesenen Elektron das zugehörige Rückstreuproton in Spektrometer A detektiert werden konnte.

Abbildung 9.14 zeigt ein flugzeitkorregiertes „missing energy"-Rohspektrum für (e,e'p) an einem ^{12}C-Target mit der Massenbelegung 30 mg/cm^2 bei einem mittleren „missing momentum" von $p_m = 350$MeV/c. Die große Häufigkeitsspitze (Peak) bei 15.96 MeV entspricht der Separationsenergie des herausgeschlagenen Protons, die übrigen Peaks den Anregungsenergien des Restkerns ^{11}B.

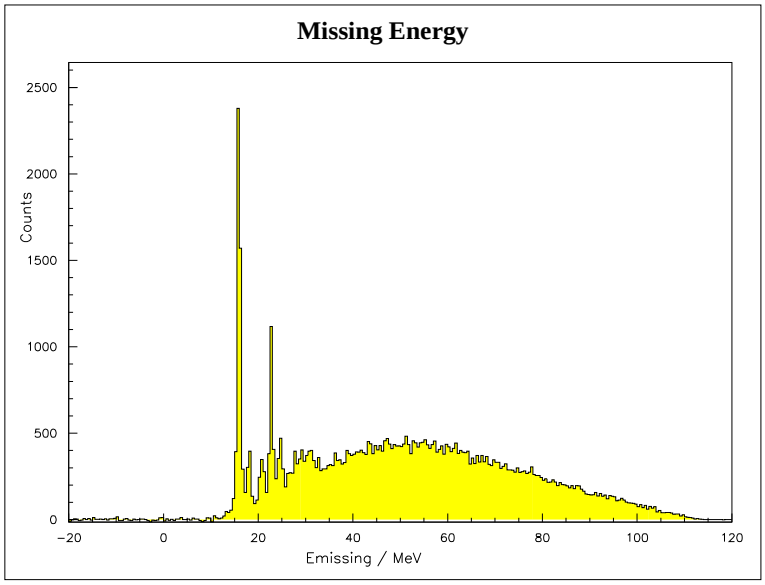

Abb. 9.14: Flugzeitkorregiertes „missing energy"-Rohspektrum für (e,e'p) an ^{12}C [Jahr94].

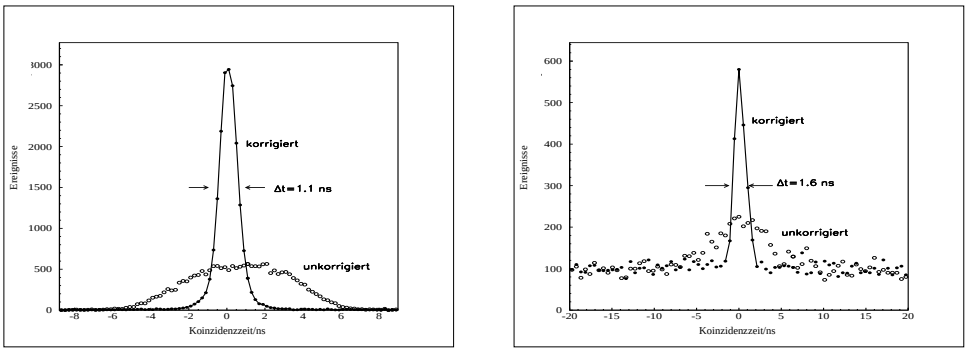

Abb. 9.15: Koinzidenzzeitspektren für ^{16}O(e,e'p) bei $p_m = 120$ und 590 MeV/c, einem Strahlstrom von 5 bzw. 25 μA und 50 mg/cm^2-Target [Jahr94].

Die Zwei-Spektrometer-Anordnung wurde durch Reproduktion von (e,e'p)-Messungen bei bereits an NIKHEF vermessenen Kinematiken überprüft [Blom95]. In den Koinzidenzzeitspektren (Abb. 9.15) zeigen die dabei erfaßten Daten ein ausgezeichnetes Signal-zu-Rausch-Verhältnis. In der Abbildung mit *korrigiert* bezeichneten Kurve sind die Weglängenenunterschiede der Elektronen und Protonen relativ zu den Zentralstrahlen von Spektrometer A und B berücksichtigt.

9.5 Erste Dreifach-Koinzidenz-Experimente

Mit der erfolgreichen Inbetriebnahme des dritten Spektrometers im Frühjahr 1995 war es erstmalig möglich, auch Dreifach-Koinzidenz-Experimente mit der Drei-Spektrometer-Anlage durchzuführen [Kahr95]. Das hierarchische Client-Server-Konzept von MECDAS erlaubte dabei die Abwicklung der Datenerfassung mit der Software, die bis dahin zufriedenstellend für Koinzidenz-Experimenten mit zwei Spektrometern eingesetzt wurde. Es waren, wie vom Konzept vorgesehen, für diesen Zweck weder spezielle Software-Entwicklungen noch Eingriffe in die existierende Software notwendig. Lediglich die Modifikation der Experimentbeschreibung war erforderlich, um das dritte Spektrometer in das Gesamtsystem einzubinden. Dabei konnte die Ähnlichkeit der Spektrometer, damit auch der für die Datenerfassung eingesetzen Hardware und folglich auch dafür notwendiger Software jedoch nur beschränkt ausgenutzt werden. Denn es kam bei Spektrometer C neben etablierter Hardware erstmals das neue TDC-System auf der Basis des im Hause entwickelten TDC-2001-Chips für die Erfassung der Driftkammerzeiten zum Einsatz [Claw95]. Die Vorkehrungen für seine Inbetriebnahme im Rahmen der Datenerfassung beschränkte sich dabei auf die Erstellung eines entsprechenden Gerätetreibermoduls.

Als Vorbereitung auf geplante (e,e'pp)-Experimente wurde ein erstes Tripel-Koinzidenz-Experiment durchgeführt, bei dem die kinematisch überbestimmte Reaktion ^{3}He(e,e'pd) untersucht wurde. Dabei wurde ausgenutzt, daß durch Impulsmessung von zweien der drei Streuprodukte der Impuls des dritten eindeutig bestimmt ist. Mit dem Nachweis von Elektronen in Spektrometer B und Protonen in Spektrometer A bei gegebenen Winkel- und Magnetfeldeinstellungen

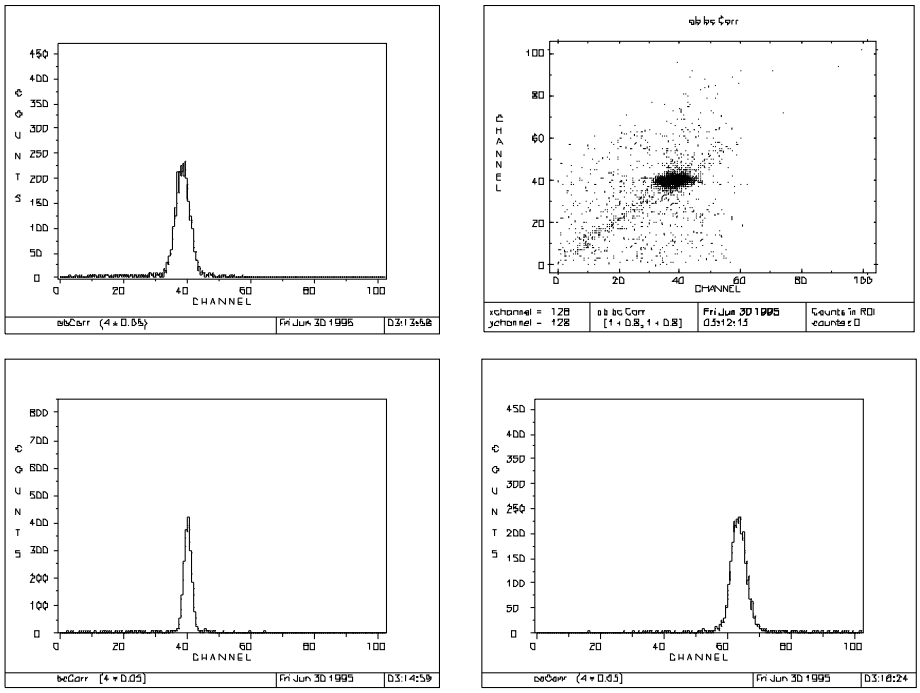

Abb. 9.16: Koinzidenzzeitspektren, die die zeitlichen Abstände zwischen dem Teilchennachweis in den drei Spektrometern für Dreifachkoinzidenzereignisse darstellen. Die Gegenüberstellung im Scatter-Plot zeigt eine deutliche Korrelation als Hinweise auf echte Koinzidenzen.

konnten in der Tat die der entsprechenden Reaktion entstammenden Deuteronen in Spektrometer C mit dem durch die Kinematik des Dreikörperproblems bestimmten Impuls detektiert und so die korrekte Funktion der Drei-Spektrometer-Anordnung verifiziert werden. Die Abbildung 9.16 zeigt die zeitlichen Korrelationen der Teilchennachweise in den drei Spektrometern, die auf die Registrierung vom i. w. echten Tripel-Koinzidenzen hinweisen.

9.6 Einsatz bei anderen Kollaborationen

Außer bei der A1-Kollaboration findet die aktuelle Version der Datenerfassungs-Software inzwischen auch bei anderen Experimenten innerhalb des Instituts ihren Einsatz. Das sind einerseits die neusten Experimente der A2-Kollaboration [Peis95]. Zur Vorbereitung des LARA-Experiments wurde ein Rechnersystem von ähnlicher Komplexität wie an der Drei-Spektrometer-Anlage eingesetzt, bestehend aus drei Master-Slave-Frontend-Rechnern und einer Workstation. Während im Frontend-Bereich E6/E5- und E6/E6-Kombinationen Verwendung fanden, wurde als übergeordneter Rechner, der der Steuerung der Datenerfassung, dem Eventbuildung und der Archivierung der Meßdaten diente, erstmals eine Alpha-Workstation unter dem Betriebssystem OSF/1 für den regulären Meßbetrieb eingesetzt. Zur On-line-Analyse wurde CERN-Software in Kombination mit MECDAS-Analyse-Software verwendet.

Als dritte Arbeitsgruppe verwendet die A4-Kollaboration standardmäßig MECDAS [Koeb95]. Hier wurde das Datenerfassungssystem von Anfang an eingesetzt. Die Messungen beschränken sich bisher jedoch mehr auf Test-Experimente. Aufgrund der damit verbundenen niedrigeren Anforderungen erwies sich eine Ein-CPU-Lösung im Frontend-Bereich vorläufig als ausreichend. Dabei konnte von der Ein-Rechner-Varianten der Datenerfassungs-Software sehr gut Gebrauch gemacht werden. Ähnlich wie bei A2 wird auch hier neben Analyse-Software von MECDAS auch CERN-Software zur On-line-Analyse der Meßdaten eingesetzt.

9.7 Software-Verfügbarkeit

Wie bereits an verschiedenen Stellen angedeutet ist das im Institut für Kernphysik eingesetzte Rechnersystem von sehr heterogener Struktur. Neben Workstations unterschiedlicher Hersteller und Bauart werden auch im Frontend-Bereich VMEbus-Rechner verschiedenen Typs eingesetzt. Von Bedeutung für die Datenerfassung sind hier einerseits Workstations der Firma Digital von inzwischen veralteten VAX-Rechnern über Workstations mit MIPS-Prozessoren bis hin zu modernen Alpha-Workstations. Andererseits finden VMEbus-Rechner auf der Basis der MC 68000-Familie Einsatz und seit einiger Zeit auch solche mit Sparc-Prozessoren, die die Grundlage der ebenfalls eingesetzten Workstations von Sun bilden.

Auch im Bereich der Betriebssysteme zeigt sich eine große Variationsbreite, die von Betriebssystemen wie VMS oder OS-9 bis hin zu einer Reihe von UNIX-Plattformen reicht. Während VMS und OS-9 ursprünglich noch die Grundlage für die Datenerfassung bildeten und in den ersten Versionen von MECDAS noch unterstützt werden mußten, haben diese Betriebssysteme in Bezug auf die Datenerfassung weitestgehend ihre Bedeutung verloren und wurden im Institut für Kernphysik nach und nach durch UNIX ersetzt. Bis auf solche VMEbus-Rechner und Workstations, die beide mit den Sparc-Prozessor arbeiten, ist jedoch jede Rechnerarchitektur mit einer anderen Varianten von UNIX ausgestattet, die sich in einzelnen Details von den anderen unterscheiden.

Dieser Sachverhalt mußte bei der Realisierung von MECDAS ebenso berücksichtigt werden wie allgemeine Richtlinien für eine bestmögliche Portabilität. Wie gut das schließlich gelungen ist, zeigt die Tatsache, daß die Anpassung der Software auf neue, ursprünglich nicht unterstützte UNIX-Varianten nur mit minimalem Aufwand verbunden war. Dieser beschränkte sich im wesentlichen auf formale Aspekte beim Übersetzen der Software in Bezug auf die Verwendung von bestimmten C-Präprozessor-Makros bzw. Header-Dateien und einer Reihe besonderer Funktionen aus Standard-Bibliotheken. Auch nicht exakte Kompatibilitäten bei Shells und anderen UNIX-Werkzeugen, die für die Generierung lauffähiger Programme aus den Quellen bzw. den späteren Betrieb eingesetzt werden, erforderten einen beschränkten zusätzlichen Aufwand. Es zeigte sich, daß bei entsprechendem Grundwissen die Anpassung für ein bisher noch nicht unterstütztes, jedoch etabliertes System innerhalb einiger weniger Arbeitstage abgewickelt werden konnte. Bemerkenswert ist in diesem Zusammenhang auch

Betriebs-system	UNIX-Kom-patibilität	Rechner-Architektur	Rechner-Typ	Hersteller Rechner/BS
Ultrix	BSD 4.3	MIPS	Workstation	Digital/Digital
		VAX	Workstation	Digital/Digital
OSF/1	BSD 4.4	Alpha	Workstation	Digital/Digital
SunOS 4.x	BSD 4.3	Sparc	Workstation	Sun/Sun
			VMEbus	Force/Sun
SunOS 5.x	System V.4	Sparc	Workstation	Sun/Sun
			VMEbus	Force/Sun
VMEbsd	BSD 4.3	MC 68030	VMEbus	Eltec/KPH
FreeBSD	BSD 4.4	Intel x86	PC	*/PD
Linux	System V.4	Intel x86	PC	*/PD
AIX	System V.4	RS 6000	Workstation	IBM/IBM
HPUX	System V.4	HPxxx	Workstation	HP/HP

Tab. 9.1: Verfügbarkeit von MECDAS

die Tatsache, daß die Inbetriebnahme der Software unter OSF/1 auf Alpha-Prozessoren keinen darüber hinausgehenden Aufwand erforderte, obwohl die 64-Bit-Architektur des Alpha-Prozessors Probleme befürchten lies, wie sie bei einem Großteil der im Institut eingesetzten Public Domain-Software infolge unsauberer Programmierung aufgetreten waren und zu einer stark verzögerten Verfügbarkeit dieser Software führten.

Inzwischen ist MECDAS in seiner neusten Version auf mehr als zehn unterschiedlichen UNIX-Varianten verfügbar. Obwohl nur ein Teil der Rechner bzw. dieser Betriebssysteme für den Frontend-Einsatz geeignet ist, alle dagegen auch als Basis für übergeordnete Systeme eingesetzt werden können, umfaßt jede Realisierung die komplette Software, die von der Experimentkonfiguration über die Datenaufnahme bis hin zur On-line-Analyse und graphischen Darstellung reicht. Tabelle 9.1 zeigt einen Überblick über die derzeit unterstützen Rechnersysteme.

Als Slave-Rechner ohne Betriebssystem kommen naturgemäß nur VMEbus-Rechner in Frage. Bisher befinden sich ausschließlich die Eltec-Rechner Eurocom 5 und Eurocom 6 im Einsatz. Eine frühere Version der Software konnte auch für VMEbus-Karten der Firma Parsitec verfügbar gemacht werden [Venu91]. Diese besitzen als Zentraleinheit einen oder mehrere Transputer vom Typ T 800, der bereits hardwaremäßig Multitaskingmöglichkeiten bietet.

Kapitel 10

Untersuchungen zur Leistungsfähigkeit von Mecdas

Zum Abschluß sollen noch einige Bemerkungen zur Leistungsfähigkeit des Datenerfassungssystems gemacht werden. Wie bereits zu Beginn erwähnt war ein wichtiges Ziel bei der Entwicklung von MECDAS, die Datenerfassung mit genügned hoher Geschwindigkeit durchführen zu können. Das System sollte in der Lage sein, bei regulären Experimenten an MAMI maximale Ereignisraten von einigen zehn bis hin zu einigen hundert Hertz zu verarbeiten. Um Fehler aufgrund hoher Totzeitverluste zu minimieren, sollte gleichzeitig die Totzeit in aller Regel deutlich unter 100% bleiben.

Die Realisierung derartiger Geschwindigkeitsanforderungen sind natürlich einerseits abhängig von Art und Umfang der zu erfassenden Daten und der dazu notwendigen Aktivitäten, andererseits aber ebenso von der eingesetzten Meßelektronik und den verwendeten Rechnersystemen. Erst an dritter Stelle steht die Software, die letztendlich an diese Rahmenbedingungen gebunden ist und diese bestmöglich ausfüllen sollte.

Mit der verfügbaren Hardware konnten die angestrebten Ziele in zufriedenstellendem Maße für die bisher durchgeführten und die in näherer Zukunft vorgesehenen Experimente erreicht werden. MECDAS läßt bei den an der Drei-Spektrometer-Anlage durgeführten Experimenten derzeit im Mittel Ereignisraten bis 250 Hz bei etwa bis zu 25% Totzeit zu. D. h., die pro Ereignis anfallende absolute Totzeit beträgt alles in allem etwa eine Millisekunde. Sie definiert die Zeit, die das Komplettsystem aus Hard- und Software zur Erfassung der zu einem Ereignis gehörenden Daten (die physikalisch relevante Information aus einem Satz von etwa 5000 einzelnen Datenkanälen) benötigt. Das entspricht genau der Zeit, wie sie zu Beginn der Entwicklungen auf der Basis der vorher mit Experimentelektronik und Rechnern gesammelten Erkenntnisse abgeschätzt worden waren. Nähere Untersuchungen haben gezeigt, daß diese Zeit nur noch durch eine maschinennahe Software-Optimierung oder in Zukunft durch Einsatz schnellerer Prozessoren deutlich verbessert werden kann.

10.1 Beiträge zur Totzeit

Die bei der Datenerfassung auftretende Totzeit setzt sich grundsätzlich aus mehreren Einzelkomponenten zusammen (s. Anhang A.3.1). Die Sachlage wird besonders dann kompliziert wenn an der Datenerfassung, wie sie im Rahmen von MECDAS durchgeführt wird, eine größere Zahl unabhängig arbeitender Rechner beteiligt ist, die im allgemeinen Fall ohne weiteres Rückwirkungen auf die Totzeit des Gesamtsystems haben können.

Durch die vorgenommene Aufgabenverteilung und die asynchrone Arbeitsweise der einzelnen Rechner zueinander bestimmen zwar in erster Linie die unter Echtzeitbedingungen arbeitenden Frontend-Rechner die Rechnertotzeit für die einzelnen, unabhängig bearbeiteten Arme des Experiments, doch können auch die übergeordneten Rechner und in besonderen Fällen auch die jeweils parallel arbeitenden Frontend-Rechner das Totzeitverhalten nachhaltig beeinflussen. Dazu können noch Effekte aufgrund der Kommunikation zwischen Master und Slave und insbesondere zwischen den Master-Rechnern und dem Host-Rechner auftreten. Nicht zuletzt kann sich in der Totzeit auch die Geschwindigkeit des Massenspeichermediums bemerkbar machen, auf dem die Daten letztendlich archiviert werden sollen.

Im allgemeinen Fall stellt das Datenerfassungssystem ein gekoppeltes System mit mehreren unterschiedlichen Zeitkonstanten dar, für dessen zeitliches Verhalten sich sehr komplizierte Verhältnisse ergeben, was letztendlich die Ermittlung der Totzeitverluste erschwert. Dabei gehen nicht nur Durchlauf- und Bearbeitungszeiten von Hard- und Software ein. Auch die Wahl des Strahlstroms, Untersetzungsfaktoren und der Anteil von erfaßten Einzelereignissen bei Koinzidenzexperimenten bestimmen das Totzeitverhalten.

Durch die bei MECDAS vorgenommene Entkopplung der einzelnen Arbeitsgänge tragen jedoch nicht alle an der Datenerfassung beteiligten Elemente zwangsläufig zur Totzeit bei. Vielmehr können die einzelnen Aufgaben von den beteiligten Rechnern sehr gut parallel abgewickelt werden. Bestimmend für die Totzeit ist letztendlich die langsamste Komponente in dieser Kette; und das ist in der derzeitigen Konfiguration für jeden Arm des experimentellen Aufbaus der Slave, also der eigentliche Datenaufnahmerechner im Frontend-Bereich. Alle in der Hierarchie über ihm befindlichen Komponenten sind in der Regel deutlich schneller.

Wie aus Abbildung 10.1 ersichtlich ist, die schematisch den Datenfluß von der Aufnahme bis zur Archivierung der Daten auf Massenspeicher zeigt, werden die einzelnen Verarbeitungsschritte bei der Datenerfassung durch explizite oder implizite Puffermechanismen voneinander entkoppelt. Das ermöglicht einerseits die bereits angesprochenen Parallelisierung, bietet aber zudem noch den Vorzug, daß das Gesamtsystem gegenüber Schwankungen der Bearbeitungszeiten der einzelnen Aufgaben unempfindlich wird. Die Pufferung gleicht kurzzeitige Belastungsspitzen aufgrund

- der statistischen zeitlichen Verteilung der physikalischen Ereignisse

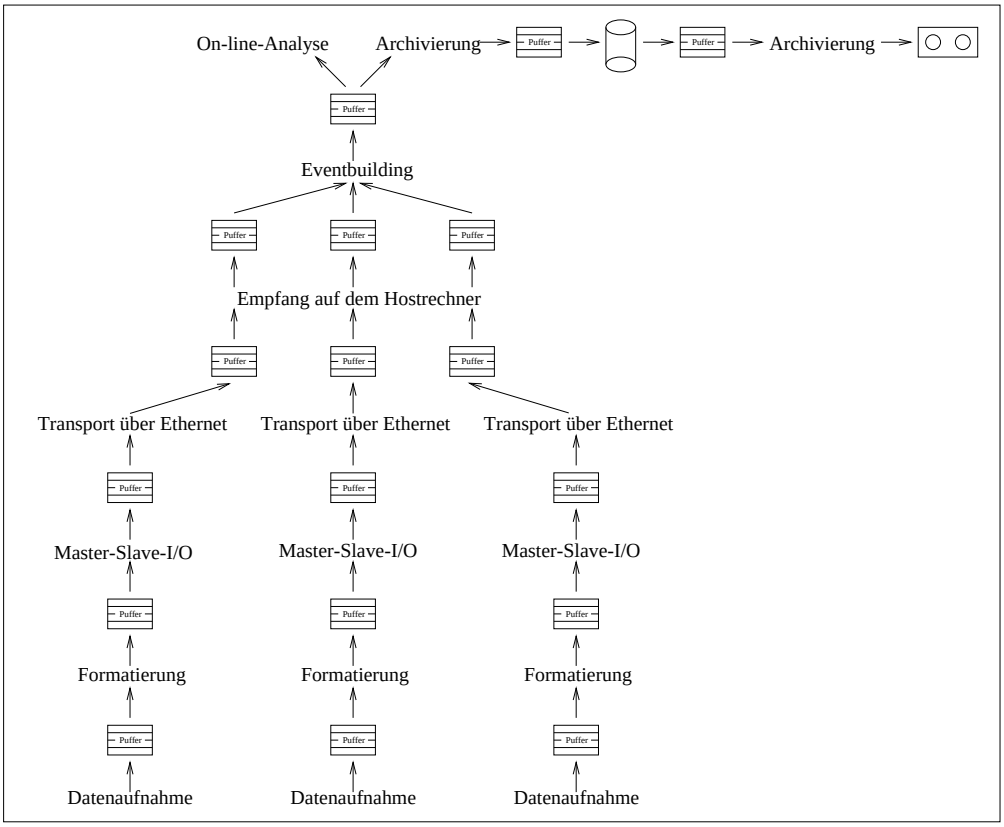

Abb. 10.1: Der Datenfluß für einen Arm einer Koiinzidenzanordnung von der Datenaufnahme bis zur Archivierung. Die einzelnen Arbeitsschritte sind entweder durch explizite in MECDAS implementierte oder implizite vom Betriebssystem vorgegebene Pufferungsmechanismen voneinander entkoppelt.

- von zeitweiser Mehrbelastung durch die parallel zur eigentlichen Datenerfassung ablaufenden Aktivitäten der im Multitasking arbeitenden übergeordneten Rechner
- von Übertragungsengpässen bei der Kommunikation insbesondere bei der Datenübertragung über ein Netzwerk wie Ethernet, an dem eine Vielzahl von Knoten für die unterschiedlichsten Anwendungen angeschlossen sind
- unterschiedlicher Datenraten von den einzelnen Armen beim Eventbuilding
- beim Zugriff auf die Massenspeicher

aus. Wichtig ist lediglich, daß jede Komponente schnell genug ist, um im Mittel mit der zu bearbeitenden Datenmenge fertig zu werden. Bei einer Ereignisrate von bis zu 250 Hz und einer mittleren Anzahl von zu erfassenden Parametern pro Spektrometer von 50, die nach ihrer Formatierung etwa 250 Bytes umfassen, liegt diese Datenrate für die einzelnen Frontends derzeit bei maximal etwa 60–70 KB/s, während Netzwerk, übergeordnete Rechner und Archivierung mit der dreifachen Datenmenge zurecht kommen müssen.

Diese Anforderungen werden in fast allen Bereichen jedoch noch sehr gut abgedeckt. Der Datentransfer über Ethernet kann, wenn auch nicht mit der maximalen Rate von 1,2 MB/s, so doch mit einigen hundert KB/s erfolgen und auch die Bandarchivierung erlaubt mit den zur Zeit verwendeten Geräten eine Geschwindigkeit von 500 KB/s und kann bei Bedarf auch größere Datenraten bewältigen. Auch das Eventbuilding und die im Zusammenhang mit der Datenerfassung auftretenden Verwaltungsaufgaben können aufgrund entsprechender Programmierung, aber insbesondere infolge der hohen Rechenleistung der als übergeordneten Rechner eingesetzten Workstation vorläufig ohne Probleme bewältigt werden. Lediglich im Frontend-Bereich zeigen sich Enpässe, einerseits durch zeitweise Überlastung des Masters aber noch viel mehr durch die grundsätzlich beschränkte, im Vergleich zu den anderen Rechnern relativ gering Rechenleistung des Slaves, der letztendlich die zentrale Rolle bei der Datenerfassung spiel. Aus diesem Grund ist es sinnvoll, die Verhältnisse dort etwas näher zu untersuchen.

10.2 Messungen der Geschwindigkeit

Aufgrund der Tatsache, daß das Datenaufnahmeprogramm in der Regel auf einem Slave-Rechner läuft, der kein Betriebssystem besitzt, existieren im Gegensatz zu komfortableren Betriebssystemen dort keine Profiling-Möglichkeiten, die es erlauben, mit wenig Aufwand den Zeitbedarf einzelner Programmteile sehr genau zu bestimmen. Da auch auf dem Master-Rechner, der eine zum Slave sehr ähnliche Architektur besitzt, diese Information nicht direkt zugänglich ist, dieser aber auch infolge seiner Nichtechtzeitfähigkeit ein ganz anderers Zeitverhalten aufweist, wurde ein einfaches Verfahren entwickelt, um Informationen über den zeitlichen Ablauf bei der Datenerfassung zu erhalten.

Dieses Verfahren basiert auf der Datenerfassung selbst und nutzt die Verfügbarkeit eines hochauflösenden Timers auf dem E5-Slave-Computer aus. Der Timer, ein mit einer Frequenz von 8 MHz getakteter Zähler, stellt die Basis für ein virtuelles Gerät dar, das wie andere CAMAC- oder Fastbus-Module im Rahmen der Datenaufnahme ausgelesen werden kann. Er bietet die Möglichkeit, mit hoher Auflösung, beliebige Teile der Software auf dem Slave zu vermessen. Das Timer-Device liefert schließlich diese Information als Daten, die als herkömmliche Meßdaten weiterverarbeitet werden können, automatisch in den Datenstrom aufgenommen und archiviert werden. Somit ist es sogar möglich, sie mit den Standardwerkzeugen der On-line-Analyse sehr einfach auszuwerten. Abbildung 10.2 zeigt einen Ausschnitt aus dem Datenerfassungsprogramm, Abbildung 10.3 demonstriert, wie eine einfache Auswertung inklusive Histogrammierung der gemessenen Zeitinformation erfolgen kann.

Die Messungen wurden mit als Slaves arbeitenden E5-CPU-Karten vorgenommen, die als Zentraleinheit mit 20 MHz getaktete MC 68020-Prozessoren besitzen, so wie sie auch im Experiment ihren Einsatz finden. Auch ansonsten wurden die Messungen unter den gleichen Bedingungen wie bei der herkömmlichen Experimentdurchführung vorgenommen.

```
Data *
rdtMain(Data *buf, size_t n)
{
        ...
        sysAdd(0, systimer(1));         /* Timer-Wert --> systime[0]       */
        ...
        dread(adc);                     /* Beliebige Aktivitaeten, deren   */
        dread(tdc);                     /* Laufzeit gemessen werden sollen */
        ...
        sysAdd(1, systimer(1));         /* Timer-Wert --> systime[1]       */
        dread(systime);                 /* Auslese des virtuellen systime  */
        ...
}
```

Abb. 10.2: Erfassung der Zeitinformation

```
void *
eventMain(void)
{
        int     i;

        for (i = 0; i < 4; ++i) {
                if (eventOk(simple.times[i]) && simple.times[i] > 0)
                        hinc1(histos.times[i], simple.times[i]);
        }

        return (NULL);
}
```

Abb. 10.3: Auswertung der Zeitinformation

10.3 Ergebnisse

Mit dem beschriebenen Verfahren wurden Ergebnisse erzielt, die sowohl mit der standardmäßig vorgenommenen Totzeitmessung verträglich sind, als auch mithilfe elektronischer Hilfsmittel wie dem Oszilloskop reproduziert werden konnten. Die Abbildungen 10.4 zeigt als Beispiel in Form eines Histogrammes die gemessene Zeit, die in der Ausleseroutine zur Datenaufnahme von etwa 50 Parametern benötigt wird.

Die Messungen einzelner Aktivitäten bei der Datenaufnahme im Frontend-Bereich brachten schließlich die in den Diagrammen 10.5 bis 10.6 dargestellten Ergebnisse. Obwohl das Meßverfahren eine wesentlich präzisere Bestimmung ermöglicht hätte, sind die angegebenen Werte nur auf etwa 10 % genau. Dieser relativ hohe Fehler mußte angenommen werden, da bei den verschiedenen im Einsatz befindlichen VMEbus-Systemen keine exakt übereinstimmenden Werte gemessen werden konnten. Die gemessenen Geschwindigkeitsunterschiede rühren wahrscheinlich daher, daß einerseits die CPU-Karten unterschiedlichen Serien entspringen, bei denen z. T. nicht identische Bauteile zum Einsatz kommen, oder gar verschiedene Layouts und in Details geänderte Schaltungen aufweisen. Andererseits spielt bei Zugriffen auf die periphere Hardware auch diese eine

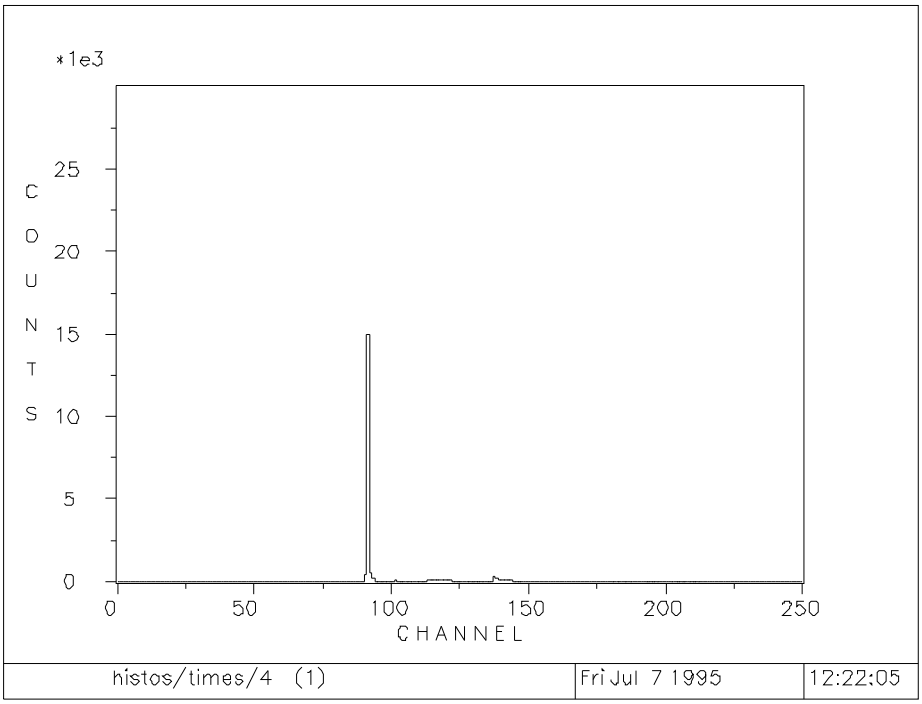

Abb. 10.4: Histogramm, das die gemessenen Zeiten wiedergibt, die in einer Ausleseroutine, die von der Komplexität her vergleichbar mit der für eines der Spektrometer ist, benötigt werden. Es zeigt einen deutlich ausgeprägten Peak und einige Erhöhungen zu größeren Zeiten hin. Der erste Peak gibt die reguläre Bearbeitungszeit an, während die größeren Zeiten durch Verzögerungen aufgrund der Zugriffe des Masters zum Datentransport verursacht werden. Dabei enspricht ein Kanal der Zeitachse 6 μs.

Rolle. Auch hier konnten meßbare Geschwindigkeitsunterschiede aus ähnlichen Gründen festgestellt werden. Im Rahmen des angegebenen Fehlers lassen sich jedoch andererseits unterschiedliche Peripherie wie CAMAC und Fastbus ganz gut vereinheitlichen.

Das Diagramm in Abb. 10.5 stellt die verschiedenen zur Verfügung stehenden Auslesearten (siehe Abschnitt 6.5.3) anhand der Zeit gegenüber, die zur Auslese eines Meßwerts benötigt werden. Wie zu erwarten ist die direkte physikalische Auslese am schnellsten. Die Bearbeitungszeit setzt sich dabei etwa zu gleichen Teilen aus Zugriff auf die externe Hardware und internen Speicherzugriffen und Aktivitäten zusammen. Dabei ist zu beachten, daß bereits hier sowohl Meßwert als auch Adreßinformation eingelesen bzw. berechnet und abgespeichert werden. Als Anhaltspunkt für den von der externen Hardware verursachten Anteil kann dabei die CAMAC-Zugriffszeit dienen, die auf dem VMEbus mit der verwendeten Rechner- und Interface-Hardware etwa 1,5 μs beträgt. Damit stellt für die direkten physikalische Auslese ermittelte Wert i. w. tatsächlich das für die Erledigung dieser Aufgaben erreichbare Minimum dar.

Da hier die CPU-Geschwindigkeit nur noch in geringem Ausmaß mitentscheidend ist, werden schnellere Prozessoren bei der direkten physikalischen Auslese nur wenig Verbesserung bringen können. Im besten Fall ist hier infolge des

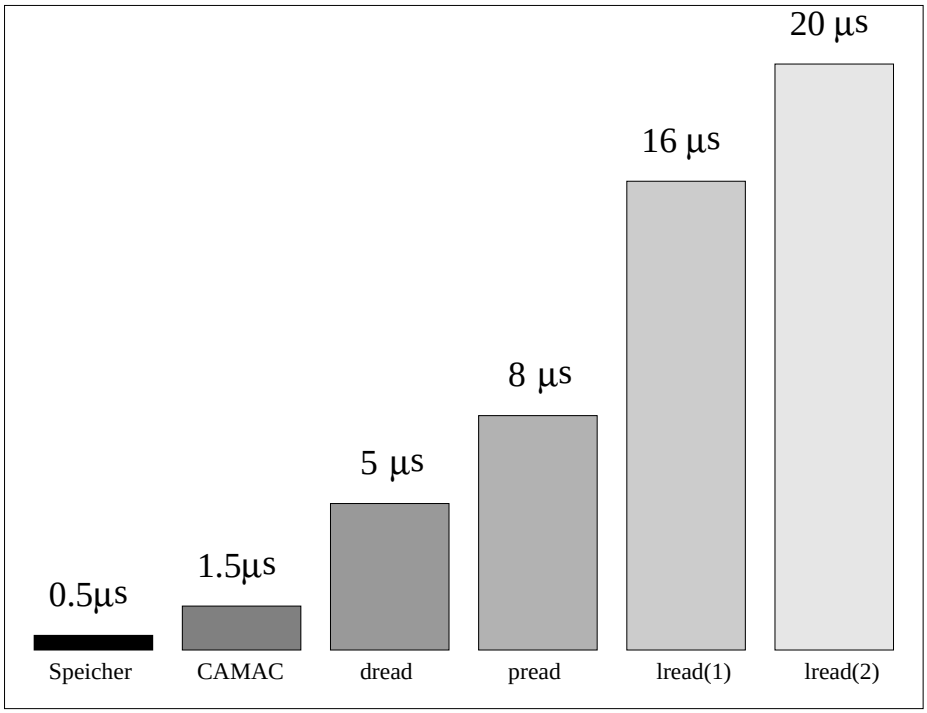

Abb. 10.5: Die gemessenen Zeiten zur Auslese eines physikalischen Meßwerts für die verschiedenen Auslesemethoden bei Einsatz einer E5-CPU-Karte im Slave-Betrieb ergänzt durch Zeiten für einfachen Zugriff auf lokalen Arbeitsspeicher und das CAMAC-System. Dabei entsprechen die mit `dread` und `pread` gekennzeichneten Balken den Zeiten bei direkter oder indirekter physikalischer Auslese. `lread(1)` kennzeichnet die Zeit die für die Auslese eines Parameters benötigt wird, wenn eine größere Datenmenge auf der Basis der logischen Experimentstruktur ausgelesen wird, und `lread(2)`, wenn ausschließlich ein einzelner Parameter auf diese Art bearbeitet wird.

Hardware-Anteils gerade eine Verdoppelung der Geschwindigkeit möglich. Einen wesentlich größeren Einfluß hat die CPU-Geschwindigkeit jedoch bei der Auslese, die sich an der logischen Experimentstruktur orientiert, die in bestimmten Fällen, insbesondere zur selektiven Bearbeitung, nutzbringend eingesetzt werden kann, deren Vorzüge jedoch durch einen beträchtlichen Software-Overhead erkauft werden müssen. Bei den derzeit im Einsatz befindlichen CPUs spielt die logische Auslese daher nur eine untergeordnete Rolle.

Zu der Bearbeitungszeit, die unmittelbar mit der Auslese verbunden ist und im wesentlichen proportional mit der Anzahl der auszulesenden Parameter wächst, kommen noch Zeiten, die von i. w. festen Aktivitäten bei der Datenaufnahme verursacht werden. Dabei lassen sich zwei Anteile unterscheiden. Der erste entspricht allgemeinen Verwaltungsaktivitäten, von denen die wichtigsten in Tabelle 10.1 aufgeführt sind. Sie sind im wesentlichen unabhängig von Art und Umfang der erfaßten Meßdaten und verursachen einen gewissen Basis-Offset, der alles in allem bei etwa 150 μs pro Ereignis liegt. Hier ist in gewissem Rahmen eine weitergehende i. w. maschinenspezifische Optimierung der Software möglich, die bisher aus Portabilitätsgründen noch nicht vorgenommen wurde. Insbesondere ist an dieser Stelle mit dem Einsatz leistungsfähigerer CPUs und Systemsoftware

Aktion	**Zeit/μs**
Interruptverarbeitung	$70 \pm 10\%$
Pufferverwaltung	$40 \pm 10\%$
Fehlerbehandlung	$40 \pm 10\%$

Tab. 10.1: Die pro Ereignis gemessenen Zeiten für verschiedenen Verwaltungsaufgaben, die unabhängig von Art und Umfang der Meßdaten sind.

bereits ohne explizites Optimieren eine deutliche Geschwindigkeitsverbesserung zu erwarten.

Der zweite Anteil wird durch experimentspezifische Vorgänge während der Auslese verursacht, die ebenso wie die Verwaltungsaktivitäten einen zwar von Experiment zu Experiment unterschiedlichen aber ansonsten konstanten Beitrag zur Bearbeitungszeit liefern. Zu diesen Vorgängen gehören einerseits Aktivitäten, die mit dem Aufsetzen von Blocktransfers oder ähnlichem verbunden sind, und andererseits solche, die zum Löschen und Zurücksetzen der Hardware notwendig sind. Das Diagramm in Abbildung 10.6 zeigt die verschiedenen Anteile innerhalb einer typischen, im Rahmen der Drei-Spektrometer-Anlage verwendeten Ausleseroutine. Die dunkel unterlegten Flächen geben dabei die Zeiten für die angesprochenen Aktivitäten wieder, während die hellen Flächen dem eigentlichen Ausleseaufwand entsprechen.

Anhand des Diagramms wird deutlich, daß in der Konfiguration, die diesem Beispiel entspricht und i. w. mit der eingesetzten identisch ist, etwa die Hälfte der Zeit innerhalb der Ausleseroutine Grund-Overhead ist, der unabhängig vom Umfang der zu erfassenden Meßdaten ist. Dieser wird nur zu einem geringen Teil von Software verursacht; es gehen vielmehr sehr stark Hardware-Eigenschaften ein. Ein besonders extremes Beispiel ist der Fastbus. Hier ist der allgemeine Aufwand, der notwendig ist, um den Fastbus korrekt anzusteuern, und ausschließlich durch die Hardware vorgegeben ist, absolut gesehen mit etwa 140 μs der bei weitem dominierende Anteil am Gesamt-Overhead. Diese Tatsache ist umso bemerkenswerter, da dieser Aufwand auch relativ, also in Bezug auf die Anzahl der in diesem Fall ausgelesenen Meßwerte — in der Regel vier von etwa 70 ADCs —, übermäßig hoch ist. Dieser Effekt zeigt deutlich, daß der Einsatz von Fastbus, der auf die Bearbeitung sehr großer Kanalzahlen ausgelegt ist, bei einer solch geringen Kanalzahl, wie sie im Rahmen der A1-Experimente benötigt wird, zumindest in Hinblick auf die Ausleseeffizienz noch nicht sinnvoll erscheint. Noch extremer werden sogar die Verhältnisse, wenn die Konversionszeit der verwendeten Fastbus-Module mit berücksichtigt wird. Diese beträgt noch einmal, unabhängig von der Kanalzahl, etwa 300 μs. Die sich damit ergebende Zeit von beinahe 500 μs, die aufgrund der Verwendung von Fastbus anfällt, ist weitestgehend unabhängig von den zur Datenaufnahme eingesetzten Prozessoren, so daß hier weder durch Optimierung der Software noch durch Einsatz schnellerer Rechner-Hardware eine durchgreifende Verbesserung zu erwarten ist, solange der Fastbus nicht durch äquivalente Hardware ersetzt wird.

Die verbleibende Grund-Overhead verteilt sich ansonsten, unter Berücksichtigung der jeweiligen Kanalzahlen, relativ gleichmäßig auf die restliche Hard-

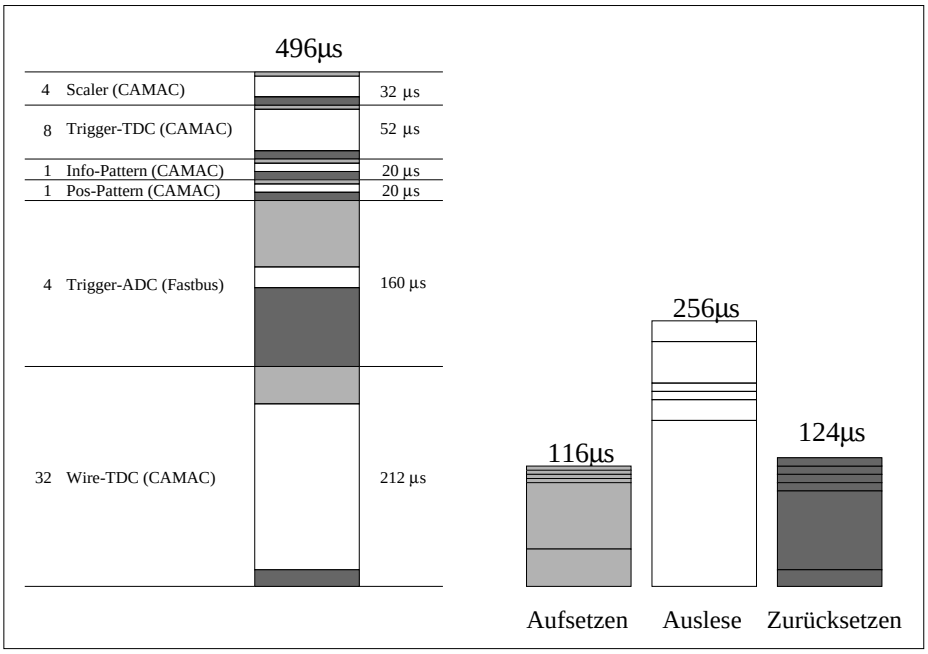

Abb. 10.6: Die einzelnen Aktivitäten innerhalb der Ausleseroutine und ihre zeitliche Verteilung.

ware. Ähnlich wie bei Fastbus, aber doch in geringerem Ausmaß als dort, wird er auch hier zu einem großen Teil von der Hardware vorgegeben.

Faßt man die einzelnen Anteile zusammen, so ergibt sich für die Gesamtbearbeitungszeit eines typischen Ereignisses etwa 650 μs. Diese wird noch durch die Hardware-Konversionszeiten von ADCs und TDCs vergrößert, die hier von der Konversionszeit der Fastbus-ADCs dominiert wird und im ungünstigsten Fall mit 300 μs beiträgt. Damit summiert sich die von Hard- und Software verursachte Totzeit auf etwa 950 μs.

Kapitel 11

Erfahrungen und Verbesserungsmöglichkeiten

11.1 Das Rechnersystem

Eine der gravierensten Schwächen des Gesamtsystems zur Datenerfassung liegt in verschiedenen Komponenten des verwendeten Rechnersystems. Hier ist eine Erneuerung von Rechner-Hardware und Betriebssystem-Software aus Gründen der Betriebssicherheit, zur Verbesserung der Leistungsfähigkeit und zur besseren Nutzung zeitgemäßer Software-Techniken dringend notwendig.

11.1.1 Verbesserung der Betriebssicherheit

Die Betriebssicherheit des Gesamtsystems zur Datenerfassung stellt eine sehr wichtige Grundvoraussetzung zu seiner Nutzbarkeit dar. Daher wurde auch sehr viel Arbeit darauf verwendet, das möglichst gut zu erreichen. Im Laufe der Zeit zeigten sich jedoch in dieser Beziehung Probleme, die sich einer einfachen Lösung auf Anwenderebene entzogen.

Im Frontend-Bereich, insbesondere bei den Master-Rechnern, zeigt sowohl die Rechner-Hardware als auch die Systemsoftware von Zeit zu Zeit ein gravierendes Fehlverhalten, das sich in sporadisch auftretenden Fehlfunktionen einzelner Programme bis hin zu vollständigen Abstürzen von Rechnern äußert. Die Ursache dafür ist wohl in erster Linie in Design-Fehlern von Schaltung und Platinen-Layout der VMEbus-Rechner zu suchen, die zu dem Zeitpunkt, an dem sie angeschafft wurden, extreme Neuentwicklungen waren und, wie erst im Nachhinein festgestellt wurde, die Serienreife noch nicht erlangt hatten. Neuere CPU-Karten dieses Typs, die ebenfalls im Institut für Kernphysik eingesetzt werden, zeigen hier ein deutlich stabileres Verhalten, was das Betriebssystem als die eigentliche Ursache der Probleme ausschließt. Nicht zu vernachlässigen ist auch die Tatsache, daß die Rechner sich inzwischen seit über fünf Jahren im Einsatz befinden, der einerseits aus einem intensiven Testbetrieb bestand mit ständigen elektrischen, thermischen und mechanischen Belastungen, andererseits in der

endgültigen Verwendung nun mit intensiver radioaktiver Strahlung verbunden ist. Alterungseffekte sind daher ebenso möglich und wurden über die zunehmende Registrierung der beschriebenen Probleme auch beobachtet.

Aber auch bei dem auf den VMEbus-Rechnern benutzten Betriebssystem selbst sind Fehler innerhalb der Systemsoftware nicht auszuschließen. Insbesondere das Zusammenarbeiten des Betriebssystems, das zu einem großen Teil aus etwa zehn Jahre alter Software besteht, mit den Systemen der übergeordneten, wesentlich neueren Rechnern, zeigt in bestimmten Fällen Probleme. Im Zusammenhang mit notwendigen System-Updates auf diesen Rechnern treten in wachsendem Maße Inkompatibilitäten zutage. Das macht deutlich, daß es sich bei VMEbsd um ein zwar noch gut funktionierendes, jedoch inzwischen veraltetes Betriebssystem handelt, das unbedingt auf einen neueren Stand gebracht werden müßte. Da es kein kommerzielles System ist, besteht für eine wie bei den anderen Systemen übliche Software-Wartung, die der Beseitigung bekannter Fehler und der Bereitsstellung von Neuerungen dient, keinerlei Möglichkeit; ebenso wie die Portierung müßte auch das ausschließlich in Eigeninitiative geleistet werden.

Als Lösung der beschriebenen Probleme bietet sich der Einsatz neuer Rechner im Frontend-Bereich an, die einerseits technisch auf dem aktuellen Stand sind und andererseits zeitgemäße Betriebssysteme besitzen. Während noch zu Beginn der Entwicklungen des Datenerfassungssystems das Angebot an geeigneten VMEbus-Rechnern sehr begrenzt war, steht heute eine große Auswahl an CPU-Karten für unterschiedlichste Anforderungen zur Verfügung. Im Gegensatz zu früher besitzt diese Hardware auch eine weitreichende kommerzielle Unterstützung durch leistungsfähige Betriebssysteme. Insbesondere Echtzeitsysteme gewinnen hier immer stärker an Bedeutung. Es ist daher angebracht, in nächster Zukunft zumindest die Master-Rechner und deren Betriebssystem durch neue Hard- und Software zu ersetzen und so die Betriebssicherheit zu verbessern. Sie kann weiter erhöht werden, wenn auch die Slaves erneuert werden bzw. diese infolge der gestiegenen Leistungsfähigkeit der neuen Rechner und deren Betriebssysteme evtl. ganz entfallen können.

Dabei ist zu hoffen, daß sich mit der Neuanschaffung tatsächlich auch die Situation verbessert, denn das Beispiel der eingesetzen Ultrix-Workstations von Digital zeigt, daß auch kommerzielle Systeme bei intensiver Beanspruchung deutliche Probleme zeigen können. Auch hier kommt es zu sporadischen Abstürzen bzw. Verklemmungen infolge starken Netzwerk- und Band-I/Os. Im Gegensatz zu den Frontend-Rechnern handelt es sich jedoch hier um wesentlich modernere Rechner und Betriebssystem-Versionen. Jedoch ist auch in diesem Falll beides inzwischen nicht mehr Stand der Technik. Das trifft insbesondere für das Betriebssystem Ultrix zu, das von Digital nur noch schlecht gewartet wird und lediglich eines von vielen Betriebssystemen dieses Herstellers darstellt. Da der Vertrieb der damit arbeitenden Rechner inzwischen eingestellt wurde und diese keine kommerzielle Bedeutung mehr haben, ist in absehbarer Zukunft mit seinem Ende zu rechnen. Auch hier bietet es sich an, neue Rechner, wie sie bereits im Institut eingesetzt werden, zu verwenden.

11.1.2 Erhöhung der Geschwindigkeit

Nicht nur die Erhöhung der Betriebssicherheit des bei der Durchführung von Experimenten eingesetzten Rechnersystems spricht für eine Modernisierung der Rechner sondern auch die damit verbundene Leistungssteigerung des Systems in den verschiedenen Einsaztbereichen. Auch hier zeigen die Erfahrungen bzw. die Analyse der im vorangegangenen Kapitel wiedergegebenen Meßergebnisse, daß sowohl die eingesetzten Workstations als auch die Frontend-Rechner zeitweise am Rande ihrer Leistungsfähigkeit arbeiten. Zwar sind sie bisher noch in der Lage, die ursprünglich einmal gestellten Anforderungen zu erfüllen, doch zeigen sich bei höheren Datenraten deutliche Engpässe, und künftigen, bereits abzusehenden Anforderungen sind sie nicht mehr gewachsen.

An erster Stelle sind hier die Slave-Rechner zu nennen, die die eigentliche Datenaufnahme ausführen, jedoch von ihrer nominellen Rechenleistung die langsamsten Teile des Rechnersystems darstellen. Ihre Leistungsfähigkeit war bisher bestimmend für die Rechnertotzeit. Der Einsatz inzwischen verfügbarer, um ein Vielfaches schnellerer Rechner würde hier, falls es die restliche Elektronik erlaubt, eine deutliche Reduzierung der Totzeit bzw. Erhöhung der maximal zu verarbeitenden Ereignisraten zulassen, was auch bei extremer mit viel Aufwand und Verschlechterung von Wart- und Portierbarkeit verbundener Software-Optimierung bei weitem nicht zu erreichen wäre.

In Grenzfällen zeigt sich aber auch, daß die Leistungsfähigkeit der Master-Rechner oder gar die der Workstations zeitweise nicht ausreicht. Hier wird die Überlastung durch die vielfältigen, parallelen Aktivitäten der Rechner verursacht, die zu einer immer wiederkehrenden, punktuellen Überbeanspruchung von Systemressourcen wie virtuellem Speicher, Platten- und Netzwerk-I/O führt. Grund dafür ist insbesondere auf den VMEbus-Rechnern der nur sehr beschränkte reale Arbeitsspeicher von effektiv etwa 2 MB, dem einige zehn MB an Speicherbedarf durch die für Datenerfassung und Steuerung der Experimentierapparatur benötigten Programme gegenüberstehen. Das führt dann in bestimmten Situationen zu ständigen Page- und Swap-Vorgängen und damit zu die CPU belastenden I/O-Aktivitäten, die bei genügend großem Speicher vermieden werden könnten.

Ähnliches gilt auch für die Workstations, die zwar mit z.B. 40 MB deutlich mehr Arbeitsspeicher besitzen, wo aber auch die Anwendungen wesentlich speicherintensiver sind. Dazu gehören einerseits neben Eventbuilding und Datenarchivierung besonders die mit der On-line-Analyse verbundenen Aufgaben, die umfangreiche und zudem rechenintensive Programme mit hohem Bedarf an Arbeitsspeicher für Meßdaten und Ergebnisse erfordern. Andererseits machen sich im Bereich der Experimentsteuerung auf den Workstations dieselben Probleme wie auf den Frontend-Rechnern bemerkbar, verursacht durch Steuerungsprozesse, die zwar in der Regel jeweils selten aktiviert werden, jedoch wegen ihrer Vielzahl zu einer ständigen Belegung von Systemressourcen führen. Dazu kommt noch der hohe Speicherverbrauch einer graphischen Bedienoberfläche, wie sie insbesondere zur Darstellung der Meßergebnisse oder im Rahmen des Steuerungssystems eingesetzt wird.

Die genannten Probleme konnten zu einem Teil durch die Verteilung der Aufgaben auf mehrere Rechner gemindert werden, doch ist diese Lösung für einige Aufgabenstellungen nicht geeignet, so daß auch in diesem Fall der Einsatz leistungsfähigerer Rechner sinnvoll ist, insbesondere, wenn dabei eine Leistungssteigerung um bis zu einer Größenordnung zu erwarten ist.

11.1.3 Ausnutzung verbesserter Betriebssystem-Funktionalität

Neben den offensichtlichen Mängeln von Rechner-Hardware und Systemsoftware zeigt das Rechnersystem aber auch eine Reihe von Nachteilen aufgrund der eingeschränkten Möglichkeiten veralteter Betriebssysteme. Zu einem geringen Teil konnte das zwar durch den Einsatz zeitgemäßer Public Domain-Software wie C++-Compiler, Editor, Debugger, Bibliotheken für verschiedenste Zwecke oder Hilfsmittel zur Erstellung graphischer Benutzeroberflächen kompensiert werden, der weitaus größere Teil aber könnte nur durch Eingriffe in den Betriebssystemkern bzw. dessen Neuorganisation erreicht werden. Neuere Betriebssysteme bieten hier Eigenschaften wie

- besseres Echtzeitverhalten sowohl im Frontend-Bereich als auch bei den Host-Rechnern,
- Unterstützung des Anwenders zur Nutzung der Echtzeitfähigkeiten,
- die Möglichkeit der parallelen Verarbeitung innerhalb von Programmen unter Benutzung von sog. Threads,
- geringere Einschränkung bei der Nutzung von System-Ressourcen,
- bessere Möglichkeiten bei der Nutzung von Shared Memory,
- Memory Mapped Files,
- modulare Systemkonzepte,
- dynamische Konfigurierbarkeit und bessere Anpaßbarkeit,

die den bisher verwendeten durchweg fehlen und bei der Beseitigung einiger bisher nur ungenügend gelöster Probleme in der Datenerfassungs-Software sehr hilfreich sein würden. Hier ist in erster Linie die Verwendung eines Echtzeitbetriebssystems bzw. eines Realtime-Kernels zu nennen, mit dessen Hilfe eine wesentlich effizientere Umsetzung der lokalen Serverfunktionalität zur Datenaufnahme erreicht werden könnte.

Wie weit die Verbesserungen jedoch tatsächlich gehen können, bleibt zukünftigen Untersuchungen überlassen. Bisher bestand auch aus zeitlichen Gründen keine Möglichkeit, zur Zeit auf dem Markt befindliche Echtzeitsysteme wie LynxOS [Lynx92] oder VxWorks [Wind93] oder herkömmliche Systeme mit begrenzten Echtzeitfähigkeiten wie SunOS/Solaris [Sun93] oder OSF/1 [DEC94] auf ihr Echtzeitverhalten und ihre Eignung für die Datenaufnahme zu untersuchen.

11.2 Datenerfassung

In einigen Teilen konnte das Programmsystem von MECDAS nur prototypartig realisiert werden. Damit konnte zwar ein lauffähiges System geschaffen werden, es bieten sich jedoch an der einen oder anderen Stelle Verbesserungsmöglichkeiten in Bezug auf Funktionalität und Leistung an.

11.2.1 Ausbau der Gerätebibliothek

Ein typisches Beispiel dafür ist die Gerätebibliothek, die ursprünglich Beschreibungen bzw. Software nur für die dringendst für die Datenerfassung benötigten Geräte enthielt. Inzwischen wurde sie zwar erweitert, doch wurde im Rahmen neuer Experimente eine Vielzahl weiterer Gerätetreiber-Module erstellt, die nach einer intensiven Überarbeitung in Hinblick auf Effizienz aber auch die saubere Einhaltung der Schnittstellen in die Bibliothek aufgenommen und damit allgemein zur Verfügung gestellt werden sollten.

Desweiteren wurde in der Gerätebeschreibung bisher noch nicht umgesetzte Mechanismen vorgesehen, die für eine effiziente und korrekte Ansteuerung der Meßelektronik sehr hilfreich sein können. Dazu gehört z. B. die Möglichkeit einer regelmäßigen Funktionsüberprüfung der Hardware inklusiver der Rechner, eine weitergehende Fehlerbehandlung oder die Kommunikation zwischen einzelnen Treiberinstanzen, was sich für verschiedene Zwecke als sinnvoll erwiesen hat. Hier ist sowohl in den Gerätetreibern selbst als auch in der sie benutzenden Software entsprechende Implementationsarbeit zu leisten. Als besonders wichtig erwies sich die Notwendigkeit, einzelne Aktivitäten aus dem Gerätetreiber der letzendlich Bestandteil des Datenaufnahmeprogrammes wird, auszulagern und in eigenständige Programme zu packen, die bei Zustandsübergängen der Datenaufnahme aktiviert werden.

11.2.2 Fehlerbehandlung während der Datenaufnahme

Die Behandlung von Laufzeitfehlern unterschiedlichster Art ist bisher nur auf das nötigste beschränkt (s. Abschnitt 5.5) und sollte in verschiedenen Bereichen verbessert werden. Von besonderer Bedeutung sind dabei

- die Behandlung von Fehlern während der Datenaufnahme unter Echtzeitbedingungen
- die Wiedersynchronisierung eines verteilten Systems bei Auftreten von Fehlern innerhalb von Subsystemen

In beiden Fällen wurden zwar die grundlegenden Arbeiten schon geleistet, doch sind aus verschiedenen Gründen Verbesserungen angebracht.

11.2.2.1 Reduzierung des notwendigen Verwaltungsaufwands

Für die unter Echtzeitbedingungen ablaufenden Datenaufnahme auf dem Slave-Prozessor ist die Ausnahmebehandlung, d. h., die Behandlung von Bus- und Adreßfehlern oder anderen vom Prozessor erkennbaren Hard- und Software-Fehlern, sehr einfach gehalten. Dabei wurde versucht, auf möglichst portable Art, diesen extrem maschinenabhängigen Problembereich zu bearbeiten. Das dabei notwendige Retten des Systemzustands, um im Fehlerfall nichtlokale Sprünge zur Wiederherstellung eines definierten Zustands vornehmen zu können, verursacht bei den verwendeten Prozessoren für jedes Ereignis einen Aufwand von einigen zehn Mikrosekunden, der reduziert werden könnte, wenn die Ausnahmebehandlung besser auf den Prozessor abgestimmt wäre. Das macht eine maschinenspezifische Analyse der vom Prozessor zur Verfügung gestellten Diagnoseinformation erforderlich, hat jedoch den Vorteil, daß der allgemeine Verwaltungsaufwand, der sich letzendlich in der Totzeit niederschlägt, stark reduziert werden kann, während das Vefahren gleichzeitig wesentlich vollständigere Information über die Fehlersituation zur Verfügung stellen kann.

11.2.2.2 Dynamische Fehlerbehandlung

Bezüglich der Fehlerbehandlung ist die bisherige Datenaufnahme-Software so angelegt, daß sie nicht-fatale Fehler so weit wie möglich abfängt und in der Lage ist, die Messung bei sporadisch auftretenden Fehlern am Laufen zu halten. Kritisch wird es jedoch bei dauerhaftem Ausfall eines Geräts, was die Druchführung einer Messung nicht mehr zuläßt. Hier bestünde aber die von der Struktur der Datenaufnahme-Software prinzipiell gegebene Möglichkeit, fehlerhafte Hardware für die Datenaufnahme auszublenden und so einen Notbetrieb zu gewährleisten, bis die Defekte behoben werden können.

11.2.2.3 Konsistenz im verteilten System

Eine wichtige Anforderung an das für Mehrarmexperimente eingesetzte verteilte Datenerfassungssystem ist die Notwendigkeit, den Zustand des Gesamtsystems und damit auch alle beteiligten Komponenten stets konsistent zu halten. Bei Zustandsänderungen darf diese Konsistenz unter keinen Umständen verloren gehen. Das Konzept des hierarchischen Client-Server-Systems bildet dafür zwar eine gute Grundlage, praktische Erfahrungen im intensiven Einsatz haben jedoch gezeigt, daß bestimmte Fehlersituationen noch nicht ausreichend gut behandelt werden. Dabei ist weniger die Lösung grundlegender Probleme erforderlich, als vielmehr die Implementation und Ergänzung von Programmcode, um einerseits bei Fehlern jeglicher Art die vollständige und korrekte Übermittlung von Statusinformation zu gewährleisten und andererseits eine konsequente und individuelle Behandlung jedes Fehlers vorzunehmen. Dabei kann gewinnbringend von den bisher gemachten Erfahrungen Gebrauch gemacht werden.

11.2.3 Verbesserung des Eventbuilding

Eine zentrale Rolle bei Mehrarmexperimenten spielt das Eventbuilding. Auch hier hat die Erfahrung gezeigt, daß die bisher verwendete Methode verbesserungsbedürftig ist. Zwar arbeitet das Verfahren im Normalfall korrekt, doch zeigen sich Probleme in Ausnahmesituationen, die in erster Linie von Timing-Problemen der Koinzidenzelektronik bzw. bei der Ansteuerung der Eventbuilder-Hardware und dadurch hervorgerufener Inkonsistenzen in den Datenströmen der Teilsysteme verursacht werden

11.2.3.1 Hardware-Fehler

Da die Elektronik nicht in der Lage ist, die zugrundeliegenden Fehlfunktionen zu erkennen, haben auch die beteiligten Rechner nicht die Möglichkeit, diese unmittelbar zu registrieren. Es sind vielmehr indirekte Methoden notwendig, die sich an den durch die Fehlfunktionen verursachten Effekten orientieren — in diesem Fall Asynchronitäten zwischen den Meßdaten der einzelnen Arme des Experiments.

Die Eventbuilder-Software führt dazu Konsistenzüberprüfungen der Meßdaten durch, die jedoch nur sehr einfach gehalten sind, ausgehend von einer korrekt arbeitenden Hardware, die nur in Ausnahmefällen versagt. Die Software ist in der Lage, Inkonsistenzen in den Datenströmen zu erkennen und darauf auf einfache Art zu reagieren, indem sie einzelne fehlerhafte Ereignisse verwirft und im Anschluß daran bestrebt ist, die Synchronität der Datenströme bei den folgenden Ereignisse wieder herzustellen. Treten Fehler auf, die nicht erkannt werden oder die mit diesen Methoden nicht mehr zu beheben sind, kann die Software im besten Fall noch mit Fehlermeldungen reagieren, aber ebenso auch zu einer permanenten Funktionsstörung führen und den Eingriff des Experimentators in den Meßablauf notwendig machen.

Der praktische Einsatz zeigte, daß solche fatalen Fehler ebenso wie die einfachen wesentlich häufiger vorkommen als ursprünglich vorausgesetzt. Um sie in den Griff zu bekommen, müssen zwei Lösungsansätze verfolgt werden:

- Untersuchung und Beseitigung der Hardware-Probleme
- Verbesserung der Fehlererkennung und -bewältigung der Eventbuilder-Software

Zum ersten Punkt wurden bereits Maßnahmen mit ersten Erfolgen eingeleitet. Der zweite Punkt läßt sich durch eine Erweiterung der Konsistenzüberprüfung lösen, indem anstelle ihrer derzeitigen Beschränkung auf ein einzelnes Ereignis auch die Umgebung des interessierenden Ereignisses, also Ereignisse davor und danach, berücksichtigt werden muß.

11.2.3.2 Überlauf des Ereigniszählers

Ergänzt werden sollten die bisher beschriebenen Maßnahmen durch eine Beseitigung der prinzipbedingten Schwächen des bisherigen Verfahrens wie insbesondere der sehr beschränkten Breite des Ereigniszählers, bei dessen Überlauf die eindeutige Ereigniszuordnung verloren gehen kann. Dieses Problem läßt sich durch einfache Modifikationen von Hard- und Software leicht vermeiden, indem bei jedem Überlauf des Zählers ein künstliches Ereignis generiert wird, das wie jedes andere Koinzidenzereignis von allen beteiligten Frontend-Rechner zu erfassen ist. Damit enthält jeder Datenstrom unabhängig von den Ereignisraten der einzelnen Arme und dem Auftreten von Synchronisationsfehlern regelmäßig Synchronisationspunkte, an denen sich die Eventbuilder-Software orientieren und jede Asynchronität wieder beseitigen kann.

11.2.4 Alternative Auslesemöglichkeiten

Das Konzept von MECDAS orientiert sich durchgängig an dem Baukastenprinzip, das es erlaubt, viele Komponenten des Systems durch äquivalente Gegenstücke zu ersetzen. Während davon z. B. im Analyse-Bereich bereits Gebrauch gemacht wird, bietet es sich auch an, bei der Datenaufnahme Alternativen zu realisieren, die i. w. die zentrale Datenerfassungs-Software für allgemeine Aufgaben nutzt, jedoch für den experimentspezifischen Teil der Auslese anstelle der standardmäßigen, nicht ganz so effizienten Lösung auf der Basis der Experimentbeschreibung eine auf bestimmte Problemstellungen optimierte spezielle Lösung verwendet. So kann es sinnvoll sein, für extreme Anforderungen und unter Beschränkung der Allgemeinheit eine stärker an die Hardware angelehnte Auslese und Verarbeitung der Daten vorzunehmen.

Wie bereits in Kapitel 6 erläutert stellt das Standardverfahren einen Kompromiß dar, um die Inhomogenität der Meßelektronik bei den MAMI-Experimenten zu bewältigen. Werden in einem Experiment jedoch ausschließlich gleichartige Geräte eingesetzt, ist damit evtl. unnötiger Overhead verbunden, der bei einer auf die verwendeten Hardware optimierten Auslese, Formatierung und Analyse der Daten wegfallen könnte. Um auch hierbei den Experimentator bestmöglich zu unterstützen, sind Werkzeuge notwendig, wie sie bereits für den allgemeinen Fall existieren. Neben einer Neuimplementierung wäre hier eine Modifikation bzw. Erweiterung der existierenden Werkzeuge sinnvoll.

11.2.5 Verbesserung der Datenarchivierung

Aufgrund des sehr flexiblen Datenarchivierungskonzepts, das auf einem indirekten Verfahren basiert, stellte die Archivierung bisher ein recht unkritisches Element bei der Datenerfassung dar. Dennoch ist ihre korrekte Funktion von zentraler Bedeutung und es ist sinnvoll, auch in diesem Bereich eine bestmögliche Lösung anzustreben. Die bisherige Verwendung von Exabyte-Laufwerken, also Technologie aus dem Video-Bereich, stellt zwar eine sehr kostengünstige

und allgemein etablierte Methode dar, doch besitzt dieses Verfahren eine Reihe von Nachteilen, die nicht nur durch die grundsätzlich höhere Fehleranfälligkeit von Tape-I/O begründet werden, sondern auch in der schlechteren Handhabbarkeit insbesondere beim späteren Zugriff auf die abgespeicherten Daten im Rahmen der Off-line-Analyse. Hier bietet sich z. B. die Verwendung von optischen Speichermedien, insbesondere von CDs, an, für die es inzwischen einerseits preisgünstige Schreibgeräte gibt, und andererseits aber auch Lesegeräte bereits zur Standardausstattung moderner Workstations und PCs gehören.

11.2.6 Das Gesamtsystem für die Drei-Spektrometer-Anlage

11.2.6.1 Bessere Koordinierung zwischen Datenerfassung und Steuerungssystem

Das Gesamtsystem zur Durchführung der Experimente an der Drei-Spektrometer-Anlage besteht aus den drei im Rahmen von verschiedenen Dotorarbeiten erstellten Komponenten

- Datenerfassung
- Steuerung der Experimentierapparatur
- Überwachung des Experiments

Ihre Zusammenarbeit war bisher aufs nötigste beschränkt, um in der Aufbau- und Entwicklungsphase unnötige Abhängigkeiten und Seiteneffekte zu vermeiden und eine möglichst unabhängige, parallele Entwicklung zu erlauben, die sich erst einmal auf die Lösung der Kernprobleme konzentrierte. Nach erfolgreicher Inbetriebnahme der Einzelsysteme ist jedoch insbesondere in Hinblick auf eine Entlastung des Experimentators bei der Durchführung zukünftiger Experimente eine weitergehende Koordinierung ihrer Aktivitäten angebracht. Ebenso wie bereits bei der Datenerfassung sollte dafür eine übergeordnete Zustandsmaschine sorgen, die das Gesamtsystem steuert und stets in einem konsistenten Zustand halten muß, und bei Bedarf die einzelnen Komponenten wieder synchronisiert.

11.2.6.2 Verbesserte Behandlung von Ausnahmesituationen

Noch viel stärker als bei der Datenerfassung allein muß innerhalb des Gesamtsystems auf eine sehr gute Behandlung von Ausnahmesituationen geachtet werden, da das System infolge seiner Komplexität sonst durch einen Nicht-Experten nicht mehr benutzbar würde.

11.2.6.3 Einheitliche Datenbank

Grundlage bei der Koordinierung zwischen Datenerfassung und Steuerungssystem sollte eine gemeinsame Datenbank sein, die u. a. für folgende Aufgaben benötigt wird:

- einheitliche Verwaltung der für Datenerfassung und Experimentsteuerung benutzen Geräte
- Unterbindung von Konflikten und Inkonsistenzen zwischen den Teilsystemen
- redundanzfreie Verwaltung von Konfigurationsinformation
- Bereitsstellung dieser Information für die einzelnen Teilsysteme
- Speicherung des Systemzustands einerseits für Dokumentationszwecke andererseits für eine eventuelle spätere Wiederherstellung
- langfristige Verwaltung der Meßdaten und der verwendeten Speichermedien

Eine professionelle Datenbank könnte für diese Zwecke neben der ausschließlichen Speicherung dieser Informationen Hilfsmittel zur Verfügung stellen etwa zur automatisierten Generierung von Eingabemenüs oder Zustandsanzeigen, Verfahren für Konsistenzüberprüfungen oder Mechansimen zur Suche von Information nach komplexen Kriterien. Auch wenn hier ein kommerzielles Produkt eingesetzt werden sollte, was im Gegensatz zu vielen anderen Aufgabenstellungen bei der Datenerfassung in diesem Fall sehr gut möglich ist, ist neben dem Entwurf angemessener Datenstrukturen und deren Beziehungen zueinander ein erheblicher Aufwand an zusätzlicher Implementationsarbeit zu leisten, um eine solche Datenbank auf geeignete Weise an das existierende Software-System anzukoppeln.

11.2.7 Graphische Bedienoberfläche

Um die Bedienung des Datenerfassungssystems zu erleichtern, sollte in Zukunft verstärkt die Möglichkeiten einer graphischen Bedienoberfläche ausgenutzt werden. Bisher beschränkt sich das System auf eine relativ einfache Nutzung des X-Window-Systems. Weitergehende Versuche im Rahmen einer Prototypimplementation zeigten jedoch deutlich die Vorteile einer an die spezielle Anwendung angepaßten graphischen Oberfläche, die sich in einer leichteren Benutzbarkeit und der deutlichen Reduzierung von Bedienungsfehlern äußert. Es ist daher sinnvoll, den bereits eingeschlagenen Weg konsequent weiter zu verfolgen.

11.3 Experimentbeschreibung

Insbesondere bei der Durchführung von neuen Experimenten hat sich die einfache Konfigurierbarkeit von MECDAS bereits mehrfach von Vorteil erwiesen. Dabei zeigten sich jedoch gleichzeitig auch einige Schwächen der derzeitigen Implementation, deren Beseitigung das System noch besser nutzbar machen würde.

11.3.1 Erhöhung der Fehlertoleranz

Die Konfigurations-Software sollte fehlerhafte Angaben jeder Art in der Experimentbeschreibung verarbeiten können und entsprechende Fehlermeldungen und Hinweise für den Benutzer generieren. Die implementierte Software erfüllt diese Anforderungen bisher jedoch nur in einem beschränkten Maße. Hier ist noch Arbeit zu leisten, um einerseits die Fehlertoleranz der Software zu erhöhen und auch bei extremen Fehlangaben definierte Software-Reaktionen zu erhalten, andererseits die Qualität der Fehlermeldungen zu verbessern. Dazu sind vielfältige Konsistenzchecks notwendig, die neben bereits realisierten Überprüfungen der physikalischen und logischen Konfiguration auch die Bearbeitungsvorschriften miteinbeziehen. Hilfreich in diesem Zusammenhang wäre ebenfalls, wie ursprünglich vorgesehen, eine graphische Eingabemöglichkeit, die von vornherein Fehler in der Experimentbeschreibung unterbinden könnte.

11.3.2 Verbesserung der Funktionalität

11.3.2.1 Statische Experimentbeschreibung

Das Konzept der virtuellen Geräte, wie es bereits bei Steuerung, Datentransfer und Eventbuilding bei komplexen Experimenten erfolgreich eingesetzt wurde, läßt sich auch in anderen Bereichen der Datenaufnahme sinnvoll einsetzen. Virtuelle Gräte repräsentieren grundsätzlich Meßdaten, für die kein unmittelbares Gegenstück in Form eines physikalischen Gerätes existiert. Insbesondere können damit zuvor aufgenommene Meßdaten unabhängig von ihrer logischen Struktur zusammengefaßt und zu einem neuen Bestandteil eines anderen Datensatzes gemacht werden. Damit ist eine noch bessere Trennung zwischen physikalischer und logischer Experimentstruktur möglich, die eine dynamische Zuordnung der Meßdaten zu ihrer logischen Bedeutung zuläßt und auch noch nachträglich bei Eventbuilding oder gar der Analyse der Meßdaten ein Umordnen der logischen Struktur erlaubt. Eine weitere Anwendung ist ähnlich wie beim Eventbuilding der Einsatz von intelligenten Subsystemen, die bereits vorformatierte Meßdaten liefern. Für den konsequenten Einsatz der virtuellen Geräte muß jedoch noch weitergehende Software erstellt werden, die virtuelle und reale Geräte einander auch in der praktischen Anwendung gleichstellt.

11.3.2.2 Bearbeitungsvorschriften

Durch weitergehende Ausnutzung des C-Parsers `mcc` kann auch die Angabe der Bearbeitungsvorschriften für den Benutzer wesentlich einfacher und gleichzeitig flexibler gestaltet werden. Im Zusammenhang mit der statischen Experimentbeschreibung ist damit nicht nur eine redundanzfreie Formulierung der für die Datenaufnahme notwendigen Aktivitäten möglich, sondern es kann durch eine detaillierte Analyse der Gesamtinformation durch spezielle, noch zu erstellende Software für die Datenaufnahme in noch stärkerem Maße von der Generierung optimierten Programmcodes Gebrauch gemacht werden, der an die individuellen Anforderungen des Experiments bestmöglich angepaßt ist.

11.4 On-line-Analyse

Bisher beschränkte sich die On-line-Analyse in MECDAS auf eine sehr rudimentäre Implementation. Im Rahmen dieser Arbeit konnte das Themengebiet nur soweit angerissen werden, daß dem Anwender grundsätzlich die Möglichkeit gegeben wurde, die mit dem Datenerfassungssystem aufgenommenen Daten mit wenig Aufwand weiterzubearbeiten. Hier wurde im wesentlichen Wert auf die korrekte Dekodierung und die Bereitsstellung der Daten gelegt und es wurden einfache Hilfsmittel für deren weitere Verarbeitung zur Verfügung gestellt. Im Rahmen anderer Arbeiten wurden zwar Teilaspekte der Analyse insbesondere von Daten aus Experimenten an der Drei-Spektrometer-Anlage bereits behandelt [Kram95, Dist95, Schr93], doch decken diese die Gesamtproblematik der On-line-Analyse nicht ausreichend ab.

11.4.1 Standard-Analyse

Ein erstes Ziel bei der Implementation von Analyse-Software war weniger die Realisierung eines Software-System für eine allgemeine und komfortable On-line-Analyse, als die Bereitstellung einer einfacheren Standard-Analyse für einzelne Experimente bzw. Experimentgruppen zu deren On-line-Überwachung und -Kontrolle. Die Erfahrungen haben dabei gezeigt, daß Experimente, die an ein und derselben Apparatur durchgeführt werden, dabei sehr gut durch einheitliche Software abgedeckt werden können. Typisch dafür ist die Drei-Spektrometer-Anlage. Hier wurde in mehreren parallelen Entwicklungen Software für diese Zwecke implementiert. Sie besteht aus Unterprogrammbibliotheken für verschiedene Aufgabenbereiche, die zusammen mit experimentspezifischer Software nach deren Übersetzung zu lauffähigen Programmen gebunden werden müssen. Änderung von Auswertebedingungen, Grenzen, Schwellen oder Histogramm-Parametern sind jedoch in der Regel nur durch ständiges Neuübersetzen und -binden der Software möglich. Erste Ansätze für abgeschlossene Standard-Programme zur On-line-Analyse [Kram95, Schr93], die etwa mit Parameterdateien arbeiten, sind jedoch noch nicht ausgereift und erfordern eine intensive Überarbeitung. Hier muß u. a. ebenso wie bei der derzeit verfügbaren Off-line-Analyse-Software wesentlich stärker und dedizierter von der in der Experimentbeschreibung enthaltenen logischen Konfigurationsinformation Gebrauch gemacht werden.

11.4.2 Dynamische Analyse

Aufbauend auf den Entwicklungen für experimentspezifische Standard-Analyse-Software ist es dringend notwendig, auch ein allgemeineres Software-System zu realisieren, das es erlaubt, einerseits mit wenig Aufwand für jedes Experiment, das mithilfe von MECDAS durchgeführt wird, auch eine adäquate Analyse bereitzustellen, andererseits aber auch komfortabel bedient zu werden, so daß auf möglichst einfache Art auch komplexe Analysevorschriften definiert und geändert werden können (siehe dazu auch Abschnitt 8.4).

Kapitel 12

Zusammenfassung

Zielsetzung der vorliegenden Arbeit war die Konzeption, die Realisierung und die Inbetriebnahme eines Datenerfassungssystems zur Durchführung der Experimente an MAMI. Dabei sollte den Anforderungen der neu aufgebauten Drei-Spektrometer-Anlage der A1-Kollaboration, die es erlaubt, präzise Koinzidenzexperimente mit virtuellen Photonen durchzuführen, besonders Rechnung getragen werden. Zu diesem Zweck wurde nach eingehenden Untersuchungen der Anforderungen und Realisierungsmöglichkeiten unter dem Namen MECDAS ein Software-System erstellt, das sich nach dem Baukastenprinzip aus Programmpaketen für die Aufgabengebiete Datenaufnahme, Datentransfer, Eventbuilding, Datenarchivierung, Meßablaufsteuerung und On-line-Analyse zusammensetzt. Es ist aufgrund der Komplexität der Detektorsysteme als ein verteiltes System konzipiert. In Hinblick auf eine möglichst weitgehende Einsetzbarkeit wurde besonderen Wert auf die Konfigurierbarkeit des Systems gelegt, die einerseits durch ein hohes Maß an Modularität in Hard- und Software erreicht werden konnte, andererseits durch ein spezielles Verfahren der Experimentbeschreibung.

MECDAS arbeitet mit einem verteilten, hierarchisch organisierten Rechnersystem unter UNIX, das je nach Komplexität des Experiments aus einer mehr oder weniger großen Anzahl von unterschiedlichen Rechnern besteht, die jeweils unabhängig voneinander Teilaufgaben übernehmen. Für experimentnahe Aufgaben werden VMEbus-Systeme eingesetzt, die als Einzelprozessorsystem oder Master-Slave-Multiprozessorsystem realisiert sind; übergeordnete Aufgabestellungen werden durch Workstations abgedeckt.

Die Software zur Datenerfassung setzt sich aus einem Satz von Programmen zusammen, die nach dem Client-Server-Prinzip arbeiten. Ein zentrales, auf der Basis einer vom Benutzer bereitgestellten Experimentbeschreibung generiertes Programm, das als einzelnes, eigenständiges Softwaremodul alle notwendigen experiment- und detektorspezifischen Informationen enthält, ist für die Auslese der Meßdaten über CAMAC, Fastbus oder VMEbus zuständig. Es arbeitet als Server, der zur Experimentsteuerung, Datenarchivierung und On-line-Analyse durch Programme ergänzt wird, die als seine Klienten Nachrichten mit ihm austauschen.

Die Experimentbeschreibung, zu deren Verarbeitung ein umfangreiches Programmpaket implementiert wurde, stellt die zentrale Datenbasis für alle Aufgaben der Datenerfassung dar. Sie wird komplett zu den zu archivierenden Daten gepackt, so daß sie auch für Off-line-Auswerteaufgaben direkt verfügbar ist. Damit sind die Daten, die letztendlich auf einem Magnetband oder einer Plattendatei abgespeichert sind, selbstbeschreibend und zu jedem Zeitpunkt ohne zusätzliche Information eindeutig identifizierbar. Die Software wird ergänzt durch Vorschriften zur Benutzung des Systems bzw. seiner einzelnen Komponenten und die Definition von Schnittstellen, die es erlauben, vom Benutzer bereitgestellte Software einfach anzukopplen und damit das System speziellen Bedürfnissen anzupassen bzw. zu erweitern.

Als Ergänzung zu den Programmen für die Datenerfassung wurde im Rahmen dieser Arbeit schließlich auch Software zur weiteren Verarbeitung der einmal aufgenommenen Meßdaten realisiert. Neben einer Reihe von Standard-Programmen konnte eine Unterprogrammbibliothek zur Verfügung gestellt werden, die es dem Experimentator erlaubt, mit wenig spezifischen Kenntnissen einfach auf die gemessenen Daten zuzugreifen und diese auszuwerten. Insbeondere besitzt MECDAS Methoden und Schnittstellen, die im On-line-Betrieb einen direkten Zugriff auf die aktuellen Meßdaten erlauben. Vom Benutzer erstellte Auswerteprogramme oder Standardprogramme für diese Zwecke können damit einfach zur On-line-Überwachung des Experiments genutzt werden. Wie bei Datenerfassung und Eventbuilding wird auch bei der Analyse von der Experimentbeschreibung Gebrauch gemacht. Die Analyse-Software wird schließlich ergänzt durch den speziellen Erfordernissen des On-line-Betriebs angepaßte Programme und Routinen zur Generierung und Bearbeitung von Histogrammen und deren graphischen Darstellung.

Die im Rahmen dieser Arbeit entwickelte Software ist vollständig in der Hochsprache C geschrieben. Bei der Implemetation wurde großen Wert auf Portabilität gelegt. Gleichzeitig lehnt sich das MECDAS zugrundeliegende Konzept weitestgehend an die Prinzipien der objektorientierten Programmierung an. Das so realisierte Datenerfassungssystem stellt einen integralen Bestandteil der Drei-Spektrometer-Anlage dar. Mit ihm wurden alle von der A1-Kollaboration durchgeführten Experimente abgewickelt. Das Datenerfassungssystem kommt inzwischen aber auch in anderen Experimenten innerhalb des Instituts für Kernphysik zum Einsatz. Dabei hat sich neben seiner Funktionalität insbesondere auch seine Konfigurierbarkeit bewährt. Das Datenerfassungssystem wurde u.a. bei (e,e'p)-Messungen an ^{12}C und ^{16}O im quasielastischen Bereich, zur Elektroproduktion von Pionen am Proton und zur Elektroproduktion von Multihadron-Endzuständen an ^{12}C mit Ereignisraten von bis zu 250 Hz erfolgreich eingesetzt und hat damit seine Funktionsfähigkeit unter Beweis gestellt.

Danksagung

An dieser Stelle möchte ich mich ganz herzlich bei allen bedanken, die zum Gelingen dieser Arbeit beigetragen haben.

Hern Prof. Dr. Th. Walcher danke ich für die Themenstellung und sein stetes Interesse am Fortgang der Arbeit.

Bei Herrn Prof. Dr. K. Merle möchte ich mich bedanken für die Betreuung, für zahlreiche Diskussionen, Gespräche und wertvolle Anregungen im Laufe unserer längjährigen Zusammenarbeit, aber auch für den mir überlassenen Freiraum.

Allen Mitgliedern der A1-Kollaboration gilt mein Dank für die gute Zusammenarbeit beim gemeinsamen Aufbau der Drei-Spektrometer-Anlage. Insbesondere möchte ich mich bei Helmut Kramer, Volker Kunde, Jörg-Michael Henneberg und Rainer Geiges für die kollegiale Zusammenarbeit bedanken. Peter Sauer, Klaus Weindel und Norbert Clawiter danke ich ganz besonders für viele hilfreiche und interessante Diskussionen.

Ebenso gilt mein Dank auch Frau Saebel, Herrn Hoff und Herrn Walther für das konstruktive Arbeitsklima und die gute Zusammenarbeit innerhalb der EDV-Gruppe.

Schließlich möchte ich mich ganz herzlich bei meinen Eltern und meinem Bruder Andreas bedanken, die stets mit viel Verständnis alle Einschränkungen und Belastungen, die mit dieser Arbeit verbunden waren, mit mir zusammen getragen haben und mich zu jedem Zeitpunkt uneingeschränkt unterstützten. Leider war es meinem Vater nicht mehr vergönnt, den Abschluß der Arbeit mitzuerleben.

Literaturverzeichnis

[A190] A1-proposal A1/1-90, *An Investigation of the Electroproduction of Multihadron Final States in Quasielastic, Dip and Δ Resonance Region*, Mainz, 1990

[A192] A1-Kollaboration, *Experimente mit virtuellen Photonen*, Jahresbericht 1992/93

[A193] A1-proposal A1/2-93, *Measurement of polarized protons from quasielastic electron scattering on ^{12}C and ^{40}Ca*, Mainz, 1993

[AT&T97] AT&T, *The System V Interface Definition (SVID)*, AT&T, Murray Hill, 1987

[Aho86] A.V. Aho, R. Sethi, J.D. Ullman, *Compilers: Principles, Techniques and Tools*, Addison-Wesley 1986

[Aho88] A.V. Aho, B.W. Kernighan, P.J. Weinberger, *The AWK Programming Language*, Addison-Wesley 1988

[Albi92] P. Albitz, C. Liu, *DNS and BIND*, O'Reilly & Associates 1992

[Auth90] K. Authenrieth, *Technik verteilter Betriebssysteme*, Hüthig 1990

[BSD86a] *UNIX Programmer's Supplementary Documents*, 4.3 Berkeley Software Distribution, Berkeley, 1986

[BSD86b] *UNIX System Managers Manual*, 4.3 Berkeley Software Distribution, Berkeley, 1986

[Bach86] M.J. Bach, *The Design of the UNIX Operating System*, Prentice Hall 1986

[Bana84] M. Banahan, A. Rutter, *UNIX lernen, verstehen, anwenden*, Hanser 1984

[Bash] B. Fox et al., *Bash Man Page*, Online-Dokumentation, Free Software Foundation

[Blom95] K.I. Blomqvist et al., *No evidence for medium effects in the $^{12}C(e,e'p)^{11}B_{g.s.}$ reaction*, Z. Physik A (im Druck)

[Böhm95] R. Böhm, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Bohn90] M. Bohn, *Analyse kernphysikalischer Experimente auf Parallelrechnern (Transputern)*, Diplomarbeit, KPH 22/90, Institut für Kernphysik, Universität Mainz, 1990

[Bolc93] Bolch/Vollath, *Prozeßautomatisierung*, Teubner 1993

[Bour83] S. Bourne, *The UNIX System*, Addison-Wesley 1983

[Bred91] H.-J. Brede, N. Josuttis, S. Lemberg, A. Lörke, *Programmieren mit OSF/Motif*, Addison-Wesley 1991

[Brun91] R. Brun et al., *PAW — Physics Analysis Workstation*, CERN, 1991

[Brun94] R. Brun et al., *CERN Software Documentation*, CERN, 1994

[CES86] CES, *CAMAC Branch Driver CBD 8210 User's Manual*, Creative Electronic Systems S.A., 1986

[Claw95] N. Clawiter, *Das TDC-2001 System des A1-Experiments: Aufbau und erste Testmessungen*, Diplomarbeit in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Come88] D. Comer, *Internetworking with TCP/IP*, Prentice-Hall 1988

[Corb91] J.R. Corbin, *The Art of Distributed Applications*, Springer 1991

[DEC94] DEC, *DEC OSF/1 Documentation*, Digital Equipment Corporation, 1994

[DIN82] DIN ISO 7498, *Informationsverarbeitung Offener Systeme, Basis-Referenzmodell*, Beuth 1982

[DIN86] DIN, Deutsche Norm 66252, *Graphisches Kernsystem (GKS)*, Beuth 1986

[Dibb89] P. Dibble, *OS-9 Insights — Ein Programmierhandbuch für OS-9/68000*, Hüthig 1989

[Dist93a] M. Distler, *Cindy++*, Interner Report A1/EDV, Institut für Kernphysik, Universität Mainz, 1993

[Dist93b] M. Distler, *Chlib++ — The Chamber Library*, Interner Report A1/EDV, Institut für Kernphysik, Universität Mainz, 1993

[Dist93c] M. Distler, *A1 tape utilities*, Interner Report A1/EDV, Institut für Kernphysik, Universität Mainz, 1993

[Dist95] M. Distler, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Edel95] R. Edelhoff, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Egan90] J.I. Egan, T.J. Teixera, *UNIX Device-Treiber*, Addison-Wesley 1990

[Elte87] Eltec, *Eurocom-5 Documentation*, Hard- und Software-Manual, Eltec Elektronik Mainz, 1987

[Elte90] Eltec, *Eurocom-6 Documentation*, Hard- und Software-Manual, Eltec Elektronik Mainz, 1990

[Erlg84] *Erlgraph — Erlanger Graphik-System*, Benutzer-Handbuch, Erlangen, 1984

[Esse88] H.G. Essel, *GOOSY Data Acquisition and Analysis*, GSI, Darmstadt, 1988

[Eth80] Digital Equipment Corpbib/Intel Corpbib/Xerox Corp., *The Ethernet — A Local Area Network, Data Link Lanyer and Physical Layer Specifications*, 1983

[Feld79] S. Feldman, *MAKE — A Program for Maintaining Computer Programs*, Software — Practice & Experience 1979

[Geig93a] R. Geiges, K. Merle, *A High Resolution TDC Subsystem*, IEEE Transactions on Nuclear Science, Vol. 41, No. 1

[Geig93b] R. Geiges, P. Gitzel, K. Merle, H. Walther, K. Weindel, *Entwicklung eines TDC-Subsystems für Spektrometer C des A1-Experiments*, Jahresbericht 1992/93, S. 274ff

[Geig93c] R. Geiges, A. Richter, H. Walther, *Entwicklung eines Eventnummerngenerators für den A1-Setup*, Jahresbericht 1992/93, S. 252f

[Grav92] M.F. Gravina, P.F. Kunz, P. Rensing, *Hippoplotamus Users Guide*, SLAC, Stanford University, 1992

[Hake93] A. Hake, *Softwarewerkzeuge zur Steuerung an MAMI-Experimenten*, Diplomarbeit, KPH 9/93, Institut für Kernphysik, Universität Mainz, 1993

[Hatl88] D.J. Hatley, I.A. Pirbhai, *Strategien für die Echtzeit-Programmierung*, Hanser 1988

[Heil91a] A. Heil, private Mitteilung, Mainz, 1991

[Heil91b] A. Heil, private Mitteilung, Mainz, 1991

[Herm76] H. Herminghaus, A. Feder, K.H. Kaiser, W. Manz, H.v.d. Schmitt, *A cascaded racetrack-microtron with high duty cycle*, Nulc. Instr. Meth. 138, page 1, 1976

[Hero92] H. Herold, *lex und yacc — Lexikalische und syntaktische Analyse*, Addison-Wesley 1992

[IEEE82] IEEE, *CSMA/CD Access Method and Physical Layer Specifications*, IEEE Standard 802.3, 1982

[IEEE91] IEEE Seventh Conference REALTIME'91 on Computer Applications in Nuclear Physics, Conference Record, Jülich, 1991

[Jell91] T. Jell, A.v. Reeken, *Objektorientiertes Programmieren in C++*, Hanser 1991

[John78] S.C. Johnson, *Yacc: Yet Another Compiler Compiler*, UNIX Programmer's Manual, Band 2, Bell Laboratories 1978

[Kahr95] M. Kahrau, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Kall93] F. Kalleicher, private Mitteilung, Mainz, 1993

[Kauf94] F.-J. Kauffels, *Lokale Netze — Grundlagen, Standards, Perspektiven*, Markt & Technik 1994

[Kern81] B.W. Kernighan, J.R. Mashey, *The UNIX Programming Environment*, IEEE Computer Magazine, 1981

[Kern83] B.W. Kernighan, D.M. Ritchie, *Programmieren in C*, Hanser 1983

[Kern86] B.W. Kernighan, R. Pike, *Der UNIX-Werkzeugkasten, Programmieren mit UNIX*, Hanser 1986

[Kern90] B.W. Kernighan, D.M. Ritchie, *Programmieren in C*, 2. Ausgabe, Hanser 1990

[Klei87a] St. Klein, *CAmac Routines for Offline Analysis*, Institut für Kernphysik, Universität Mainz, 1987

[Klei87b] K. Kleinknecht, *Detektoren für Teilchenstrahlung*, Teubner 1987

[Klöp91] B. Klöppel, *Compilerbau am Beispiel der Programmiersprache SIMPL*, Vogel 1991

[Knut68] D.E. Knuth, *The Art of Computer Programming*, Addison-Wesley 1968

[Koeb95] S. Koebis, Diplomarbeit, Institut für Kernphysik, Universität Mainz, 1995

[Kram94] H. Kramer, private Mitteilung, Mainz, 1994

[Kram95] H. Kramer, *Grundlagen für das Steuerungs- und Überwachungssystem der Drei-Spektrometer-Anlage am Elektronenbeschleuniger MAMI*, Dissertation, Institut für Kernphysik, Universität Mainz, 1995

[Kryg86] K.W. Krygier, *Mikroprozessorgesteuerte Datenerfassung bei (e,e'x)-Koinzidenz-Experimenten*, Diplomarbeit, KPH 6/86, Institut für Kernphysik, Universität Mainz, 1986

[Kryg87] K.W. Krygier, V. Kunde, K. Merle, *Datenerfassung bei (e, e'x)-Experimenten unter Einsatz von VME-Systemen*, Jahresbericht 1986/87, S. 281ff

[Kryg89a] K.W. Krygier, V. Kunde, K. Merle, MECDAS — *ein flexibles Datenerfassungssystem für Koinzidenzexperimente*, Jahresbericht 1988/89, S. 223–227

[Kryg89b] K.W. Krygier, V. Kunde, K. Merle, *VMEbus-Systeme im Institut für Kernphysik*, Jahresbericht 1988/89, S. 220ff

[Kryg89c] K.W. Krygier, K. Merle, *Netzwerk-Software unter OS-9*, Jahresbericht 1988/89, S. 227ff

[Kryg91] K.W. Krygier, J. Weiß, *VMEbsd — BSD-UNIX für den E6*, Interner Report EDV/KPH, Institut für Kernphysik, Universität Mainz, 1991

[Kryg92] K.W. Krygier, K. Merle, J. Weiß, *UNIX-Portierung für VMEbus-Rechner*, Jahresbericht 1990/91, S. 202f

[Kryg93] K.W. Krygier, K. Merle, MECDAS — *A Distributed Data Acquisition System for Experiments at MAMI*, Eighth Conference on Rela-Time Computer Applications in Nuclear Physics, Particle and Plasma Physics, Vancouver, 1993

[Kryg94] K.W. Krygier, K. Merle, MECDAS — *A Distributed Data Acquisition System for Experiments at MAMI*, IEEE Transactions on Nuclear Science, Vol. 41, p. 86–88, 1994

[Kryg95a] K.W. Krygier, MECDAS — *Datenerfassung*, Interner Report EDV/KPH, Institut für Kernphysik, Universität Mainz, 1995

[Kryg95b] K.W. Krygier, MECDAS — *Experiment-Konfiguration*, Interner Report EDV/KPH, Institut für Kernphysik, Universität Mainz, 1995

[Kryg95c] K.W. Krygier, MECDAS — *Analyse*, Interner Report EDV/KPH, Institut für Kernphysik, Universität Mainz, 1995

[Korn95] M. Korn, *Entwicklung des Bahnrückverfolgungsverfahrens für die Drei-Spektrometer-Anlage und experimentelle Bestimmung der Abbildungseigenschaften der Spektrometer A und B mit elastischer Elektronenstreuung*, Dissertation, KPH 4/95, Institut für Kernphysik, Universität Mainz, 1995

[Kühn88] W. Kühn, private Mitteilung, Mainz, 1988

[Kund88] V. Kunde, *Aufbau eines Datenerfassungssystems für Vielparameterexperimente und dessen Erprobung an MAMI-A*, Diplomarbeit, KPH 11/88, Institut für Kernphysik, Universität Mainz, 1988

[Kund95] V. Kunde, *Aufbau des Experimentsteuerungssystems der Drei-Spektrometer-Anlage und Messung von $^{12}C(e,e'x)$ mit der BGO-Kristallkugel*, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Kunz91] P.F. Kunz, *Object Oriented Programming*, SLAC-PUB-5629, Stanford Linear Accelerator Center, Stanford, 1991

[Kunz92] M. Kunze, *Experience with mixed language programming*, New Computing Techniques in Physics Research II — Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics, 1992

[Kuss95] M. Kuss, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Lang94] H. Langendörfer, B. Schnor, *Verteilte Systeme*, Hanser 1994

[Lecr1] LeCroy Research Systems SA, *Fastbus Model 1881 Programmable Segment Manager/Interface, Users Manual*

[Lecr2] LeCroy Research Systems SA, *Fastbus Model 1882 N/F 96 Channel ADC, Users Manual*

[Lecr3] LeCroy Research Systems SA, *CAMAC Model 4299 TDC Readout System, Users Manual*

[Leff89] S. Leffler, M. McKusick, M. Karels, J. Quaterman, *Das 4.3-BSD-UNIX-Betriebssytstem, Design und Implementation*, Addison-Wesley, 1989

[Lesk78] M.E. Lesk, E. Schmidt, *Lex: A Lexical Analyzer Generator*, UNIX Programmer's Manual, Band 2, Bell Laboratories 1978

[Levi90] J.R. Levine, T. Mason, D. Brown, *lex & yacc*, O'Reilly & Associates 1990

[Lies95] A. Liesenfeld, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Lipp91] S. Lippman, C++ Primer, Addison-Wesley 1991

[Louk93] M. Loukides, *System Performance Tuning*, O'Reilly & Associates 1993

[Lpr] UNIX Programmer's Manual

[Lynx92] Sysgo, LynxOS-Produktinformation, Sysgo Mainz, 1992

[Mans90] N. Mansfield, *Das Benutzerhandbuch zum X Window-System*, Addison-Wesley 1990

[Mart93] Ch. Martin, *Implementation und Test der A1-Targetsystemsoftware und Integration ins Gesamtsystem für die Kontrolle von (e,e'p)-Messungen*, Diplomarbeit, Institut für Kernphysik, Universität Mainz, 1993

[Mart94] J. Martin, J. Leben, *TCP/IP-Netzwerke — Architektur, Administration und Programmierung*, Prentice Hall 1994

[Märt94] Ch. Märtin, *Rechnerarchitektur — Struktur, Organisation, Implementierungstechnik*, Hanser 1994

[Meiß85] K. Meißner, *Arbeitsplatzrechner im Verbund*, Hanser 1985

[Merl83] K. Merle, K.W. Krygier, *Experiment-Datenerfassung mit UNIX*, Jahresbericht 1982/83, S. 138ff

[Merl88a] K. Merle, U. Müller, H. Walther, *VMEbus-Interface für einen FASTBUS-Segment-Manager*, Jahresbericht 1988/89

[Merl88b] K. Merle, H. Walther, K. Weindel, *Schnelle optische Datenstrecke zur Kopplung von VMEbus-Systemen*, Jahresbericht 1988/89

[Micr87] Microware, *OS-9/68000 Operating System, Technical Manual*, Microware Systems Corporation, 1987

[Mock87] P.V. Mockapetris, *Domain Names, Concepts and Facilities*, RFC 1034, 1987

[Murp89] R. Murphey, *GNU graphics — Utilities for plotting scientific data*, Free Software Foundation, 1989

[Nico86] Rüdiger Nicolovius, *Graphik mit GKS*, Hanser 1986

[OSI83] ISO, *Information Processing Systems — Open Systems Interconnection — Basic Reference Model*, ISO/DIS 7498, 1983

[Offe93] E.A.J.M. Offerman, *ESPACE — Event Scanning Programm for A1 Collaboration Experiments*, Interner Report, Institut für Kernphysik, Universität Mainz, 1993

[Oust94] J.K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley 1994

[Peis89] J. Peise, *Kohärente Bremsstrahlung von Elektronen in einem Diamantkristall*, Diplomarbeit, Institut für Kernphysik, Universität Mainz, 1989

[Peis95] J. Peise, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Pete89] W.D. Peterson, *The VMEbus Handbook — A User's Guide to the IEEE 1014 and IEC 821 Microcomputer Bus*, VITA, 1989

[Plot] UNIX Programmer's Manual

[Posi88] *IEEE P1003.1 Portable Operating System Interface for Computer Environments (POSIX)*, Institute of Electrical and Electronic Engineers, 1988

[Posp95] Th. Pospischil, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Post80] J.B. Postel, *User Datagram Protocol*, RFC 768, 1980

[Post81a] J.B. Postel, *Internet Protocol*, RFC 791, 1981

[Post81b] J.B. Postel, *Internet Control Message Protocol*, RFC 792, 1981

[Post81c] J.B. Postel, *Transmission Control Protocol*, RFC 793, 1981

[Post93] J.B. Postel, *Internet Official Protocol Standards*, RFC 1540, 1993

[Pres88] W.H. Press et al., *Numerical Recipies in C*, Cambridge University Press 1988

[Quar93] Quarrie et al., *An Object Oriented Run control environment for the CEBAF Data Acquisition System*, Proceedings of the International Conference on Computing in High Energy Physics '92, Annecy, France

[Rich95] A. Richter, *Trennung des longitudinalen, transversalen und longitudinal-transversal interferierenden Anteils des Wirkungsquerschnitts der Reaktion H(e,e'π^+) in der Nähe der Pionenschwelle*, Dissertation, Institut für Kernphysik, Universität Mainz, 1995

[Ritc74] D.M. Ritchie, K.L. Thompson, *The UNIX Time-sharing System*, Communications of the ACM, 1974

[Ritc79] D.M. Ritchie, *The evolution of the UNIX time-sharing system*, Symposium on Language Design and Programming Methodology, 1979

[Roch75] M. Rochkind, *The source code control system*, IEEE Trans. on Software Engineering 1975

[Rsh] UNIX Programmer's Manual

[SCSI] ANSI, *SCSI — Small Computer System Interface*, ANSI X3T9.2/82-2, Rev. 9, 1983

[SFB92] *Finanzierungsantrag 1993–1995 Sonderforschungsbereich 201 Mittelenergiephysik mit elektromagnetischer Wechselwirkung*, Mainz, 1992

[SFB95] *Finanzierungsantrag 1996–1998 Sonderforschungsbereich 201 Mittelenergiephysik mit elektromagnetischer Wechselwirkung*, Mainz, 1995

[Sand85] R. Sandberg, *The Design and Implementation of the Sun Network File System*, USENIX Association Conference Proceedings, 1985

[Sant89] L.L. Santoline et al., *Multiprocessor shared-memory information exchange*, IEEE Transactions on Nuclear Science, Vol 36, 1989

[Sant90] M. Santifaller, *TCP/IP und NFS in Theorie und Praxis*, Addison-Wesley 1990

[Saue95] P. Sauer, *Entwicklung, Aufbau und Inbetriebnahme der vertikalen Driftkammern der Drei-Spektrometer-Anlage am Mainzer Mikrotron MAMI und Studium der Reaktion* $^{12}C(e,e'x)^{11}B$ *für mittlere und hohe Nukleonenimpulse im Kern*, Dissertation, KPH 12/95, Institut für Kernphysik, Universität Mainz, 1995

[Scha95] St. Schardt, *Aufbau und Erprobung der Drei-Spektrometer-Anordnung für Koinzidenzexperimente mit Elektronen am 855 MeV-Elektronenbeschleuniger MAMI*, Dissertation, Institut für Kernphysik, Universität Mainz, 1995

[Schm89] F. Schmidt, private Mitteilung, Mainz, 1989

[Schr85] A.T. Schreiner, G. Friedman, *Compiler bauen mit UNIX — Eine Einführung*, Hanser 1985

[Schr87] A.-T. Schreiner, *Professor Schreiners UNIX-Sprechstunde*, Hanser 1987

[Schr89] A.-T. Schreiner, *C-Praxis mit curses, lex und yacc*, Hanser 1989

[Schr93] Ch. Schrimpf, *Entwicklung und Test von Analysemethoden im Rahmen der* MECDAS-*Datenerfassung*, Diplomarbeit, KPH 15/93, Institut für Kernphysik, Universität Mainz, 1993

[Stef93] St. Steffens, *Entwicklung und Erprobung einer „X-Windows"-Benutzeroberfläche zur rechnergesteuerten und -kontrollierten Durchführung von Koinzidenzexperimenten an MAMI*, Diplomarbeit, KPH 7/93, Institut für Kernphysik, Universität Mainz, 1993

[Ster91] H. Stern, *Managing NFS und NIS*, O'Reilly & Associates 1991

[Ster93] H. Stern, *NFS und NIS, Managing von UNIX-Netzwerken*, Addison-Wesley 1993

[Stro92] B. Stroustrup, *The C++ programming language*, Addison-Wesley 1992

[Stev90] W.R. Stevens, *UNIX network programming*, Prentice Hall 1990

[Stev92] W.R. Stevens, *Advanced Programming in the UNIX environment*, Addison-Wesley 1992

[Stev94] W.R. Stevens, *TCP/IP Illustrated*, Addison-Wesley 1994

[Stol87] Stollmann, *UNIX System V.3.1 Documentation*, Stollmann GmbH, 1987

[Stol88] P. Stoll, *Datenerfassung bei (e,e,'x)-Koinzidenzexperimenten unter Einsatz von Mehrprozessorsystemen*, Diplomarbeit, KPH 16/88, Institut für Kernphysik, Universität Mainz, 1988

[Sun87] Sun Microsystems, *XDR: External Data Representation Standard*, RFC 1014, 1987

[Sun88a] Sun Microsystems, *RPC: Remote Procedure Call, Protocol Specification*, RFC 1057, 1988

[Sun88b] Sun Microsystems, *NFS: Network File System Protocol Specification*, RFC 1094, 1988

[Sun93] Sun, *Solaris Documentation*, Sun Microsystems, 1993

[Syslog] UNIX Programmer's Manual

[Tane90] A.S. Tanenbaum, *Betriebssysteme*, Hanser 1990

[Thom78] K. Thompson, *UNIX Implementation*, The Bell System Technical Journal 1978

[Venu91] V. Venuß, *Erweiterung der Meßdatenerfassung MECDAS auf ein Multiprozessorsystem unter Einsatz von UNIX und Transputern*, Diplomarbeit, KPH 11/91, Institut für Kernphysik, Universität Mainz, 1991

[Voeg82] N. Voegler, J. Friedrich, Nucl. Instr. and Meth. 149 (1982) 293

[Voge84] U. Vogel, *Ein universelles Programmsystem zur Datenerfassung sowie zur „on-line" und „off-line" Analyse im Rechnerverbund bei vielparametrigen Experimenten*, Diplomarbeit, Institut für Kernphysik, Universität Mainz, 1984

[Volc88] R. Volck, *Software Engineering mit UNIX-Workstations*, Hanser 1988

[vdSc89] H.v.d. Schmitt, *RTF/68K Real-Time Fortran-77 for 68K Processors — Manual of Compiler and Run-Time Library Vers. 3.4*, Physikalisches Institut, Universität Heidelberg, 1989

[Wagn95] A. Wagner, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[Weis91] J. Weiß, private Mitteilung, Mainz, 1991

[Wind93] WindRiver, *VxWorks Product Information*, WindRiver Systems, 1993

[Wirt77] N. Wirth, *Compilerbau*, Teubner 1977

[Wirt83] N. Wirth, *Algorithmen und Datenstrukturen*, Teubner 1983

[Wolf95] St. Wolf, Dissertation in Vorbereitung, Institut für Kernphysik, Universität Mainz, 1995

[X1188a] A. Nye, *Xlib Programming Manual, Xlib Reference Manual*, O'Reilly & Associates 1988

[X1188b] T. O'Reilly, V. Queria, L. Lamb, *X Window System User's Guide*, O'Reilly & Associates 1988

[X1188c] O. Jones, *Introduction to the X Window System*, Prentice-Hall 1988

[X1190a] D. Gilly, T. O'Reilly, *The X Window System in a Nutshell*, O'Reilly & Associates 1990

[X1190b] R.W. Scheifler, J. Gettys *X Window System*, Digital Press 1990

[X11doc] Online-Dokumentation zum X-Window-System Version 11, Release 5

Anhang A

Grundlagen

A.1 Der VMEbus

Der VMEbus basiert auf einer Entwicklung des Hardware-Herstellers Motorola und ist als ein offenes, inzwischen von vielen Herstellern unterstütztes Bus-System standardisiert [Pete89]. Er ist ein modulares System, für das eine Vielzahl von Einschubkarten in Einfach- oder Doppel-Europakarten-Format verfügbar sind. Eine breite Angebotspalette reicht von einfachen Digital-I/O-Interfaces über ADCs, DACs oder Kommunikationsschnittstellen bis hin zu leistungsfähigen Rechnern. Aus historischen Gründen spielten dabei ursprünglich nur Mikrocomputer auf der Basis von Motorola-Prozessoren eine Rolle. Im Laufe der Zeit kamen aber auch Prozessoren anderer Hersteller hinzu, und seit geraumer Zeit sind moderne RISC-CPUs verfügbar.

Der Anwender hat damit die Möglichkeit, ein Prozeßrechnersystem nach problemorientierten und wirtschaftlichen Gesichtspunken aus einzelnen Komponenten des einen oder anderen Herstellers zusammenzustellen und es so optimal auf gegebene Anforderungen und Randbedingungen anzupassen. Er erhält ein wartungsfreundliches und flexibles System, bei dem auch eine nachträgliche Erweiterung oder der Ersatz einzelner veralteter Komponenten einfach möglich ist. Der VMEbus-Standard gewährleistet dabei eine gute Interoperabilität, d. h., I/O-Karten und CPUs sollten unabhängig vom Hersteller gut zusammenarbeiten.

Der VMEbus ist je nach Ausbaustufe 16- bzw. 32-bit-orientiert und erlaubt eine Datenübertragungsrate bis hin zu 50 MB/s. Er ist ein Multimaster-System, bei dem über eine Arbitrierungs-Logik mehrere Karten aktiv Kontrolle über die verbindenden Busleitungen haben können, und ist damit multiprozessorfähig. In einem VMEbus-Überrahmen können also mehrere CPU-Karten parallel arbeiten und über den VMEbus auf dieselbe Hardware zugreifen. Ein einfaches Interrupt-System dient dabei zur Verwaltung von Unterbrechungsanforderungen.

Die Standardisierung des VMEbus hat zur Folge, daß neben der Verfügbarkeit von umfangreicher Hardware auch Software-Unterstützung auf breiter Basis existiert. Zusammen mit einer Reihe von Betriebssystemen für unterschiedliche

Anwendungsgebiete sind Compiler und Entwicklungsumgebungen für die wichtigsten Programmiersprachen und umfangreiche, weitere Software verfügbar

A.2 Workstations

Workstations sind leistungsfähige Mikrocomputer auf der Basis moderner CISC- oder RISC-Prozessoren, die in erster Linie durch ihre graphische Bedienkonsole mit leistungsfähigem Bildschirm, Tastatur und Graphikeingabegerät (meist eine sog. Maus) gekennzeichnet sind. Sie verfügen in der Regel über Arbeitsspeicher von einigen 10 MB und Anschlußmöglichkeiten für externen Massenspeicher, oftmals auch integrierte Festplatten, und besitzen durchweg eine Netzwerk-Schnittstelle. Sie sind als Arbeitsplatz-Rechner konzipiert, die zwar im Gegensatz zu PCs im Multitasking/Multiuser-Betrieb arbeiten, in der Regel aber nur von wenigen Personen gleichzeitig interaktiv genutzt werden.

Ähnlich wie der VMEbus sind Workstations keine Spezialentwicklungen für den naturwissenschaftlichen Bereich sondern inzwischen vielmehr in Industrie und Wirtschaft weit verbreitet und in großen Stückzahlen im Einsatz. Sie sind selbst keinem Standard unterworfen, während im Bereich der Peripherie vielfältig von Standards wie SCSI, Ethernet, V.24 oder Centronics Gebrauch gemacht wird. Aber auch auf Software-Ebene existiert eine sehr gute Kompatibilität, da die Workstations fast aller Hersteller unter Derivaten des Betriebssystems UNIX arbeiten. In vielen Fällen sind solche Workstations bzgl. Funktion und Rechenleistung gut gegeneinander austauschbar. Der dadurch starke Konkurenzkampf auf dem Workstation-Markt führte zu einer anhaltend günstigen Entwicklung des Preis/Leistungsverhältnisses sowohl bei den Workstations selbst als auch bei Arbeits- und Massenspeicher.

A.3 Die Echtzeit-Problematik

A.3.1 Die Totzeit

Eine wichtige, vom Experiment her vorgegebene Anforderung ist eine möglichst hohe Geschwindigkeit bei der Datenaufnahme, um die Totzeit und die damit verbundenen Verluste an nützlichen Ereignissen möglichst klein zu halten. Die Totzeit τ, also die Zeit während der Bearbeitung eines Ereignisses, in der keine neuen Ereignisse registriert werden können, setzt sich aus mehreren Komponenten zusammen. Im wesentlichen kann man zwischen dem von Detektoren und Experimentelektronik verursachten Anteil τ_{intr}, der bei der Analog/Digital-Konvertierung auftretenden Konversionszeit τ_{conv} und der durch Hard- und Software herbeigeführten Rechnertotzeit τ_{comp} unterscheiden; es gilt also:

$$\tau = \tau_{intr} + \tau_{conv} + \tau_{comp}$$

Während τ_{intr} weit unterhalb einer Mikrosekunde liegt und bei zu erwartenden Raten von weniger als 1000 zu erfassender Ereignisse pro Sekunde erst einmal

vernachlässigt werden kann, spielen die Konversionszeiten von ADCs und TDCs schon eher eine Rolle. Sie liegen bei der eingesetzten Hardware bei bis zu einigen 100 μs. Für die Größe der Rechnertotzeit lassen sich a priori keine konkreten Angaben machen. Hier bestimmen neben der Geschwindigkeit der Rechner-Hardware wesentlich Konzept und Realisierung der Software, wie schnell ein Ereignis bearbeitet werden kann.

Generell läßt sich auch die Rechnertotzeit in mehrere Anteile unterteilen. Sie besteht zum einen aus der Reaktionszeit τ_{react} auf das von der Meßelektronik generierten und evtl. um die Konversionszeit verzögerten Trigger-Signal, das die Registrierung eines gültigen Ereignisses durch die Detektoren anzeigt. Hinzu kommen die Zeiten τ_{cs1} und τ_{cs2}, die benötigt werden, um den Programm-Kontext umzuschalten (sog. Context Switch), und die eigentliche Bearbeitungszeit τ_{acq}, in der der Rechner alle Aktionen zur Datenerfassung ausführt. Es gilt:

$$\tau_{comp} = \tau_{react} + \tau_{cs1} + \tau_{acq} + \tau_{cs2}$$

Abbildung A.1.a verdeutlicht diese Tatsache. Bei gegebener Prozessorleistung wird τ_{acq} wesentlich von der Anzahl und Organisation der zu bearbeitenden Parameter und der Art der Schnittstellen zur Experimentelektronik bestimmt. τ_{react}, τ_{cs1} und τ_{cs2} sind diesbezüglich konstant, hängen aber von den Methoden ab, wie der Rechner auf das Trigger-Signal reagiert.

A.3.2 Reaktion auf externe Ereignisse

Für einen Computer stellt die Registrierung eines physikalischen Ereignisses ein sog. externes Ereignis dar, für dessen Bearbeitung prinzipiell zwei verschiedene Mechanismen existieren. Das erste Verfahren wird Polling genannt. In diesem Fall überprüft der Rechner in bestimmten Zeitabständen aktiv die Signalquelle, die das externe Ereignis mit einen Zustandswechsel anzeigt. Sobald dieser registriert wurde, kann der Rechner alle Aktivitäten zur Bearbeitung dieses Ereignisses einleiten.

Bei der zweiten Methode, der Interrupt[1]-Verarbeitung, kann der Rechner durch ein spezielles, bei Auftreten des externen Ereignisses erzeugten Unterbrechungssignals veranlaßt werden, ein gerade in Ausfühung befindliches Programm zu unterbrechen, um die Bearbeitung des Ereignisses vorzunehmen. Dazu wird ein spezielles Unterprogramm, die sog. Interrupt Service Routine (ISR), gestartet, nach deren Beendigung das ursprüngliche Programm an der unterbrochenen Stelle und im selben Zustand wie vor der Unterbrechung weitergeführt wird.

Beide Verfahren haben Vor- und Nachteile, die man je nach Einsatzzweck gegeneinander abwägen muß. Polling hat den Vorteil, daß bei geeigneter Programmierung und einer kleinen Anzahl zu überwachender Geräte die schnellst mögliche Reaktion auf ein externes Ereignis erfolgen kann (Abb. A.1.b). τ_{react} ist sehr klein, während τ_{cs1} und τ_{cs2} im Idealfall ganz verschwinden. Das ist aber verbunden mit den Nachteil, daß die CPU zwischen zwei Ereignissen keine anderen Aufgaben erledigen kann, ohne die Reaktionszeit negativ zu beeinflussen.

[1] Unterbrechung

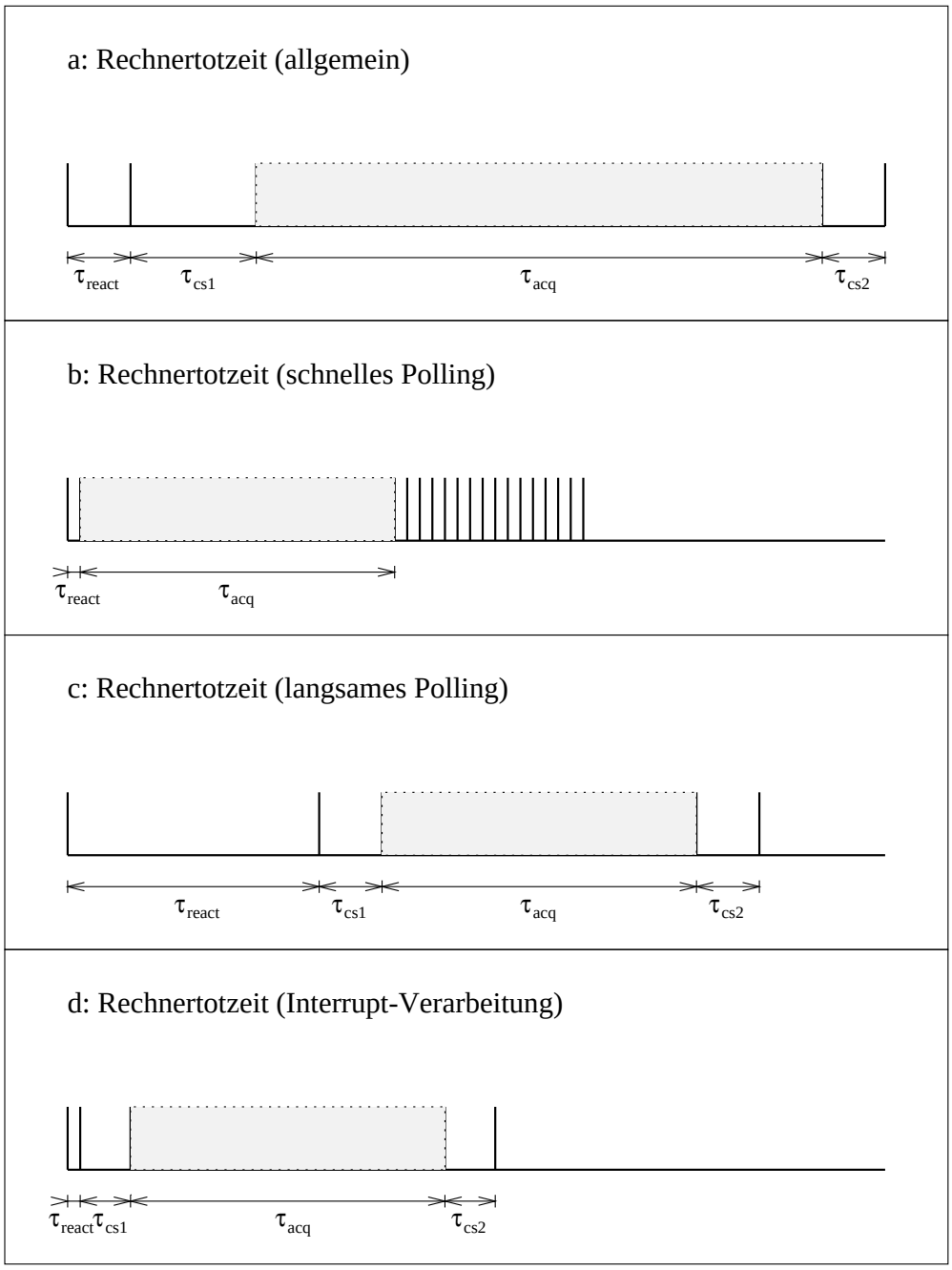

Abb. A.1: Schematische Darstellung der Rechnertotzeit

Größere Poll-Intervalle können zwar dieses Problem verbunden mit z. T. aufwendiger Programmierung beseitigen, vergrößern jedoch die Reaktionszeit und haben endliche Kontext-Umschaltzeiten zur Folge. (Abb. A.1.c).

Anders ist es dagegen bei der Interrupt-Verarbeitung (Abb. A.1.d). Hier entsteht zwar bei der Kontext-Umschaltung zwischen Hauptprogramm und ISR stets ein gewisser, prozessorspezifischer Overhead, doch ist dieser erst einmal unabhängig von der Anzahl der zu überwachenden Geräte und der Rate, mit der die Interrupts auftreten. Zugleich kann in den nicht für die Interrupt-Bearbeitung genutzten Zeiten in gewissen Grenzen die Bearbeitung beliebiger Programme erfolgen und so die CPU bestmöglich ausgenutzt werden.

A.3.3 Echtzeitsystem

Im Experiment müssen nun außer der Datenaufnahme noch eine ganze Reihe weiterer Aufgaben gleichzeitig von den beteiligten Rechnern bearbeitet werden. Daher reichte es nicht aus, das System ausschließlich auf hohe Datenerfassungsgeschwindigkeit zu optimieren. Möglichst schnelle Reaktion auf externe Ereignisse, d. h. kurze Reaktions- und Kontextumschaltzeiten sind für ein gut funktionierendes Meßsystem eine Grundvoraussetzung. Somit fielen für dessen Realisierung die typischen Anforderungen der Echtzeitdatenverarbeitung an.

Nach allgemeiner Definition ist ein Echtzeitsystem ein System zur simultanen Steuerung und Abwicklung von technischen Prozessen[2]. Hierbei besteht das System in der Regel aus Hard- und Software-Komponenten, und die zu bearbeitenden Prozesse werden durch eine Umgebung vorgegeben, mit der das System in dauerndem Informationsaustausch steht. Ein Echtzeitsystem ist ausgezeichnet durch die Eigenschaft, auf Anfragen oder Umweltänderungen innerhalb kurzer, definierter Zeiten reagieren zu können.

Bei der Experimentdatenverarbeitung treten nun in mehrfacher Hinsicht die Anforderungen eines Echtzeitsystems auf. Mit der eigentlichen Datenaufnahme sind extrem harte Echtzeitanforderungen verbunden. Hier sind Reaktionszeiten im Mikrosekunden-Bereich gefordert. Auch die Abwicklung des Informationsaustauschs mit der Umwelt muß mit höchster Geschwindigkeit vor sich gehen. Im Zusammenhang mit der Steuerung der Meßapparatur, wo Überwachungs- und Regelungsaufgaben anfallen, die eher mit klassischen Echtzeitanwendungen zu vergleichen sind, sind die Anforderungen nicht ganz so hoch. Aber auch hier muß das System je nach Anwendung in Zeiten zwischen einer Millisekunde bis hin zu etwa einer Sekunde auf externe Ereignisse definitiv reagieren.

Um sowohl die harten als auch die weichen Echtzeitanforderungen zu erfüllen, bot die verfügbare Rechner-Hardware eine sehr leistungsfähige Grundlage. Doch nur mit geeigneter Software war es möglich, diese auch weitestgehend zu nutzen.

[2] An dieser Stelle ein Hinweis auf die Doppeldeutigkeit des Begriffs Prozeß: Während er in diesem Zusammenhang technische Vorgänge und Abläufe bezeichnet, versteht man im Bereich der Datenverarbeitung unter diesem Begriff in der Regel ein sich in Abarbeitung befindliches Programm auf einem Rechner. In dieser Bedeutung wird er an späterer Stelle auch in dieser Arbeit gebraucht

A.4 Betriebssystem

Betriebssystem[3] ist die zusammenfassende Bezeichnung für Software, die die Ressourcen eines Computers kontrolliert und die Basis für die Entwicklung von Anwenderprogrammen ist. Eine grundlegende Funktion des Betriebssystems besteht darin, den Anwender bei der Benutzung des in der Regel sehr komplexen Computersystems und dessen Peripherie zu unterstützen. Es nimmt ihm wesentliche Aufgaben bei der direkten Ansteuerung der Hardware ab und stellt sich dem Anwender als eine Schnittstelle oder virtuelle Maschine dar, die einfach zu verstehen und zu programmieren ist und für deren Betrieb keine Detailkenntnisse über Funktionsweise und Programmierung der Hardware notwendig sind.

Der Betriebssystemkern deckt dabei im wesentlichen vier wichtige Aufgabenbereiche ab:

Die **Ein- und Ausgabe** wird nicht direkt von den Anwenderprogrammen vorgenommen, sondern über entsprechende hardwareunabhängige Systemaufrufe vom Betriebssystem. Es verwaltet die I/O-Geräte selbständig und sorgt automatisch für das korrekte Ansprechen auf ihre Hardware.

Es existieren Mechanismen zur **Speicherverwaltung**. Das Betriebssystem kontrolliert den auf dem Computersystem zur Verfügung stehenden Arbeitsspeicher und teilt ihn Benutzerprogrammen bei Bedarf zu. Bei Verwendung von sog. virtuellem Speicher sorgt es für die Auslagerung von Arbeits- auf Massenspeicher (sog. Paging und Swapping) und nimmt dessen Verwaltung vor. Nicht zuletzt tragen Speicherschutzmechanismen zu einem fehlerfreien Betrieb bei.

Ein **Dateisystem** sorgt für die Verwaltung bei externer Informationsspeicherung, wenn Daten auf Massenspeichern abgelegt werden, und stellt dem Anwender ein einheitliches, geräteunabhängiges Dateikonzept zur Verfügung. Es regelt den Zugriff auf Dateien und sorgt für den korrekten Datentransfer. In der Regel vergibt und überprüft es Zugriffsberechtigungen auf einzelne Dateien oder ganze Dateigruppen.

Zu dem Zweck, auf einem Rechner zur selben Zeit mehrere Programme quasi parallel ablaufen zu lassen, sorgt schließlich die **Prozeßverwaltung**. Sie nimmt die Verwaltung der CPU und ihre Zuteilung auf die verschiedenen Programme vor. Unter seiner Kontrolle werden Prozesse aktiviert und terminiert oder während ihres Ablaufs gesteuert. Im Gegensatz zu dem auch oben benutzten Sinn eines technischen Vorgangs versteht man in diesem Zusammenhang unter einem Prozeß ein in Ausführung befindliches Programm. Ein Prozeß besteht aus dem ausführbaren Programm, den Programmdaten und dem aktuellen Programmkontext, der sich aus Programmstack, Befehlszähler, Stackzeiger und anderen Speicherinformationen, die zum Lauf benötigt werden, zusammensetzt. Je nach Anforderungen gibt es unterschiedliche Verfahren, wie im Mehrprogrammbetrieb (Multitasking) die CPU von Programm zu Programm umgeschaltet wird (Scheduling). Zuteilungsalgorithmen entscheiden, wann ein Prozeß unterbrochen und ein anderer getartet wird. Das Betriebssystem verwaltet hierzu Prozeßzustände

[3] englisch Operating System

und Prozeßhierarchien und bietet oftmals Mechanismen zur Interprozeßkommunikation an.

Zur Erfüllung dieser Aufgaben arbeitet der Betriebssystemkern in aller Regel in einem speziellen, privilegierten CPU-Modus, in dem er gegen Fehleingriffe der Benutzer hardwaremäßig geschützt ist. Anwenderprogramme können seine Dienste über spezielle Mechanismen, sog. Systemaufrufe, nutzen.

Der Betriebssystemkern wird ergänzt durch weitere Systemprogramme wie Kommandointerpreter (Shell), Compiler, Editoren oder ähnliche anwendungsunabhängige Programme. Im Gegensatz zum Systemkern arbeiten diese Programme meist im selben Prozessor-Modus wie herkömmliche Anwenderprogramme.

Da das Betriebssystem an jeder Operation einer Rechenanlage in irgendeiner Weise beteiligt ist, hängt die Leistungsfähigkeit der Anlage ganz entscheidend von einer effizienten Implementierung der Betriebssystemkomponenten ab. Für die verschiedenen Anwendugsbereiche existieren eine ganze Reihe unterschiedlichster Betriebssystemvarianten. Die Bandbreite reicht von einfachen Singletasking-Systemen über Multitasking/Multiuser-Systeme bis hin zu leistungsfähigen Multiprozessor-Betriebssystemen. Von besonderer Bedeutung im Zusammenhang mit der Datenerfassung sind speziell auf die Anforderungen der Prozeßdatenverarbeitung optimierte Echtzeit-Betriebssysteme.

Ein Betriebssystem für einen bestimmten Computer muß auf dessen Hardware bestmöglich angepaßt sein. Da umgekehrt ein komplexer Rechner ohne unterstützende Systemsoftware nur schwierig zu nutzen ist, bieten in der Regel die Computer-Hersteller Betriebssysteme zusammen mit der Hardware oder als nachträgliche Ergänzung an. Zum Teil ist Systemsoftware auch von Dritten erhältlich. Ähnlich wie bei anderer kommerzieller Software fallen üblicherweise auch bei der Systemsoftware Kosten an, die bei der Konzeption des Systems zu berücksichtigen sind. Die Nutzung der Software wird dann durch spezielle Lizenzbestimmung geregelt.

A.5 Das Betriebssystem OS-9

OS-9/68000 ist ein Echtzeitbetriebssystem für die MC 68000-Prozessorfamilie der Firma Motorola, d. h. es unterstützt Rechner auf der Basis des MC 68000 bis hin zum MC 68040 und gewährleistet so eine einheitliche Software-Umgebung in einem weiten Leistungsbereich. Es ist ein Multitasking/Multiuser-System und damit nicht nur in der Lage, mehrere Programme zur gleichen Zeit zu bearbeiten, sondern es besitzt auch eine Benutzerverwaltung, die mehreren Personen zur gleichen Zeit Zugang zum System bietet. Einfache Mechanismen sorgen dabei für gegenseitigen Zugriffsschutz.

OS-9 ist modular aufgebaut. Der Betriebssystemkern besteht aus einzelnen, auch noch nachträglich ladbaren und je nach Hard- und Software-Anforderungen kombinierbaren Modulen. Sie gehören wie etwa Filemanager oder Kernel zum rechnerunabhängigen Basissystem, sorgen in Form von konfigurierbaren Gerätetreibern für das korrekte Ansprechen der Peripherie über standardisierte I/O-

Schnittstellen oder können für Spezialaufgaben auch vom Benutzer selbst implementiert werden. Das Betriebssystem läßt sich daher einfach an unterschiedliche Anforderungen anpassen.

Für Zwecke der Software-Entwicklung und komplexe Aufgabenstellungen besonders wichtig ist die Unterstützung von Festplatten mit einem hierarchischen Filesystem durch das Betriebssystem. OS-9 kann aber ebenso vollständig plattenlos betrieben werden, indem es mit Software arbeitet, die entweder in ROMs[4] abgelegt ist oder von außen über VMEbus oder Netzwerkschnittstellen ins RAM[5] geladen werden kann.

OS-9 besitzt eine einfache Speicherverwaltung, die direkten Zugriff auf lokale oder über den VMEbus zugängliche Hardware zuläßt. Es besitzt unterschiedliche Mechanismen zur Interprozeßkommunikation und über zusätzlich installierbare Software auch Netzwerkzugang, bietet jedoch direkt keine Unterstützung zur Realisierung von Multiprozessorsystemen.

Sowohl System-Schnittstellen als auch Dienstprogramme von OS-9 sind stark an das Betriebssystem UNIX angelehnt, bieten jedoch keine vollständige Kompatibilität. Zur Software-Entwicklung stehen neben Editoren, Assembler und Debugger unter OS-9 auch Compiler für Hochsprachen wie C oder Fortran zur Verfügung.

A.6 Das Betriebssystem UNIX

UNIX ist ein Multitasking/Multiuser-System, dessen Entwicklung 1969 in den Bell-Laboratorien von AT&T auf der Basis von Minicomputern der PDP-Serie von DEC begann. Seitdem wurde es ständig verbessert und weiterentwickelt und auf eine Vielzahl anderer Rechnersysteme portiert. Mit der Zeit fand es eine weite Verbreitung im wissenschaftlichen und dann auch im kommerziellen Bereich. Es wird inzwischen von einer großen Zahl von Computer-Herstellern und Software-Firmen unterstützt.

UNIX ist ein leistungsfähiges, trotz seines Alters modernes und gut strukturiertes Betriebssystem. Es ist gekennzeichnet durch Einfachheit, Modularität und Flexibilität. Es besteht aus einem Systemkern, der je nach Systemvariante mehr oder weniger monolithisch aufgebaut ist und in weiten Grenzen konfigurierbar ist, und einer Vielzahl von Dienstprogrammen, die im Gegensatz zu vielen anderen Betriebssystemen auf normaler Anwenderebene laufen.

Die Benutzerschnittstelle — standardmäßig realisiert als Kommandointerpreter (Shell) — ist einfach und bietet trotzdem alle Dienste an, die der Anwender benötigt. Das System verfügt über Möglichkeiten, komplexe Programme aus vielen einfachen zusammenzufügen. Es stellt seinen Benutzern ein breites Spektrum von Software-Werkzeugen zur Verfügung. Diese Werkzeuge können so einfach kombiniert und manipuliert werden, um eine Vielfalt von Aufgaben

[4] Read Only Memory; in der Regel als EPROM (Erasable Promable ROM) realisiert
[5] Random Access Memory; i. w. der Arbeitsspeicher des Rechners

zu lösen, daß die Benutzer des Systems anspruchsvolle Arbeiten durchführen können, ohne überhaupt Programme zu schreiben.

Der Standard-UNIX-Werkzeugkasten enthält Programme jeder Komplexität, vom einfachen Dienstprogramm, welches die Anzahl der Worte in einer Datei zählt, bis hin zu einem Übersetzer-Generator, der aus einer formalen Grammatik einen Erkenner für die entsprechende Sprache erzeugt. Die Werkzeuge, die Dateihaltung, das Ein- und Ausgabesystem, die sehr leistungsfähige Kommandosprache (mit Recht eine Programmiersprache) und die Kombination dieser Komponenten sorgt in UNIX für eine angenehme Benutzung und für eine sehr produktive Umgebung zur Software-Entwicklung [Bana84].

Das System verfügt in der Regel standardmäßig über Compiler, Assembler, Linker und Debugger und einer Vielzahl von Editoren. In vielen Fällen kann diese Entwicklungsumgebung ergänzt oder ersetzt werden durch ein breites Angebot an kommerzieller Software. Gleichzeitig ist UNIX eine Hauptplattform für die Entwicklung von Public Domain Software, so daß auch aus diesem Bereich eine breite Palette an leistungsfähiger und zugleich kostenloser Software zur Verfügung steht.

Das Betriebssystem besitzt ein hierarchisch aufgebautes Dateisystem, das leichte Pflege und effizientes Implementieren gestattet. Es verwendet ein einheitliches Dateiformat, den Bytestream, wodurch das Schreiben von Anwenderprogrammen wesentlich vereinfacht wird. Auf der Basis diese Dateikonzepts ist eine einfache und einheitliche Schnittstelle zu allen peripheren Geräten (Drucker, Terminal, Magnetbandgerät) aber auch für Zwecke der Interprozeßkommunikation (Pipes, Sockets, Streams) realisiert.

UNIX besitzt eine virtuelle Speicherverwaltung, die das teilweise oder vollständige Auslagern von Programmcode und -daten aus dem Arbeitsspeicher auf Massenspeicher kontrolliert und somit die gleichzeitige Ausführung vieler, auch großer Programme erlaubt, deren Speicherbedarf weit über den verfügbaren Arbeitsspeicher hinausgeht. Es bietet einen Speicherschutz, der den Zugriff auf den Arbeitsspeicher von Betriebssystem und Prozessen durch andere Prozesse unterbindet, stellt aber gleichzeitig auch Mechanismen zur Verfügung, die es Anwenderprogrammen erlauben, über gemeinsame Speicherbereiche (sog. Shared Memory) zu kommunizieren. Auch die für die Prozeßdatenverarbeitung wichtige Möglichkeit des direkten Hardware-Zugriffs von Anwederebene aus ist gegeben.

Zur rechnerübergreifenden Kommunikation bietet UNIX ebenfalls Mechanismen an. Von besonderer Bedeutung ist hier das Netzwerk-Protokoll TCP/IP. Es war ursprünglich nur in der von der Universität von Berkeley weiterentwickelten UNIX-Version (BSD[6]) verfügbar, ist inzwischen aber zu einem Standard-Bestandteil von UNIX geworden [Leff89]. Es bietet über Netzwerke wie Ethernet oder auch das herkömmliche Telefonnetz Zugang zu dem weltweit arbeitenden Internet und ist die Grundlage für eine Vielzahl von Netzwerkdiensten von einfachem Filetransfer (ftp) über Remote-Login (telnet) bis hin zu verteilten Filesy-

[6] Berkeley Software Distribution

stemen (NFS[7], DFS[8]). Auch existieren auf dieser Basis eine Reihe von Ansätzen zur Realisierung von Multiprozessor-Systemen.

UNIX ist fast ausschließlich in einer Hochsprache, der maschinennahen Programmiersprache C [Kern83], geschrieben. Im Gegensatz zu herkömmlichen Betriebssystemen ist hier nur ein minimaler Anteil des Systemkerns in Assembler — also in der Maschinensprache des Rechners — implementiert. Da die Quellprogramme insbesondere für Universitäten in einem gewissen Rahmen relativ gut verfügbar sind, ist es damit verhältnismäßig einfach, das Betriebssystem zu erweitern und existierende Betriebssystemteile zu modifizieren und speziellen Bedürfnissen bzw. Hardware-Anforderungen anzupassen. Auch die Diagnose und Beseitigung von auftretenden Fehlern wird damit erleichtert. Oftmals ist hierzu ein Eingriff in den eigentlichen Systemkern gar nicht notwendig; in vielen Fällen reicht es aus, das eine oder andere Dienstprogramm abzuändern, zu ersetzen oder durch benutzerspezifische Programme zu ergänzen.

Die Verwendung der Hochsprache C bei der UNIX-Implementation ist eine sehr gute Grundlage dafür, das System auf andere Rechner zu übertragen, was dann auch wesentlich zu seiner weiten Verbreitung beigetragen hat. Inzwischen ist es wie kein anderes Betriebssystem auf einer großen und ständig wachsenden Vielfalt von Rechnern verfügbar. Es läuft auf einem breiten Spektrum von Rechnern vom Mikroprozessor bis zum Großrechner und auf Produkten verschiedenster Hersteller.

Viele Elemente von UNIX sind im Laufe der Zeit auch von anderen Betriebssystemen übernommen worden. Insbesondere wurden ausgehend von der UNIX-Implementation viele betriebssystemübergreifende nationale und internationale Software-Standards entwickelt. Dazu gehören z. B. POSIX 1003 basierend auf der Definition der UNIX-Systemschnittstellen oder die Sprachdefinition von C, die sich in einem ANSI-Standard[9] niedergeschlagen hat. Auch die BSD-Implementation von TCP/IP ist Grundlage für viele Standards im Internet-Bereich.

A.7 Die Programmiersprache C

C ist eine Hochsprache, die strukturiertes und modulares Programmieren unterstützt, aber zusätzlich auch die Möglichkeit der maschinennahen Programmierung bietet. Dazu verfügt die Sprache analog zu den entsprechenden Maschinenbefehlen über arithmetische und logische Operationen zur Manipulation einzelner Bits, elementarer Datenworte und Adressen.

C ist eine Programmiersprache für allgemeine Anwendungen, entstand jedoch im Rahmen der UNIX-Entwicklung mit dem Ziel, vieles im Bereich der Systemprogrammierung, was bis dahin nur in der Maschinensprache des jewei-

[7] Network File System

[8] Distributed File System; Bestandteil von ... DMS

[9] American National Standards Institute; nationaler Normenausschuß der USA, vergleichbar mit dem Deutschen Institut für Normung (DIN)

ligen Rechners programmiert werden konnte, nun auch in einer Hochsprache formulieren zu können. Sowohl der weit überwiegende Teil des Systemkerns von UNIX als auch die meisten Programme, die hier eingesetzt werden, sind in C geschrieben. Damit war es schließlich möglich, UNIX sehr einfach auf unterschiedliche Maschinenarchitekturen zu portieren — eine wesentliche Ursache für die weite Verbreitung diese Betriebssystems. Die Sprache selbst ist jedoch nicht von einem bestimmten Betriebssystem oder einer Maschine abhängig.

Die Programmiersprache C besitzt eine einfache und übersichtliche Struktur. Sie unterstützt zwar die wesentlichen elementaren und daraus ableitbare Datentypen und enthält all die fundamentalen Kontrollstrukturen wie sie auch in anderen Programmiersprachen verfügbar sind, um eine wohlstrukturierte Programmierung zu ermöglichen, doch verfügt sie über keine Operationen, mit denen man zusammengesetzte Objekte wie Zeichenketten, Mengen, Listen oder Vektoren direkt bearbeiten kann. Die Sprache selbst definiert keine Speicherverwaltung außer statischer Definitionen und der Stack-Verwaltung, die für lokale Variablen in Funktionen besteht. Schließlich hat C selbst keine Einrichtungen für Ein- und Ausgabe; es gibt keine Read- oder Write-Anweisungen und keine eingebauten Techniken für Dateizugriff.

Für diese abstrakten Mechanismen existieren in aller Regel aber explizit aufzurufende, standardisierte Funktionen, die in Form von Bibliotheken zur Verfügung stehen. Sie gewährleisten einen kompatiblen Zugriff auf das Betriebssystem, sorgen für formatierte Ein- und Ausgabe, die Speicherverwaltung und vieles andere mehr. Die Programmiersprache ist zusammen mit der zentralen Bibliothek in einem ANSI-Standard [Kern90] definiert, der die Basis für die meisten Compiler darstellt.

Eine Besonderheit von C ist die Verwendung eines Preprozessors, der Makros im Programmtext ersetzt, Quelldateien kombiniert und bedingte Übersetzung ermöglicht.

Obgleich C den Fähigkeiten vieler Rechner angepaßt ist, ist die Sprache doch unabhängig von einer bestimmten Maschinenarchitektur. Mit einer gewissen Sorgfalt ist es leicht, portable Programme zu schreiben, d. h. Programme, die ohne Änderung auf einer Reihe von Computern ausgeführt werden können. Der Standard stellt Portabilitätsprobleme klar heraus und schreibt einen Satz Konstanten vor, die die Maschine charakterisieren, auf der das Programm ausgeführt wird. Compiler für die Programmiersprache C sind schon seit langer Zeit auf allen wichtigen Rechnern und Betriebssystemen verfügbar.

A.8 Interprozeßkommunikation unter UNIX

Eine universelle Kommunikationsmöglichkeit bilden unter UNIX die sogenannten **Pipes** (Pipelines). Sie erlauben einen synchronisierten Datentransfer zwischen zwei Prozessen unter Ausnutzung der geräteunabhängigen Standard-Ein/Ausgabemechanismen. Sie basieren auf dem verallgemeinerten I/O-Konzept von UNIX und stellen virtuelle Ein/Ausgabegeräte mit FIFO-Funktionalität dar

mit denselben Schnittstellen wie bei herkömmlichen Geräten und Dateien. Damit unterscheidet sich der Datentransfer zwischen zwei Prozessen programmtechnisch nicht von dem auf Files oder mit einem Benutzer-Terminal und ist so sehr einfach zu implementieren und zu testen.

Pipes werden in vielen Bereichen des UNIX-Systems genutzt. Sie sind sehr hilfreich, um komplexe Anwendungen aus kleinen, übersichtlichen, voneinander unabhängigen und einfach zu handhabenden Bausteinen effizient zusammenzusetzen. Insbesondere die Shell bietet zur Nutzung dieser Vorzüge durch den Anwender weitreichende Möglichkeiten an.

Pipes sind als grundlegender Kommunikationsmechanismus effizient implementiert. Sie bieten standardmäßig eine sehr leistungsfähige Kommunikationsbasis im Bereich vieler Aufgabenstellungen. Nur in wenigen Fällen, wo extrem hohe Datenübertragungsraten benötigt werden bzw. nur ein extrem geringer I/O-Overhead erlaubt ist, muß auf andere Mechanismen zurückgegriffen werden.

Einen sehr schnellen Austausch auch großer Datenmengen zwischen zwei Programmen gewährleistet **Shared Memory**. Hier bietet das Betriebssystem die Möglichkeit, Bereiche im Arbeitsspeicher des Rechners zu definieren, auf die mehrere Anwenderprozesse gemeinsam Zugriff haben. Mit Programmiersprachen wie C, die die Verwendung von Zeigern erlauben, besitzen die Programme freien Zugriff auf diese Speicherbereiche und können ihn genauso wie ihren eigenen, programminternen Speicher nutzen. Damit wird in vielen Fällen der explizite Datentransfer unnötig, was natürlich die Effizienz der Kommunikation erhöht.

Shared Memory bietet im Gegensatz zu den Pipes keine Möglichkeiten zur Synchronisierung der Zugriffe von verschiedenen Seiten. Hierfür stehen jedoch unabhängig davon Mechanismen zur Erzeugung, Verwaltung und Nutzung von speziellen **Semaphoren** zur Verfügung. Auch UNIX-**Messages** können zu diesem Zweck genutzt werden. Sie bieten darüber hinaus noch die Möglichkeit, beschränkte Informationen in Form von Nachrichten auszutauschen.

Im Gegensatz zu Pipes sind zu Shared Memory, Semaphoren und Messages vergleichbare IPC-Mechanismen in der einen und anderen Form oftmals auch unter anderen Betriebssystemen verfügbar.

Ähnlich wie bei Pipes funktioniert auch die Kommunikation über **UNIX-Domain-Sockets**. Bis auf die Initialisierungsphase wird auch hier der Datenaustausch über Standard-I/O-Mechanismen durchgeführt. Analog zu den herkömmlichen Dateinamen besitzen sie wie die sog. Named Pipes durch spezielle Einträge im UNIX-Filesystem ebenfalls Namen, über die sie symbolisch ansprechbar sind. Im Gegensatz zu Pipes arbeitet die Socket-Kommunikation jedoch bidirektional und bietet eine Reihe nützlicher Konfigurationsmöglichkeiten.

UNIX-Domain-Sockets stellen einen Sonderfall allgemeiner **Sockets** dar, die nicht nur die Interprozeßkommunikation unterstützen, sondern mit identischen Mechanismen die Basis für eine rechnerübergreifende Kommunikation bilden.

A.9 Das OSI-Schichtenmodell

Im Jahre 1979 entwickelte die ISO[10] einen Vorschlag für ein Schichtenmodell (bekannt als OSI-Schichtenmodell[11]), das Richtlinien für die Übertragung von Daten in Netzwerken vorgibt. Das Schichtenmodell unterteilt den Dialog und den Datenaustausch zwischen den Rechnern in sieben Schichten oder Ebenen. Die Schnittstellen zwischen den Ebenen sind definiert. Innerhalb der einzelnen Schichten besteht eine gewisse Freiheit hinsichtlich der verwendeten Hard- und Software. Die Datenübermittlung nach dem OSI-Modell schafft eine Übereinstimmung zwischen der Sende- und der Empfangsstation, auch wenn völlig unterschiedliche Rechnersysteme verwendet werden.

Die Protokolle für die zur Kommunikation zwischen Prozessen einer Schicht erforderlichen Verfahren bilden eine Protokollhierarchie mit nach oben zunehmendem Leistungsumfang. Die auf allen Rechnern eines Rechnernetzes eingerichtete Protokollhierarchie bewirkt eine einheitliche Sichtweise und eine standardisierte Abwicklung aller Kommunikationsvorgänge. Im einzelnen haben die Schichten folgende Aufgaben:

1. **Übertragungsschicht** oder **Physikalische Schicht** (physical layer) ist für die physikalische Verbindungen zwischen Datenendeinrichtungen zuständig und leistet den Transport der Daten in Form von Bitströmen über ein geeignetes Medium. Dabei sorgt diese Schicht für die Umwandlung der bitcodierten Nachricht in die für das Übertragungsmedium geeignete Form und umgekehrt.

2. **Datensicherungsschicht** oder **Leitungsschicht** (data link layer) kontrolliert die Datenübertragung. Sie definiert das Format der Daten im Netz, d. h., wie die Bitströme zu handhabbaren Datenblöcken, den Paketen mit netzspezifischer Adreßinformation, Prüfsummen und Nutzdaten, zusammengesetzt werden und sorgt für deren Übertragung.

3. **Vermittlungsschicht** oder **Netzwerkschicht** (network layer) stellt Hilfsmittel bereit, um Daten zwischen Rechenanlagen austauschen zu können, auf denen Anwenderprozesse ablaufen. Sie ist der Datenübertragungsdienst, der die unterschiedlichen Eigenschaften von Datenverbindungen der 1. und 2. Schicht vor den darüberliegenden Schichten verbirgt. Die Netzwerkschicht sorgt für das sog. Routing, um Daten von einem zu einem anderen entfernten Netzwerk zu transportieren. Sie benutzt dazu im Gegensatz zur 2. Schicht eine hardwareunabhängige, logische Adreßierung.

4. **Transportschicht** (transport layer) bereitet Nachrichten für den Weitertransport vor. Sie steilt benutzerdefinierte Datenpuffer in sog. Datagramme mit der von der Netzwerkschicht vorgegebenen Größe auf bzw. setzt

[10] International Organisation for Standardisation; Dachorganisation von über 50 nationalen Normenausschüssen mit Sitz in Genf mit der Aufgabe, die von den einzelnen Ländern vorgeschlagenen Standardisierungen abzustimmen und zu vereinheitlichen

[11] Open System Interconnection

empfangene Datagramme wieder zusammen. Außerdem muß sie bei Bedarf Fehlerkorrekturmechanismen bereitstellen, um eine zuverlässige Zustellung der zu übertragenden Daten zu gewährleisten..

5. **Sitzungsschicht** (session layer) legt Aufbau und Dauer von Netzverbindungen fest.

6. **Darstellungsschicht** (presentation layer) definiert das Format zum Austausch von Daten über die Verbindungen der Sitzungsschicht. Sie dient der Umwandlung der maschinenoriertierten Sichtweise der Layer 1 bis 5 in eine problemorientierte.

7. **Anwendungsschicht** (application layer) stellt die direkte Schnittstelle zur eigentlichen Anwendung dar.

Die einzelnen Schichten bzw. ihre Protokolle können sowohl in Hardware als auch Software realisiert sein. Für die unterschiedlichsten Anwendungen im Kommunikationsbereich, die z. T. auch ganz verschiedene Anforderungen stellen, sind auch sehr unterschiedliche Realisierungen verfügbar. Die meisten lassen sich zwar ganz grob mit dem Schichtenmodell beschreiben, halten sich jedoch aufgrund ihrer historischen Entwicklung nicht exakt daran.

Für die physikalische Schicht, die in der Regel in Hardware ausgeführt ist, existiert eine breite Palette an Möglichkeiten von der einfachen seriellen Datenübertragung auf herkömmlichen Leitungen (V.24) über spezielle lokale Netzwerke (LAN[12]) bis hin zu höchst leistungsfähigen Übertragungsverfahren z. B. auf der Basis von Lichtleitern. In der Regel ist die Leitungsschicht und manchmal auch höhere Schichten in Form intelligenter Controller verfügbar. Die Umsetzung der darüberliegenden Schichten bleibt der Software vorbehalten.

Die meisten Rechner besitzen die Hardware bzw. bieten Anschlußmöglichkeiten dafür, um damit eine mehr oder weniger leistungsfähige Kommunikation durchzuführen, aber erst „anspruchsvollere“ Betriebssysteme liefern dafür die entsprechende Unterstützung. Die Systemsoftware sorgt dann in aller Regel für das korrekte Ansprechen der Hardware, muß aber auch wesentliche Teile der höheren Schichten abdecken und entsprechende Schnittstellen zur Verfügung stellen.

Die TCP/IP-Protokollfamilie, die standardmäßig die Basis der Netzwerk-Kommunikation bei UNIX ist, und inzwischen auch von vielen anderen Betriebssystemen unterstützt wird, läßt sich funktional ganz gut in das beschriebene Schichtenmodell einorden: das Internet-Protokoll (IP) entspricht im wesentlichen der Netzwerkschicht, während TCP[13] und das ebenfalls auf dieser Ebene verfügbare UDP[14] der Transportschicht zuzuordnen sind. Beide Schichten stehen dem Anwender über das bereits im Rahmen der Interprozeßkommunikation

[12] Local Area Network
[13] Transmission Control Protocol
[14] User Datagram Protocol

erwähnte Socket-Interface als Systemschnittstelle zur Verfügung. Die Sitzungsschicht wird z. B. durch PRC[15] repräsentiert, die ebenso wie die Darstellungsschicht, die z. B. in Protokollen wie XDR[16] realisiert ist, üblicherweise in Form von Unterprogrammbibliotheken verfügbar ist. Zur Anwendugsschicht gehören die Protokolle, wie sie in den Programmen `ftp` zum Filetransfer oder `telnet` zum Remote-Login benutzt werden.

A.10 Ethernet

Ethernet ist ein sehr weit verbreitetes Rechnernetz, das auf einer busartigen Struktur basiert. Es wurde Ende der 70er Jahre von der Firma Xerox entwickelt, und 1985 im wesentlichen unverändert als einer der ersten Standards nach dem OSI-Schichtenmodell festgeschrieben. Als Standard ISO 8802/3 bzw. IEEE 802.3[17] deckt es die physikalische Schicht und wesentliche Teile der Leitungsschicht ab.

Übertragungsmedium beim Ethernet ist ein 50-Ω-Koaxialkabel. Es verbindet alle Computer eines Netzwerks über entsprechende Interfaces, die die eigentliche Kommunikation abwickeln. Die Datenübertragungsrate im Ethernet beträgt 10 MBit/s. Ethernet arbeitet paketorientiert mit Paketen einer Länge zwischen 64 und 1514 Bytes.

Ethernet arbeitet nach dem CSMA/CD-Verfahren[18], d. h., die Interfaces überprüfen bei einer Sendeabsicht den Zustand des Netzes, um festzustellen, ob es bereits von einen anderen Teilnehmer benutzt wird, und warten gegebenenfalls mit dem Senden der Daten darauf, daß es wieder frei wird. Beim Senden auftretende Kollisionen werden erkannt und führen zur Wiederholung des Sendevorgangs nach einem Zeitintervall, dessen Länge durch einen Zufallsalgorithmus bestimmt wird.

Auf der Netzwerkschicht erlaubt Ethernet eine Vielzahl von unabhängigen Kommunikationsprotokollen. Die wichtigsten gehören zu den Protokollfamilien wie DECnet und TCP/IP, die auch wesentliche Teile der darüberliegenden Schichten abdecken (s. o.).

A.11 Byte-Orientierung

Die Byte-Orientierung von Daten ist eine wichtige Eigenschaft, in der sich trotz vieler Übereinstimmungen bei der Definition von elementaren Datenworten von Prozessoren die Rechner unterscheiden können. Sie beschreibt die Zuordnung der Wertigkeiten der einzelnen Bytes, aus denen sich ein Speicherwort zusammensetzt, zu ihren relativen Speicherpositionen innerhalb des Wortes. Ein 16-Bit-Wort besteht aus zwei Bytes, wobei eines den höherwertigen Anteil des Wortes

[15] Remote Procedure Calls
[16] eXternal Data Representation
[17] Institute of Electrical and Electronics Engineers in den USA
[18] Carrier Sense Multiple Access/Collision Detection

repräsentiert, das zweite den niederwertigen. Formal ausgedrückt gilt für ein Wort W und dessen höher- bzw. niederwertiges Byte H und L:

$$W = H \cdot 256 + L$$

Die beiden Bytes sind im Speicher hintereinander abgelegt. Von vornherein liegt nicht fest, welcher Speicherposition welche Wertigkeit zugeordnet ist. Dafür gibt es grundsätzliche zwei Möglichkeiten. Die erste Alternative, bei der das erste Byte die höhere Wertigkeit besitzt und das zweite das niederwertige ist, ist bekannt unter dem Namen „Big Endian“, die zweite wird „Little Endian“ genannt und beschreibt den Fall, daß das erste Byte die niedere Wertigkeit besitzt und das zweite die höhere.

Beide Möglichkeiten finden bei den einzelnen Prozessoren Verwendung. MC 68000- und Sparc-Prozessoren besitzen z. B. eine Big-Endian-Architektur, während VAX- oder Intel x86-Prozessoren typische Little-Endian-Maschinen sind. Die im Institut eingesetzten MIPS-Rechner unterstützen zwar prinzipiell beide Alternativen, werden jedoch unter dem eingesetzten Betriebssystem Ultrix nur im Little-Endian-Modus betrieben.

Wenn binäre Daten zwischen Rechnern ausgetauscht werden, macht eine unterschiedliche Byte-Orientierung eine entsprechende Anpassung notwendig. Das heißt, bei Worten, die mehr als ein Byte enthalten, müssen die Bytes entsprechend ihrer Wertigkeiten vertauscht werden, allgemein unter dem Begriff „Byte-Swapping“ bekannt. Der Einsatz eines verteilten, heterogenen Systems bei der Datenerfassung macht nun den ständigen Datenaustausch zwischen Rechnern unterschiedlicher Byte-Orientierung notwendig. Dabei kann der Datenaustausch nicht nur explizit über eine direkte Rechnerkommunikation, sondern auch implizit durch Bewegen von Datenträgern, auf denen die Daten abgespeichert sind, erfolgen.

Um eine korrekte Datenübertragung zu gewährleisten, müssen die Daten entweder immer einheitlich abgespeichert werden, oder es wird Information über die Byte-Orientierung der abgespeicherten und übertragenen Meßdaten benötigt. Eine einheitliche Abspeicherung — ein Verfahren, wie es bei dem unter TCP/IP arbeitenden Kommunikationsprotokoll XDR u. a. auch im Steuerungssystem der A1-Experimentierapparatur benutzt wird [Kram95] — hat zur Folge, daß auch bei der Datenübertragung zwischen zwei Rechnern mit identischer Byte-Orientierung stets ein Hin- und Rückwandlung erfolgen muß, obwohl es in diesem Fall gar nicht nötig wäre. Der damit verbundene Aufwand kann sich insbesondere bei der Datenaufnahme nachteilig bemerkbar machen. Diese Nachteile entfallen bei der zweiten Methode.

Anhang B

Beispiele zur Experimentbeschreibung

B.1 Einfaches Experiment

B.1.1 Physikalische Beschreibung: `simple.phys`

```
/* --- Verweise auf Geraete/Hardware-Beschreibungen aus Bibliothek --- */

extern Device   lecroy_2228;            /* ADC von LeCroy, Typ 2228    */
extern Device   lecroy_2249;            /* TDC von LeCroy, Typ 2249    */
extern Device   microbusy;              /* Event-FF KPH-Microbusy      */
extern Device   pattern_unit;           /* Pattern-Unit, Standard-Typ  */
extern Device   ces_a2;                 /* A2-Crate-Controller von CES */
extern Device   cbd_8210;               /* Branch-Controller CBD 8210  */
extern Device   e5;                     /* Eltec E5 VMEbus CPU         */

/* ------------------- Definition von Modulgruppen ------------------- */

Group   adc;                            /* logische Gruppe der ADCs    */
Group   tdc;                            /* logische Gruppe der TDCs    */
Group   eventff;                        /* Verriegelungselektronik     */
Group   pattern;                        /* logische Gruppe Hit-Pattern */
Group   slave;                          /* Slave-Rechner zur Datenerf. */

/* ------------------------- Hardware-Setup -------------------------- */
/* ----------------- CAMAC-Crate mit Readout-Hardware ---------------- */

Crate crate1  = {
        MODULE(lecroy_2228, adc),       /* ADC in Slot 1               */
        EMPTY,                          /* Slot 2 leer                 */
        EMPTY,                          /* Slot 3 leer                 */
        EMPTY,                          /* Slot 4 leer                 */
        MODULE(lecroy_2249, tdc),       /* TDC in Slot 5               */
        EMPTY,                          /* Slot 6 leer                 */
        EMPTY,                          /* Slot 7 leer                 */
```

```
        EMPTY,                          /* Slot 8 leer                */
        EMPTY,                          /* Slot 9 leer                */
        EMPTY,                          /* Slot 10 leer               */
        EMPTY,                          /* Slot 11 leer               */
        MODULE(microbusy, eventff)      /* Eventff in Slot 12         */
        EMPTY,                          /* Slot 13 leer               */
        MODULE(pattern_unit, pattern)   /* Pattern Unit in Slot 14    */
};

/* ------------------------- CAMAC-Branch -------------------------- */
Branch branch = {
        CRATE(ces_a2, crate1)           /* obiges CAMAC-Crate m. A2-CC */
};

/* ------------------------- CAMAC-System -------------------------- */
Camac camac = {
        BRANCH(cbd_8210, branch)        /* obiger Branch mit CBD 8210  */
};

/* --------------- An Datenaufnahme beteiligte Rechner --------------- */
Computer computer = {
        FRONTEND(e5, slave)             /* E5 als Slave-Prozessor      */
};
```

B.1.2 Logische Beschreibung: `simple.log`

```
struct simple {                         /* Definition der log. Konfig. */
        struct {                        /* Definition eines Detektors  */
                short energy;           /* Energieinformation (ADC)    */
                short time;             /* Zeitinformation (TDC)       */
        } naj[8];                       /* acht NaJ-Detektoren         */
        short pattern;                  /* Hit-Pattern                 */
} simple;                               /* symbolischer Experimentname */
```

B.1.3 Physikalisch-logische Zuordnung: `simple.att`

```
simple.naj[0, 3].energy = adc[0, 3];          /* Energie-Info. stammt aus */
simple.naj[4].energy    = adc[8];             /* ADC-Modul(en), dessen 4. */
simple.naj[5, 7].energy = adc[5,7];           /* Kanal defekt ist.        */
simple.naj[0, 7].time   = tdc[0, 7];          /* Zeiten werden von TDC-   */
                                              /* Modul geliefert          */
simple.pattern          = pattern[0];
```

B.1.4 Bearbeitungs-Vorschriften: `simple.rdt`

```
static int      events;

Data *
rdtStart(Data *buf, size_t n)
{
        events = 0;

        return (buf);
}

Data *
rdtMain(Data *buf, size_t n)
{
        int     i;

        lread(simple.pattern);
        for (i = 0; i < 8; ++i)
                if (simple.pattern & (1 << i))
                        lread(simple.naj[i]);
        buf = pack(simple, &simple, buf, 0, events);

        return (buf);
}

Data *
rdtStatus(Data *buf, size_t n)
{
        sprintf(buf, "Events: %d\n", events);

        return (buf);
}
```

B.2 Die Drei-Spektrometer-Anlage

B.2.1 Spektrometer A

B.2.1.1 Physikalische Beschreibung: `a.phys`

```
extern Device smi_1821, lecroy_1882n, lecroy_4434_24, lecroy_2228a;
extern Device lecroy_4299, pattern_unit, scaler_kph, inreg;
extern Device sync_master, microbusy, ces_a2, cbd_8210, e5, sysinfo;

Group   trigger_adc, trigger_tdc, pm_scaler_dE, pm_scaler_ToF;
Group   cer_scaler, various_scaler, ps_scaler, paddle_scaler, info_pattern;
Group   position_pattern, wire_tdc;
Group   eventff, slave;
Group   sync_pattern, systime;
```

```
Crate detector_crate_1 =
{
        EMPTY,                  /* LeCroy 4413 discriminator dE_l     01 */
        EMPTY,                  /* LeCroy 4413 discriminator dE_r     02 */
        EMPTY,                  /* LeCroy 4516 logic coinc de_l/de_r  03 */
        MODULE(lecroy_4434_24, pm_scaler_dE),   /* PMs dE             04 */
        EMPTY,                  /* LeCroy 4413 discriminator tof_l    05 */
        EMPTY,                  /* LeCroy 4413 discriminator tof_r    06 */
        EMPTY,                  /* LeCroy 4516 logic coin tof_l/tof_r 07 */
        MODULE(lecroy_4434_24, pm_scaler_ToF),  /* PMs tof            08 */
        MODULE(lecroy_4434_24, paddle_scaler),  /* paddles            09 */
        EMPTY,                  /* LeCroy 4413 discrim. cherenkov     10 */
        MODULE(lecroy_4434_24, cer_scaler),     /* cherenkov + other  11 */
        EMPTY,                  /* LeCroy 4508 PLU                    12 */
        MODULE(sync_master, sync_pattern),      /* KPH sync module    11 */
        MODULE(lecroy_2228a, trigger_tdc),      /* LeCroy 2228A TDC   14 */
        EMPTY,                  /* VUCAM                              15 */
        MODULE(scaler_kph, ps_scaler),          /* prescaled events   16 */
        MODULE(scaler_kph, various_scaler),     /* various            17 */
        MODULE(microbusy, eventff),             /* microbusy          18 */
        MODULE(pattern_unit, info_pattern),     /* KPH Pattern Unit   19 */
        EMPTY,                  /* LeCroy 2132 HV-Interface           20 */
        MODULE(lecroy_4299, wire_tdc),          /* TDC Interface      21 */
        MODULE(inreg, position_pattern),        /* Wedel Input        22 */
        EMPTY                   /*                                    23 */
};

Crate detector_crate_2 =
{
        EMPTY,                  /* slot 1 !!!!!                          */
        EMPTY,
        EMPTY,
        EMPTY,                  /* slot 4 + 5: LeCroy 1821 SMI           */
        EMPTY,                  /*           : LeCroy 1821 SMI           */
        EMPTY,                  /* slot 6                                */
        EMPTY,
        MODULE(lecroy_1882n, trigger_adc),
        EMPTY,
        EMPTY,                  /* slot 10                               */
        EMPTY,
        EMPTY,
        EMPTY,
        EMPTY,
        EMPTY,                  /* slot 15                               */
        EMPTY,
        EMPTY,
        EMPTY,
        EMPTY,
        EMPTY
};

Branch branch =
{
        CRATE(ces_a2, detector_crate_1),
        CRATE(ces_a2, detector_crate_3),
        EMPTY
};
```

```
Camac camac =
{
        BRANCH(cbd_8210, branch),    /* Branch No. 1                        */
        EMPTY
};

Fastbus fastbus =
{
        BRANCH(smi_1821, detector_crate_2), /* Segment No. 8 (Hardware)     */
        EMPTY
};

Computer computer =
{
        FRONTEND(e5, slave)
};
```

B.2.1.2 Logische Beschreibung: a.log

```
#include "spectrometer.h"
#include "statistics.h"

struct dataA {
        struct runInfo   run;
        struct beamInfo  beam;
        struct detA      det;
        struct eventInfo coinc;
};

struct DataA a;
```

B.2.1.3 Physikalisch-logische Zuordnung: a.att

```
/********************************* VDC *********************************/

a.det.vdc.x1[0, 399]                      = wire_tdc[0, 399];
a.det.vdc.s1[0, 319]                      = wire_tdc[416, 735];
a.det.vdc.x2[0, 415]                      = wire_tdc[736, 1151];
a.det.vdc.s2[0, 319]                      = wire_tdc[1152, 1471];
a.det.vdc.s2[320, 335]                    = wire_tdc[400, 415];

/***************************** scintillator ****************************/

a.det.trigger.scint.de.pad[0, 14].right.energy  = trigger_adc[ 1, 15];
a.det.trigger.scint.de.pad[0, 14].left.energy   = trigger_adc[20, 34];
a.det.trigger.scint.tof.pad[0, 14].right.energy = trigger_adc[40, 54];
a.det.trigger.scint.tof.pad[0, 14].left.energy  = trigger_adc[60, 74];

a.det.trigger.scint.de.pad[0, 14].right.hits    = pm_scaler_dE[16, 30];
a.det.trigger.scint.de.pad[0, 14].left.hits     = pm_scaler_dE[0, 14];
a.det.trigger.scint.tof.pad[0, 14].right.hits   = pm_scaler_ToF[16, 30];
a.det.trigger.scint.tof.pad[0, 14].left.hits    = pm_scaler_ToF[0, 14];
```

```
a.det.trigger.scint.de.pad[0, 14].hits          = paddle_scaler[0, 14];
a.det.trigger.scint.tof.pad[0, 14].hits         = paddle_scaler[16, 30];

a.det.trigger.scint.de_tof_time                 = trigger_tdc[0];

a.det.trigger.scint.ps_hits                     = ps_scaler[8];

/******************************** cerenkov *******************************/

a.det.trigger.cerenkov.mirror[0, 11].energy     = trigger_adc[80, 91];
a.det.trigger.cerenkov.mirror[0, 11].hits       = cer_scaler[16, 27];
a.det.trigger.cerenkov.ps_hits                  = ps_scaler[2];

/********************************* top ***********************************/

a.det.trigger.top.energy                        = trigger_adc[92];
a.det.trigger.top.ps_hits                       = ps_scaler[9];


/************** trigger ( scintillator & cerenkov & top ) ***************/

a.det.trigger.scint_ch_time                     = trigger_tdc[7];
a.det.trigger.scint_top_time                    = trigger_tdc[5];
a.det.trigger.info                              = info_pattern[0];

/***************************** general info ******************************/

a.det.sync_info                                 = sync_pattern[0];

/***************************** run statistics ****************************/

a.run.runtime                                   = various_scaler[11];
a.run.realtime                                  = various_scaler[10];
a.run.ps_runtime                                = ps_scaler[11];
a.run.ps_realtime                               = ps_scaler[10];
a.run.ps_interrupts                             = ps_scaler[5];

/************************* conincidence info *************************/

a.coinc.ab.time[0]                              = trigger_tdc[3];
a.coinc.ab.time[1]                              = trigger_tdc[4];
a.coinc.ab.time[3]                              = dummy_group[1];
a.coinc.ca.time[0]                              = trigger_tdc[1];
a.coinc.ca.time[1]                              = trigger_tdc[2];
a.coinc.ca.time[2]                              = trigger_tdc[6];
a.coinc.bc.time[0]                              = dummy_group[2];
a.coinc.bc.time[1]                              = dummy_group[3];
a.coinc.bc.time[2]                              = dummy_group[4];

a.coinc.ab.dead1                                = cer_scaler[0];
a.coinc.bc.dead1                                = cer_scaler[1];
a.coinc.ca.dead1                                = cer_scaler[2];
a.coinc.ab.dead1                                = cer_scaler[3];
a.coinc.ca.dead1                                = cer_scaler[4];
a.coinc.bc.dead1                                = cer_scaler[5];
a.coinc.abc_dead                                = cer_scaler[6];
```

```
a.coinc.abc_coinc                                = ps_scaler[12];
a.coinc.ab.coinc                                 = ps_scaler[3];
a.coinc.ca.coinc                                 = ps_scaler[13];
a.coinc.bc.coinc                                 = ps_scaler[14];

/***************************** beam info *****************************/

a.beam.photo                                  = various_scaler[7];
a.beam.foerster                               = various_scaler[6];
a.beam.faraday                                = various_scaler[0];
a.beam.ps_photo                               = ps_scaler[7];
a.beam.ps_foerster                            = ps_scaler[6];
a.beam.ps_faraday                             = ps_scaler[0];
```

B.2.1.4 Bearbeitungs-Vorschriften: `a.rdt`

```
struct RunStatistics  rs;

void    read_speca(void);
void    read_scaler(void);
void    read_various_scaler(void);

/*****************************************************************************/

Data *
rdtMain(register Data *buf, size_t n)
{
        rs.total = total;

        pread(sync_pattern);
        if (a.det.sync_info == 0)
                warning("sync_info == 0");

        read_various_scaler();
        read_speca();

#ifdef SINGLE
        buf = pack(a, &a, buf, 1, total & 0x1fff);
#else
        buf = pack(a, &a, buf, (a.det.sync_info >> 13) & 0x7,
          a.det.sync_info & 0x1fff);
#endif

        return (buf);
}

/*****************************************************************************/

Data *
rdtStart(Data *buf, size_t n)
{
        clear(various_scaler);
        clear(ps_scaler);
```

```
        clear(pm_scaler_dE);
        clear(pm_scaler_ToF);
        clear(paddle_scaler);
        clear(cer_scaler);
        clear(sync_pattern);

        memset(&rs, 0, sizeof (rs));

        return (buf);
}

/*****************************************************************************/

Data *
rdtStop(Data *buf, size_t n)
{
        read_various_scaler();
        read_scaler();
        buf = pack(a, &a, buf, 0, 0);

        return (buf);
}

/*****************************************************************************/

void
read_speca()
{
        if (!(total & 0x1ff))          /* total % 512 */
                read_scaler();

        dread(trigger_tdc);
        clear(trigger_tdc);

        dread(info_pattern);
        clear(info_pattern);

        dread(position_pattern);
        clear(position_pattern);

        dread(trigger_adc);
        clear(trigger_adc);

        dread(wire_tdc);
        clear(wire_tdc);
}

/*****************************************************************************/

void
read_scaler()
{
        pread(pm_scaler_dE);
        clear(pm_scaler_dE);

        pread(pm_scaler_ToF);
```

```
        clear(pm_scaler_ToF);

        pread(paddle_scaler);
        clear(paddle_scaler);

        pread(cer_scaler);
        clear(cer_scaler);

        lread(a.run.ps_runtime);
        lread(a.run.ps_realtime);
        lread(a.run.ps_interrupts);
        lread(a.det.trigger.scint.ps_hits);
        lread(a.beam.ps_photo);
        lread(a.beam.ps_foerster);
        lread(a.beam.ps_faraday);
        lread(a.det.trigger.top.ps_hits);
        lread(a.det.trigger.cerenkov.ps_hits);
        lread(a.coinc.abc.coinc);
        lread(a.coinc.ab.coinc);
        lread(a.coinc.ac.coinc);
        lread(a.coinc.bc.coinc);

        rs.total        = total;
        rs.interrupts   = a.run.ps_interrupts            & (~0x8000);
        rs.dE_ToF_coinc = a.det.trigger.scint.ps_hits    & (~0x8000);
        rs.ps_runtime   = a.run.ps_runtime               & (~0x8000);
        rs.ps_realtime  = a.run.ps_realtime              & (~0x8000);
        rs.ps_foerster  = a.beam.ps_foerster             & (~0x8000);
        rs.ps_faraday   = a.beam.ps_faraday              & (~0x8000);
        rs.ps_photo     = a.beam.ps_photo                & (~0x8000);
        rs.ps_top       = a.det.trigger.top.ps_hits      & (~0x8000);
        rs.ps_cher      = a.det.trigger.cerenkov.ps_hits & (~0x8000);
        rs.a_b_c_coinc  = a.coinc.abc.coinc              & (~0x8000);
        rs.a_c_coinc    = a.coinc.ab.coinc               & (~0x8000);
        rs.a_b_coinc    = a.coinc.ca.coinc               & (~0x8000);
        rs.b_c_coinc    = a.coinc.bc.coinc               & (~0x8000);
}

/*****************************************************************************/

void
read_various_scaler()
{
        dread(various_scaler);
        clear(various_scaler);
}

/*****************************************************************************/

char *
rdtStatus(char *buf)
{
        printstatus(buf, &rs, "A");

        return (buf + strlen(buf) + 1);
}
```

B.2.2 Allgemeine Konfigurations-Information

B.2.2.1 detector.h

```
#ifndef _DETECTOR_H_
#define _DETECTOR_H_

/****************************** VDCs ******************************/

#define AX1SIZE         400
#define AS1SIZE         320
#define AX2SIZE         416
#define AS2SIZE         336

#define BX1SIZE         368
#define BS1SIZE         336
#define BX2SIZE         400
#define BS2SIZE         368

#define CX1SIZE         448
#define CS1SIZE         352
#define CX2SIZE         448
#define CS2SIZE         352

struct vdcA {
        short   x1[AX1SIZE];
        short   s1[AS1SIZE];
        short   x2[AX2SIZE];
        short   s2[AS2SIZE];
};

struct vdcB {
        short   x1[BX1SIZE];
        short   s1[BS1SIZE];
        short   x2[BX2SIZE];
        short   s2[BS2SIZE];
};

struct vdcC {
        short   x1[CX1SIZE];
        short   s1[CS1SIZE];
        short   x2[CX2SIZE];
        short   s2[CS2SIZE];
};

/*************** Trigger (Cerenkov & Scintillators) ***************/

#define ACPADDLES       15
#define BPADDLES        14
#define ACMIRRORS       12
#define BMIRRORS         5

struct pm {
        short   energy;
        long    hits;
};
```

```
struct cerenkovAC {
        struct pm       mirror[ACMIRRORS];
        short           ps_hits;
};

struct cerenkovB {
        struct pm       mirror[BMIRRORS];
        short           ps_hits;
        short           pattern;
};

struct paddle {
        struct pm       right, left;
        long            hits;
};

struct planeAC {
        struct paddle   pad[ACPADDLES];
};

struct scintAC {
        struct planeAC  tof, de;
        short           de_tof_time;
        short           ps_hits;
};

struct planeB {
        struct paddle   pad[BPADDLES];
        short           pattern;
};

struct planeBDe {
        struct pm       pad[BPADDLES];
        short           pattern;
};

struct scintB {
        struct planeB   tof;
        struct planeBDe de;
        short           de_tof_time;
        short           ps_hits;
};

struct topScintABC {
        short           energy;
        short           ps_hits;
};


struct triggerAC {
        struct topScintABC      top;
        struct cerenkovAC       cerenkov;
        struct scintAC          scint;
        short                   scint_top_time;
        short                   scint_ch_time;
        short                   info;
};
```

```
struct triggerB {
        struct topScintABC      top;
        struct cerenkovB        cerenkov;
        struct scintB           scint;
        short                   scint_top_time;
        short                   scint_ch_time;
        short                   info;
};

#endif
```

B.2.2.2 spectrometer.h

```
#ifndef _SPECTROMETER_H_
#define _SPECTROMETER_H_

#include "detector.h"

/************************* Beam Information  *************************/

struct beamInfo {
        short   foerster;
        short   ps_foerster;
        short   photo;
        short   ps_photo;
        short   faraday;
        short   ps_faraday;
        short   position;
        short   position_sem;
};

/*************************  Run Information  *************************/

struct runInfo {
        short   runtime;        /* general gate on (100 us)            */
        short   realtime;       /* micro not busy (100 us)             */
        short   ps_runtime;     /* prescaled runtime (1 s)             */
        short   ps_realtime;    /* prescaled realtime (1 s)            */
        short   ps_interrupts;  /* prescaled interrupts (1 k)          */
};

/************************* Event Information *************************/

struct times {
        short   time[3];
        short   coinc;
        long    dead1;
        long    dead2;
};

struct eventInfo {
        struct times    ab;
        struct times    bc;
        struct times    ca;
```

```
        short           abc_coinc;
        long            abc_dead;
};

/************************* Detector System A *************************/

struct detA {
        struct vdcA     vdc;
        struct tiggerAC trigger;
        short           sync_info;
};

/************************* Detector System B *************************/

struct detB {
        struct vdcB     vdc;
        struct triggerB trigger;
        short           sync_info;
};

/************************* Detector System C *************************/

struct detC {
        struct vdcC     vdc;
        struct triggerAC trigger;
        short           sync_info;
};

#endif
```

B.2.2.3 statistics.h

```
#ifndef _STATISTICS_H_
#define _STATISTICS_H_

struct runStatistics {
        long    total;          /* total number of events                 */
        short   interrupts;     /* prescaled number of interrupts         */
        short   dE_ToF_coinc;   /* prescaled number of scint. coincidences */
        short   ps_foerster;    /* prescaled foerster detector            */
        short   ps_photo;       /* prescaled photo effect                 */
        short   ps_faraday;     /* prescaled faraday cup                  */
        short   ps_runtime;     /* prescaled runtime (microbusy)          */
        short   ps_realtime;    /* prescaled realtime (microbusy)         */
        short   ps_top;         /* prescaled top scintillator rates       */
        short   ps_cher;        /* prescaled cherenkov rates              */
        short   a_b_c_coinc;    /* prescaled abc coincidences             */
        short   a_b_coinc;      /* prescaled ab coincidences              */
        short   a_c_coinc;      /* prescaled ac coincidences              */
        short   b_c_coinc;      /* prescaled bc coincidences              */
};

#endif
```

B.2.2.4 printstatus.c

```
#include <math.h>
#include "statistics.h"

int
printstatus(char *buf, struct MpsRunStatistics *rs, char *setup)
{
        int     hours, minutes, seconds;
        long    total;          /* total number of events */
        short   interrupts;     /* prescaled number of interrupts */
        float   dE_ToF_coinc;   /* prescaled num of scint. coincidences */
        float   ps_foerster;    /* prescaled foerster detector */
        float   ps_photo;       /* prescaled photo effect */
        float   ps_faraday;     /* prescaled faraday cup */
        short   ps_runtime;     /* prescaled runtime  -> microbusy */
        short   ps_realtime;    /* prescaled realtime -> microbusy */
        short   ps_top;         /* prescaled top scintillator rates  */
        short   ps_cher;        /* prescaled cherenkov rates */
        short   a_b_coinc;      /* prescaled ab coincidences */
        double  runtime;
        int     n;

        total           = rs -> total;
        interrupts      = rs -> interrupts;
        dE_ToF_coinc    = rs -> dE_ToF_coinc * 5;
        ps_foerster     = rs -> ps_foerster * 5;
        ps_photo        = rs -> ps_photo * 5;
        ps_faraday      = rs -> ps_faraday * 5;
        ps_runtime      = rs -> ps_runtime;
        ps_realtime     = rs -> ps_realtime & 0x3fff;
        ps_top          = rs -> ps_top;
        ps_cher         = rs -> ps_cher;
        a_b_coinc       = rs -> a_b_coinc;

        hours = ps_runtime / 3600;
        minutes = (ps_runtime - hours * 3600) / 60;
        seconds = ps_runtime % 60;
        runtime = ps_runtime ? ps_runtime : HUGE;

        sprintf(buf, "Experiment Status of Setup %s\n", setup);
        n = strlen(buf);
        buf += n;
        while (--n > 0)
                *buf++ = '-';
        *buf++ = '\n';
        sprintf(buf, "Processed Events   : %8d   (%.1f Hz)\n", total,
            (double) total / runtime);
        buf += strlen(buf);
        sprintf(buf, "A/B Coincidences   : %8d   (%.1f Hz)\n",
            a_b_coinc, (double) a_b_coinc / runtime);
        buf += strlen(buf);
        sprintf(buf, "dE/ToF Coincidences: %8.0f k (%.1f Hz)\n",
            dE_ToF_coinc, 1000.0 * (double) dE_ToF_coinc / runtime);
        buf += strlen(buf);
```

```
        sprintf(buf, "Cerenkov          : %8d k (%.1f Hz)\n",
            ps_cher, 1000.0 * (double) ps_cher / runtime);
        buf += strlen(buf);
        sprintf(buf, "Foerster          : %8.0f k (%.3f kHz (uA))\n",
            ps_foerster, (double) ps_foerster / runtime);
        buf += strlen(buf);

        /*
         * Photoeffekt: 17.945 kHz = 1 uA (Stand 21.12.93, 13 h)
         *              17.879 kHz              --"--      18 h)
         * RANGE_HI      0.712 kHz = 1 uA
         *               0.6468 kHz = 1 uA  23.12.93, 17 h
         *               0.371 kHz = 1 uA 12.1.94, 17 h)
         */

        sprintf(buf, "Photoeffect       : %8.0f k (%.3f kHz (uA))\n",
            ps_photo, ((double) ps_photo / runtime) / 0.371);
        buf += strlen(buf);

        /*
         * Faraday: 145 Hz Offset
        */

        sprintf(buf, "Faraday           : %8.0f k (%.3f kHz (uA))\n",
            ps_faraday, ((double) ps_faraday - 0.145 * (double) ps_runtime) /
           runtime);
        buf += strlen(buf);
        sprintf(buf, "Real Time         : %8d s (%.1f %%)\n",
            ps_realtime, 100.0 * (double) ps_realtime / runtime);
        buf += strlen(buf);
        sprintf(buf, "Dead Time         : %8d s (%.1f %%)\n",
            ps_runtime-ps_realtime,
            100.0 * (double) (ps_runtime-ps_realtime) / runtime);
        buf += strlen(buf);
        sprintf(buf, "Run Time          : %8d s = %02d:%02d:%02d h\n",
            ps_runtime, hours, minutes, seconds);
        buf += strlen(buf);
        sprintf(buf, "\n");

        return (0);
}
```

B.2.3 Gesamt-Setup

```
/* ------------------------ logische Beschreibung ------------------------ */

struct abc {
        struct dataA    a;                          /* 1. Teilbaum/Exp.-Arm */
        struct dataB    b;                          /* 2. Teilbaum/Exp.-Arm */
        struct dataC    c;                          /* 3. Teilbaum/Exp.-Arm */
} abc;                                              /* Gesamtexperiment     */

/* --------------------- physikalische Beschreibung ---------------------- */

extern Device   rsh;             /* virt. Geraet (Frontend-Ansteuerung)   */
extern Device   eventbuilder;    /* virt. Geraet (Eventbuilding auf Host) */

Group speka, spekb, spekc, host;

Computer computer = {
        FRONTEND(eventbuilder, host),          /* uebergeordneter Host     */
        FRONTEND(rsh, speka),                  /* Frontend mit Namen speka */
        FRONTEND(rsh, spekb),                  /* Frontend mit Namen spekb */
        FRONTEND(rsh, spekc),                  /* Frontend mit Namen spekc */
};

/* ------------------ physikalisch/logische Zuordnung -------------------- */

abc.a = speka[0];            /* 1. Teilbaum bearbeitet von Frontend speka */
abc.b = spekb[0];            /* 2. Teilbaum bearbeitet von Frontend spekb */
abc.c = spekc[0];            /* 3. Teilbaum bearbeitet von Frontend spekc */
```

B.3 Gerätebeschreibungen

B.3.1 Einfaches CAMAC-Modul

```
Device  lecroy_2228a = {
        MCAMAC, MMODULE, MTDC,        /* module type                        */
        "lecroy_2228a",               /* module name                        */
        8,                            /* number of subaddresses             */
        1,                            /* 1 word to read, no block transfer */
        2048,                         /* range 0 ... 2047                   */
        CAMAC16 | CAMACx | CAMACq | CAMAClam | DEVoverflow(11),   /* flags */
        NULL,                         /* no expansion                       */
        FNULL,                        /* no setup function                  */
        F(0),                         /* read CNAF                          */
        F(24),                        /* initialization CNAF (disable lam) */
        F(9),                         /* clear module (and lam) CNAF        */
        FNULL,                        /* no special termination function    */
        FNULL,                        /* no special reset function          */
        FNULL,                        /* no special check funtion           */
        FNULL                         /* no special control funtion         */
};
```

B.3.2 Komplexes Gerät

```
#include <stdlib.h>
#include "gen/camac.h"
#include "cfg/item.h"

#define FREAD           2
#define FCLEAR          16
#define FINIT           16
#define NSCALER         32

static int      setup(), init(), clear(), term();
static Cword    *read();

Device  lecroy_4434_24 = {
        MCAMAC, MMODULE, MSPECIAL, /* module type */
        "lecroy_4434_24",       /* module name */
        1,                      /* number of subaddresses */
        NSCALER,                /* up to NSCALER words to read */
        0x10000,                /* range 0 ... 0xffff */
        CAMAC24 | CAMACx | CAMACq,
        NULL,                   /* no expansion */
        FUN(setup),             /* setup function */
        FUN(read),              /* readout function */
        FUN(init),              /* initialization function */
        FUN(clear),             /* clear function */
        FUN(term),              /* term function */
        FNULL, FNULL,           /* reserved */
};

struct cnafs {
        Cnaf    high;
        Cnaf    low;
        Cnaf    init;
        Cnaf    clear;
};

static int
setup(struct item *item, Cnaf (*memmap)())
{
        struct device   *dev;
        int     b, c, n, addr[10];
        struct cnafs    *cnafs;

        if (item -> special == NULL) {
                if ((item -> special = malloc(sizeof (struct cnafs))) == NULL)
                        return (-1);
        }
        cnafs = (struct cnafs *) item -> special;
        if (getaddr(addr, 10, &dev, item -> hw) != 5)
                return (-1);
        b = addr[0];
        c = addr[1];
        n = addr[2];
        cnafs -> clear = memmap(b, c, n, 0, FCLEAR, CAMAC16);
        cnafs -> init  = memmap(b, c, n, 0, FINIT, CAMAC16);
        cnafs -> high = memmap(b, c, n, 0, FREAD, CAMAC24);
```

```
        cnafs -> low = memmap(b, c, n, 0, FREAD, CAMAC16);

        return (0);
}

static int
init(struct item *item)
{
        *((struct cnafs *) item -> special) -> init = 0x40;

        return (0);
}

static int
clear(struct item *item)
{
        *((struct cnafs *) item -> special) -> clear = 0x40;

        return (0);
}

static int
term(struct item *item)
{
        return (0);
}

static Cword *
read(struct item *item, register Cword *buf)
{
        register Cnaf   high, low;
        register Cword  data, h, l;
        register int i;

        high = ((struct cnafs *) item -> special) -> high;
        low = ((struct cnafs *) item -> special) -> low;
        *((struct cnafs *) item -> special) -> init =
            NSCALER - 1 << 8 | 0xe0;
        for (i = 0; i < NSCALER; ++i) {
                h = *high;
                l = *low;
                if (h > 0 || l) {
                        *buf++ = i;
                        *buf++ = h;
                        *buf++ = l;
                }
        }

        return (buf);
}
```