Physik-Department E 12
der Technischen Universität München

Die Datenaufnahmesteuerung für das HADES Detektorsystem

Diplomarbeit
von
Benjamin Sailer

München, 15. Januar 2001

Zusammenfassung

Im Rahmen der vorliegenden Arbeit wurde eine Steuerung für die HADES-Datenaufnahme geplant, implementiert und am Detektor in Betrieb genommen.

HADES ist ein Dileptonenspektrometer, das sich derzeit an der Gesellschaft für Schwerionenforschung (GSI) Darmstadt in der Endphase des Aufbaus befindet. Es soll zur Messung von leptonischen Vektormesonenzerfällen dienen, die bei großer Statistik präzise Aussagen über die Eigenschaften von Hadronen in Materie erlauben. Die ATM-basierte Datenaufnahme (DAQ) liest über 70 000 Kanäle mit einer primären Triggerrate von 100 kHz aus. Eine große Anzahl an speziell für diesen Zweck konstruierten Einzelkomponenten bedarf der Konfiguration, Überwachung, Steuerung und Dokumentation.

Unter Verwendung des Softwarepakets EPICS wurde ein verteiltes System entwickelt, das eine flexible Konfiguration des Steuerungsverhaltens erlaubt. Schnittstellen für die Hardwarekonfiguration wurden ebenso geschaffen wie solche zur allgemeinen Datenhaltung des HADES-Experiments. Das Konzept von endlichen Automaten dient innerhalb der Datenaufnahmesteuerung als Ablaufsteuerung. Für den Experimentator wurde eine graphische Benutzeroberfläche implementiert, die eine einfache Bedienung der Datenaufnahmesteuerung erlaubt.

Besonderer Wert wurde auf die Modularität des Systems gelegt, was die einfache Erweiterung der Datenaufnahmesteuerung ermöglicht. Die Beschränkung auf standardisierte Programmierschnittstellen soll den flexiblen Einsatz der Steuerung auf unterschiedlichen Plattformen gewährleisten.

Somit wurde ein Programmpaket erstellt, das alle Anforderungen an Software im allgemeinen und eine Datenaufnahmesteuerung im besonderen erfüllt oder zumindest auf einfache Weise dahingehend erweitert werden kann.

Testaufbauten wurden sowohl im Labor in München als auch am Experimentierort GSI erfolgreich installiert und betrieben. Sie zeigen die Tragfähigkeit des Konzepts. Während der letzten Strahlzeit mit dem nahezu vollständigen Spektrometer gelang die Migration der Testversionen zu einem Teil der neuen Datenaufnahmesteuerung. Erste vorläufige Meßergebnisse von der HADES-Inbetriebnahme werden vorgestellt.

Inhaltsverzeichnis

Kapitel 1

Motivation

1.1 Dileptonenspektroskopie mit HADES

Im Gegensatz zu normaler Materie, wie z.B. Atome oder Atomkerne, ist bei Hadronen die Masse nicht im wesentlichen durch die Masse ihrer Bestandteile — also der Quarks — bestimmt. Durch die starke Bindungsenergie, die durch die starke Wechselwirkung dominiert ist, erhöht sich die Masse auf ein Vielfaches der Quarkmassen. Der hauptsächliche Beitrag entsteht durch Gluonen sowie $\bar{q}q$-Paare, die durch Vakuumpolarisation entstehen. Dieser Mechanismus hängt von der Umgebung in starkem Maße ab, so daß sich die Eigenschaften von Hadronen, insbesondere die Masse, innerhalb von Kernmaterie von denen im Vakuum unterscheiden. Eine zusätzliche Erhöhung der Dichte und / oder der Temperatur können diese Effekte verstärken [Rap99] [Cas99]. Darüberhinaus wird erwartet, daß bei Dichten ρ über $5 \cdot \rho_0$ bzw. Temperaturen T ab etwa 150 MeV das Quark-Antiquark Kondensat $\langle \bar{q}q \rangle$, der Ordnungsparameter der QCD, verschwindet und die chirale Symmetrie restauriert wird. Einen Eindruck von der funktionalen Abhängigkeit gewährt Abbildung 1.1 [Wei94]. Aus ihr wird auch ersichtlich, daß bereits bei moderater Dichte merkliche Abweichungen vom Vakuumwert erwartet werden und damit die entsprechende Modifikationen der Hadroneneigenschaften.

Nun ist eine solche Umgebung experimentell schwer zugänglich. In der Natur kommen Dichten in der Größenordnung von mehrfacher Grundzustandsdichte von Kernmaterie lediglich in Neutronensternen oder Supernovae vor, Temperaturen jenseits der 100 MeV nur in geringem zeitlichen Abstand zum Urknall.

Es ist allerdings möglich, derartige Zustände in modernen Teilchenbeschleunigern zu erreichen, wenn auch sowohl räumlich als auch zeitlich stark begrenzt. In einer zentralen Schwerionenkollision mit einer Schwerpunktsenergie in der Größenordnung von 1 GeV pro Nukleon entstehen im Kollisionszentrum Dichten von etwa $3 \cdot \rho_0$. Als Sonden zur Messung der Eigenschaften solcher Zustände eignen sich unter anderem leichte Vektormesonen, die ebenfalls in derartigen Kollisionen entstehen. Diese sollten eine geringere Masse haben, als das im Vakuum

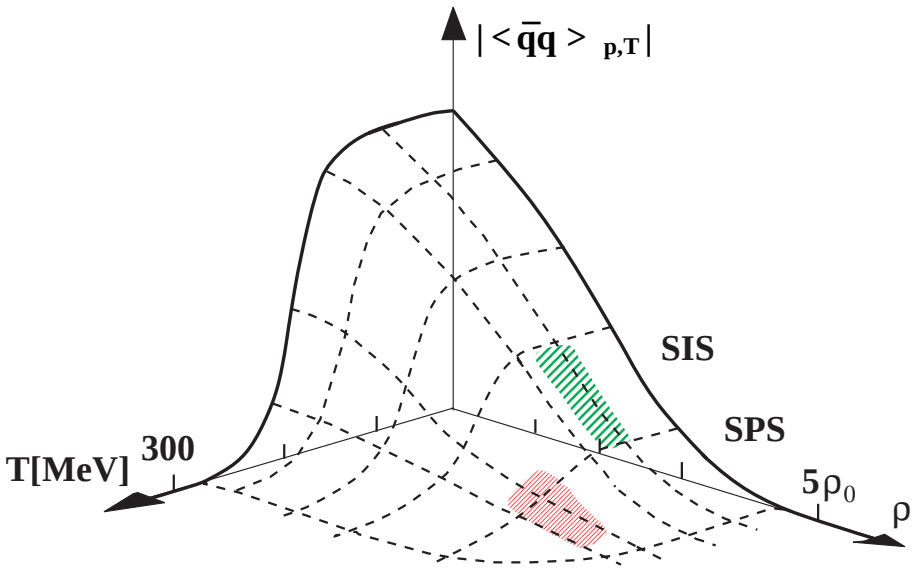

Abbildung 1.1: Restauration der chiralen Symmetrie bei hohen Temperaturen und Dichten nach Weise [Wei94]. Die Wahrscheinlichkeitsdichte des Quark-Antiquark-Kondensats nimmt ab, was mit einer Verringerung der Masse von leichten Vektormesonen einhergeht.

der Fall wäre. Wenn sie dann innerhalb der heißen und dichten Phase zerfallen, ist die Messung dieser Masse im Prinzip über die Spektroskopie der Zerfallsprodukte möglich.

Wenn man bedenkt, daß die Teilchen einerseits bei der zur Verfügung stehenden Energie in ausreichendem Maße erzeugt werden müssen, auf der anderen Seite aber auch kurzlebig genug sein sollen, daß eine endliche Wahrscheinlichkeit des Zerfalls innerhalb des Kernes besteht, wird klar, warum sich vor allem die leichten Vektormesonen ρ, ω und ϕ, deren Eigenschaften in Tabelle 1.1 zu finden sind, zur Messung eignen.

Vektormeson	Verzweigungsverhältnis in e^+e^-	Mittlere Zerfallszeit [fmc^{-1}]
ρ	$4.6 \cdot 10^{-5}$	1.3
ω	$7 \cdot 10^{-5}$	23.4
ϕ	$3 \cdot 10^{-4}$	44.4

Tabelle 1.1: Verzweigungsverhältnisse der leichten Vektormesonen in den leptonischen Zerfallskanal e^+e^- [Sch95].

Um aus einer solchen Messung stichhaltige Aussagen extrahieren zu können, ist es notwendig, daß die Zerfallsprodukte möglichst ohne weitere Wechselwirkung die heiße und dichte Phase verlassen können. Damit scheiden stark wech-

selwirkende Teilchen aus und es bleiben Leptonen und Photonen. Vorteilhaft ist es darüber hinaus, wenn es sich um einen Zweikörperzerfall handelt, da dies die Messung erleichtert. Unter Berücksichtigung der zur Verfügung stehenden Energie erweist sich schließlich der Zerfall in ein Elektron-Positron-Paar als ein sehr gutes Instrument. Diese Leptonen unterliegen lediglich der elektroschwachen Wechselwirkung (sowie der Gravitation, die hier völlig zu vernachlässigen ist), die um den Faktor $\alpha^2 = (\frac{1}{137})^2$ gegenüber der starken Wechselwirkung unterdrückt ist. Leider reduziert dieser Faktor nicht nur die Wahrscheinlichkeit für eine Endzustandswechselwirkung, sondern ebenso die Wahrscheinlichkeit für einen derartigen Zerfall selbst. Um also eine ausreichende Statistik für eine präzise quantitative Aussage zu erreichen, ist ein Spektrometer notwendig, das sowohl eine hohe Akzeptanz und eine große Ratenfestigkeit besitzt, als auch eine wirksame Methode, die Dileptonenpaare aus den restlichen Zerfallsprodukten der Mesonen und — was noch weit mehr ins Gewicht fällt — aus den Resten der bei der Kollision zerplatzenden Kerne herauszufiltern.

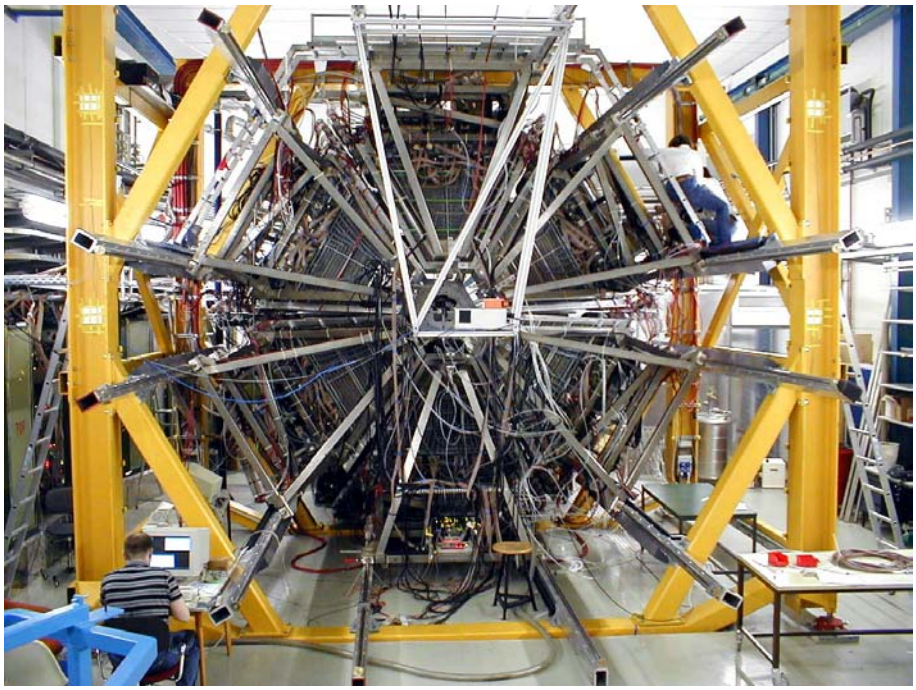

Abbildung 1.2: HADES im November 2000.

Zu diesem Zweck wurde das neue e^+e^--Paar Spektrometer HADES[1] [HAD94] [Fri99] konzipiert. HADES ist ein Detektorsystem, das an der GSI[2] Darmstadt aufgebaut wurde und im November 2000 seine erste Strahlzeit mit allen Detek-

[1] *H*igh *A*cceptance *Di*electron *S*pectrometer
[2] *G*esellschaft für *S*chwer*i*onenforschung mbH

torkomponenten hatte. HADES ist ein Detektorsystem der zweiten Generation, d.h. es widmet sich dieser Aufgabenstellung im SIS[3] Energiebereich bis 2 AGeV Projektilenergie, nachdem sie bereits vorher von DLS[4] angegangen wurde. Im Vergleich zu DLS ist die Akzeptanz von HADES allerdings um einen Faktor 100 höher, dazu kommt eine Auflösung der invarianten Masse der Zerfallsedukte von 1 % gegenüber 12 % bei DLS. Außerdem ist HADES wesentlich ratenfester als DLS. Damit sollte es möglich sein, zum ersten Mal nicht nur leichte Vektormesonen über dem kombinatorischen Untergrund [Sch95] aus Konversionselektronen und Dalitzzerfällen zu erkennen, sondern auch präzise Aussagen über Lage und Breite ihrer Massenverteilung im invarianten Massenspektrum machen zu können.

1.2 Notwendigkeit einer Datenaufnahmesteuerung

Das neue Spektrometer wird in seiner Endausbaustufe aus ca. 62 Detektor- bzw. Subsystemen mit insgesamt ca. 70 000 Kanälen bestehen und wird von einer Kollaboration aus ca. 120 Wissenschaftlern aufgebaut. Die einzelnen Subsysteme müssen weitgehend unabhängig auszulesen und zu überwachen sein und verlangen alle nach individuellen Initialisierungs- und Kalibrierverfahren. Ein derartig komplexes System kann nicht mehr von einem einzelnen Experimentator bedient werden und bedarf einer weitgehend automatisierten Detektorsteuerung und -überwachung. Darüberhinaus ist insbesondere eine benutzerfreundliche Datenaufnahmesteuerung unerläßlich, um den Experimentierbetrieb über wochenlange Perioden mit wechselnden Einstellungen zu ermöglichen.

Eine Detektorsteuerung (Slow Control) ist die Summe aller Einrichtungen, die es dem Experimentator während eines Experimentes erlauben, den Status des Detektorsystems zu überwachen, Detektorparameter vor und während der Datennahme zu ändern sowie Änderungen zu dokumentieren. Darüberhinaus soll es dem Experimentator erleichtert werden, die Informationen, die er über den Zustand des Detektors erhält, zu ordnen, zu analysieren und Wichtiges von Unwichtigem zu trennen, damit er einen Fehlerfall schnell erkennen und geeignete Gegenmaßnahmen ergreifen kann. Ansonsten besteht die Gefahr, daß unnütze Daten genommen werden (wenn z.B. die Hochspannung an einem Detektor ausfällt) oder gar die Hardware beschädigt wird (weil Gefahren wie zu hohe Ströme an Detektoren nicht rechtzeitig erkannt werden). Immer wiederkehrende Aufgaben sollten darüberhinaus von der Detektorsteuerung automatisiert werden.

[3] *S*chwer*i*onen*s*ynchrotron; Teilchenbeschleuniger an der GSI.

[4] *Di*lepton *S*pectrometer — Ein Experiment, daß am BEVALAC in Berkeley, USA aufgebaut wurde, um Dileptonenpaare zu messen.

Des weiteren sollte die Detektorsteuerung bei größeren Detektoren, die während der langen Konstruktionsphase ebenso wie im Produktionsbetrieb in immer wechselnden Konfigurationen für Tests, Kalibration und verschiedene Experimente betrieben werden, es dem Experimentator erlauben, diese Konfigurationen ohne aufwendige Programmiertätigkeiten über eine angemessen zu bedienende Schnittstelle auszuwählen und zu verändern. Auch diese Änderungen sollten automatisch dokumentiert werden, so daß es zu jedem späteren Zeitpunkt möglich ist, die im jeweiligen Experiment gewonnenen Daten korrekt zu interpretieren.

Die Aufgabe der Datenaufnahme (DAQ[5]) eines Experimentes ist es, die Informationen, die die Detektoren des Aufbaus liefern, zu sammeln, nach Ereignissen zu ordnen und Massenspeichermedien zuzuführen, wo sie dann der Analyse zur Verfügung stehen. Insofern ist die Datenaufnahme genauso ein eigenständiger Bestandteil des Experiments wie die Detektoren.

Die Datenaufnahmesteuerung (Run Control) ist der Bestandteil der Detektorsteuerung, der die Bedienung der Datenaufnahme betrifft. Gerade in der Datenaufnahme ist es notwendig, die Bedienung auch Experimentatoren zu erlauben, die keine Detailkenntnisse über das Datenaufnahmesystem selbst haben. Meist sind Konfigurationsänderungen nötig, um einen einzelnen Detektor zu testen oder zu kalibrieren. Damit ist zwar für den Detektor selbst ein Experte verfügbar, nicht aber unbedingt ein Experte für die Datenaufnahme.

Für ein verteiltes System, wie es eine Datenaufnahmesteuerung aufgrund der verteilten Natur einer Datenaufnahme immer ist, kann das Client-Server-Modell[6] angewendet werden [Mac96]. Danach werden Prozesse in zwei verschiedene Kategorien eingeteilt:

- Server-Prozesse laufen auf jeder Komponente der Datenaufnahme, die ihr zugeordnete Hardware kontrollieren und überwachen soll. Dazu ist im Normalfall eine CPU mit einem Betriebssystem notwendig, für das eine definierte Programmierschnittstelle zur Verfügung steht. In heterogenen Systemen sind hier Standards wichtig, die plattformübergreifende Programmierung ermöglichen (z.B. POSIX C-Programmierschnittstellen für ANSI C/C++).

- Client-Prozesse kommunizieren mit den Servern und stellen dem Experimentator eine Benutzeroberfläche zur Verfügung, mittels der er die Steuerungsaufgaben wahrnehmen kann. Client-Prozesse können in hierarchisch aufgebauten Systemen allerdings auch Server für andere Prozesse darstellen. So können Aufgaben zusammengefaßt und Abläufe automatisiert werden.

[5] *D*ata *Acq*uisition

[6] Neben der Client-Server-Nomenklatur wird hier zum Teil auch eine andere Sprechweise verwendet: Agent-Manager. Die Rolle des Servers übernimmt der Agent, während der Client zum Manager wird. Der Unterschied liegt darin, daß ein Server normalerweise viele Clients bedient, während viele Agenten für einen Manager arbeiten. Insofern wäre die zweite Bezeichnung passender. Allerdings verwendet sowohl [Mac96] als auch EPICS [Dal00] die Bezeichnungen Server und Client, so daß in der vorliegenden Arbeit beide Nomenklaturen zu finden sind.

In der folgenden Auflistung sind die wichtigsten Anforderungen an die Datenaufnahmesteuerung zusammengefaßt:

- Überwachung und Protokollierung der Betriebsparameter
- Aufbereitung der Daten durch eine graphische Benutzerschnittstelle
- Automatisierung immer gleich oder ähnlich strukturierter Abläufe, wo immer es möglich ist
- Schutz vor Fehlbedienung — Fehlertoleranz
- Konfigurierbarkeit der DAQ, ohne Komponenten neu programmieren oder kompilieren zu müssen
- Fehlerbehandlung für Ereignisse außerhalb der normalen Betriebszustände
- Modularität für die Entwicklungsphase des Detektors
- Dokumentation und Hilfesystem, das den Normalbetrieb abdeckt

Im Rahmen der vorliegenden Arbeit wurden die Anforderungen an eine derartige Datenaufnahmesteuerung untersucht und darauf aufbauend ein Konzept und ein Prototyp entwickelt.

Kapitel 2

Die HADES Datenaufnahme

2.1 Der Aufbau von HADES

Um eine Vorstellung davon zu erhalten, welche Anforderungen an die Steuerung von HADES und speziell der HADES-Datenaufnahme bestehen, wird zunächst in groben Zügen der Aufbau des Detektorsystems beschrieben, wie ihn Abbildung 2.1 zeigt. HADES ist ein Magnetspektrometer, dessen Geometrie für die Gegebenheiten eines Fixed-Target-Experiments optimiert ist [HAD94] [Sal95] [Sch95] [Fri99]. Sowohl das Magnetfeld als auch die Driftkammern (MDC[1]), die zur Detektion der Teilchendurchstoßpunkte und damit zur Spurrekonstruktion dienen, befinden sich aus Sicht des Strahls in Vorwärtsrichtung und decken Polarwinkel von ca. 15° bis 85° ab.

Das Magnetfeld wird von einem supraleitenden Magneten mit sechs Spulen erzeugt. Die Form des Magnetfeldes ist ein Toroid, d.h. die Magnetfeldlinien verlaufen ringförmig um die Strahlachse. Teilchen werden daher lediglich in polarer Richtung abgelenkt. Diese sechs Spulen legen gleichzeitig die sechsfache Rotationssymmetrie des gesamten Detektorsystems fest.

Die Driftkammern sind paarweise vor und hinter dem Magnetfeld angeordnet, um so die Trajektorie der Teilchen vor und nach der Ablenkung im Magnetfeld messen zu können. Daraus und aus der Kenntnis der Magnetfeldstärke [Bre99] kann der Impuls berechnet werden, und zusammen mit der durch die Teilchenart festgelegten Ruhemasse ergibt sich der Viererimpuls. Wenn man nun diesen für beide Teilchen aus einem Zweikörperzerfall hat, kann man daraus auch den Viererimpuls und damit die Ruhemasse des Ausgangsteilchens errechnen [Sch95].

Um trotz der in Tabelle 1.1 aufgeführten geringen Verzweigungsverhältnisse der Sonden in den dileptonischen Zerfallskanal statistisch aussagekräftige Meßergebnisse zu erhalten, müssen möglichst viele Ereignisse gemessen werden. HADES muß also eine hohe Ratenfestigkeit besitzen. Als Ziel ist ein Primärstrahl von 10^8 Teilchen pro Sekunde angestrebt. Bei 1 % Wechselwirkung im Target und einer

[1] *M*ini *D*rift *C*hambers

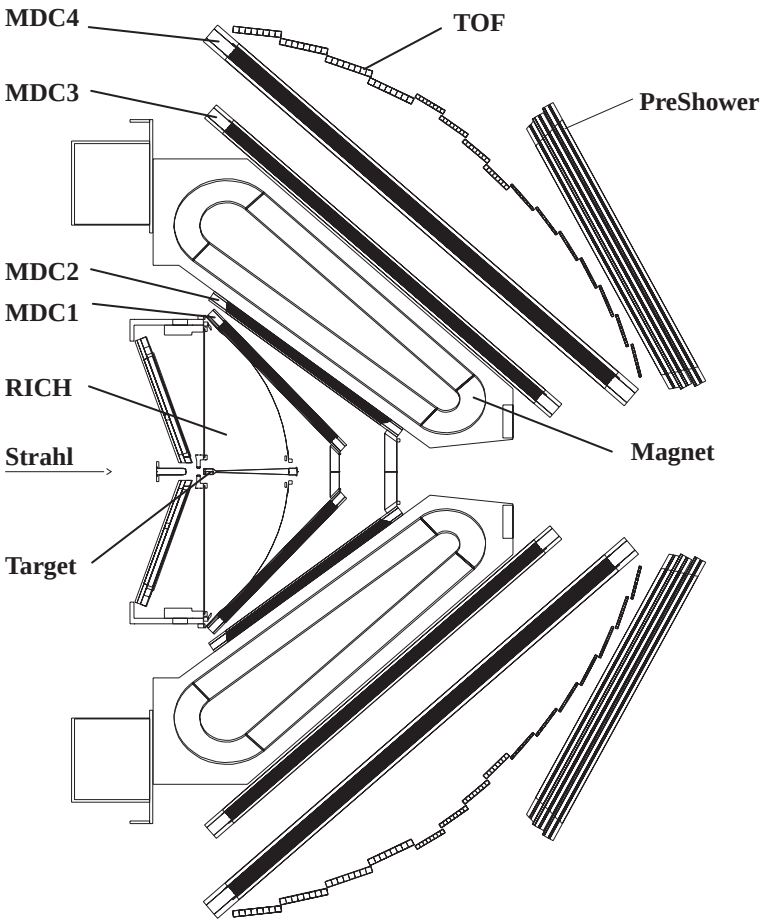

Abbildung 2.1: Schnitt durch das HADES Detektorsystem.

Beschränkung auf die zentralsten 10 % der Stöße ergibt sich eine Rate von $10^5\,\mathrm{s}^{-1}$ für die erste Triggerstufe. Es sollen die 10^2 Ereignisse pro Sekunde weggeschrieben werden, die die aussichtsreichsten Kandidaten für die gesuchten leptonischen Zerfälle sind.

Diese hohen geforderten Raten führen zusammen mit der hohen Multiplizität bei Schwerionenstößen, die beispielhaft in Abbildung 2.2 dargestellt ist, zu der Notwendigkeit einer leistungsfähigen Teilchenidentifikation, die eine Reduzierung der aufgenommenen Daten auf interessante Ereignisse bereits online, d.h. noch als Teil der Datenaufnahme, erlaubt. Dazu werden drei weitere Detektoren eingesetzt:

Der Ring abbildende Cherenkov Detektor (RICH[2]) [Ger96] [Kas00] (noch innerhalb der ersten Driftkammerebene gelegen) nutzt den Cherenkov-Effekt aus. Dabei werden von Teilchen, die sich durch ein Medium schneller als Licht be-

[2] *R*ing *I*maging *Ch*erenkov

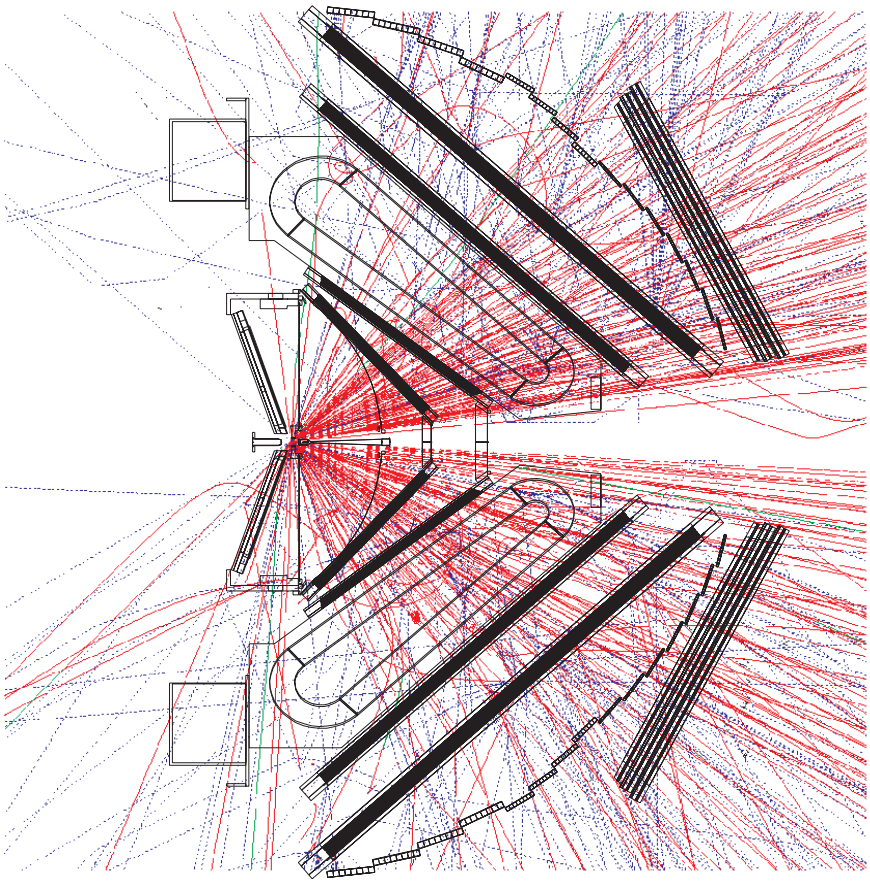

Abbildung 2.2: Bei einer zentralen Kollision zweier Goldkerne mit einer Schwerpunktsenergie von 1 AGeV werden etwa 200 geladene Teilchen erzeugt.

wegen, Photonen in einem festen Winkel zur Teilchentrajektorie ausgesandt. Im HADES-RICH wird die Lichtgeschwindigkeit mittels eines Gasradiators so nahe bei dem Wert der Vakuumlichtgeschwindigkeit eingestellt, daß lediglich die leichten Elektronen und Positronen genügend Energie besitzen, um die Cherenkov-Schwelle zu überschreiten. Neben dem Radiatorvolumen besteht der HADES-RICH aus einem Spiegel, der den Cherenkov-Kegel auf einen Ring abbildet, und einem in Rückwärtsrichtung befindlichen Photonendetektor, der in der Bildebene des Spiegels liegt. Dieser ist zweidimensional ortsauflösend und erlaubt somit die Identifikation der schnellen Leptonen über die Erkennung von Ringen.

Ebenso benötigt man auch hinter dem Magneten und den äußeren Driftkammern Detektoren zur Teilchenidentifikation: Bei kleinen Polarwinkeln dient hierzu ein PreShower-Detektor, der durch den Vergleich von Ladungssignalen vor und hinter Bleikonvertern Elektronen und Positronen anhand der von ihnen erzeugten elektromagnetischen Schauer identifizieren kann.

Bei größeren Polarwinkeln können Teilchen über ihre Flugzeit identifiziert werden. Eine Flugzeitwand (TOF[3]) [Ago98], bestehend aus stangenförmigen Plastikszintillatoren und Photomultipliern, dient zusammen mit dem Startdetektor ebenfalls zur Erkennung von Leptonen.

Die Flugzeitwand wird zu kleinen Polarwinkeln hin durch einen wesentlich schwächer segmentierten Flugzeitdetektor mit der Bezeichnung „TOFino“ fortgesetzt. Dieser ist eine vorläufige Lösung und soll später durch eine Flugzeitwand bei kleinen Polarwinkeln ersetzt werden, die eine ähnlich hohe Segmentierung aufweist wie die äußere TOF.

Des weiteren kommt noch ein achtfach segmentierter Startzähler zum Einsatz, der das Startsignal für TOF liefert, sowie ein ebenso strukturierter Vetodetektor, der hinter dem Target angebracht ist und das Startsignal inhibiert, wenn ein Strahlteilchen das Target ohne Wechselwirkung passiert.

Teilsystem	Anzahl der Kanäle	Ereignisgröße im ersten Puffer[4] [KB]	Primäre Datenmenge [GBs^{-1}]	Weggeschriebene Datenmenge [MBs^{-1}]
MDC	26 400	11.2	1.65	10.99
RICH	27 648	0.9	0.09	0.87
PreShower	18 432	4.7	1.76	4.54
TOF	416	1.2	0.11	1.16
Gesamt	72 896	18.0	3.61	17.56

Tabelle 2.1: Datenraten der verschiedenen Detektoren von HADES (nach [Mün01]), ohne dritte Triggerstufe.

[3] *T*ime *O*f *F*light

[4] Der Datenstrom des PreShower ist in der ersten Pufferstufe nicht nullenunterdrückt.

2.2 Die Datenaufnahme

Die zwei großen Bereiche, in die sich die HADES Datenaufnahme gliedern läßt, sind zum einen Elektronik zur Digitalisierung und Auslese (Hardware-Auslese) und zum anderen softwarebasierter Datentransport und das Zusammensetzen der Detektordaten zu Ereignissen (Software-Auslese und Eventbuilding). Dazu kommt eine dreistufige Auslesesteuerung (Trigger). Die erste Triggerstufe ist ein Auslesesignal, daß aus einer Kombination von Startzählersignal, Multiplizität in der Flugzeitwand und fehlendem Signal im Vetozähler hinter dem Target gewonnen wird. Die zweite Triggerstufe ist in Hardware implementiert, während als dritte Triggerstufe eine Softwarelösung geplant ist.

2.2.1 Hardware-Trigger

Experiment	KaoS [Sen00]	HADES [Mün01]	LEP [Mat99]	LHC [Mat99]
Zahl der Kanäle	$3 \cdot 10^3$	10^5	10^5	$10^6 - 10^8$
Ereignisgröße [KB]	3	300	100	1 000
LVL1-Triggerrate [s^{-1}]	10^4	10^5	$5 \cdot 10^4$	$4 \cdot 10^8$
LVL1-Datenrate [GBs^{-1}]	0.03	3.6	0.1	10^3
LVL2-Triggerrate [s^{-1}]	10^3	10^3	50	10^5
LVL2-Datenrate [MBs^{-1}]	3	18	20	10^5
LVL3-Triggerrate [s^{-1}]		10^2	10	10^3
LVL3-Datenrate [MBs^{-1}]		1.8	5	10^3
Datenreduktion	10	$2 \cdot 10^3$	20	10^3

Tabelle 2.2: Kennzahlen für Triggersysteme verschiedener Experimente der Kern- und Hochenergiephysik [Mat99] (LEP und LHC stehen stellvertretend für typische (bei LHC: geplante) Experimente an den entsprechenden Beschleunigern).

Um die oben angesprochene Online-Datenreduktion zu erreichen, werden im Gegensatz zu kernphysikalischen Experimenten früherer Generationen, wie z.B. KaoS [Sen93], bei HADES (wie auch in anderen modernen Experimenten der Hochenergiephysik, z.B. den CERN[5]-Experimenten COMPASS [COM96] und ALICE [Mor99]) die Daten bereits sehr früh am Detektor digitalisiert. Die Notwendigkeit der Datenreduktion verdeutlicht Tabelle 2.2. Dies erlaubt das Zwischenspeichern der Informationen, während das Triggersystem die Entscheidung darüber trifft, ob die Daten zu interessierenden Ereignissen (in unserem Falle solche mit einem dileptonischen Zerfall) gehören und damit weiterverarbeitet werden

[5] *C*onseil *E*uropéen pour la *R*echerche *N*ucléaire

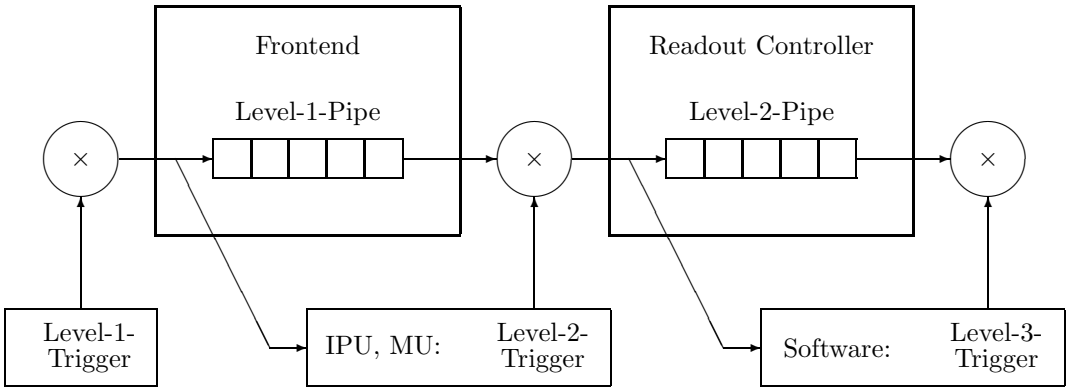

Abbildung 2.3: Datenpfade von Frontend- und Readout-Elektronik.

sollen. Damit weiter Daten aufgenommen werden können, solange die Triggerentscheidung noch nicht gefallen ist, ist in diesen Zwischenspeichern jeweils Platz für mehrere Ereignisse, wie Abbildung 2.3 zeigt.

Für das Treffen der Triggerentscheidung ist es notwendig, eine Kopie der Daten sogenannten Bildverarbeitungseinheiten (IPUs[6]) zuzuführen. Solche gibt es für alle Detektoren, die zur Teilchenidentifikation bestimmt sind, also RICH, PreShower und TOF. Die IPU des RICH erkennt Ringe in dem ihr übermittelten Abbild der Detektorfläche [Mün95] [Leh00]. Die IPU des PreShowers [Pet00] identifiziert analog dazu Leptonen an den unterschiedlichen Ladungssignalen, die in den verschiedenen Ebenen des Detektors erzeugt werden. Die IPU der TOF [Lin01] filtert die schnellen ($\beta \approx 1$) Teilchen aus den Daten der Flugzeitwand heraus. Die IPUs geben dann die so ermittelten Leptonenkandidaten und deren Koordinaten an ein zentrales Modul weiter. Dieses überprüft, ob die Einzelereignisse zusammenpassen. Dies ist der Fall, wenn eine sinnvolle Teilchenbahn durch die Durchstoßpunkte in den einzelnen Detektoren gelegt werden kann (Matching). Auf dieser Grundlage trifft das zentrale Modul die positive oder negative Triggerentscheidung. Es wird daher als „Matching Unit“ (MU) bezeichnet [Tra99]. Der gesamte Vorgang wird in Abbildung 2.4 illustriert.

2.2.2 Hardware-Auslese

Bei allen Detektoren von HADES läßt sich die Ausleseelektronik in Module zur Digitalisierung, Nullenunterdrückung und dem als Level-1-Pipe bezeichneten Zwi-

[6] *I*mage *P*rocessing *U*nits

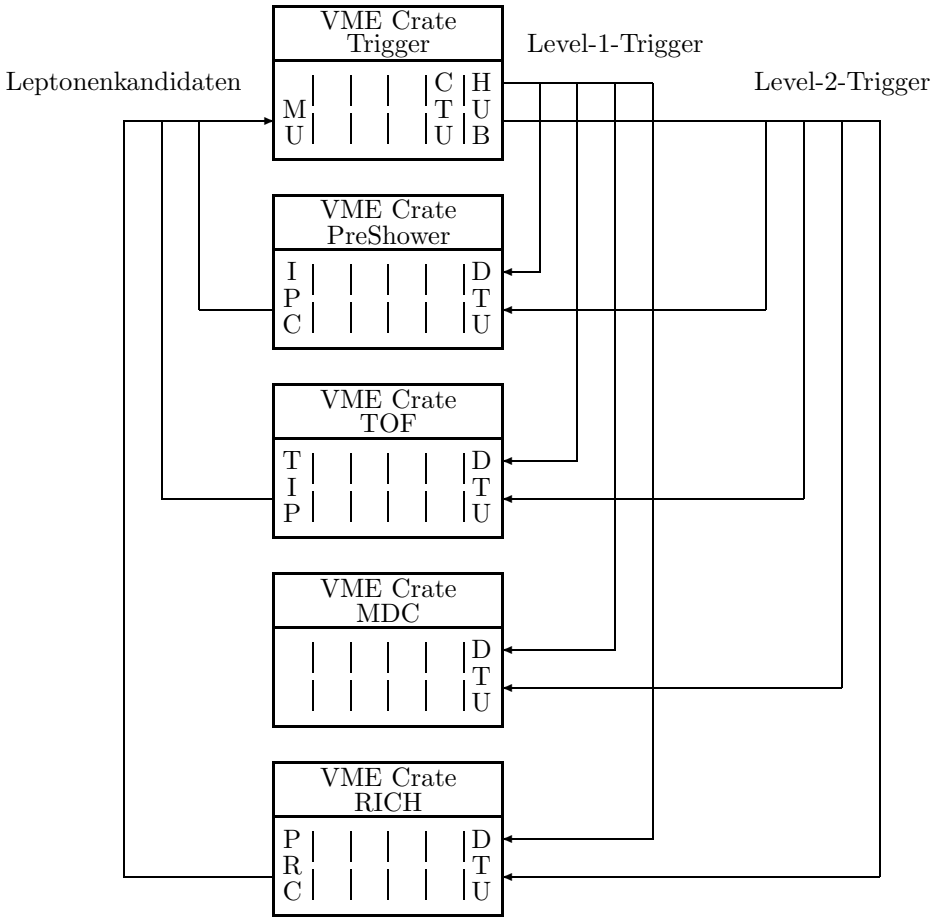

Abbildung 2.4: Aufbau des mehrstufigen HADES Triggersystems; IPC, TIP und PRC sind die Bildverarbeitungseinheiten der verschiedenen Detektoren.

schenspeicher einerseits und Module zur Auslese nach positiver Triggerentscheidung der zweiten Stufe und einem weiteren Zwischenspeicher (der analog als Level-2-Pipe bezeichnet wird) andererseits aufteilen. Die Auslese der Level-2-Pipe erfolgt jeweils von einer VME-CPU[7].

Daneben gibt es noch die oben bereits erwähnten Bildverarbeitungseinheiten sowie in jedem VME-Crate[8] eine Detektortriggereinheit (DTU) [Lin98]. Die zentrale Triggereinheit (CTU) im Trigger-Crate ist für die Verteilung der Triggerentscheidung zuständig und erzeugt darüberhinaus eine eigene Nummer für

[7]VME — *V*ersa *M*odule *E*urocard; ein Datenbussystem für Industrieanlagen, das auch in der Hochenergiephysik immer breitere Anwendung findet.

[8]Ein Crate ist ein Einschubgehäuse für Karten eines Bussystems. Ein VME-Crate beherbergt also VME-Karten.

jedes Ereignis, die den Daten nach der Digitalisierung beigefügt werden. Über diese kann jedes Datenpaket im gesamten Datenpfad als zu einem bestimmten Ereignis zugehörig identifiziert werden. Mit dieser Information können im „Event Building“ zusammengehörige Daten zu Ereignissen zusammengesetzt werden.

2.2.3 Software-Auslese und Eventbuilding

Bis hier wurden die Daten von speziell dafür gebauter Hardware verarbeitet, um die hohen Raten von bis zu 10^5 zentralen Ereignissen pro Sekunde überhaupt erreichen zu können. Nach der zweiten Triggerstufe ist die Datenmenge bereits so weit reduziert, daß sie von gängigen Computersystemen weiter verarbeitet werden kann.

Der grundsätzliche Mechanismus bei der Software-Auslese ist in Abbildung 2.5 skizziert. Sowohl auf Seiten der Auslese-CPUs, als auch bei dem Eventbuilder erhält ein Prozess die Daten und schreibt sie in einen offenen Speicherbereich (shared memory), ein anderer liest sie aus diesem aus, verarbeitet sie und leitet sie weiter. Auf den Auslese-CPUs ist die Datenquelle für den ersten Prozess die oben beschriebene spezielle Hardware und das Ziel des zweiten Prozesses die Netzwerkkarte. Auf dem Eventbuilder empfängt der erste Prozess die Daten von der Netzwerkkarte, während der zweite Prozess zusammengehörige Ereignisbestandteile, sogenannte Subevents, erkennt und jeweils vollständige Sätze zu kompletten Ereignissen zusammensetzt und wegschreibt (daher die Bezeichnung „Eventbuilding“) bzw. unvollständige Ereignisse verwirft. Da das gesamte Triggersystem nicht erlaubt, daß ein Datenpaket ein anderes mit älteren Ereignissen „überholt“, kann der Eventbuilder das Fehlen eines Subevents erkennen, sobald er aus dem entsprechenden Empfangskanal ein Subevent mit einer höheren Ereignisnummer erhält (Abbildung 2.6).

Datentransport mit ATM

Der Datentransport wird aufgrund der Anforderungen nicht über ein gängiges Netzwerkprotokoll wie Ethernet, sondern über das ursprünglich aus dem Telefonbereich stammende ATM[9] abgewickelt. Im speziellen Fall der Datenaufnahme ist die Software-Auslese zusammen mit dem Eventbuilding trotz des mehrstufigen Triggersystems der Engpaß beim Datentransport. Es würde also keinen Sinn machen, verlorengegangene Datenpakete per Protokoll zu erkennen und ein weiteres Mal anzufordern. Wenn das System Daten an der Grenze der Leistungsfähigkeit transportiert, würde das nur zum Verlust anderer Pakete und insgesamt zu einem geringeren Datendurchsatz wegen des größeren Protokoll-Overheads führen. Die vergleichsweise einfach strukturierte ATM-Hardware auf der Empfangsseite

[9] *A*synchronous *T*ransfer *M*ode — ein Protokoll für breitbandige Glasfasernetze, das einzelnen Verbindungen Mindestbandbreiten garantiert werden können. ATM verzichtet auf Empfangsbestätigung der Pakete.

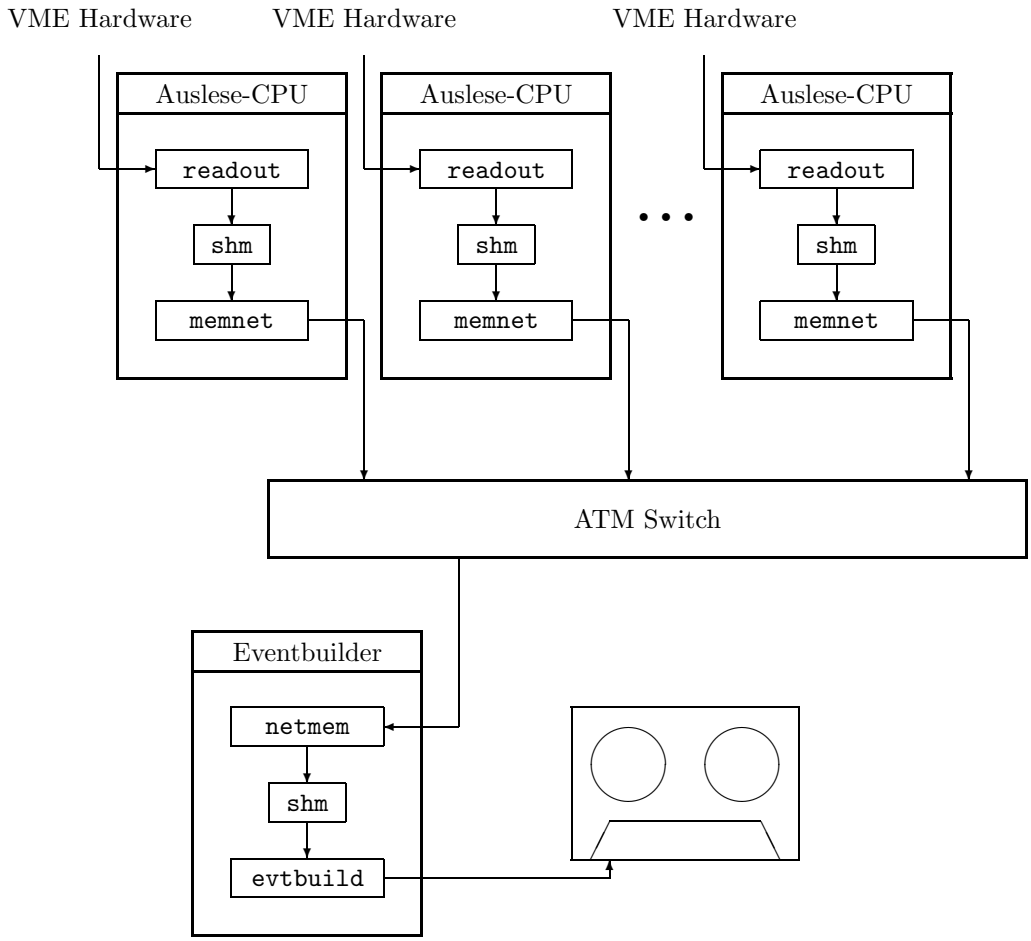

Abbildung 2.5: Datentransport mit ATM sowie Datenaufnahmeprozesse in Datenauslese-CPUs und Eventbuilder.

sowie im Verhältnis zu den Puffertiefen großen Datenraten erfordern eine kurze und garantierte Reaktionszeit des Betriebssystems. Bei Übertragungsraten von etwa 16 MBs^{-1} [Kya98] und einer Puffergröße von 256 KB bleiben dem Betriebssystem, das die Daten empfängt, lediglich etwa 15 ms, um die Daten abzuholen. Ein System, das Antwortzeiten garantieren kann, wird als echtzeitfähig bezeichnet.

Als ATM-Switch kam ein Fore ASX-200WG zum Einsatz. Die verwendeten ATM-Adapter waren ForeRunner®[10] 200E PCI-Karten im Eventbuilder und CES ATM 848x PMC-Karten auf den VME-CPUs.

[10]ForeRunner® ist ein eingetragenes Warenzeichen der Marconi Corporation plc.

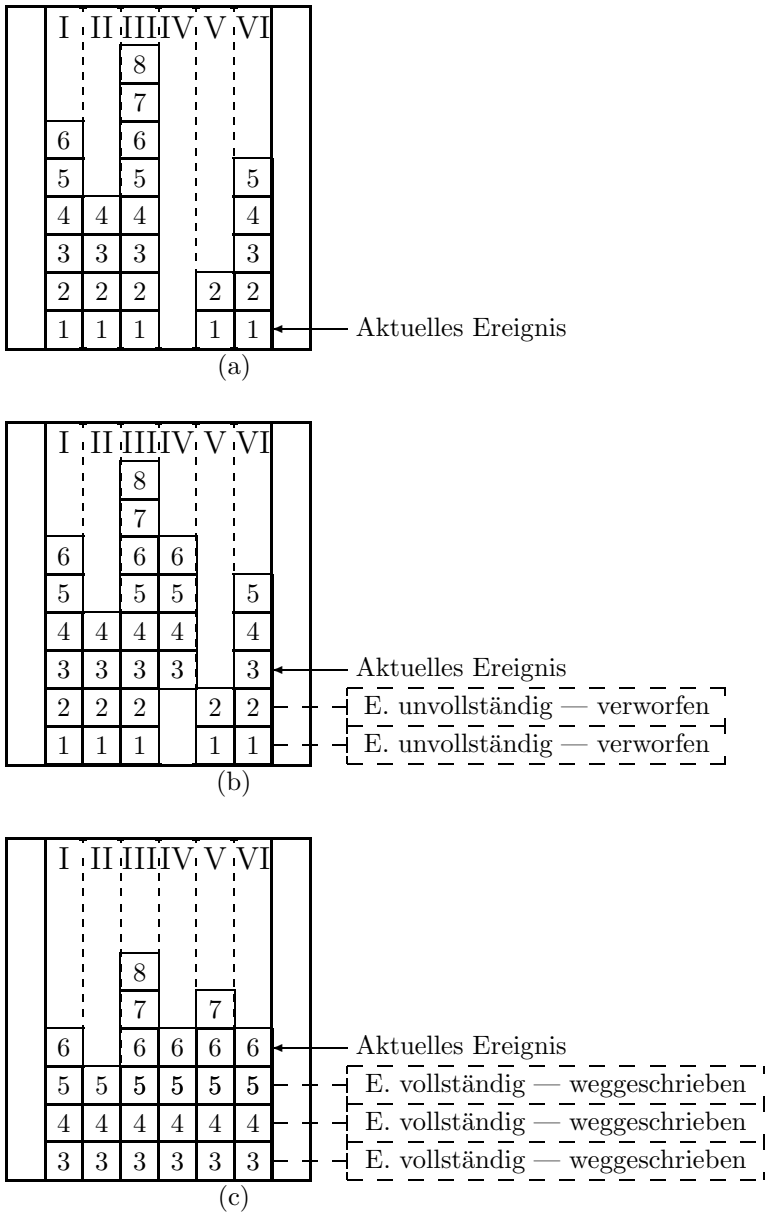

Abbildung 2.6: Prinzip des Eventbuildings — (a) Kein vollständiges Ereignis, Eventbuilder wartet auf ATM-Pakete; (b) Von Quelle IV fehlen die Ereignisse 1 und 2, alle übrigen Daten dieser Ereignisse werden verworfen, der Zähler für das aktuelle Ereignis wird auf „3“ erhöht; (c) Ein Paket aus Quelle V mit Daten zu den Ereignissen 3–7, Ereignisse 3–5 sind vollständig und werden weggeschrieben; Zähler für das aktuelle Ereignis wird auf „6“ erhöht.

Rechnerplattformen

Auf Seiten der Auslese-CPUs wird als Betriebssystem LynxOS[11] eingesetzt, welches auf VME-CPUs vom Typ RIO 8062 der Firma CES[12] läuft.

Im Bereich des Eventbuilders kann gängige Rechnerhardware verwendet werden, solange das Betriebssystem verfügbare ATM-Hardware unterstützt und die Echtzeitanforderungen erfüllt. Die wachsende Verbreitung und damit Unterstützung des freien Betriebssystems Linux, verbunden mit der Möglichkeit, damit günstige Hardware verwenden zu können, erlaubt es, dieses System für das Eventbuilding zu verwenden. In neueren Versionen ist sowohl eine Echtzeiterweiterung, als auch zumindest experimentelle Unterstützung des ATM-Protokolls, enthalten. Ebenso gibt es hier bereits seit längerem SMP[13]-Unterstützung, die den Einsatz von Mehrprozessorrechnern erlaubt. Konkret wurde eine x86-Doppelprozessor-Plattform eingesetzt, um den beiden Datenaufnahmeprozessen möglichst ohne Unterbrechung jeweils eine CPU zur Verfügung zu stellen. Ansonsten träten bei jedem Prozesswechsel Verzögerungen auf, die durch die eher lange garantierte Antwortzeit der Linux-Echtzeiterweiterung zum Datenverlust führen könnten. Um für jede Datenquelle einen genügend großen Speicherbereich zur Verfügung zu stellen, enthält der Eventbuilder zur Zeit 512 MB Hauptspeicher.

Geforderte Eigenschaft	Auslese-CPU	Eventbuilder
Echtzeitfähigkeit	√	√
Bussysteme	VME, PCI	PCI, PCI64
ATM	√	√
Bandzugriff	–	√
Hauptspeicher	32 MB bis 64 MB	512 MB bis 2 GB

Tabelle 2.3: Anforderungen an Hardware und Betriebssystem für den softwaregesteuerten Teil der Auslese und das Eventbuilding.

Die Strahlzeit im November 2000 hat darüberhinaus gezeigt, daß das sequentielle Schreiben auf Festplatte durch ein Dateisystem mit Datenpufferung für das Schreiben großer Datenmengen hinderlich ist. Festplatten, die normalerweise einen Datendurchsatz von $15\,\mathrm{MBs}^{-1}$ und mehr haben, können in dieser Hinsicht nicht mit einem leistungsfähigen Bandlaufwerk mithalten, obwohl das eingesetzte Quantum[14] DLT 8000 lediglich etwa $6\,\mathrm{MBs}^{-1}$ schreiben kann. Der Grund hierfür liegt in der Verwendung des Dateisystems, das versucht, die zu schreibenden Daten zunächst einmal zu puffern. Dies ist für die gängige Verwendung der Dateien zwar von Vorteil, da häufig gelesene und geschriebene Daten

[11] LynxOS ist ein eingetragenes Warenzeichen von Lynx Realtime Systems.
[12] CES ist ein eingetragenes Markenzeichen der Creative Electronic Systems SA.
[13] *S*ymmetric *M*ulti*p*rocessing
[14] Quantum ist ein eingetragenes Markenzeichen der Quantum Corporation.

einfach im Hauptspeicher verbleiben und damit sehr gute Zugriffszeiten die Regel sind. Die Datenaufnahme ist leider in dieser Hinsicht eine sehr untypische Anwendung: Sie schreibt eine große Menge Daten, ohne danach lesend auf sie zugreifen zu wollen. Das Dateisystem versucht nun, die Daten möglichst lange im Hauptspeicher zu halten, ohne tatsächlich auf Platte zu schreiben. Wenn kein Hauptspeicher mehr übrig ist, beginnt das Dateisystem nun verzögert, die Daten doch auf die Platte zu schreiben, allerdings wieder gerade so viele, wie notwendig sind, um weiterarbeiten zu können. Dieser Mechanismus, der die Daten einmal zusätzlich kopiert und den gesamten nicht anderweitig genutzten Hauptspeicher belegt, führt letztendlich zu der stark verringerten Schreibleistung auf die Festplatten. Wenn man das Dateisystem umgeht und direkt den Plattentreiber zum Beschreiben der Laufwerke verwendet, kann man dieses Problem umgehen.

Zusammenfassend sind in Abbildung 2.7 noch einmal die wichtigsten Elemente der Datenaufnahme in ihrer hierarischen Struktur dargestellt. Die Übersicht zeigt alle vier Stufen des Datenpfades innerhalb der HADES-Datenaufnahme. Die ersten beiden Schritte sind noch in Digitalelektronik auf der Basis konfigurierbarer Logik implementiert. Die anderen beiden Stufen der Datenaufnahme verwenden Software auf gängigen Rechnersystemen zum Datentransport.

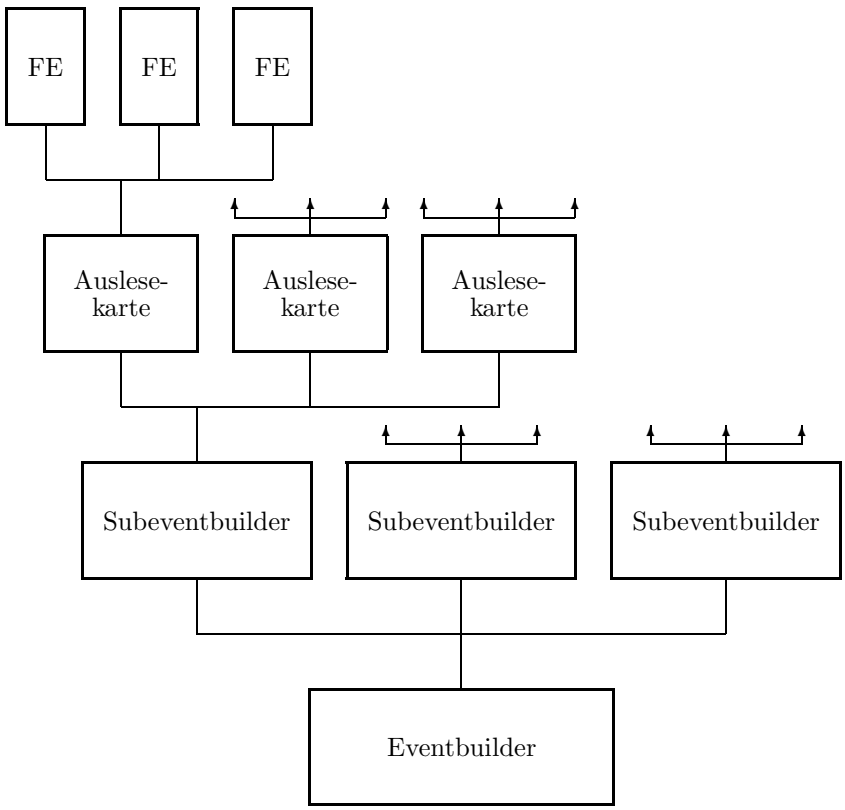

Abbildung 2.7: Überblick über die hierarchische Struktur der Datenaufnahme.

2.2.4 Software-Trigger

In der ursprünglichen Planung von HADES sollte eine weitere Datenreduktion durch eine dritte Triggerstufe erreicht werden. Diese sollte vollständig in Software realisiert werden. Analog zur Level-1-Pipe würde die Auslese der Daten aus der Level-2-Pipe erst nach positiv gefallener Entscheidung der dritten Triggerstufe stattfinden. Im Laufe der Entwicklung wurde die Leistung der Datenaufnahme über den Designwert gebracht. Wenn die Datenreduktion der zweiten Triggerstufe die geplanten Werte erreicht, wäre es damit schon jetzt möglich, alle Daten ohne eine dritte Triggerstufe aufzunehmen. Für den Fall, daß die zweite Triggerstufe die angestrebte Unterdrückungsraten nicht erreicht, bleibt die Option auf die dritte Triggerstufe notwendig.

Zusätzlich zu den in der zweiten Triggerstufe verwendeten Informationen aus den Detektoren zur Teilchenidentifikation sollen bei der dritten Triggerstufe auch noch Daten aus den MDC-Detektoren genutzt werden, die eine einfache Spurrekonstruktion („Tracking“) erlauben. Aus dieser erwartet man noch einmal eine um den Faktor 10 reduzierte Datenmenge. Für die technische Implementierung wäre ATM das optimale Werkzeug. Aufgrund der großen Bandbreitenreserven könnten nämlich alle weiteren Datenpfade, die nach der Extraktion der relevanten Daten aus dem Rohdatenstrom zur Triggerentscheidung nötig wären, über das ATM-Netzwerk laufen. Im Falle einer dritten Triggerstufe wäre eine dem Rechenaufwand angepaßte Menge an Tracking-Rechnern direkt über ATM an das Datenaufnahmesystem angebunden und würde wechselweise mit Ereignissen versorgt, über die sie dann die Triggerentscheidung treffen könnten, die wiederum über ATM an die Ausleserechner verteilt werden kann, wie Abbildung 2.8 skizziert.

Würde die dritte Triggerstufe nicht implementiert werden und die Datenmenge dadurch die Kapazitäten eines Eventbuilders überschreiten, könnte der gleiche Mechanismus auch die Lastverteilung für das Eventbuilding übernehmen.

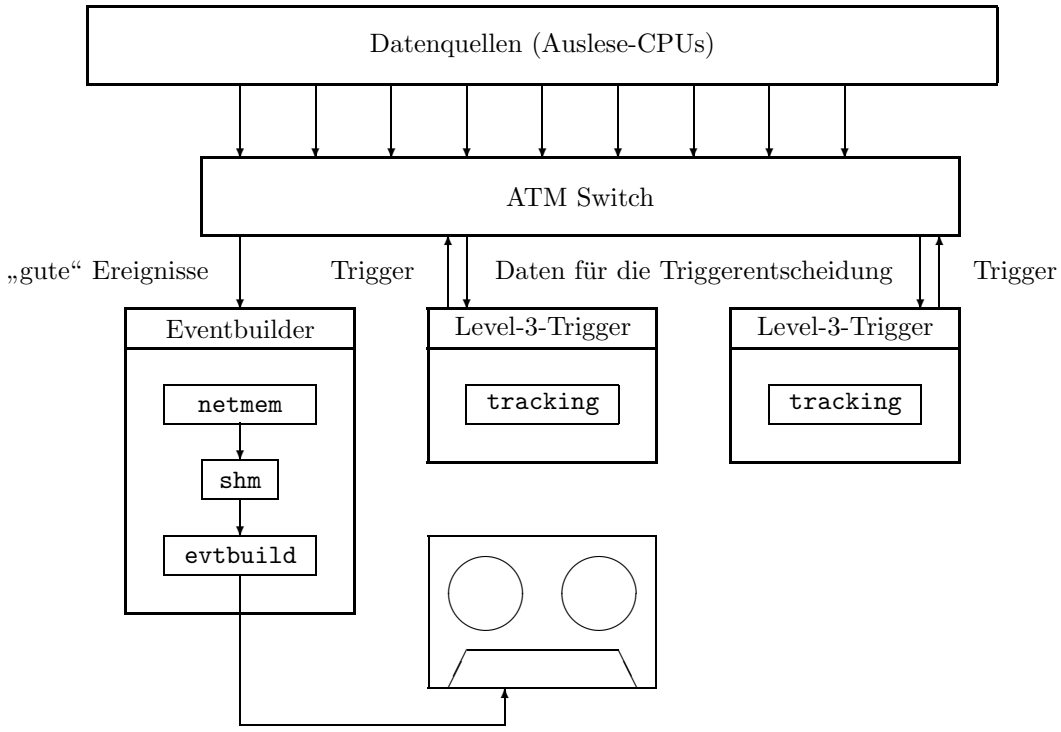

Abbildung 2.8: Möglicher Aufbau der dritten Triggerstufe.

Kapitel 3

Die Datenaufnahmesteuerung

3.1 Vorbetrachtungen

In diesem Abschnitt sollen allgemeine Anforderungen an die Datenaufnahmesteuerung erörtert werden. Dazu gehört vor allem die Analyse bereits existierender Lösungsansätze.

3.1.1 Aufgabenstellung

In Abbildung 2.7 wurde die logische Struktur der Datenaufnahme illustriert. Dies übersetzt sich für die Datenaufnahmesteuerung in einen etwas komplizierteren Baum, der in Abbildung 3.1 zu sehen ist. Daraus sind viele der Anforderungen an die Datenaufnahmesteuerung direkt ablesbar. Die konfigurierbare Logik der Auslese-Karten und der Frontends muß nach jeder Abschaltung neu geladen werden. Außerdem benötigt sie eine Reihe von Betriebsparametern, unter anderem eine Schwelle für jeden der 70 000 Kanäle, sowie teilweise Adressinformationen, die beide für die Nullenunterdrückung notwendig sind.

Die in der Software-Auslese eingesetzten Prozesse müssen genauso bedient werden wie die Hardware der Digitalelektronik. Dabei sind für die verschiedenen Phasen des des Betriebes jeweils eine Reihe von Bedingungen einzuhalten. Beispielsweise sind auf jedem Rechner die Prozesse in einer bestimmten Reihenfolge zu starten und in der umgekehrten Reihenfolge anzuhalten. Des weiteren muß beim Start der Datenaufnahme die CTU immer als letzte Komponente in Betrieb genommen werden.

Einige Regelungen sind außerdem noch von der aktiven Konfiguration abhängig. Je nachdem, welche Bildverarbeitungseinheiten aktiv sind, muß z.B. die MU unterschiedlich initialisiert werden. All diese Regelungen sind von der Datenaufnahmesteuerung anzuwenden und zu überwachen.

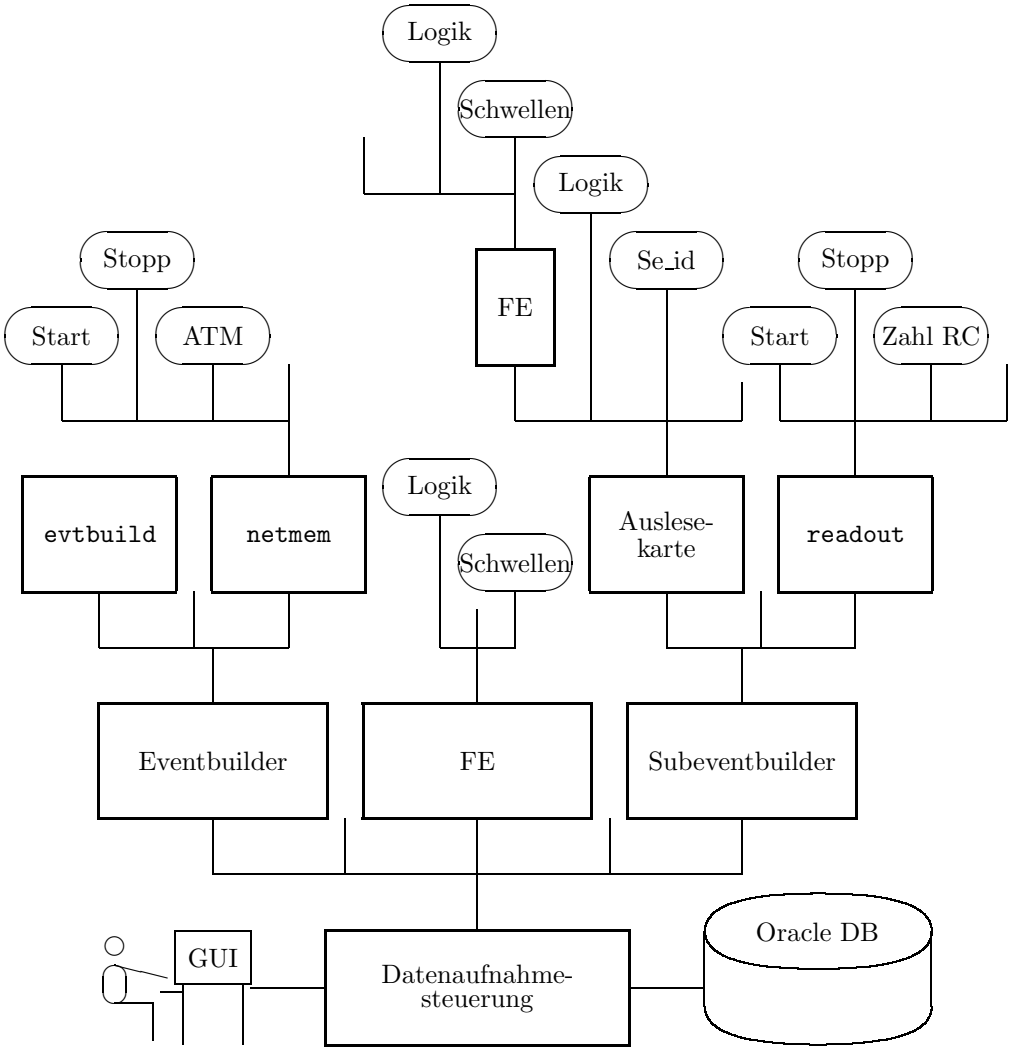

Abbildung 3.1: Die Struktur der Datenaufnahme aus Sicht der Datenaufnahmesteuerung; nur ein kleiner Teil der bereitzustellenden Funktionalität ist in den Blättern des Baumes dargestellt.

3.1.2 Andere Experimente

Zunächst erscheint ein Blick auf bereits implementierte Datenaufnahmesteuerungen sinnvoll.

Für die Ablaufsteuerung haben sich in einer ganzen Reihe von Großexperimenten, wie auch HADES eines ist, endliche Automaten (FSM[1]) als das Mittel der Wahl erwiesen. Die Implementierung unterscheidet sich häufig nur in Details. In [Mac96] werden eine Auswahl von Datenaufnahmesteuerungen verschiedener Experimente in Hinblick auf verwendete Konzepte und deren Vor- und Nachteile

[1] *F*inite *S*tate *M*achines

untersucht. Diese Untersuchung fand im Rahmen der Planung der Datenaufnahmesteuerung von ALICE [Mor99] statt und zeigt auch für die Problemstellungen der HADES-Datenaufnahmesteuerung Lösungsmöglichkeiten auf.

Im einzelnen wurden die folgenden Datenaufnahmesteuerungen analysiert:

ALEPH ALEPH wird von einer Anzahl statischer FSMs gesteuert, die die asynchrone Ausführung von Kontrollroutinen beherrschen. Der Hauptnachteil dieser Struktur ist die statische Implementierung, die eine einfache Erweiterung der Steuerung stark behindert.

DELPHI Die Datenaufnahmesteuerung von DELPHI besitzt eine hierarchische Struktur. Der Detektor wird in verschiedene „Domänen“ aufgeteilt, die jeweils ein Einzelsystem repräsentieren und von einem „State Manager“ kontrolliert werden. Jede dieser Domänen enthält wiederum FSMs, die das entsprechende Einzelsystem kontrollieren. Diese Struktur erlaubt eine einfachere Modifikation, da Änderungen durch die weitgehend autarken Domänen wesentlich weniger Auswirkungen auf den Rest der Detektorsteuerung haben.

ZEUS ZEUS verwendet eine Datenaufnahmesteuerung, die der von DELPHI ähnlich ist. Darüberhinaus werden Techniken der „künstlichen Intelligenz“ eingesetzt, um Fehler bei der Datennahme zu erkennen, ihre Ursachen mittels Heuristik zu ermitteln und die Probleme zu beheben. Alle grundlegenden Anforderungen von Benutzerseite werden erfüllt. Neben der grundsätzlichen Gefahr, das Programm gegenüber dem Operateur mit zu viel Kompetenz auszustatten, wurde das System zu komplex und unübersichtlich für die Wartung. Dies führte zu einer komplett neuen Implementierung.

CODA Die ehemals bei CEBAF[2] eingesetzte Datenaufnahme ist inklusive ihrer Steuerung eher eine Auswahl von Werkzeugen als ein fertiges System. Sie zeichnet sich durch die Flexibilität aus, für unterschiedliche Detektoren eingesetzt zu werden.

BaBar BaBars Datenaufnahmesteuerung ist der von DELPHI sehr ähnlich, was nicht zuletzt daher rührt, daß sowohl Konzepte, als auch konkrete Implementierungen von DELPHI wiederverwendet werden.

3.1.3 Hilfesystem

Zuletzt ist noch das Hilfesystem zu erwähnen, das wie für jedes gute Softwarepaket auch für die Datenaufnahmesteuerung wichtig ist. Die Online-Hilfe sollte sich in das jeweilige Hilfesystem des Betriebssystems einfügen, um dem Benutzer einen intuitiven Zugang zu ermöglichen. Verbreitet sind bei UNIX-Systemen von allem

[2] heute Thomas Jefferson Laboratory.

die Manual-Seiten[3] sowie die durch die Verbreitung des WWW in letzter Zeit immer beliebteren HTML-Hilfen[4], die den Vorteil der Plattformunabhängigkeit genießen.

Das Hilfesystem soll es dem Benutzer erlauben, die Datenaufnahmesteuerung selbständig in Betrieb zu nehmen und zu bedienen. Im Idealfall werden auch Lösungen für häufige, einfache Fehlersituationen beschrieben.

[3] über den UNIX-Befehl `man` verfügbar gemachte Hilfe-Dateien.
[4] HTML — *H*yper*T*ext *M*arkup *L*anguage

3.2 Organisation der Datenaufnahmesteuerung

Im folgenden soll betrachtet werden, wie die Aufgaben der Datenaufnahmesteuerung durch bekannte Konzepte aus der Informatik beschrieben und gelöst werden können.

3.2.1 Verteilte Systeme

Da die Datenaufnahme selbst ein verteiltes System ist, wird automatisch auch ihre Steuerung zu einem solchen. Auf jedem Rechner, auf dem Teile des Datenaufnahmesystems (Hard- und Software) gesteuert und überwacht werden sollen, muß ein Prozess laufen, der diese Aufgaben übernimmt und selbst wiederum von zentraler Stelle bedient werden kann. Solch ein Prozess wird Agent genannt[5]. Ein Ensemble mehrerer Agenten mit voneinander unabhängigen, aber zentral steuerbaren Prozessen heißt auch „Multi-Agenten-System“.

Ein verteiltes System ist nach [Lan94] und [Slo89] definiert als eine Gruppe von mehreren Prozessorelementen, auch Knoten genannt, sowie Speichereinheiten, die alle mittels Nachrichten miteinander kommunizieren. Das Nachrichtensystem wird als „Message Passing System“ bezeichnet. Die Verbindungen, die zwischen den einzelnen Knoten und Speichereinheiten bestehen, bilden insgesamt das Netzwerk.

Im Gegensatz zu einem hochintegrierten System erfordert ein verteiltes System neben einer guten Leistungsfähigkeit der Prozessoreinheit selbst auch ein schnelles Netzwerk[6], wobei je nach Anwendung die Gewichtung dieser beiden Anforderungen unterschiedlich ist. Je stärker ein Problem parallelisierbar ist, desto weniger Kommunikation zwischen den einzelnen Knoten ist erforderlich und desto weniger Netzwerkleistung ist notwendig.

Im Falle der HADES-Datenaufnahmesteuerung ist die Leistung nicht entscheidend, da die Anzahl der Aktionen, die pro Sekunde ausgeführt werden müssen, gering ist. Schließlich ist die Datenaufnahmesteuerung ein Bestandteil der „Slow Control“, die nur die langsam veränderlichen Parameter des Systems überwacht. Speziell die Last, die die Steuerungsprozesse für die CPU bedeuten, sollte in dieser Hinsicht vernachlässigbar sein, aber auch die Belastung des Netzwerkes ist gering. Dafür sollte großer Wert auf Ausfallsicherheit gelegt werden.

Die Datenaufnahmesteuerung selbst beinhaltet bis jetzt noch keine Prozesssteuerung in dem Sinne, daß eine zentrale Stelle über die Ausführung von Steue-

[5]Die Bezeichnung „Agent“ ist ebenso wie die Bezeichnung „Server“ nicht eindeutig. Sowohl das Programm mit oben beschriebener Funktionalität, als auch der Rechner, auf dem dieses Programm läuft, wird häufig als „Agent“ bzw. „Server“ bezeichnet.

[6]Wobei die Geschwindigkeit eines Netzwerkes wiederum durch zwei unabhängige Kennzahlen beschrieben wird: Zum einen die Latenzzeit (die Zeit, die zum Transport einer Information benötigt wird) und zum anderen der Datendurchsatz (die Menge an Daten, die pro Zeiteinheit transportiert werden können).

rungsprozessen wacht. Anstelle einer genauen Kontrolle, wo welcher Steuerungsprozess wann gestartet oder gestoppt wird, wird angestrebt, die Steuerungsprozesse als Serverprozesse andauernd laufen zu lassen (beginnend mit dem Systemstart) und die Änderungen von Konfigurationen während der Laufzeit zu erlauben. Dies ist nicht zu verwechseln mit dem Start und Stopp der Datenaufnahmeprozesse, die sehr wohl notwendig ist.

Eine zentrale Kontrolle der Steuerungsprozesse, wie sie in der vorher bereits vorhandenen Datenaufnahmesteuerung mittels RPC[7] [Blo92] implementiert war, ist eigentlich nur notwendig, wenn von dieser zentralen Stelle aus auch Konfigurationsparameter verteilt werden. Dies führt aber dazu, daß veränderte Parameter entweder mitverteilt werden müssen oder eine Änderung der Parameter immer mit einem Neustart aller Agenten einhergehen muß (wie es bei der alten Implementation der Fall war).

3.2.2 Ein endlicher Automat als Ablaufsteuerung

Die Möglichkeit, die Elemente der Datenaufnahme einzeln ansteuern zu können, macht noch keine benutzbare Datenaufnahmesteuerung aus. Im Falle der HADES Datenaufnahme ergeben sich unter Berücksichtigung aller Hardwarekomponenten und einzelnen Datenaufnahmeprozesse eine Anzahl von weit über 300 zu bedienenden Parametern, die zum großen Teil voneinander abhängig sind und in einer bestimmten Reihenfolge an- bzw. ausgeschaltet werden müssen. Darüberhinaus ändert sich die Systemkonfiguration gerade in der Aufbau- und Erprobungsphase des Detektors und der Elektronik sowie bei späteren Erweiterungen und Spezialmessungen noch häufig, wodurch auch die Abhängigkeiten der einzelnen gesteuerten Einheiten untereinander beeinflußt werden.

Aus diesem Grund ist ein weiterer Bestandteil der Datenaufnahmesteuerung notwendig: Eine Ablaufsteuerung, die es durch die Festlegung bestimmter Regeln ermöglicht, immer wiederkehrende Abläufe zu automatisieren und die Integrität des Systems zu gewährleisten. Zur Lösung derartiger Problemstellungen bietet die Informatik die Automatentheorie.

Ein endlicher Automat ist definiert als eine Menge von Zuständen und Übergängen zwischen diesen, die durch ein sogenanntes Eingabealphabet ausgelöst werden. Der Grund dafür, daß die endlichen Automaten ein sehr formal beschriebenes Gebiet der Informatik darstellen, liegt darin, daß sie eigentlich ein Mittel zur Verifizierung formaler Sprachen darstellen. Wie aber schon die Bezeichnung vermuten läßt, ist dies weder die einzige noch die ursprüngliche Anwendung der endlichen Automaten. Bei der Datenaufnahmesteuerung wird vor allem die Eigenschaft der endlichen Automaten ausgenutzt, daß durch die Festlegung des Automaten in der Beschreibung bereits gewährleistet wird, daß sich das System vom Standpunkt der Steuerung aus immer in einem definierten (und erlaubten)

[7] *R*emote *P*rocedure *C*all; ein Modell zur Steuerung verteilter Prozesse.

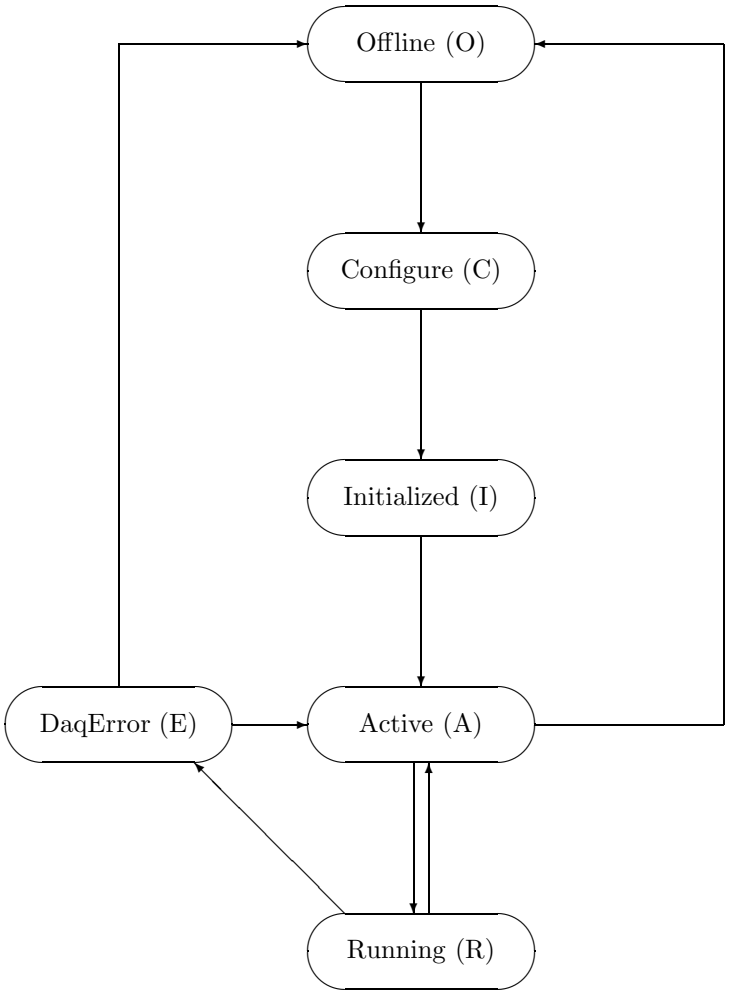

Abbildung 3.2: Beispiel eines endlichen Automaten zur Steuerung einer Datenaufnahme.

Zustand befindet. Übergänge, die nicht von vorneherein vorgesehen worden sind, können nicht durch Fehlbedienung des Automaten erreicht werden. Formal wird ein endlicher Automat durch ein 5-Tupel

$$(Q, \Sigma, \delta, q_0, F)$$

beschrieben. Hierbei sind

Q die Menge der Zustände des Automaten. Q ist endlich im Falle eines endlichen Automaten.

Σ das Eingabealphabet. Σ ist immer endlich (auch bei einem nicht endlichen Automaten).

δ die Übergangsfunktion, eine Abbildung von $Q \times \Sigma$ auf Q. Über sie ist definiert, in welchen Zustand der Automat bei Eingabe eines Zeichens aus dem Eingabealphabet übergeht, und zwar abhängig vom aktuellen Zustand.

q_0 der Anfangszustand, ein Element aus Q.

F die Menge der Endzustände, eine Teilmenge von Q.

Das Erreichen eines Endzustandes ist in der Automatentheorie nicht gleichbedeutend mit dem Ende der Automatentätigkeit. Stattdessen dienen die Endzustände lediglich zur Verifikation, ob ein Eingabewort gültig, d.h. ein Wort der durch den Automaten definierten formalen Sprache ist. In unserem Fall ist eine derartige Verifikation nicht notwendig. Sie würde lediglich dann wichtig werden, wenn wir den Automaten wirklich zum Erkennen einer formalen Sprache einsetzen würden. Damit verliert die Menge der Endzustände für die Datenaufnahmesteuerung ihre Bedeutung. Der Anfangszustand ist im Beispiel von Abbildung 3.2 der Zustand „Offline“, die Definition der Endzustände ist willkürlich und könnte lediglich aus intuitiven Gründen „Active“ sein.

Das Eingabealphabet ist eine endliche Menge von Zeichen, die der Automat akzeptieren kann. In unserem Beispiel können alle dem Automaten übergebenen Steuersignale ebenso als Elemente des Eingabealphabets aufgefaßt werden, wie die Signale, die der Automat durch die Überwachung des Datenaufnahmesystems erhält. Das komplexeste Element des 5-Tupels ist die Übergangsfunktion. Sie ordnet im allgemeinen einer Untermenge von $(Q \times \Sigma)$, also einem Teil der Kombinationen aus Zuständen und Zeichen des Eingabealphabets, Übergänge in neue Zustände zu. In der realen Situation ist ein solcher Übergang auch mit einer Menge von Aktionen verbunden, die durchgeführt werden. Diese beinhalten im Beispiel der Ablaufsteuerung die Bedienung der Datenaufnahmeelemente. Im abstrakten Fall spielen diese Aktionen keine Rolle, von Bedeutung ist lediglich das Ergebnis, also der Zustand nach dem Übergang. Den Definitionen in [Hop79] folgend kann die Übergangsfunktion immer vollständig ergänzt werden, so daß alle Symbole, die während eines Zustandes keinem Übergang zugeordnet sind, den Automaten einfach wieder in den selben Zustand überführen. Diese Erweiterung wird bei uns genutzt. Dagegen sind Spezialfälle wie nichtdeterministische Automaten[8] und ϵ-Übergänge[9] nicht notwendig.

Anhand des Beispiels aus Abbildung 3.2 sollen die Begriffe erläutert werden. Die Menge der Zustände kann daraus direkt entnommen werden:

$$Q = \{\mathrm{O}, \mathrm{C}, \mathrm{I}, \mathrm{A}, \mathrm{E}, \mathrm{R}\}.$$

[8]Bei denen die Übergangsfunktion zu einer Relation verallgemeinert wird. Damit können einer Kombination $(q, a) \in Q \times \Sigma$ auch mehrere Übergänge zugeordnet werden.

[9]Übergänge, die kein Element aus Σ als Eingabe erfordern.

Auch der Anfangszustand wurde bereits erwähnt:

$$q_0 = \mathrm{O}.$$

Die Menge der Endzustände ist unerheblich kann wieder beliebig gewählt werden:

$$F = \{\mathrm{A}\}.$$

Die Ereignisse, auf die der Automat reagieren soll, teilen sich in zwei Kategorien auf: Auf der einen Seite sind dies Anweisungen, die vom Operateur an die Datenaufnahmesteuerung gegeben werden, auf der anderen Seite stehen die Rückmeldungen der Datenaufnahme (sowohl im normalen Betrieb als auch im Fehlerfall). In die erste Kategorie gehören die Befehle

- Initialize (i)
- Activate (a)
- Start (s)
- Halt (h)
- Recover from Error (r)
- Go offline (o)

Die Meldungen der Datenaufnahme sind

- Configuration ready (c)
- DaqError detected (e)

Also ist

$$\Sigma = \{\mathrm{i}, \mathrm{a}, \mathrm{s}, \mathrm{h}, \mathrm{r}, \mathrm{o}, c, e\}.$$

Tabelle 3.1 stellt die Übergangsfunktion dar, die sich aus dem Sachverhalt ergibt. Dies definiert den Automaten und damit auch sein Verhalten vollständig.

3.2.3 Konfiguration mit Hilfe von Parametern

Sowohl die Agenten des verteilten Systems, als auch die endlichen Automaten, enthalten keine statischen Informationen über den zu steuernden Systemaufbau, sondern gewinnen diese erst zur Laufzeit. Dazu dient eine externe, zentrale Quelle, die die notwendigen Informationen, die im folgenden als (Betriebs-)Parameter bezeichnet werden, bereitstellt. Es liegt nahe, das Kommunikationssystem des verteilten Systems auch zum Transport dieser Betriebsparameter zu verwenden.

	O	C	I	A	R	E
i	C	C	I	A	R	E
a	O	C	A	A	R	E
s	O	C	I	R	R	E
h	O	C	I	A	A	E
r	O	C	I	A	R	A
o	O	C	I	O	R	O
c	O	I	I	A	R	E
e	O	C	I	A	E	E

Tabelle 3.1: Die Übergangsfunktion δ; links ist das Eingabealphabet, oben die Zustände aufgetragen. Die meisten Felder enthalten als Eintrag den Ausgangszustand, d.h. das eingegebene Zeichen ändert nichts am Systemzustand.

3.2.4 Datenbank und Datenbankanbindung

Als Quelle für die Parameter wird ein weiterer Agent verwendet, der seinerseits verschiedene Quellen konsultieren kann. Zum einen soll es möglich sein, die Parameter in Form von Dateien zur Verfügung zu stellen, in denen die Parameter linear aufgelistet sind. Zum anderen (und dies ist das eigentliche Ziel) sollte die Quelle der Parameter die eigens für HADES entwickelte Datenbank sein. Diese Modularität erlaubt sowohl ein Arbeiten mit der zentralen Datenbank, als auch einen Wechsel zu einer davon unabhängigen Konfiguration — was zumindest für die Entwickler außerhalb der GSI immer möglich bleiben sollte — ohne Änderungen an der Software.

In der Datenbank werden diese Informationen unter anderem bereits für Analysezwecke gehalten. Aus Gründen der Datenkonsistenz ist es also dringend zu empfehlen, dieselben Parameter auch für die Datenaufnahme zu verwenden [Mat98]. Dies erfordert neben der Anbindung des Agenten an die Datenbank das Design eines Teiles der Datenbank selbst (wobei mit Datenbank in diesem Falle die Datenstruktur — speziell die Struktur der Tabellen, Ansichten etc. — gemeint ist[10]).

[10]Der Begriff Datenbank ist vieldeutig. Auf der einen Seite beschreibt er die Software, die es ermöglicht, auf den Datenbestand strukturiert zuzugreifen; die korrekte Bezeichnung hierfür ist (*R*elationales) *D*aten*b*ank *M*anagement *S*ystem ((R)DBMS). Des weiteren kann mit Datenbank auch der konkrete Datenbestand gemeint sein, auf den man mit dem RDBMS zugreift. In dieser Weise soll im folgenden der Begriff Datenbank verwendet werden.

3.3 Verwendete Werkzeuge

Im vorangegangenen Abschnitt wurden die Anforderungen festgelegt, die an die Datenaufnahmesteuerung gestellt werden. Im folgenden werden die aus diesen Anforderungen und weiteren Bedingungen (wie z.B. der Situation der bereits relativ weit implementierten Detektorsteuerung) resultierenden Entscheidungen über die eingesetzten Werkzeuge beleuchtet sowie letztere vorgestellt.

Die dominierende Rolle hierbei spielt EPICS[11]. Seine Komponenten werden sowohl als Message Passing System, als auch als Zugriffsfunktionen auf Datenaufnahmehard- und -software eingesetzt. Der EPICS `sequencer` wird sogar als Ablaufsteuerung im Sinne eines endlichen Automaten verwendet.

Die HADES Analyse hatte bereits Oracle 8.0.x für Linux als RDBMS ausgewählt. Um die in Kapitel 3.2 erwähnte Datenkonsistenz sicherzustellen, bietet sich eine Nutzung dieser Datenbank für die Datenaufnahmesteuerung an. In der Tat erfüllt Oracle auch alle Anforderungen, die generell an ein RDBMS gestellt werden: Es unterstützt mit SQL[12] den verbreitetesten Abfragestandard und skaliert als professionelles Produkt sehr gut, was bei intensiver Nutzung von entscheidender Bedeutung sein kann.

3.3.1 EPICS

Nachdem die Detektorsteuerung auf EPICS basiert, lag es nahe, dessen Eignung auch in Bezug auf die Datenaufnahmesteuerung näher zu untersuchen. EPICS ist in erster Linie eine Client-Server-Anwendung zur Steuerung von VME-Hardwarekomponenten. Es wurde ursprünglich als Steuerung für Beschleunigeranlagen entwickelt. Verschiedene Gruppen an den Beschleunigern des ANL[13], des CERN und von DESY[14] übernahmen dabei Teilbereiche der Entwicklung und führen sie bis heute fort.

Das zentrale Konzept, auf dem die gesamte Kommunikation von EPICS basiert, sind Prozessvariablen (PVs). Prozessvariablen sind in der Regel entweder die Repräsentation eines Hardwareobjektes (zum Beispiel eines Schrittmotors) oder aus anderen Prozessvariablen errechnete Werte. Sie werden alleine über ihren Namen identifiziert und sind im ganzen Netzwerk transparent verfügbar, d.h. der Client muß nicht wissen, welcher Server die Prozessvariable bereitstellt. Die grundsätzlichen Operationen, die auf die Prozessvariablen angewendet werden können, sind:

[11] *E*xperimental *P*hysics and *I*ndustrial *C*ontrol *S*ystem

[12] *S*tructured *Q*uery *L*anguage — eine im ISO-Standard 9075 definierte sogenannte Sprache der 4. Generation (4GL). Die aktuelle Definition stammt von 1997 und wird als SQL3 bezeichnet.

[13] *A*rgonne *N*ational *L*aboratory

[14] *D*eutsches *E*lektronen-*Sy*nchrotron

Read Ein Wert wird gelesen. Im Falle des Schrittmotors könnte das Lesen der Prozessvariablen etwa die Position zurückliefern. Je nach Abfrage würde sie entweder neu gemessen oder der letzte Meßwert übertragen.

Write Ein Wert wird geändert. Der Schrittmotor zum Beispiel könnte damit den Befehl erhalten, in eine neue Stellung zu fahren.

Monitor Über Änderungen des Wertes wird informiert. Wenn der Schrittmotor (auf Weisung eines anderen Clients) verfahren wird, benachrichtigt EPICS denjenigen, der den Monitor eingeschaltet hat, über die ausgeführte Änderung.

Prozessvariablen enthalten nicht nur ihren Wert, sondern weitere Informationen, wie zum Beispiel

- die Einheit, in der dieser Wert zu verstehen ist
- Grenzen, innerhalb derer sich der Wert befinden soll
- Einen Alarm-Status, der anzeigt, ob sich der Wert auch innerhalb dieser Grenzen befindet
- Die Zeit des letzten Lesevorgangs

Die wesentlichen Komponenten von EPICS, die die Benutzung der Prozessvariablen ermöglichen, sind:

IOC Der Standard-Server; er stellt im Netz PVs zur Verfügung, die den Zustand von Hardware repräsentieren und durch die Manipulation der Variablen die Hardware steuern lassen. Bis zur Version 3.13.3 stand der IOC nur für das Echtzeitbetriebssystem VxWorks zur Verfügung, für das wiederum keine Datenaufnahme existiert. Zum IOC gehört eine ganze Reihe von Hardware-Treibern, die es erlauben, auf Standard-VME-Hardware zuzugreifen, ohne die Zugriffe selbst programmieren zu müssen.

ca Channel Access; das Message Passing System von EPICS. Die entsprechende Bibliothek übernimmt sowohl die Suche nach den PVs als auch den Datentransport über das Netzwerk und die automatische Konvertierung verschiedener Variablentypen ineinander.

`cas` Channel Access Server; von EPICS vorgeschlagene Schnittstelle zur Entwicklung eigener Server unter anderen Betriebssystemen als VxWorks.

Db database; Definition der von einem IOC bereitgestellten PVs auf der Seite des Channel Access; dazu gehören auch Beziehungen zwischen PVs. Darüber und mit dem Einsatz verschiedener PVs ohne direkte Hardwareanbindung können logische und arithmetische Operationen mit den PVs direkt in die database-Definitionen gesteckt werden.

`medm` Auf der Graphik-Bibliothek Motif aufbauende Entwicklungsumgebung für GUIs[15] sowie Darstellungspaket selbst; nicht Bestandteil des EPICS Basispaketes. In EPICS werden Benutzerschnittstellen oft auch als OPI[16] bezeichnet.

`sequencer` Enthält den Präcompiler[17] `snc`[18], der aus einer Beschreibung endlicher Automaten C-Code erstellt; aus diesem entsteht durch Kompilation ein endlicher Automat mit Anbindung an PVs über Channel Access. Seit R3.14 (`sequencer` Version 2.0.0) ist er nicht mehr Bestandteil des Basispaketes.

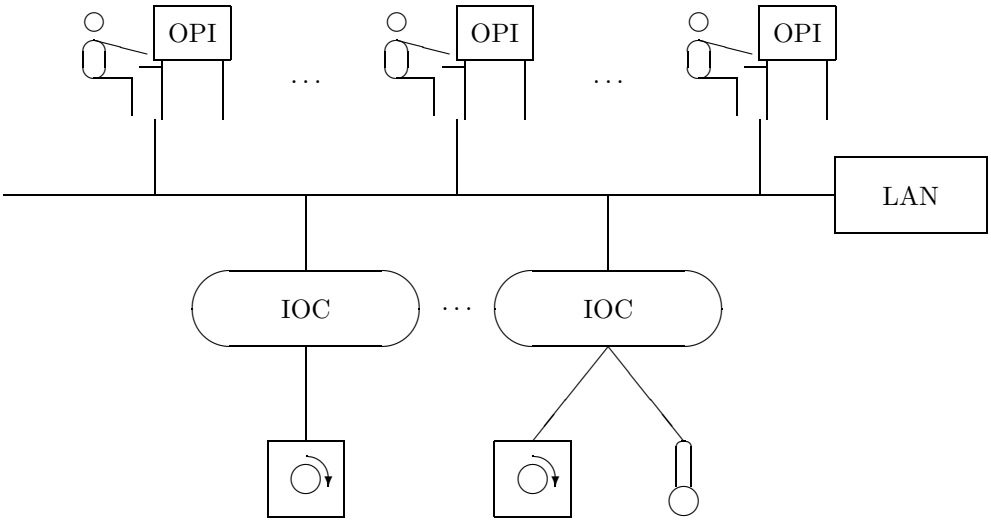

Abbildung 3.3: Organisation von EPICS [Dal00]; Die Client-Anwendungen stellen die Schnittstelle zu den Operateuren dar, während die IOCs die Hardware (in diesem Falle zwei Schrittmotoren und ein Temperaturmeßfühler) bedienen und / oder überwachen.

[15] *G*raphical *U*ser *I*nterfaces

[16] *Op*erator *I*nterface

[17] Ein Präcompiler ist ein Programm, das Quellcode einer Programmiersprache in Quellcode einer anderen übersetzt. Im Gegensatz dazu erzeugt ein Compiler ausführbaren Code (also Maschinensprache). Allerdings ist diese Unterteilung nicht frei von Ausnahmen. Ein typischer C-Compiler zum Beispiel erzeugt Assembler-Code, der dann erst von einem Assembler ist Maschinensprache übersetzt wird.

[18] *S*tate *N*otation *C*ompiler

Eigene Entwicklung mit Channel Access / `cas`

Da der IOC nur für VxWorks zur Verfügung stand, insbesondere die Datenaufnahmesoftware aber nur für POSIX[19]-Betriebssysteme vorhanden war, wurden die Funktionen zur Steuerung der Soft- und Hardware nicht als Treiber für den IOC, sondern als Bestandteile in den `cas` eingebunden. Für diese Komponente existierte bereits eine Beispielimplementierung, die als Basis zur Entwicklung der Agenten diente.

Die explizite Verwendung der Funktionen aus der `ca`-Bibliothek war lediglich für die Parameterbibliothek notwendig, da die verwendeten, im folgenden näher beschriebenen Client-Anwendungen die Zugriffe auf Prozessvariablen bereits enthalten.

Der `sequencer`

Eine dieser Client-Anwendungen sind die endlichen Automaten der Ablaufsteuerung, die mit Hilfe des `sequencer`-Paketes entwickelt werden können. Die neueste offiziellen Version von EPICS (R3.13.3) ermöglicht die Entwicklung von FSMs lediglich für VxWorks. Die angekündigte Version R3.14 bietet diese Möglichkeit auch für POSIX-konforme Betriebssysteme, so daß in Hinblick auf eine später mögliche Migration das `sequencer`-Paket eingesetzt wurde. Zur Definition der endlichen Automaten dient in diesem Paket SNL[20]. SNL ist eine Beschreibungssprache für endliche Automaten. In ihr können Zustände (States) definiert werden, zu denen man jeweils erlaubte Übergänge (Transitions) sowie die Bedingungen angibt, unter denen die Übergänge stattfinden. Diese Sprache beinhaltet eine Untermenge von C und verwendet auch ähnliche Sprachkonstrukte. Tatsächlich aber wird der SNL-Code von einem Präcompiler mit dem Namen `snc` erst in C-Code umgewandelt, der wiederum mit einem normalen C-Compiler (bzw. oben erwähntem Cross-Compiler) kompiliert wird.

Die vom `sequencer`-Paket erzeugten Automaten akzeptieren als Eingabealphabet im wesentlichen Änderungen von Prozessvariablen, die sie überwachen. Ein Ausschnitt des SNL-Codes, der den in Kapitel 3.2.2 beschrieben Automaten steuern soll, könnte etwa wie im folgenden aussehen:

Zunächst werden den relevanten Prozessvariablen Programmvariablen zugewiesen und der Monitor aktiviert:

```
pvAssign(error, "DAQERROR_DETECTED");
pvMonitor(error);
```

[19] POSIX ist die gängige Bezeichnung für den IEEE Std 1003.1-1990 [Lew91] Standard, der Schnittstellen zur Benutzung von und Programmierung für Betriebssysteme festlegt. Praktisch alle gängigen UNIX-Derivate halten sich in weiten Teilen an diesen Standard, um portierbare Programmierung von Anwendungen zu erleichtern. Auch die Datenaufnahme von HADES verwendet ausschließlich POSIX-Programmierschnittstellen.

[20] *S*tate *N*otation *L*anguage

Jeder Zustand enthält eine Liste der von ihm ausgehenden Übergänge, jeweils mit der dazu notwendigen Bedingung, die ein Zeichen des Eingabealphabets darstellt.

```
state RUNNING {
    when (error == 1) {
        printf("Transition to State DAQERROR\n");
    } state DAQERROR

⋮

}
state DAQERROR {
```

Auf diese Weise kann der Automat sowohl Aufgaben der Steuerung, als auch der Regelung wahrnehmen. Im ersten Fall sind die überwachten Variablen von den Experimentatoren zu setzen, im zweiten Fall enthalten sie Informationen über die geregelten Komponenten.

Der `medm`

Die zweite Client-Anwendung, die für die Datenaufnahmesteuerung benötigt wird, ist die graphische Benutzeroberfläche. Generell sind eine ganze Reihe von Werkzeugen zur Erstellung graphischer Benutzeroberflächen verfügbar. Allerdings wäre die generische Unterstützung von Channel Access eine große Erleichterung für die Entwicklung. EPICS bietet hier vor allem zwei Wege an: Die Motif-basierte `medm`-Oberfläche, deren einzelne Masken ohne Programmierung von der gleichen Anwendung generiert werden können, in der sie dann auch dargestellt werden, und zum anderen EPICS-spezifische Erweiterungen für Tcl/Tk [Ous94], das eine Kombination aus Skriptsprache und graphischer Bibliothek ist.

Während der Strahlzeit im November 2000 wurde der `medm` als Grundlage für das GUI gewählt. Dies geschah zunächst aufgrund der bereits implementierten Detektorsteuerung. Um der variablen Natur der Datenaufnahme Rechnung zu tragen, wird die Tatsache ausgenutzt, daß die Struktur der `medm`-Masken in sogenannten adl-Dateien[21] abgelegt ist. Diese offene Schnittstelle ermöglicht es, auf einfache Weise ein Programm zu erstellen, das die aktuelle Konfiguration aus der Parameterquelle einliest und die adl-Dateien und damit die `medm`-Masken dynamisch erzeugt.

Eigentlich ist allerdings ein Werkzeug zu bevorzugen, das diese Konfiguration selbst übernehmen kann. Die im oben erwähnten Zwischenschritt dynamisch erzeugten Dateien haben nämlich den Nachteil, daß zu manchen Zeitpunkten die (aktuelle und damit korrekte) Information in der primären Parameterquelle

[21] ADL — *A*SCII *D*isplay *L*ist

nicht mit dem Inhalt der adl-Dateien übereinstimmt. Dies ist ein inkonsistenter Zustand, der eigentlich durch den Einsatz der zentralen Verwaltung der Konfiguration verhindert werden sollte. Es bleibt also zu untersuchen, ob die Tcl/Tk-Schnittstelle von EPICS diesen Anforderungen zur dynamischen Konfiguration eher gewachsen ist.

Namenskonvention der Prozessvariablen

Da die einzige Information, die für den Zugriff auf eine Variable benötigt wird, ihr Name ist, kann durch eine durchdachte Namenskonvention für die Variablen der Zugriff standardisiert werden.

Die Namen der PVs werden bei der HADES Datenaufnahmesteuerung aus vier Bestandteilen zusammengesetzt, die voneinander jeweils durch „:“ getrennt werden. Der erste Teil lautet immer „HAD“ und zeigt die Zugehörigkeit zu HADES an. Der nächste Teil identifiziert das Subsystem des Detektors. Dies kann zum Beispiel „RICH0“ sein. Zu beachten ist hierbei, daß der RICH nicht ein, sondern drei Subsysteme stellt, da er drei VME-Crates besitzt. Der dritte Bestandteil des Namens ist die HADES-weit eindeutige Bezeichnung für eine einzelne Komponente. Dies könnte zum Beispiel ein Ausleseprozess sein, der „RICH0GP“[22] heißt. Der letzte Namensbestandteil ist das Attribut, das durch die Variable repräsentiert werden soll, also zum Beispiel „INIT“, der Status der Initialisierung. Somit ergibt sich als Name „HAD:RICH0:RICH0GP:INIT“. Für Betriebsparameter tritt anstelle des Subsystems die Bezeichnung „PARAM“. Eine Abfrage, die eine willkürlichen Größe „size“ für obigen Ausleseprozess herausfinden soll, übersetzt sich also in den PV-Namen „HAD:PARAM:RICH0GP:SIZE“.

3.3.2 Oracle

Das eingesetzte RDBMS Oracle enthält neben dem eigentlichen Datenbankkern, der sich um die Datenhaltung und die Extraktion der Daten mittels SQL kümmert, noch eine ganze Reihe von Schnittstellen, die es erlauben, von externen Anwendungen aus auf die Daten zuzugreifen. Zur Extraktion der Daten für die Parameterbibliothek wurde der Präcompiler `ProC/C++` verwendet. Diese einfache Schnittstelle bietet zwar eine geringe Flexibilität, da sie schon zum Übersetzungszeitpunkt nicht nur die syntaktische Gültigkeit, sondern auch die konkrete Verfügbarkeit der abgefragten Daten (oder zumindest der sie enthaltenden Strukturen, sprich: Tabellen) überprüft. Aufgrund der relativ einfachen Struktur der Parameterschnittstelle fällt dieser Nachteil aber nicht ins Gewicht.

Zusätzliche Arbeit bereitet die Tatsache, daß die Struktur der Parameter während des Transports im Netz von der innerhalb der Datenbank gänzlich verschieden ist. Der Channel Access beinhaltet keine Strukturierung der Prozessva-

[22] GP — *G*et *P*rocedure (im Gegensatz zu Put Procedure) ist der Prozess, der Daten ausliest, also hier der `daq_readout`.

riablen, während eine Datenbank die Daten in normalisierter Form [Mat98] halten soll, die selbige sehr stark untergliedert, wodurch die Informationen über viele Tabellen verteilt werden.

Alle derartigen Informationen müssen bereits in der Datenbank so formatiert werden, daß sie für die Parameterbibliothek geeignet sind. Dies geschieht in Form virtueller Tabellen (Views oder Ansichten genannt). Diese Ansichten sind nichts anderes als abgelegte Abfragen auf tatsächlich vorhandene Tabellen oder andere Ansichten. Durch Kaskadierung dieser Ansichten ist es möglich, alle zum Betrieb der Datenaufnahmesteuerung notwendigen Parameter in einer großen (virtuellen) Tabelle zur Verfügung zu stellen. Aus dieser wird dann der entsprechende Parameter allein durch Angabe der gewünschten Gruppe (in der Nomenklatur der Parameterschnittstelle als „name“ bezeichnet) und des Parameternamens (aus historischen Gründen in der gleichen Nomenklatur „idx“ genannt) identifiziert. In Anhang A wird dies anhand eines Beispiels verdeutlicht.

Diese Strategie zur Extraktion der Parameter aus der Datenbank ist eleganter als eine Konvertierung der Werte außerhalb der Datenbank. Trotzdem kann je nach Ausbau dieser Lösung das Problem entstehen, daß die Abfrageschnittstelle der Datenbank durch die Vielzahl von Abfragen überfordert wird. Während EPICS laut Spezifikation den Transport von etwa 1 000 Prozessvariablen pro Sekunde ermöglichen soll, können Abfragenraten in dieser Größenordnung auf eine sehr komplizierte Struktur die Datenbank durchaus überfordern. Oracle, das auch in professionellen Umgebungen mit wesentlich höheren Datendurchsätzen vielfach eingesetzt wird, läßt erwarten, daß einfaches Aufrüsten der Hardwarekapazitäten eine Leistungssteigerung ohne zusätzliche Entwicklungsarbeit erlaubt. Ausgeklügelte Caching-Mechanismen können darüberhinaus erkennen, daß sich das Ergebnis der komplexen nachgeordneten Abfragestruktur nicht ändert, solange die Tabellen, auf denen die Abfrage operiert, gleich bleiben. Das sollte zumindest jeweils für eine Konfiguration der Fall sein — die Initialisierung und Inbetriebnahme einer Konfiguration sollte je nach Ausbaustufe etwa 200 bis 4 000 Parameterabfragen erfordern.

3.4 Die implementierten Elemente

Im Rahmen der vorliegenden Arbeit wurden die folgenden Elemente der Datenaufnahmesteuerung implementiert:

`daq_seb_cas`, `daq_eb_cas` Agent zur Kontrolle der Hard- und Softwarebestandteile der Datenaufnahme auf den Auslese-CPU (SEB)[23] respektive auf dem Eventbuilder (EB); der Agent verwendet die von den Hard- und Softwareentwicklern bereitgestellten Steuerungsfunktionen.

`daq_param_cas` Agent zur zentralen Bereitstellung verschiedenster Konfigurationsparameter über Channel Access

`daq_seb_seq`, `daq_eb_seq` Auf dem EPICS `sequencer` basierende FSM zur Ablaufsteuerung innerhalb von SEB und EB

`daq_seq` Ebenfalls auf dem EPICS `sequencer` basierende FSM zur Steuerung der untergeordneten `sequencer` `daq_seb_seq` und `daq_eb_seq`

`daq_man` Werkzeug zur automatischen Generierung der zu dem jeweils aktuellen Setup gehörenden `medm`-Maskendefinitionen

Notwendige Vorarbeiten waren die Definition der Parameterschnittstelle sowie die Bereitstellung der zugehörigen Bibliotheken.

Zusätzlich wurden die Teile der Datenaufnahme, die Daten auf Band schreiben, neu implementiert. Dies bedeutete einen merklichen Leistungsgewinn beim Eventbuilding.

3.4.1 Vorarbeiten

Verbesserung des Bandzugriffs der Datenaufnahme

In der ursprünglichen Implementierung der Datenaufnahme wurden die aufgenommenen Daten über externe Programmaufrufe auf Band geschrieben. Dies führte zu einer höheren Prozessorlast. Der Aufruf des Programms (`dd`), das dann die Daten auf Band schreibt, erfordert nämlich den Transport der Daten zwischen den beiden Programmen, was einem weiteren Umkopieren der Daten entspricht. Außerdem bedingt der Wechsel des Kontextes bei einem Betriebssystem Latenzzeiten, die bei nicht ausreichender Echtzeitfähigkeit im Falle von zeitkritischen Anwendungen — wie sie der Empfang von Daten per ATM mit einem kleinen Empfangspuffer darstellt — zu Fehlern, in diesem speziellen Fall zu Datenverlust führen kann. Gerade die Betriebssysteme, die für den Eventbuilder in Frage kommen (z.B. OSF1 und Linux), sind aber nicht von Haus aus echtzeitfähig, sondern

[23] *S*ub *E*vent *B*uilder

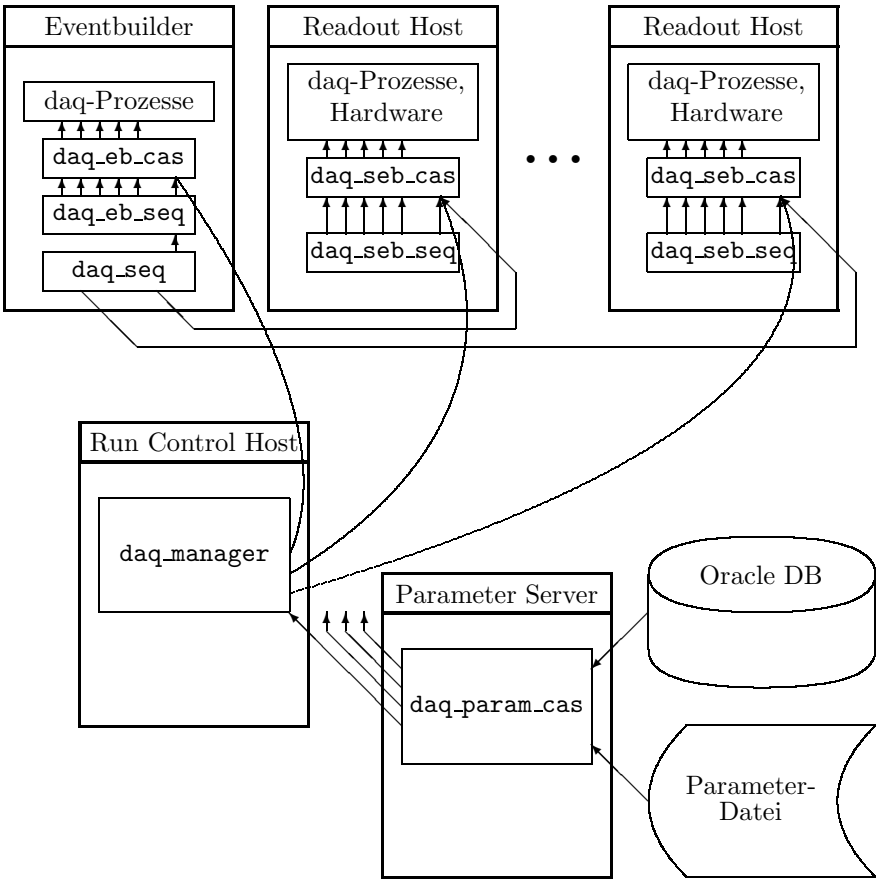

Abbildung 3.4: Die Verwendung der einzelnen Elemente der Datenaufnahmesteuerung. Eine relationale Datenbank dient als Quelle der Parameter, die über Channel Access transportiert werden. Die verschiedenen Agenten kontrollieren Datenaufnahmehard- und -software. Komplexe Abläufe werden von den Sequencern gesteuert. Die Benutzerschnittstelle erlaubt die Gesamtsteuerung von einem beliebigen weiteren Rechner.

erhalten die Echtzeitfähigkeit erst über Erweiterungen. Daher sind die garantierten Reaktionszeiten des Systems zum Teil nicht für den Datenempfang per ATM ausreichend.

Die Integration von Funktionen, die das Schreiben der Daten auf Band übernehmen, in den `daq_evtbuild` vermeidet zum einen die zeitraubende Kommunikation der beiden Prozesse und zum anderen Kontextwechsel des Betriebssystems.

Die grundsätzlichen Anforderungen an die Funktionen zum Beschreiben der Magnetbänder waren:

Format Beschreiben der Bänder entsprechend dem im ANSI X3.27-1978 Standard beschriebenen Format, das neben den eigentlichen Daten einen vorausgehenden Datenblock mit Verwaltungsinformationen (Header) und einen nachfolgenden ebensolchen Datenblock (Footer) fordert.

Bandinitialisierung Neben Verwaltungsinformationen für die einzelnen Dateien fordert ANSI X3.27-1978 auch Verwaltungsinformationen für das Medium selbst (Volume Label).

Pufferung Um Hardwarezugriffe, die als Systemaufrufe ebenso wie ein Prozesswechsel einen gewissen Aufwand mit sich bringen, auf ein sinnvolles Maß zu beschränken, sollten die wegzuschreibenden Daten gepuffert werden. Die hier verwendete Puffergröße gibt bei einer Reihe von Magnetbandtreibern zugleich die Blockgröße der weggeschriebenen Daten vor.

In Tabelle 3.2 ist ein grober quantitativer Vergleich der Eventbuilding-Leistung auf der bis zur Strahlzeit im Mai 2000 verwendeten Eventbuilderhardware angegeben. Der dort aufgeführte Auslese-Prozess ist ein sogenannter Software-Readout, d.h. ein Prozess, der ohne großen Rechenaufwand Daten produzieren kann. Als geeignete Kennzahl für den Aufwand A, normiert auf den Aufwand, Ereignisse zusammenzubauen und auf Band wegzuschreiben, läßt sich das Verhältnis der Prozessorauslastungen wie folgt angeben:

$$A = \frac{PL_{\texttt{daq\_evtbuild}} + PL_{\texttt{dd}}}{PL_{\texttt{daq\_readout}}}$$

Die Normierung auf die Prozessorlast des Software-Readout erwächst aus der Tatsache, daß bei mehr freier CPU-Zeit dieser auch mehr Daten produziert. Somit spiegelt sie einfach den größeren Datendurchsatz wieder.

Prozess	Prozessorlast (PL) mit altem Bandzugriff	Prozessorlast (PL) mit neuem Bandzugriff
`daq_readout`	$(6 \pm 3)\,\%$	$(13 \pm 2)\,\%$
`daq_evtbuild`	$(12 \pm 2)\,\%$	$(22 \pm 2)\,\%$
`dd`	$(24 \pm 2)\,\%$	$0\,\%$
A	6.0	1.7

Tabelle 3.2: Prozessorauslastung auf einer Alpha Station 250 4/266 ohne und mit verbessertem Bandzugriff.

Unter Berücksichtigung der Meßwerte aus Tabelle 3.2 hat sich die Leistung des Bandzugriff zumindest um den Faktor 3 verbessert, und dies unter der Annahme, daß der Beitrag des Eventbuildings ohne Bandzugriff vernachlässigbar ist.

Parameterbibliothek

Zur Konfiguration der Bestandteile der Datenaufnahme / Datenaufnahmesteuerung wurde eine einheitliche Schnittstelle geschaffen, die (je nach eingebundener Bibliothek) die Fähigkeit haben sollte, aus unterschiedlichen Quellen Parameter zu erhalten. Im einfachsten Fall war dies eine Datei, die diese Parameter in einem bestimmten Format bereit hielt.

Die Idee hinter dieser Schnittstelle ist, daß ohne Umprogrammierung (später vielleicht auch ohne Neuübersetzung) der Datenaufnahmesteuerung unterschiedliche Parameterquellen gewählt werden können, z.B. der Agent zur Parameterbereitstellung (oben bereits erwähnter `daq_param_cas`, siehe Abbildung 3.4) oder eine Textdatei (siehe Abbildung 3.5). Für jede Parameterquelle wurde also eine Bibliothek programmiert, die diese Schnittstelle zur Verfügung stellt.

Die Bibliotheken[24] im einzelnen sind:

`libtclParam.a` Extraktion der Parameter aus einer Datei unter Verwendung des `tcl`-Interpreters [Ous94]. Dies ermöglicht im Prinzip die dynamische Berechnung bzw. Erzeugung von Parametern. In der Praxis wurde von dieser Möglichkeit allerdings noch kein Gebrauch gemacht.

`libfileParam.a` Für Systeme, auf denen kein `tcl` zur Verfügung steht, wurde als Ausweichlösung ein kleiner `tcl`-Interpreter selbst geschrieben, der allerdings nur das kleinstmögliche Subset dieser Sprache zur Verfügung stellt. Lediglich die Definition der Parameter ist möglich.

`liboraParam.a` Hier dient als Quelle für die Parameter eine Ansicht der HADES-eigenen Oracle-Datenbank. Diese Ansicht sammelt wiederum die Parameter aus den einzelnen Tabellen in geeigneter Form zusammen, so daß nach und nach der gesamte Konfigurationsparametersatz hierüber bereitgestellt werden kann. Diese Bibliothek verwendet den von Oracle bereitgestellten Präcompiler, mit dem SQL-Zugriffe von C-Programmen aus realisiert werden können.

`libpsqlParam.a` Ähnlich wie Oracle bietet auch das freie Datenbankprojekt PostgreSQL[25] Schnittstellen zur Abfrage aus C-Funktionen heraus. Anders als Oracle kommt hier kein Präcompiler zum Einsatz. Stattdessen wird der Zugriff direkt über Bibliotheksfunktionen realisiert. Um die Datenbankstruktur neben der tatsächlich eingesetzten Datenbank auch an kleinen

[24] Bibliotheken im Kontext der Programmierung sind Sammlungen von Funktionen.
[25] Siehe http://www.postgresql.org/

Testtabellen überprüfen zu können, wurde die Parameterschnittstelle auch für PostgreSQL eingeführt.

`libcaParam.a` Der Transport der Parameter über das Netzwerk soll mittels Channel Access durchgeführt werden. Die dazu benötigten Abfragefunktionen werden mit dieser Bibliothek bereitgestellt.

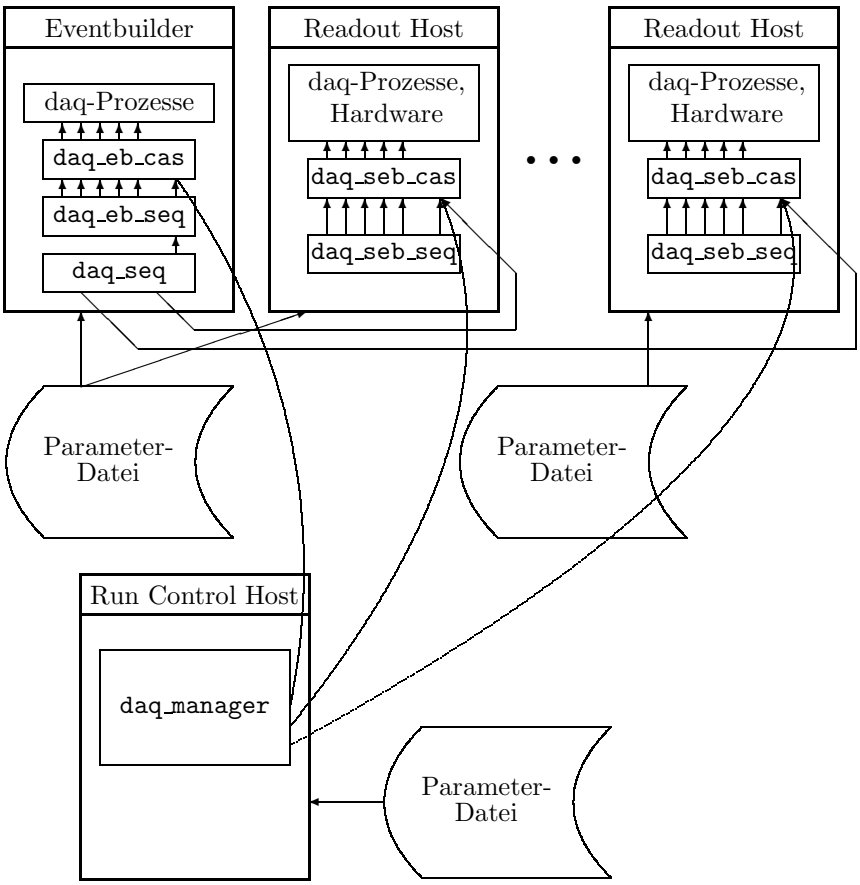

Abbildung 3.5: Bereitstellung der Parameter zu Testzwecken über lokal verfügbare Parameterdateien. Der `daq_manager` kann die Hauptvariablen der einzelnen Crates entweder direkt kontrollieren, oder dies dem `daq_seq` überlassen. Parameter werden den einzelnen Komponenten über Parameterdateien zur Verfügung gestellt, die auf dem (lokalen oder über das Netzwerk bereitgestellten) Dateisystem liegen.

Eine erste, bereits vorhandene Version der Parameteraufrufe enthielt lediglich Abfragen nach einzelnen Parametern, und zwar in Form von Ganzzahlen und Zeichenketten. Die einzig mögliche Quelle waren Textdateien, in denen die Parameter als tcl-Variablen gesetzt waren. Deren Quellcode diente als Basis zur Entwicklung von `libfileParam.a`. Der erste Schritt war die Erweiterung der Funktionalität in Hinblick auf geordnete Listen von Zeichenketten und Ganzzahlen. Diese müssen unter Angabe einer maximalen Anzahl der Elemente abgefragt werden, welche bei der Antwort aber durchaus auch unterschritten werden kann. Es ist also nicht notwendig von vorne herein die Länge der abgefragten Liste zu kennen. Weitere Änderungen im Vergleich zu der ursprünglichen Version der Parameterschnittstelle war eine einfache, aber durchaus den Anforderungen genügende Form der Fehlerbehandlung, in der der Rückgabewert eventuelle Fehler anzeigt und zusätzlich eine Fehlermeldung bereitgestellt wird. Die genaue Definition der Parameterschnittstelle sowie Beispiele zur Verwendung sind in Anhang C zu finden.

Entsprechend dieser Schnittstellendefinition mußten die anderen Quellen nun ebenfalls implementiert werden. Um zu gewährleisten, daß die bereitgestellten Funktionen sich korrekt und insbesondere unabhängig von der verwendeten Parameterquelle gleich verhalten (so die Parameterquelle die gleichen Parameter zur Verfügung stellt), wurde zusätzlich noch eine Testumgebung programmiert, die das Verhalten der Parameterbibliotheken überprüft [All00]. Dort wird die Antwort der Parameterbibliotheken auf eine Reihe verschiedener legaler und illegaler Anfragen ausgewertet.

3.4.2 Die Agenten

Agenten zur Hard- und Softwaresteuerung (`daq_seb_cas`, `daq_eb_cas`)

Vor der Migration zur neuen Datenaufnahmesteuerung wurde die Aufgabe der Hardwarekonfiguration von Steuerungsprogrammen (im folgenden `ctrl`-Programme genannt[26]) wahrgenommen. Diese wurden über Skripte aufgerufen, deren Inhalt damit die jeweilige Hardwarekonfiguration vorgab. Diese Möglichkeit zur Hardwarekonfiguration soll für die Entwicklung der Hardware weiterhin bestehen. Für eine konsistente Entwicklung der Funktionen, die die Hardware kontrollieren, sind diese in Bibliotheken zusammengefaßt, deren Schnittstelle sowohl die `ctrl`-Programme, als auch die Agenten nutzen. Dieser Zusammenhang ist in Abbildung 3.6 dargestellt.

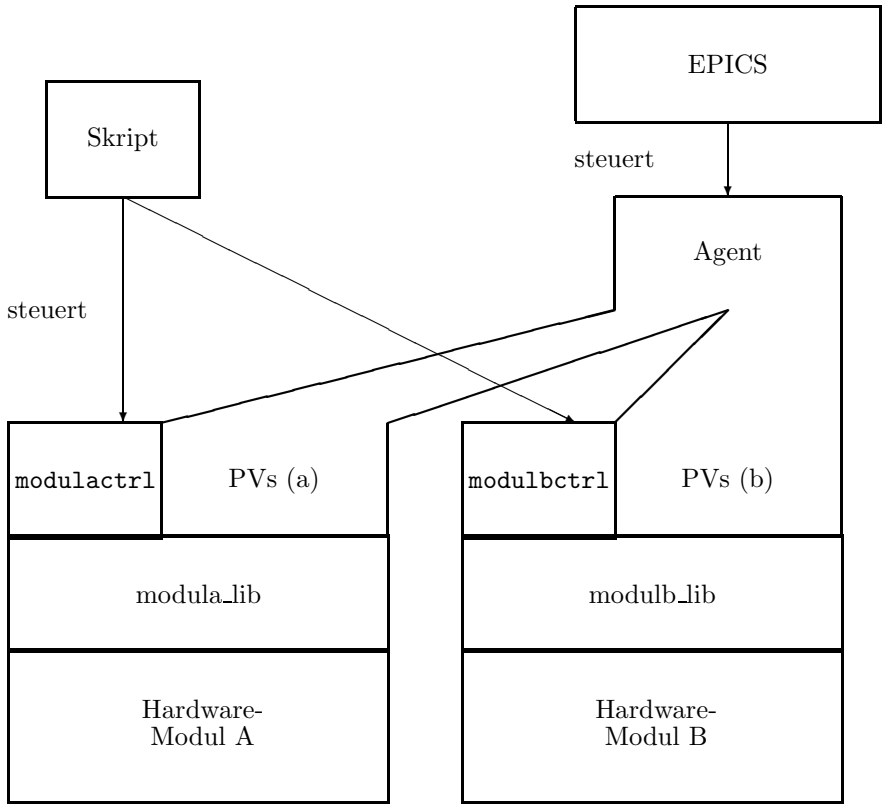

Abbildung 3.6: Gemeinsame Nutzung der Hardwarebibliotheken durch `ctrl`-Programme und Agenten.

Die Schnittstellendefinition enthält die in Tabelle 3.3 aufgelisteten Funktio-

[26]Die `ctrl`-Programme unterliegen einer strengen Nomenklatur: Sie heißen `modulctrl`, wobei `modul` jeweils durch den Inhalt der ersten Spalte der Tabelle B.1 ersetzt wird.

nen. Ihnen wird lediglich der Name des zu verwaltenden Moduls sowie eine Referenz auf die Parameterquelle übergeben. Durch Verwendung einer neu eingerichteten Verzeichnisstruktur sowie von CVS[27] wurde es ermöglicht, die Zugriffsfunktionen zugleich als Bestandteil der einzelnen Steuerungsprogramme der Entwickler als auch als Teil der Datenaufnahmesteuerung zu entwickeln. Analog zur Hardware wurden auch für die Softwarekomponenten der Datenaufnahme Steuerungsfunktionen entwickelt.

Funktion	Beschreibung
Bereits implementierte Funktionen	
Init	Initialisierung — Laden der Hardware-Designs in FPGAs[28]und DSPs[29]; bis jetzt ist auch das Laden der Schwellen in diese Funktion integriert.
Reset	Zurücksetzen der Hardware in einen definierten Zustand (Löschen der Fehler-Bits, Leeren der FIFOs, etc.); vor jedem Start muß per Definition ein Reset durchgeführt werden [Hom99].
Start	Ein Auslesezyklus beginnt; die Hardware kann Daten aufnehmen (wartet auf Trigger, etc.).
Stop	Ein Auslesezyklus endet
Status	Bis jetzt erscheint die Ausgabe des Hardwarestatus noch auf der Standard-Fehlerausgabe. Sie kann in dieser Form nicht vom Agenten, sondern nur von den zur Entwicklung gedachten `ctrl`-Programmen aufgerufen werden.
Noch zu implementierende Funktionen	
Thresholds	Führt das Laden der Schwellen durch
InitStatus	Rückgabewert zeigt an, ob die Hardware initialisiert ist
ResetStatus	Rückgabewert zeigt an, ob die Hardware zurückgesetzt ist
RunStatus	Rückgabewert zeigt an, ob die Hardware im Daten aufnimmt
BusyStatus	Rückgabewert zeigt an, ob die Hardware ein Busy erzeugt
ErrorStatus	Rückgabewert zeigt an, ob die Hardware Fehler erkannt hat

Tabelle 3.3: Die zur Steuerung benötigten Funktionsaufrufe.

Auch die Datenaufnahmesoftware selbst wurde zu Teilen bereits auf die Verwendung der Parameterquelle umgestellt. Die Komplettierung dieser Arbeit be-

[27] *C*oncurrent *V*ersions *S*ystem; ein Werkzeug zur Verwaltung von Quellcode von Softwareprojekten, die von verschiedenen Programmierern an unterschiedlichen Orten gleichzeitig entwickelt werden sollen. Nähere Informationen dazu sind unter http://www.cvshome.org/ zu finden.

[28] *F*ield *P*rogrammable *G*ate *A*rrays

[29] *D*igital *S*ignal *P*rocessors

dingt allerdings tiefgreifendere Veränderungen der Software, die sich unter Umständen negativ auf die Leistung der Datenaufnahme auswirken kann. Darüberhinaus sollte die Bedienung der Software auch ohne die Verwendung der Steuerungsfunktionen weiterhin möglich sein, was zusätzliche Anforderungen an die Benutzerschnittstelle bedingt. Aus diesen Gründen muß gerade die Umstellung der Datenaufnahmesoftware mit besonderer Sorgfalt geschehen.

Der Agent selbst erwartet keine ihm übergebenen Parameter. Dies erlaubt, ihn bei Anfragen automatisch vom Betriebssystem zu starten. Danach kann er sich nach folgendem Verfahren selbst konfigurieren:

1. Der Name der CPU wird über einen Betriebssystemaufruf ermittelt.

2. Grundlegende Eigenschaften der CPU werden über die Parameterquelle in Erfahrung gebracht.

3. Alle im Subsystem eingesetzten Komponenten liest der Agent ebenfalls aus der Parameterquelle.

4. Für die einzelnen Komponenten werden jeweils die Prozessvariablen erzeugt.

5. Es folgen Prozessvariablen für etwaige zusätzlich zu überwachende Informationen der Datenaufnahmesoftware, die durch Zugriffe auf offene Speicherbereiche[30] gelesen werden können.

6. Auch für die Hardware werden entsprechende Prozessvariablen erzeugt. Diese erhalten ihren Inhalt durch Lesen dedizierter VME-Register der entsprechenden Karte.

7. Zuletzt werden Prozessvariablen erzeugt, die den Zustand des gesamten Subsystems repräsentieren.

Die Agenten für die Auslese-CPUs (oder auch SEB[31]) unterscheiden fünf verschiedene Grundkonfigurationen, die die Wahl der eingesetzten Hardware einschränken. Dies dient der frühzeitigen Erkennung von Fehlkonfigurationen. Im Falle der Datenbank als Parameterquelle kann die Konsistenz besser dort verifiziert werden, so daß diese Trennung dann wegfallen könnte.

[30] sog. Shared Memory oder kurz shm — Bereiche im Hauptspeicher, auf die mehr als ein Prozess zugreifen kann.

[31] Die Bezeichnung Sub Event Builder ist streng genommen lediglich für die CPUs gültig, auf denen auch Datenaufnahmeprozesse laufen. Dieser Agent soll in der endgültigen Konfiguration unter anderem auch zur Steuerung von Hardware dienen, die nicht direkt von CPUs ausgelesen wird, namentlich die ADC / TDC-Module der TOF. Dort können anstelle der teuren CES RIO 8062 CPUs mit ATM-Schnittstelle auch günstigere CPUs verwendet werden, wie z.B. auf der 68k-Architektur basierende Eltec E7.

Eine grundlegende Unterscheidung wird als Konvention zwischen dem Agenten für Auslese-CPUs (`daq_seb_cas`) und für den Eventbuilder (`daq_eb_cas`) getroffen. Der Eventbuilder stellt gar keine Prozessvariablen für Hardware zur Verfügung, sondern lediglich für die Software. Dafür kommen neben den „Verwaltungsvariablen“ für den Eventbuilder (der in diesem Zusammenhang gleichberechtigt mit einem Auslese-VME-Crate behandelt wird) noch Verwaltungsvariablen für den Ausleseteil der Datenaufnahme und die Datenaufnahme selbst hinzu. Diese Verwaltungsvariablen werden später als Mittel zur Steuerung der FSMs wichtig.

Nach der Erzeugung sämtlicher Prozessvariablen beginnt der Agent mit seiner eigentlichen Arbeit: Er stellt die Prozessvariablen zur Verfügung und führt bei der Modifikation selbiger die dazu korrespondierenden Routinen aus. In der momentanen Ausbaustufe werden die Zustände der Variablen jeweils im Agent selbst abgelegt. Zukünftige Funktionen in den Hardwarebibliotheken zur Extraktion der Zustände, wie sie in Tabelle 3.3 angedacht sind, sollen eine echte Überwachung der Hardware ermöglichen.

In Tabelle B.1 ist eine Auflistung der zur Zeit vorhandenen VME-Hardwarekomponenten zu finden. Die Prozessvariablen, die die verschiedenen Komponenten repräsentieren, wurden unter Verwendung des Vererbungskonzeptes implementiert, wodurch eine einfache Erweiterbarkeit des Codes um neue Module gewährleistet wird. Dies wurde in der Strahlzeit im November 2000 durch die schnelle Integration von zwei neuen Modulen, des Triggerhubs und des Fanout-Boards (siehe Tabelle B.1), in die Datenaufnahmesteuerung unter Beweis gestellt. Dieser Vorteil wird sicherlich auch in Zukunft noch weiter relevant sein, wenn zusätzliche VME-Karten, wie z.B. die Startzählerelektronik, in die Datenaufnahmesteuerung integriert werden sollen.

Agent zur Bereitstellung von Parametern (`daq_param_cas`)

Zur Bereitstellung aller Informationen für die einzelnen Bestandteile der Datenaufnahmesteuerung über Channel Access unter Verwendung verschiedener Quellen wurde mit dem `daq_param_cas` ein weiterer Agent programmiert. Im Gegensatz zu den Agenten, die im vorherigen Abschnitt besprochen wurden, werden hier die Prozessvariablen nicht bereits beim Start des Agenten statisch festgelegt. Erst, wenn eine Anfrage nach einer Prozessvariablen eintrifft, überprüft der Parameteragent, ob seine Parameterquelle den entsprechenden Parameter enthält. Die Parameter werden in der Abfrage durch Angabe von Name und Index eindeutig identifiziert. Die Umsetzung in den Namen der Prozessvariablen, die in Kapitel 3.3.1 bereits angesprochen wurde, kann nun allgemeiner gefaßt werden: Die Abfrage eines Parameters mit dem Namen „name“ und dem Index „idx“ wird in den PV-Namen „HAD:PARAM:NAME:IDX“ übersetzt. Bei den meisten Abfragen ist mit „name“ der Name der Hardwarekomponente gemeint, was dazu führt, daß die Definitionen hier und in Kapitel 3.3.1 übereinstimmen.

Erkennt der Parameteragent die erfragte Prozessvariable als Parameteranfrage und enthält die von ihm konsultierte Parameterquelle den Parameter, so erzeugt der Parameteragent die Prozessvariable und beantwortet die Anfrage nach dem Wert. Die Prozessvariable bleibt danach erhalten. Abfragen nach ihr ziehen aber jeweils wieder Anfragen bei der Parameterquelle nach sich, da der Wert während der Laufzeit des Agenten im allgemeinen ja variabel sein soll.

Es ist möglich, auch mehrere Parameteragenten parallel zu betreiben. Die einzige Anforderung, die dabei beachtet werden muß, ist, daß nicht zwei Parameteragenten den gleichen Parameter als zu ihrem Bereich gehörig erachten, da Channel Access Eindeutigkeit der Prozessvariablennamen voraussetzt. Z.B. ist in der Implementierungsphase der Betrieb eines Parameteragenten, der die Oracle-Datenbank konsultiert, und eines anderen, der die noch nicht in den Datenbank verfügbaren Parameter aus einer `tcl`-Datei liest und ebenfalls bereitstellt, problemlos möglich. Im vollen Ausbau der Datenbank führt die Möglichkeit mehrerer Parameteragenten zu einer guten Skalierbarkeit in Bezug auf den Datendurchsatz. Dabei wird vorausgesetzt, daß der Engpaß tatsächlich bei dem einzelnen Agenten liegt. Dies sollte allerdings gemäß der Spezifikationen von EPICS (Bereitstellung von etwa 1 000 Prozessvariablen innerhalb einer Sekunde) und Oracle (gut skalierbares, professionelles Serverprodukt) gegeben sein.

Da im Falle von verteilten Parameterdateien für jede Anwendung einzeln darauf geachtet werden muß, daß sowohl die richtige Datei als Parameterquelle betrachtet wird, als auch der Inhalt dieser Datei auf dem aktuellen Stand ist und die korrekten Werte enthält, ist der Parameteragent für ein weit verzweigtes System die in Hinblick auf Konsistenz sicherere Lösung. Solange an Einzelsystemen getestet wird, rechtfertigt allerdings der Nutzen eines Parameteragenten nicht unbedingt den Aufwand. Die Konsistenz der Parameterdateien könnte auch durch ein verteiltes Dateisystem erreicht werden (wie z.B. bei den Auslese-CPUs an der GSI, die alle das gleiche per NFS bereitgestellte Dateisystem nutzen), allerdings ist diese Lösung nicht bei allen Systemen möglich oder praktikabel. Dafür wäre der uneingeschränkte Zugriff auf die Systeme inklusive der Administrationsrechte vonnöten, der aber einem Experimentator in der Regel nicht zur Verfügung steht.

3.4.3 Die Ablaufsteuerung (`daq_seb_seq`, `daq_eb_seq`, `daq_seq`)

Die Implementierung der endlichen Automaten zur Ablaufsteuerung, die im folgenden auch kurz als Sequencer bezeichnet werden, wurde bis zum jetzigen Zeitpunkt wegen fehlender Unterstützung alternativer Betriebssysteme lediglich für VxWorks entwickelt. Die dabei zu lösenden Probleme waren hauptsächlich technischer Natur. Zunächst erlaubt VxWorks, anders als klassische Betriebssysteme, kaum interaktive Bedienung. Insbesondere stellt es selbst keine Entwicklungsumgebung zur Verfügung. Vielmehr muß der ausführbare Programmcode auf einer anderen Plattform erzeugt und mittels eines Cross-Compilers[32] übersetzt werden. Als Entwicklungsplattform stand lediglich Windows NT auf x86-kompatiblen PCs zur Verfügung.

Wir verwendeten die `sequencer`-Version 1.9.5 zusammen mit der EPICS-Release R3.13.3. Bereits angekündigte Versionen von EPICS (R3.14) werden zusammen mit `sequencer`-Versionen ab 2.0.0 [Lup98] erlauben, die Sequencer auf verschiedenen Plattformen zu entwickeln. Dies ermöglicht es, die Ablaufsteuerung und die zugehörigen Agenten auf der gleichen CPU laufen zu lassen. Dies ist die optimale Konfiguration in Hinblick auf die Netzlast, da die weitaus meisten vom Sequencer zu überwachenden Prozessvariablen lokal zu finden sind. Tatsächlich war aber selbst die in dieser Hinsicht wesentlich ungünstigere Konfiguration, wie sie in Abbildung 3.7 skizziert ist, während der Tests auch in Bezug auf die Netzlast unkritisch.

Die eigentliche Programmierung spaltete sich auf in die Erzeugung von SNL-Code, mit dem die Eigenschaften der endlichen Automaten definiert werden können, und die Programmierung von Subroutinen (geschrieben in ANSI-C[33] [Ker90]) zur Unterstützung.

Die Sequencer ermitteln, genau wie die Agenten, zunächst ihre Konfiguration mit Parameterabfragen. Dieser Schritt wird in Abbildung 3.8 mit dem Zustand „CONFIG“ identifiziert, der sich je nach Art des `sequencers` in unterschiedlich viele Einzelzustände aufspaltet. Alle nachfolgenden Schritte sind unabhängig vom konkret überwachten Hardwaretyp und damit auch mit dem gleichen SNL-Code realisiert. Der `sequencer` geht zunächst davon aus, daß die Hardware nicht initialisiert ist. Wenn die Kontrollvariable „HAD:DAQ:CRATE_ID:INIT“ auf „1“ gesetzt wird, setzt der Sequencer alle ihm untergeordneten Variablen „HAD:-CRATE_ID:UNIT_ID:INIT“ auf „1“, was nun den Agenten dazu veranlaßt, die darunterliegenden Einheiten zu initialisieren. Hierbei ist „CRATE_ID“ durch den eindeutigen Namen der vom Sequencer kontrollierten Einheit, z.B. das bereits erwähnte „RICH0“ zu ersetzen. „UNIT_ID“ ist der Name der Einheit, die der Agent bei Änderung der Prozessvariablen bedient (z.B. „RICH0GP“).

[32] Ein Cross-Compiler erzeugt Maschinencode nicht für die Plattform und nicht für das Betriebssystem, auf der bzw. unter dem er läuft.

[33] Die Programmiersprache ist nach ANSI X3.159-1989 standardisiert.

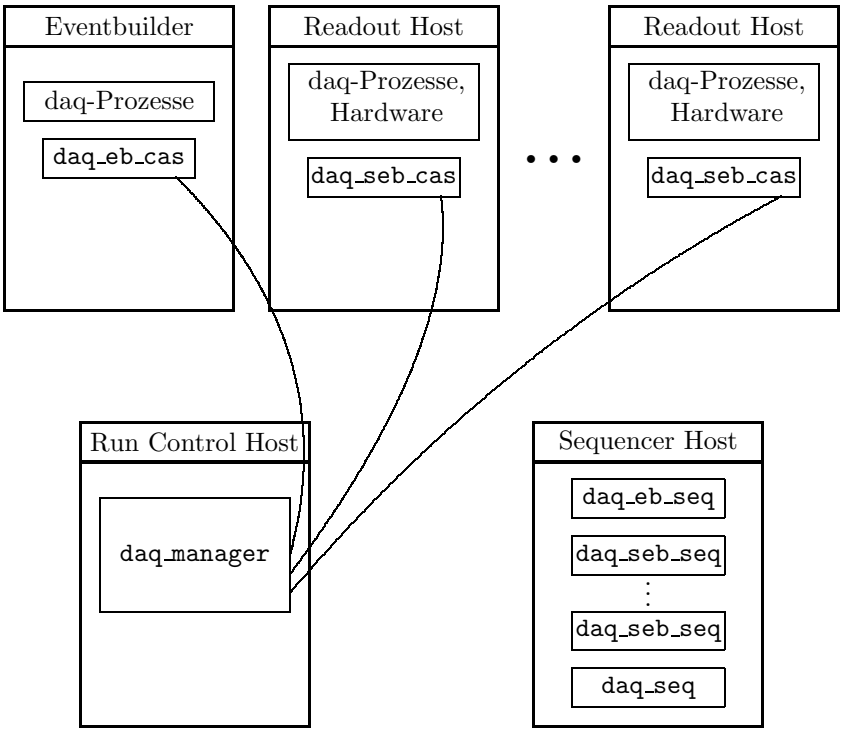

Abbildung 3.7: Übergangsweise sind die Prozesse der Ablaufsteuerung noch nicht auf die zu kontrollierenden Rechner verteilt, sondern laufen zentral auf einer CPU unter VxWorks. Um die Darstellung nicht zu unübersichtlich zu gestalten, wurden die Verbindungen, die zwischen den Agenten und den Sequencern in Abbildung 3.4 und Abbildung 3.5 bestehen, weggelassen.

Hier, wie auch in den darauffolgenden analogen Schritten bei „RESET“ und „RUN“, ist zu beachten, daß die Initialisierung nur ausgeführt wird, wenn der Wert nicht bereits vorher auf „1“ stand. Ansonsten würde das Anhalten und der Neustart des Sequencers zwingend eine Initialisierung der gesamten Hardware nach sich ziehen. Im Prinzip wird durch dieses Verhalten gewährleistet, daß Ausfälle bei der Steuerung nicht die Datenaufnahme selbst beeinträchtigen. Um auch den Ausfall eines Agenten unbeschadet zu überstehen, muß dieser die Fähigkeit besitzen, bei einem Neustart den aktuellen Zustand der Hardware in Erfahrung zu bringen. Dies ist ein wichtiger Grund für die Erweiterung der Kontrollfunktionen, wie sie in Tabelle 3.3 vorgeschlagen sind. Wenn alle Variablen erfolgreich gesetzt werden konnten, befindet sich der Sequencer nun im Zustand

„INITIALIZED“, von dem aus ein Setzen der Datenaufnahme auf einen definierten Stand erforderlich ist. Dies geschieht analog zur Initialisierung durch Setzen der Variable „HAD:DAQ:CRATE_ID:RESET“. Scheitert dagegen das Setzen einer der untergeordneten Prozessvariablen oder fällt eine Prozessvariable nach dem Initialisieren wieder in den Zustand „0“ zurück, so gilt auch das ganze Crate nicht mehr als initialisiert, d.h. „HAD:DAQ:CRATE_ID:INIT“ fällt auf den Wert „0“ zurück und der Sequencer geht in den Zustand „NOT_INITIALIZED“ über.

Ein normaler Ablauf in dem Sequencer aus Abbildung 3.8 könnte z.B. so aussehen:

- Die Sequencer werden gestartet.
  → Zustand CONFIG
- Die Sequencer lesen ihre Konfiguration.
  → Zustand NOT_INITIALIZED
- Der Experimentator gibt die Anweisung zur Initialisierung.
  → Zustand INITIALIZED
- Der Experimentator gibt die Anweisung zum Reset.
  → Zustand RESET
- Der Experimentator startet die Datenaufnahme.
  → Zustand RUNNING
- Die Datenaufnahme entdeckt einen Fehler und stoppt den Datenaufnahmeprozess. Der Sequencer entdeckt dies und stoppt die gesamte Datenaufnahme.
  → Zustand INITIALIZED
- Der Experimentator gibt die Anweisung zum Reset.
  → Zustand RESET
- usw.

Ebenso wie die Initialisierung wird im Prinzip auch das Zurücksetzen auf einen definierten Stand („RESET“) sowie das Starten der Datenaufnahme gehandhabt. Eine zusätzliche Einschränkung ist allerdings, daß die Datenaufnahme nie direkt von dem Zustand „RUNNING“ in den Zustand „RESET“ übergehen kann. Jeder Übergang weg vom Zustand „RUNNING“ setzt vielmehr alle RESET-Variablen auf „0“. Dies geschieht, da die Spezifikation der Datenaufnahme generell nicht festgelegt hat, daß mit einem Stopp der Datenaufnahme ein definierter Zustand erreicht wird. Stattdessen erfordert jeder Start der Datenaufnahme unbedingt zuvor einen Reset, was in [Hom99] explizit festgelegt wurde. Hier wird der Vorteil der Verwendung endlicher Automaten deutlich: Eine klare Definition der erlaubten Übergänge genügt, um die Einhaltung dieser Festlegung zu erzwingen.

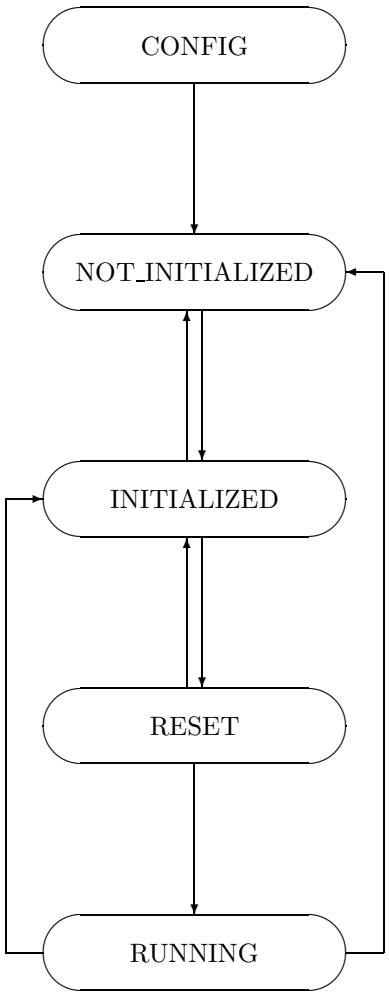

Abbildung 3.8: Ablaufdiagramm für die Komponenten eines Auslesesubsystems.

3.4.4 Die graphische Benutzeroberfläche (`daq_manager`)

Die Komponente, die der Operateur der Datenaufnahme bedient, ist die graphische Benutzeroberfläche. Das Werkzeug, das im Verlaufe der vorliegenden Arbeit als Basis zur Entwicklung gedient hat, ist der `medm`. Der `medm` ist auf der einen Seite ein Programm zur Darstellung von graphischen Masken, die in adl-Dateien definiert werden, auf der anderen Seite dient er auch als Entwicklungswerkzeug für die adl-Dateien. Da die adl-Dateien aber selbst statisch sind, wurde nur die erste Funktionalität tatsächlich angewendet. Die Erzeugung der adl-Dateien muß dynamisch erfolgen, um stets an wechselnde Konfigurationen der Datenaufnahme angepaßt sein zu können.

Dazu wurde ein Programm entwickelt, daß wiederum die Parameterquelle zur Feststellung der aktuellen Konfiguration heranzieht. Zunächst wird das Hauptmenü generiert. Dort findet sich für jedes Crate, das in der aktuellen Konfiguration vorhanden ist — wobei hier wie schon des öfteren der Eventbuilder ebenfalls ein Crate darstellt — zum einen der Status (und zwar aufgegliedert in „INIT“, „RESET“ und „RUN“) und zum anderen die Möglichkeit, detailliertere Informationen abzurufen, indem man ein entsprechendes Untermenü aufruft. Zusätzlich wird noch der Gesamtstatus der Datenaufnahme angezeigt. Dabei sind natürlich wieder die drei Einzelinformationen zu finden. Für jede dargestellte Prozessvariable ist dabei neben der Anzeige auch eine Kontrolleinheit zu finden, über die direkt der Wert geändert werden kann. Alle Prozessvariablen, die bis hierher angesprochen wurden, sind vom einfachst möglichen Typ, nämlich boolesche Variablen, die lediglich die Werte „falsch“ oder „wahr“ annehmen können. Die einzelnen Einheiten sind nach Typ gegliedert. Z.B. finden sich alle Crates des RICH in einer Gruppe, davon getrennt alle Crates der TOF und so weiter.

Die den einzelnen Crates zugeordneten adl-Dateien weisen eine analoge Struktur auf. Lediglich stehen anstelle der einzelnen Crates nun die einzelnen Einheiten innerhalb des Crates und anstelle der globalen Datenaufnahmevariablen die Prozessvariablen, die den Status des gesamten Crates repräsentieren. In Abbildung 3.9 ist ein Beispiel eines solchen Menüs zu sehen. Mit Einheiten sind hier sowohl die einzelnen Hardwarekomponenten, als auch alle Datenaufnahmeprozesse gemeint. Die Gliederung erfolgt entsprechend nach dem Typ der Hardwarekomponente. In diesem Sinne ist auch ein Datenaufnahmeprozess wiederum eine eigene „Hardwarekomponente“.

Gerade graphische Benutzeroberflächen eignen sich aufgrund ihrer Struktur sehr gut zur Anwendung des objektorientierten Modells. Dadurch, daß alle Bestandteile der Oberfläche als Objekte aufgefaßt werden, die als Entitäten einer Klasse bestimmte Attribute besitzen, die innerhalb der Klassenhierarchie vererbt werden können, läßt sich eine graphische Anwendung sehr effizient programmieren. Nicht ohne Grund sind deshalb gerade Graphikbibliotheken bereits sehr häufig in objektorientierten Programmiersprachen wie C++ geschrieben und nutzen dabei auch das Instrumentarium der Objektorientierung stark aus. Auch der

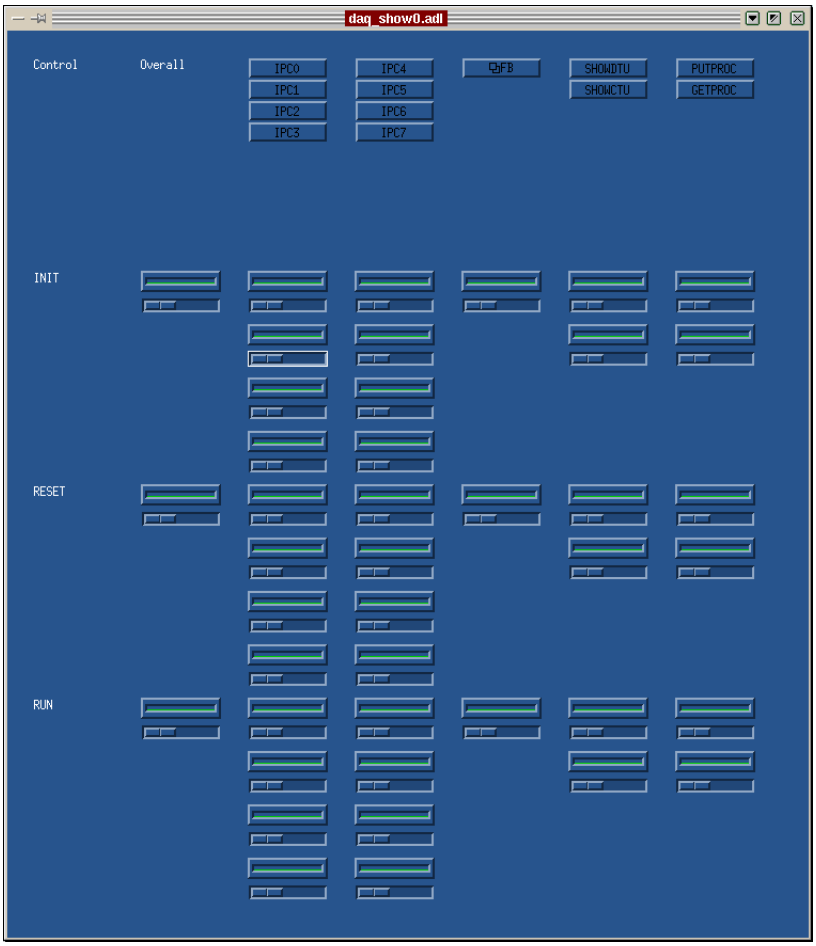

Abbildung 3.9: Die dynamisch erzeugte, graphische Benutzeroberfläche; im oberen Bereich der Maske sind Bestandteile des kontrollierten VME-Crates aufgelistet. In diesem Fall ist der Inhalt des PreShower-Auslese-Crates dargestellt. Die ersten beiden Spalten enthalten die Bildverarbeitungseinheiten, dann folgt ein „Fanout Board“, die DTU und die CTU und ganz rechts die beiden Datenaufnahmeprozesse. Unter den Knöpfen, die zu Detailinformationen über die Module führen, finden sich Statusanzeigen und Kontrollelemente für die jeweiligen „INIT“-, „RESET“- und „RUN“-PVs. Links davon finden sich die entsprechenden Elemente für das gesamte VME-Crate.

`medm` folgt diesem Konzept, was sich in der Struktur der adl-Dateien deutlich zeigt. Dort sind die einzelnen Elemente der jeweiligen Masken als Objekte mit bestimmten Eigenschaften angegeben.

Dies konnte man sich nun zunutze machen und mit einfacher objektorientierter Programmierung unter Ausnutzung der bestehenden Klassenstruktur sehr schnell ein Werkzeug erstellen, das die adl-Dateien nach den Vorgaben der Parameterquelle erzeugt.

Da die tatsächliche Struktur der von den Agenten zur Verfügung gestellten Detailinformationen der einzelnen Einheiten noch nicht besonders ausgeprägt war, wurde auch in die Gestaltung der Darstellung dieser Detailinformationen noch nicht sehr weit vorangetrieben. Im Prinzip sollte auch hier wieder die Parameterquelle das Aussehen der einzelnen Masken bestimmen. Bis jetzt allerdings erwies sich eine statische Kollektion von Masken, die nur von der Art der dargestellten Hardware abhing, als ausreichend.

Ebenso wurden die ergonomischen Gesichtspunkte, die neben der technischen Realisierung bei der Gestaltung einer solchen Oberfläche ebenfalls eine wichtige Rolle spielen, zunächst in den Hintergrund gestellt. Zum einen bietet ja gerade das objektorientierte Erzeugungswerkzeug gute Voraussetzungen, um die Anordnung, Größe, Art und auch Zahl der Kontrollelemente mit geringem Aufwand ändern zu können, zum anderen ist, wie bereits angedeutet und im Ausblick genauer ausgeführt, die Wahl des Darstellungswerkzeuges noch nicht endgültig.

Kapitel 4

Einsatz im Experiment

In der Zeit zwischen dem 09. 11. 2000 und dem 21. 11. 2000 wurde nach mehreren Teststrahlzeiten der HADES-Detektor zum ersten Mal in einem vollständigen Aufbau eingesetzt. Dazu wurden fünf Sektoren des RICH, zehn der 13 bis dahin vorhandenen MDC-Module, alle sechs Sektoren der TOF und viereinhalb Sektoren des PreShower-Detektors zu einem gemeinsamen Datenaufnahmesystem zusammengesetzt.

Zum ersten Mal wurden dabei für eine längere Zeit Messungen mit eingeschaltetem Magnetfeld durchgeführt, was die Analyse in die Lage versetzte, aus den gewonnenen Daten über die Ablenkung im Magnetfeld Teilchenimpulse zu bestimmen.

Erstmals wurde ein Datenaufnahmesystem mit einer Anzahl an Modulen aufgebaut, die mit der Anzahl der Module im Endausbau vergleichbar war. Dabei wurden die Grenzen der alten, provisorischen Datenaufnahmesteuerung, vor allem in Bezug auf Start- und Stoppzeiten, aufgezeigt. Abgesehen davon arbeitete die Datenaufnahme trotz der großen Anzahl von SEBs (hier wurde bereits die ursprünglich geplante Zahl von sieben erreicht) immer sehr zuverlässig.

In einem Testaufbau kam die neue Datenaufnahmesteuerung erstmals zum Einsatz. Dabei wurde der PreShower-Detektor mehrfach initialisiert und die Datenaufnahme in einer ganzen Reihe von Testläufen gestartet und gestoppt. Die Tragfähigkeit des Konzeptes wurde dabei nachgewiesen. Die Umstellung auf die neue Datenaufnahmesteuerung wurde in weiten Teilen vorbereitet (die neuen, an den Einsatz innerhalb der Agenten angepaßten Bibliotheksfunktionen fanden bereits in den `ctrl`-Programmen Verwendung). Um die Messungen während der Strahlzeit nicht zu beeinträchtigen, wurde die eigentliche Umstellung auf einen späteren Zeitpunkt verschoben.

4.1 Allgemeine Daten zur Strahlzeit

Strahl, Target In der Strahlzeit im November 2000 wurde ein ^{12}C-Strahl mit einer Energie von 1.5 AGeV verwendet. Es kam ein ^{12}C-Target zum Einsatz,

das eine Wechselwirkungswahrscheinlichkeit von 2 % hatte. Als Strahlintensität wurden etwa $1 - 2 \cdot 10^6$ Teilchen pro Strahlpuls gewählt, bei einer Pulslänge von wenigen Sekunden.

Datenmenge Insgesamt wurden etwa 60 GB an Daten gesammelt, von denen etwa 25 GB mit Strahl und laufendem Gesamtsystem gewonnen wurden. Der Rest entstand bei Kalibrationsläufen der einzelnen Detektoren.

Zahl der Ereignisse Als primäre Triggerbedingung wurden verschiedene Kombinationen aus Mindestmultiplizitäten in der TOF (mehr als vier Signale) und das Fehlen eines Signals im Vetozähler verwendet. Es wurden etwa $1.1 \cdot 10^7$ Ereignisse aufgezeichnet, davon $2 \cdot 10^6$ mit eingeschaltetem Magnetfeld.

Aufgezeichnete Ereignisrate Die primäre Triggerrate wurde wegen auftretenden Instabilitäten der Auslesehardware bei hohen Datenraten auf maximal 2 000 Hz begrenzt.

4.2 Der Testaufbau der Datenaufnahmesteuerung

In dem Testaufbau für die Datenaufnahmesteuerung wurde ein einfaches und ausgiebig getestetes Subsystem verwendet. Alle in den vorangehenden Kapiteln vorgestellten Komponenten der Datenaufnahmesteuerung kamen dabei zum Einsatz.

Auf einem zentralen Linux-Rechner der GSI wurde ein Parameteragent gestartet, der alle Informationen über die aktuelle Konfiguration bereitstellte. Diese wurden zunächst von dem GUI-Generator dazu verwendet, eine Kollektion von adl-Dateien zu erstellen, mittels derer später die Steuerung des Systems erfolgen konnte. Danach wurden auf dem Eventbuilder und der Auslese-CPU die Agenten zur Hard- und Softwaresteuerung gestartet. Diese erhielten die Informationen über ihre Konfiguration über eine lokale Kopie der Parameterdatei, die auch der Parameteragent zur Verfügung stellte.

Die Bibliothek zur Verwendung des Parameteragenten als Quelle wurde bei den Agenten nicht eingesetzt, da sie bei vorherigen Versuchen in einer RICH-Konfiguration als mögliche Fehlerursache abgeschaltet worden waren. Die tatsächliche Ursache für die fehlgeschlagenen Versuche konnte innerhalb der kurzen verbleibenden Zeit nicht geklärt werden. Es ist aber wahrscheinlich, daß fehlerhafte Migration einer Steuerungsfunktionen hin zu den neuen Parameteraufrufen auf der einen Seite und nicht einwandfrei arbeitende Hardware auf der anderen Seite zu nicht reproduzierbaren Ergebnissen in der Initialisierung führten.

Danach wurden auf einer VME-CPU unter dem VxWorks-Betriebssystem die drei notwendigen Sequencer-Prozesse gestartet, die ihre Konfiguration wiederum vom Parameteragenten erhielten.

Das damit vollständige System wurde nun dazu benutzt, zunächst die Hardware (acht IPCs, eine DTU, eine CTU und ein Fanout Board — siehe hierzu Anhang B) mehrmals korrekt zu initialisieren und danach die Datenaufnahme wiederholt zu starten und zu stoppen. Auch das korrekte Verhalten bei einem Verlust der Initialisierung wurde verifiziert. Da Funktionen, die den Verlust der Initialisierungsinformation anzeigen können, noch nicht implementiert sind, geschah dies durch einfaches Setzen einer beliebigen init-Prozessvariablen auf „0“ (nicht initialisiert). Das System beendete die Datenaufnahme korrekt.

4.3 Ausgewählte Ergebnisse

Da die Testsysteme in München vor allem Komponenten des RICH verwendeten und der Aufbau an der GSI mit dem PreShower-Detektor durchgeführt wurde, werden in diesem Abschnitt bevorzugt Daten dieser Detektoren vorgestellt. Zur Analyse wurde das auf ROOT[1] [Bru00] basierende HYDRA [Sán99] verwendet.

4.3.1 Leptonenidentifizierung mit RICH und PreShower

Das einwandfreie Arbeiten der Einzeldetektoren ist die Grundvoraussetzung für ein funktionierendes Zusammenspiel des Gesamtsystems. Abbildung 4.1 zeigt hierzu ein Ereignis des RICH, anhand dessen die gute Unterdrückung von Rauschen und hadronischem Untergrund illustriert wird.

Sodann kann der Mechanismus der Teilchenidentifikation untersucht werden. Hier geben vor allem Korrelationen zwischen verschiedenen Detektoren Auskunft, bei denen nach und nach immer stärkere Bedingungen eingeschaltet werden:

- Zunächst werden alle Ereignisse beachtet, in denen im RICH ein Ring gefunden wurde und sich an entsprechender Stelle im PreShower auch ein Durchstoßpunkt befand. Diese Ereignisse bilden den „PreShower-Treffer“-Graphen in Abbildung 4.2. Da es bei zentralen Stößen selbst in leichten Systemen bereits einen hadronischen Untergrund gibt, der im gleichen Polarwinkelbereich zu weiteren hadronischen Treffern im relativ schwach segmentierten PreShower-Detektor (32×32 Felder in jedem Sektor) führt, findet man noch viele zufällige Koinzidenzen und eine breite Verteilung über den gesamten Winkelbereich.

- Wird nun weiter verlangt, daß der Treffer im PreShower auch einen Schauer in den letzten beiden der drei Kammern gebildet hat, fällt der größte Teil

[1] Eine objektorientierte Klassenhierarchie, die von René Brun und Fons Rademakers am CERN zur Datenanalyse in der Teilchenphysik entwickelt wurde.

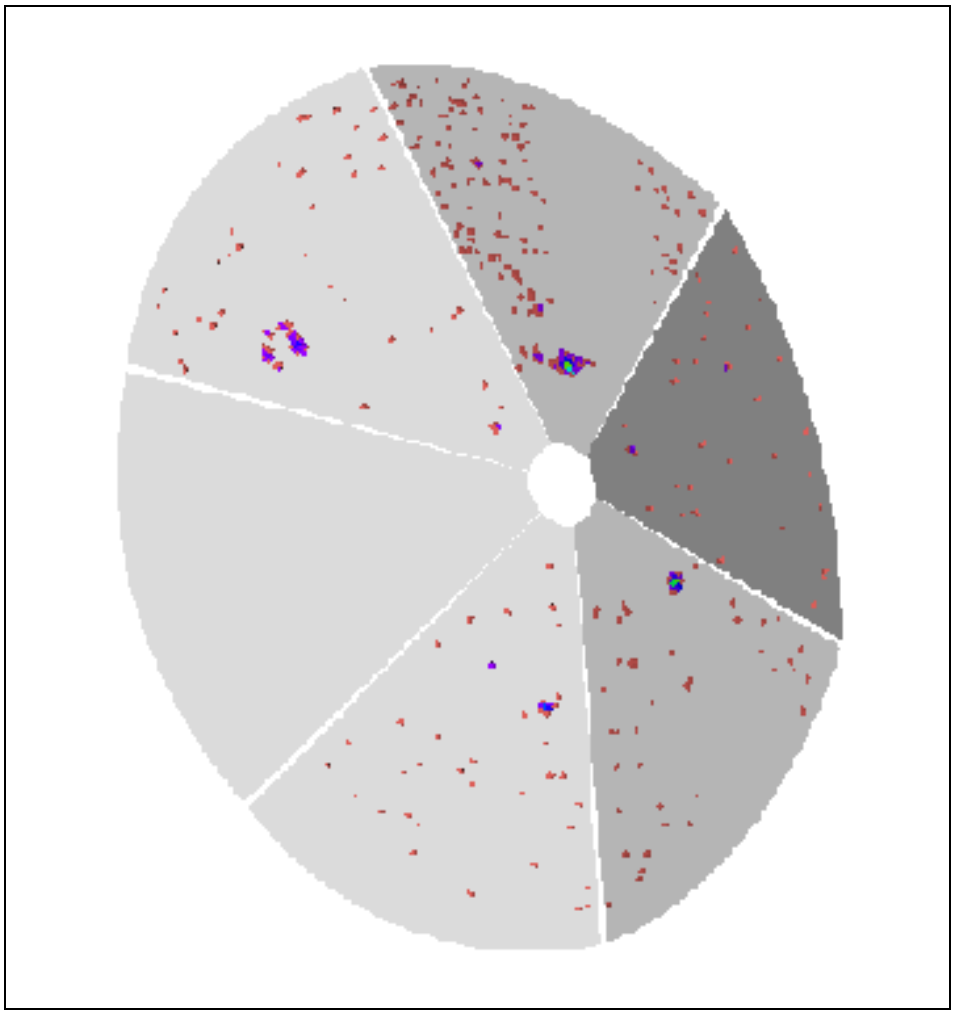

Abbildung 4.1: Ein Einzelereignis des RICH, in dem sich ein Ring befindet. Der Sektor ohne Signal ist der einzige noch nicht mit Elektronik bestückte Sektor. In diesem Ereignis war kein Magnetfeld eingeschaltet.

dieses Untergrundes weg. Ereignisse dieser Art sind in Abbildung 4.2 mit dem „PreShower-Schauer“-Graphen eingezeichnet.

- Wenn abschließend als zusätzliche Einschränkung verlangt wird, daß der TOFino-Detektor vor dem PreShower an gleicher Stelle ein schnelles Teilchen ($\beta \approx c$) detektiert hat, reduziert sich die Zahl der Einträge auf den „PreShower-TOFino“-Graphen. Diese letzte Bedingung ist ein Plausibilitätstest, der aber zu keiner wesentlichen Verringerung der Einträge führt. Die Identifikation durch den PreShower scheint also bereits eine gute Unterdrückung des Untergrundes zu bewirken.

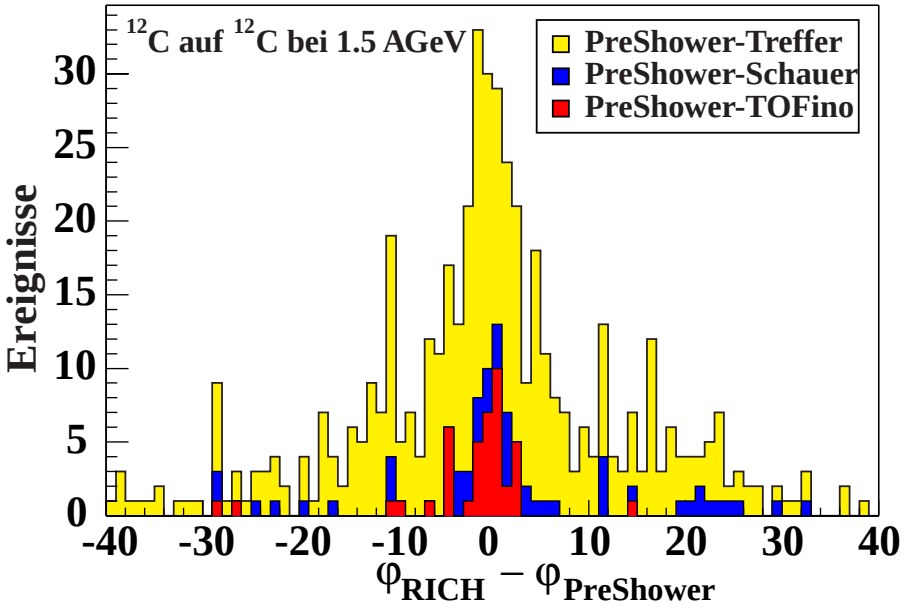

Abbildung 4.2: Die diesem Spektrum zugrundeliegenden Daten wurden ohne Magnetfeld aufgenommen. Gezeigt sind Ereignisse mit Signalen, die in beiden Detektoren jeweils den gleichen Polarwinkel haben [Fab00].

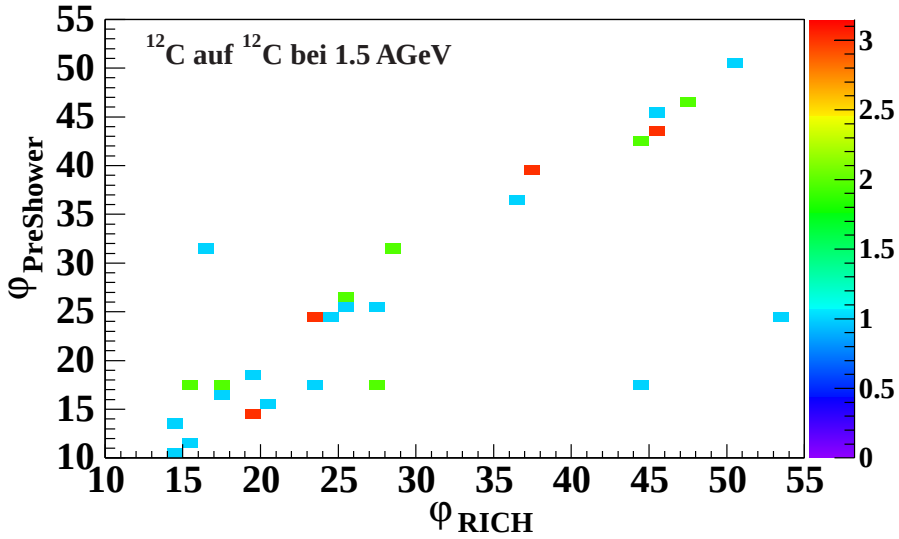

Abbildung 4.3: Hier sind für alle „PreShower-TOFino“ Ereignisse aus Abbildung 4.2 die Polarwinkel φ_{RICH} und $\varphi_{\mathrm{PreShower}}$ gegeneinander aufgetragen [Kid00].

Die gute Übereinstimmung der gefundenen Leptonenkandidaten im RICH und im PreShower wird auch in Abbildung 4.3 illustriert.

4.3.2 Impulsmessungen mit Magnetfeld und Spurrekonstruktion

Einer der entscheidenden Durchbrüche der Strahlzeit war die Aufzeichnung von Ereignissen mit Magnetfeld und hoher Statistik, die es zum ersten Mal erlaubt, über die Richtungsänderung im Magnetfeld den Impuls von Teilchen zu messen. Zur Bestimmung der Impulsänderung sind neben der Magnetfeldstärke mindestens drei Punkte der Teilchentrajektorie notwendig, die nicht alle auf der gleichen Seite des Feldes liegen dürfen. Die Berechnung wird vereinfacht, wenn vor und hinter dem Magnetfeld je zwei Punkte der Trajektorie bekannt sind. Dies erlaubt eine direkte Bestimmung des Krümmungsradius innerhalb des Feldes.

Dennoch ermöglichen bereits drei Meßpunkte die vollständige Bestimmung der Impulsänderung und damit des Gesamtimpulses. Für Leptonen, die ja als einzige Teilchen auch im RICH detektiert werden, ergäben sich diese drei Punkte bereits ohne Verwendung der MDC-Ebenen mit dem Targetpunkt, dem Durchstoßpunkt durch den Spiegel des RICH und dem Ort in den Teilchenidentifikationsdetektoren hinter dem Magneten (namentlich PreShower und TOF). Die im Vergleich zu den MDC wesentlich geringere Ortsauflösung der Teilchenidentifikationsdetektoren verringert allerdings entsprechend die Auflösung bei der Impuls- und damit auch der Energiebestimmung. Wenn zur Bestimmung der Richtung vor dem Magnetfeld die MDC-Ebenen 1 und 2 herangezogen werden, können Impulse von allen Teilchen bestimmt werden.

Dabei kommt zur Impulsbestimmung der sogenannte „Kickplane-Algorithmus“ zum Einsatz, der vereinfachend annimmt, daß der Impulsübertrag im Durchstoßpunkt der Teilchentrajektorie durch eine festgelegte Fläche innerhalb des Magnetfeldes stattfindet.

Wenn man diese Impulse nun gegen die Flugzeit aufträgt, erhält man ein Spektrum wie in Abbildung 4.4. Die Teilchen werden hier nach Massen separiert. Mit Flugzeit und Impuls kann man nun über die Weglänge direkt die Ruhemasse der Teilchen bestimmen. Dies ist in Abbildung 4.5 gezeigt.

Dieses Energie- oder auch Massenspektrum — der Unterschied liegt lediglich in einem Faktor c^2 — ist aber nicht zu verwechseln mit einem invarianten Massenspektrum, in dem die aus zwei Vierimpulsen berechnete Ruhemasse des Ausgangsteilchens eingeht.

In Abbildung 4.5 ist die Trennung zwischen Pionen und Protonen zu erkennen. Allerdings sieht man auch die geringe Energieauflösung, die durch die grobe Bestimmung des Ortes mittels TOF und PreShower begrenzt ist. Durch die MDC-Ebenen 3 und 4 läßt sich im Endausbau eine wesentlich bessere Auflösung erreichen.

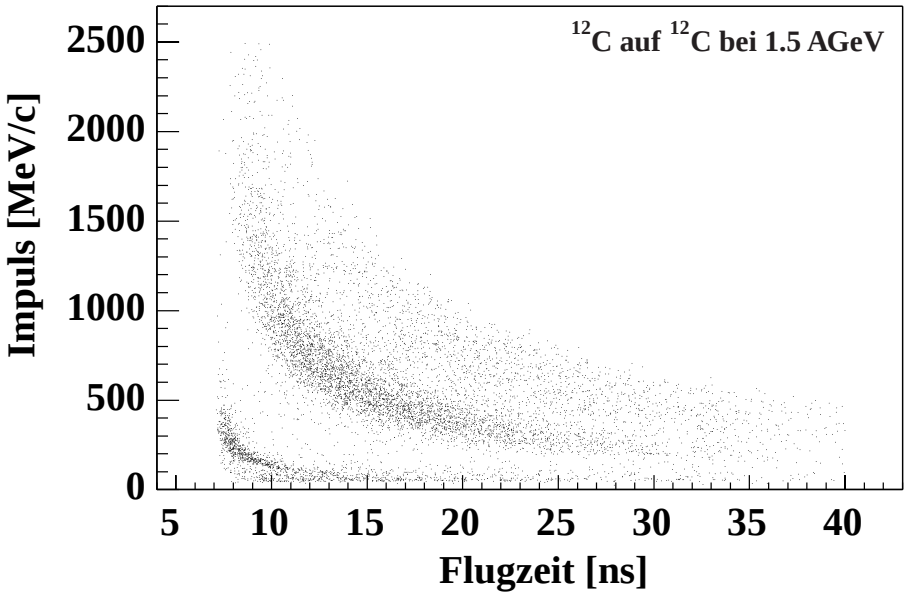

Abbildung 4.4: Impulse gegen die Flugzeit aufgetragen [Sán00].

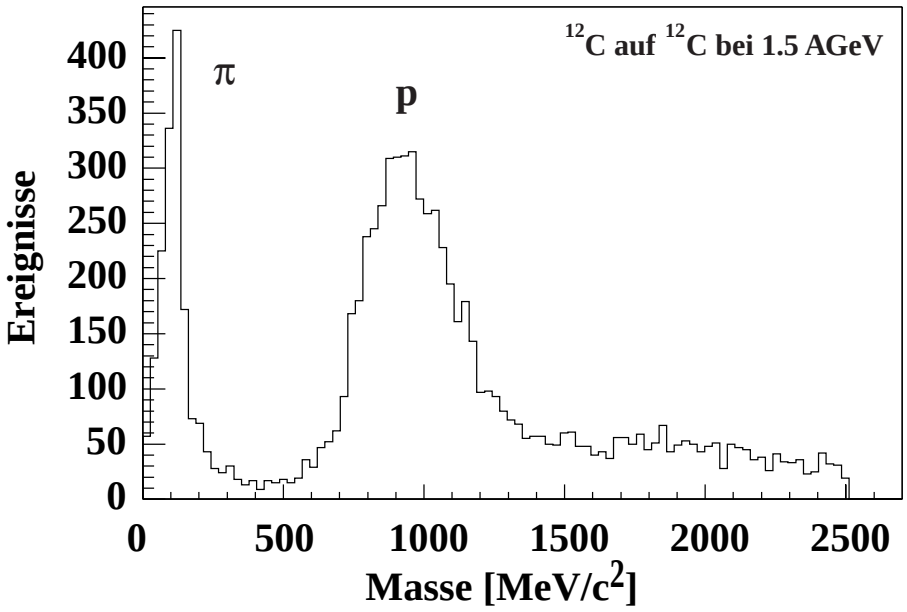

Abbildung 4.5: Spektrum der Gesamtenergie, das π^+ und π^- von Protonen separiert [Sán00].

4.3.3 Hadronenunterdrückung mit der zweiten Triggerstufe

Die Fähigkeit des HADES-RICH, Leptonen aus dem hadronischen Untergrund herauszufiltern, wird deutlich, wenn man Abbildung 4.6 und Abbildung 4.7 ver-

gleicht. In Abbildung 4.6 ist ein Flugzeitspektrum dargestellt, wie es HADES-TOF erzeugt. Die Gesamtzahl der Einträge liegt bei $\approx 675\,000$. Zu erkennen ist ein Maximum bei schnellen Teilchen, das hauptsächlich Pionen, aber auch in geringer Anzahl Elektronen / Positronen enthält. Daneben ist eine breite Verteilung langsamerer Teilchen zu sehen, die hauptsächlich aus Protonen und wenigen schwereren Kernfragmenten besteht.

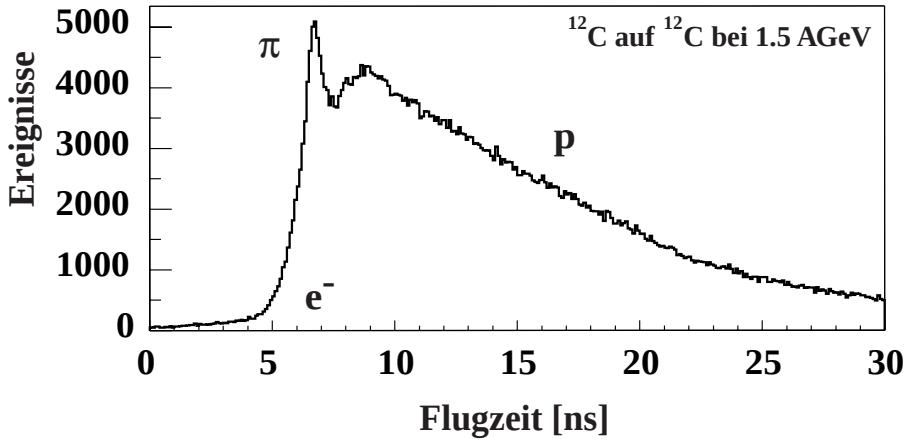

Abbildung 4.6: Ein Flugzeitspektrum von HADES-TOF [Tlú00].

Werden nun aus diesem Spektrum lediglich diejenigen Ereignisse herausgesucht, deren Signal mit einem Ring im RICH korreliert ist, erhält man Abbildung 4.7 mit nur noch 3 128 Einträgen. Das Spektrum zeigt sehr gut, daß lediglich die schnellsten Teilchen dieser Bedingung genügen.

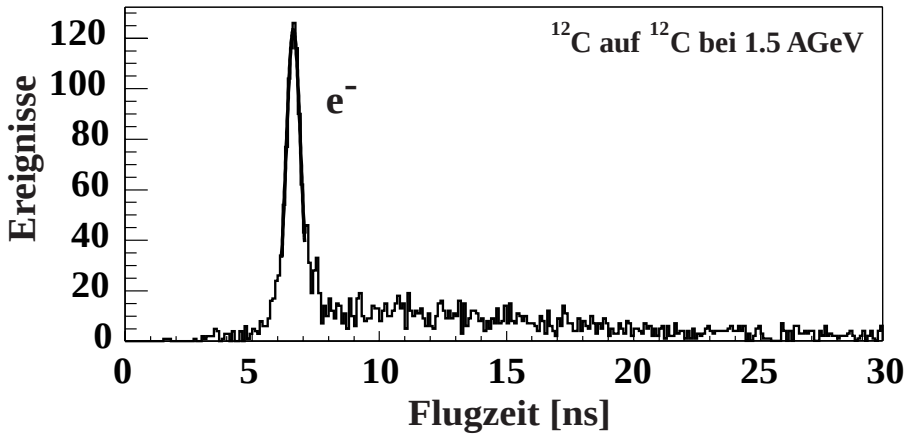

Abbildung 4.7: Alle Einträge aus Abbildung 4.6, die mit einem Leptonenkandidaten des RICH korreliert sind [Tlú00].

Diese gute Möglichkeit zur Leptonenidentifizierung alleine reicht allerdings nicht aus, um HADES bei den geforderten, hohen Raten betreiben zu können. Es muß auch gewährleistet sein, daß die Identifizierung von den Bildverarbeitungseinheiten übernommen werden kann, da nur diese die Daten während der Laufzeit nach Leptonenkandidaten durchsuchen kann. Diese Fähigkeit hat die RICH-IPU in der letzten Strahlzeit nachgewiesen. Hier wurde nicht nur das Funktionieren aller Datenpfade des Triggersystems gezeigt, sondern auch die tatsächliche Eignung zur Erkennung von Leptonen. Um die Qualität der Hardware-Ringsuche zu untersuchen, wurde die Triggerentscheidung nicht aufgrund der gefundenen Ringe, sondern immer positiv getroffen. Abbildung 4.8 zeigt eine Korrelation mit der IPU gefundenen Ringmittelpunkten gegen solche, die die Analysesoftware gefunden hat.

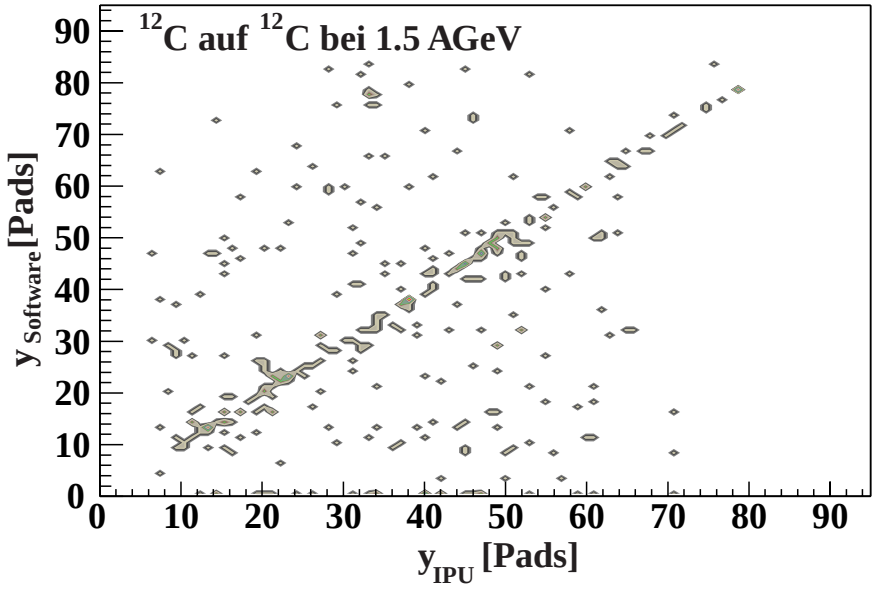

Abbildung 4.8: Radiale Ortsverteilung von Ringmittelpunkten, die „online“ in der IPU gefunden wurden, gegen die Mittelpunkte, die nachträglich mit der Analysesoftware aus allen Daten herausgefiltert wurden [Küh00]. Die Korrelation ist eindeutig zu erkennen.

Die IPU bietet die Möglichkeit, die Mindestanforderungen an gefundene Ringe über eine Schwelle einzustellen. Eine tiefergehende Analyse der Daten muß noch zeigen, ob die nicht korrelierten Datenpunkte auf die zunächst sehr niedrige eingestellte Schwelle zurückzuführen ist, oder aber auf Ringe, die die Analysesoftware als solche erkannt hat, die IPU aber nicht.

Kapitel 5

Ausblick

Im Rahmen der vorliegenden Arbeit wurde vor allem die grundsätzliche Struktur der Datenaufnahmesteuerung erarbeitet und implementiert. Es gibt allerdings eine Reihe von Details, die es noch zu verbessern gilt.

Zunächst müssen die Funktionen in der Datenaufnahmehard- und software zu realisiert werden, mittels derer der Status in Bezug auf Initialisierung und Datenaufnahmelauf extrahiert werden kann. Dies ermöglicht erst die tatsächliche Überwachung der Datenaufnahme, die ja ein wesentliches Kriterium für die Datenaufnahmesteuerung ist. Sowie die Funktionen geschrieben sind, ist es ohne größeren Aufwand möglich, sie auch in die Datenaufnahmesteuerung zu integrieren. Die cas-Bibliothek bietet zusätzlich auch die Möglichkeit, Funktionen asynchron ausführen zu lassen. Dies erlaubt es, während zeitraubenden Aktionen wie z.B. der Initialisierung von Auslesekarten weiterhin noch Prozessvariablen zur Verfügung zu stellen.

Danach ist die Migration zu der neuen EPICS-Release zu nennen, die es in Zukunft erlauben soll, die Sequencer auf den Ausleserechnern selbst zu übersetzen und laufen zu lassen. Erste Versuche in dieser Richtung wurden bereits unternommen. Die Migration des SNL-Codes sollte mit einem lauffähigen neuen EPICS- und Sequencer-Paket reibungslos vonstatten gehen. Eine Umstellung auf asynchrone Bearbeitung der Schreibzugriffe auf Prozessvariablen, wie sie in obigen Absatz vorgeschlagen wird, müßte einhergehen mit einer Verfeinerung des Modells der Datenaufnahmesteuerung innerhalb der Automatentheorie. Zusätzliche Zustände wie „mit der Initialisierung beschäftigt“ wären vonnöten, um die Konsistenz zu erhalten. Dieses Modell entspräche dann aber auch mehr der Realität als das jetzige, in dem sich der Automat zum Teil nicht genau in dem gleichen Zustand befindet wie die Hardware, die er repräsentiert.

Außerdem fehlen noch Wege zur automatischen Protokollierung verschiedener Betriebsparameter. Bis jetzt ist lediglich der Weg der Parameter aus der Datenbank heraus implementiert, nicht aber der Weg in die Datenbank hinein. Grundsätzlich kann hier ebenfalls der Parameterserver zum Einsatz kommen, wenn schreibende Funktionen für die verschiedenen Parameterquellen implemen-

tiert werden. Dies ist bis zu einem gewissen Punkt einfacher, da der Parameterserver die tatsächliche Existenz des individuellen Parameters gar nicht verifizieren, sondern einfach alle Parameter, die in einer bestimmten Form vorliegen, protokollieren muß. Im Falle der Datenbank kann die Einordnung der Werte in die korrespondierenden Tabellen über Trigger[1] erfolgen.

Hier sind wir auch schon bei den umfangreichsten noch ausstehenden Arbeiten angelangt: Bei dem Entwurf der Datenbank sowie ihrer Benutzerschnittstelle selbst. Um zwei der Hauptanforderungen der Datenaufnahmesteuerung

- Fehlertoleranz, Konsistenz
- Benutzerfreundlichkeit, Bedienbarkeit

gleichzeitig erfüllen zu können, ist es notwendig, daß der Operateur auf eine benutzerfreundliche Art und Weise die Konfiguration der Datenaufnahme abändern kann. Dies hat sich zwar bis zum jetzigen Zeitpunkt bereits auf die Änderung der Parameter reduziert, allerdings müssen die Parameter selbst nun auch vernünftig zugreifbar sein. Zur Wahrung der Konsistenz ist ein sorgfältiger Entwurf der Tabellenstruktur notwendig, zur Bedienbarkeit sind intuitive Eingabemöglichkeiten zu erzeugen.

Darüberhinaus stehen noch eine Reihe von Erweiterungen an, die zukünftige Entwicklungen wie z.B. die Implementierung der dritten Triggerstufe oder die Verwendung mehrerer Eventbuilder auch in der Steuerung berücksichtigen.

Alle diese Verbesserungen sind aber bereits bedacht und werden von der jetzigen Implementierung getragen, so daß sie (zum Teil sogar sehr leicht) in die bestehende Steuerung integriert werden können, ohne daß ein größeres Redesign notwendig wird.

[1]Der erweiterte SQL-Standard sieht sogenannte Trigger vor, die es erlauben, bestimmte SQL-Befehle, die der Datenbank übergeben werden, durch andere Befehlsfolgen zu ersetzen. Damit kann man z.B. erreichen, daß anstelle einer virtuellen Tabelle bzw. Ansicht Datensätze einem Algorithmus folgend in tatsächlich existierende Tabellen eingefügt werden.

Anhang A

Beispiel zur Konvertierung der Datenstruktur

Um zu illustrieren, in welcher Weise die Daten zwischen der Datenbank und dem Transport durch das Netz restrukturiert werden müssen, wird hier eine Komponente der Datenaufnahme herausgegriffen. Der RICH verwendet zur Auslese der Daten nach der Triggerentscheidung der zweiten Stufe eine VME-Auslesekarte, der sogenannte RACE[1].

Ein RACE besitzt die Fähigkeit, die Frontend-Elektronik eines halben RICH-Sektors auszulesen. Dazu hat er acht Eingänge, die jeweils mittels eines Flachbandkabels an eine sogenannte „Backplane“ angeschlossen sind, an welche wiederum vier bis fünf Frontends angesteckt werden können. Eine weitere Eigenschaft des RACE ist, daß er als VME-Karte ausgeführt ist und als solche eine Basisadresse auf dem VME-Bus erhält, über die er mit anderen Teilnehmern (in unserem Falle im wesentlichen mit der Auslese-CPU) kommunizieren kann. Unter anderem besitzt ein RACE auch einen Speicherbaustein, der dazu verwendet werden kann, den Teil der Daten, der Adressinformation enthält, umzusetzen, so daß der Datenstrom zum Beispiel anstelle eher unhandlicher Informationen, wie Nummern der Eingänge und Frontends, gleich etwas leichter verwendbares, wie z.B. Zeilen- und Spaltennummer auf der Detektorfläche, enthält. Zuletzt erzeugt jeder RACE ein eigenes Subevent, das durch eine Nummer, die sogenannte Subevent-Id identifiziert wird.

Somit benötigt die Funktion, die den RACE initialisiert, neben einer ganzen Reihe anderer Informationen, unter anderem diese:

- Welche Basisadresse hat die VME-Karte?
- Welche Subevent-Id erzeugt dieser RACE?
- Welche Datei wird verwendet, um den Speicher (DPR[2]) zu beschreiben?

[1] *R*ICH *A*cquisition- and *C*ontrol *E*lectronics
[2] *D*ual *P*orted *M*emory

- An welchen Eingängen sind Backplanes angeschlossen?

Nun hat die Funktion bereits einen generischen Namen für diesen RACE erhalten, über den sie ihn bei der Parameterquelle eindeutig identifiziert. In unserem Beispiel sei dies „RACE0“. Die Liste der benötigten Parameter kann nun für die Funktion in Tabellenform gebracht werden. Das Ergebnis könnte etwa Tabelle A.1 sein.

Name	Index	Nummer	Wert
RACE0	CARDBASE	0	0x10000000
RACE0	SE_ID	0	101
RACE0	DPRFILE	0	race/dpr/rc00.dpr
RACE0	PORT_ENABLE	0	1
RACE0	PORT_ENABLE	1	1
⋮	⋮	⋮	⋮

Tabelle A.1: Beispiel für die Struktur der Parameter einer Hardwarekomponente beim Transport über das Netzwerk.

In der Datenbank ist es allerdings sinnvoll, die Daten in einer gänzlich anderen Struktur bereitzuhalten. Die Aufteilung der Informationen in verschiedene Tabellen sollte dem Entity-Relationship-Modell entsprechen [Mat98]. Dies wird bei relationalen Datenbanken durch die sogenannte Normalisierung erreicht. Der Zweck der Umformung liegt zum einen darin, daß die Struktur der Realität mit der der Daten übereinstimmt — was es erleichtert, die Informationen zu extrahieren — und außerdem gewährleistet wird, daß Informationen nicht redundant gehalten werden. Eine Redundanz führt nämlich normalerweise auf lange Sicht auch zu einer Inkonsistenz der Daten, da die Änderung einer Information ja in der Datenbank unter Umständen an mehreren Stellen geschehen muß, die der Ändernde oft gar nicht alle kennt.

In unserem Beispiel sind die gesuchten Informationen innerhalb der Datenbank in drei verschiedenen Tabellen abgelegt: Tabelle A.2 enthält eine Liste aller VME-Karten, die von der Datenaufnahme verwaltet werden. Um die einzelnen VME-Karten identifizieren zu können, wird eine (im Unterschied zum Namen zeitlich unveränderliche) eindeutige Nummer für jede Karte vergeben. Außerdem ist noch der Typ der Karte abgelegt sowie ihre augenblickliche VME-Basisadresse. Weitergehende Informationen sind in unserem Falle kartenspezifisch und werden daher in eigenen Tabellen abgelegt, von denen es zum Beispiel auch eine für den Typ RACE gibt. Dies ist Tabelle A.3.

Daten, die lediglich für einen spezifischen Kartentyp sinnvoll sind, werden generell in den spezielleren Tabellen abgelegt, um zum einen Speicherplatz zu sparen und, heute wesentlich wichtiger, eine Abfrage auf eigentlich sinnlose Tabelleneinträge von vorneherein abzufangen. Es ist dann einfach nicht mehr möglich, den

CARD_ID	CARD_NAME	CARD_TYPE	CARDBASE
17	RACE0	race	0x10000000
⋮	⋮	⋮	⋮

Tabelle A.2: Liste der VME-Karten in der Datenbank.

CARD_ID	SE_ID	DPRFILE	...
17	101	race/dpr/rc00.dpr	...
⋮	⋮	⋮	

Tabelle A.3: Parameter der VME-Karten vom Typ „RACE“.

Namen der DPR-Datei für ein Fanout-Board abzufragen. Dagegen werden Daten, die jede VME-Karte haben muß, auch in der allgemeinen Tabelle abgelegt. In unserem Fall erlaubt es diese Vorgehensweise zum Beispiel, Konflikte bei der Vergabe von VME-Basisadressen einfach zu erkennen, auch wenn sie zwischen Karten unterschiedlichen Typs auftreten.

BACKPLANE_ID	CARD_ID	PORT_NR	...
36	17	0	...
48	17	1	...
⋮	⋮	⋮	

Tabelle A.4: Liste der Backplanes.

In Tabelle A.4 wiederum sind die Backplanes aufgeführt, die mit den verschiedenen Eingängen des RACE verbunden sind.

Es ist nun notwendig, die Informationen aus Tabelle A.2 bis Tabelle A.4 so umzuformen, daß das Ergebnis Tabelle A.1 ist. Für derartige Aufgaben wurde die Abfragesprache SQL entwickelt. Zunächst müssen die einzelnen Ergebnisse in die Form der Ergebnistabelle gebracht werden. Mit

```
SELECT
    CARD_NAME AS Name, "CARDBASE" AS Index, 0 As Nummer,
        CARDBASE AS Wert
FROM
    VME_CARDS;
```

erhält man aus Tabelle A.2 als Ergebnis Tabelle A.5. Neben dem Index „CARDBASE“ wird auch die laufende Nummer „0“ fest eingetragen. Schließlich wird für die einzelne Ausgabezeile keine Sortierreihenfolge benötigt.

Name	Index	Nummer	Wert
RACE0	CARDBASE	0	0x10000000
⋮	⋮	⋮	⋮

Tabelle A.5: Zwischenergebnis der Umformung von Tabelle A.2.

Auch Daten aus mehreren Tabellen miteinander zu verbinden, ist problemlos möglich:

```
SELECT
    VME_CARDS.CARD_NAME AS Name, "DPRFILE" AS Index, 0 As Nummer,
        RACES.DPRFILE AS Wert
FROM
    VME_CARDS, RACES
WHERE
    VME_CARDS.CARD_ID = RACES.CARD_ID;
```

Das Ergebnis dieser Abfrage ist Tabelle A.6.

Name	Index	Nummer	Wert
RACE0	DPRFILE	0	race/dpr/rc00.dpr
⋮	⋮	⋮	⋮

Tabelle A.6: Zwischenergebnis der Umformung von Tabelle A.2. und Tabelle A.3.

Natürlich sind mehrzeilige Ergebnistabellen erlaubt. Die Abfragen, die zu den Einzelergebnissen führen, können als Ansichten abgelegt („CREATE VIEW“-Befehl) und dann zu einer großen Tabelle zusammengefaßt („UNION“-Befehl) werden, die dann genauso wie Tabelle A.1 aussieht.

Anhang B

Liste der Hardwaremodule

Bibliotheks-name	gesteuerte Komponente	Beschreibung	Anzahl im Vollausbau
dtu	DTU und CTU [Lin98]	die zur Verteilung der LVL1- und LVL2-Triggerinformation eingesetzten VME-Karten	9 + 1
fb	Fanout Board [Kaj98]	Verteilung der Triggerinformation vom PreShower-Crate zu der PreShower-Frontend-Elektronik	1
hub	Triggerhub [Kaj00]	Komponente zur sternförmigen Verteilung der Triggerinformation an die DTUs; Damit wird die zeitweise verwendete Busverkabelung abgelöst	2
ipc	PreShower Image Processing Card [Pet00]	Bildverarbeitungseinheit zum Erkennen von elektromagnetischen Schauern	12
lm	Link Module [Pet00]	Stellt die Verbindung zwischen den IPCs und der Matching Unit her	1

Bibliotheks-name	gesteuerte Komponente	Beschreibung	Anzahl im Vollausbau
mu	Matching Unit [Tra99]	Sammelt Informationen über Leptonenkandidaten von IPC, PRC und TIP und trifft auf dieser Grundlage die LVL2-Triggerentscheidung; übergibt diese Information über VME an die im gleichen Crate befindliche CTU zur Weiterverteilung	1
prc	RICH Pattern Recognition Card [Leh00]	Bildverarbeitungseinheit zur Ringerkennung	6
race	RICH Acquisition and Control Electronics [Böh00]	VME Auslesemodul für den RICH	12
sam	GSI Steuerungs- und AusleseModul [Hof00]	VME Auslesemodul für MDC	24
sis3801	Scaler	Gewinnt Informationen über die Totzeit der DAQ, indem er LVL1-Trigger- und Triggertypen während anliegendem Busy-Signal mitzählt	1
tip	TOF Image Processing [Lin01]	Sowohl Auslesemodul für die v878-Module als auch Erkennung von Leptonenkandidaten über die Flugzeit und Konzentrator-Modul für die anderen TIPs, die die VME-Crates selbständig auslesen	3 + 1
v878	CAEN 32-Channel ADC / TDC	Frontend-Elektronik für TOF	ca. 64

Tabelle B.1: Liste aller Hardwaremodule, die in die Datnaufnahmesteuerung integriert sind. Jedes der Module verfügt über die in Tabelle 3.3 festgelegten Bibliotheksfunktionen zur Steuerung.

Anhang C

Beschreibung der Parameterschnittstelle

Um Parameter aus einer Parameterquelle zu Extrahieren, werden vier verschiedene Funktionen bereitgestellt:

`Param_getString()` liefert eine einzelne Zeichenketten

`Param_getInt()` liefert eine einzelne Ganzzahlen

`Param_getStringArray()` liefert eine geordnete Liste von Zeichenketten

`Param_getIntArray()` liefert eine geordnete Liste von Ganzzahlen

Bevor sie benutzt werden können, muß zunächst eine Parameterstruktur erzeugt werden, in der Daten für den Zugriff auf die Parameterquelle enthalten sind. Dies geschieht mit

```
#include <allParam.h>
#include <stdlib.h>

⋮

{
    Param *param;
    param = malloc(sizeof(Param));
    if(param != 0) {
        if(conParam(param) != 0) {
            sprintf(stderr, "conParam() for param failed.\n");
            exit(-1);
        }
    } else {
        sprintf(stderr, "malloc() for param failed.\n");
```

```
        exit(-1);
    }
}
```

Danach kann die Struktur zum extrahieren der Daten verwendet werden. Der Speicherplatz für die Daten muß von der aufrufenden Funktion bereitgestellt werden. Sie muß sich später auch um die Freisetzung des belegten Speichers kümmern. Am Ende muß der Speicherplatz für die Parameterstruktur wieder freigegeben werden. Dies geschieht mittels

```
{
    desParam(param);
    free param;
}
```

Im Fehlerfall liefert die jeweils aufgerufene Parameterfunktion den Wert -1 zurück. Dann kann über `Param_getErrStr()` eine Fehlermeldung abgerufen werden. Alle Funktionsdeklarationen aus `allParam.h` sind im folgenden aufgelistet:

```
int conParam(Param *);
void desParam(Param *);

int Param_getString(const Param *, const char *, const char *,
  int *, char *);
int Param_getInt(const Param *, const char *, const char *, int *,
  unsigned long int *);
int Param_getStringArray(const Param *, const char *, const char *,
  int, int *, char **);
int Param_getIntArray(const Param *, const char *, const char *,
  int, int *, unsigned long int *);

const char *Param_getErrStr(const Param *);
```

Zuletzt finden sich noch vier Beispiele für die Benutzung der Parameterfunktionen.

```
/* Param_getString */
    const char *name = "tname";
    const char *idx1 = "tstring";
    char values[PARAM_MAX_VALUE_LEN];
    int row;
    if(Param_getString(param, name, idx1, &row, values) != 0) {
        sprintf(stderr, "%s : Param_getString() failed: %s",
            argv[0], Param_getErrStr(param));
```

```
        exit (-1);
    } else if(row == 0) {
        sprintf(stderr, "%s(%s) not found.\n", name, idx1);
    } else {
        printf("%s(%s): %s\n", name, idx1, values);
    }

/* Param_getInt */
    const char *idx2 = "tint";
    unsigned long int valuei;
    if(Param_getInt(param, name, idx2, &row, &valuei) != 0) {
        sprintf(stderr, "%s : Param_getInt() failed: %s",
            argv[0], Param_getErrStr(param));
        exit (-1);
    } else if(row == 0) {
        sprintf(stderr, "%s(%s) not found.\n", name, idx2);
        exit (-1);
    } else {
        printf("%s(%s) : %s\n", name, idx2, valuei);
    }

/* Param_getStringArray */
    const char *idx3 = "tstringa";
    char *valuesa[LEN];
    int maxrows = LEN;
    int rows;
    int i;
    for (i = 0 ; i < maxrows ; i++) {
        valuesa[i] = malloc(PARAM_MAX_VALUE_LEN * sizeof(char));
        if(valuesa[i] == 0) {
            sprintf(stderr, "malloc() for value failed.\n");
        exit(-1);
        }
    }
    if(Param_getStringArray(param, name, idx3, maxrows, &rows,
        valuesa) != 0) {
        sprintf(stderr, "%s : Param_getStringArray() failed: %s",
            argv[0], Param_getErrStr(param));
        exit (-1);
    } else if(row == 0) {
        sprintf(stderr, "%s(%s) not found.\n", name, idx3);
        exit (-1);
    } else {
```

```
        for (i = 0 ; i < maxrows ; i++) {
            printf("%s(%s %d) : %s\n", name, idx3, i, valuesa[i]);
            free(valuesa[i]);
        }
    }

/* Param_getIntArray */
    const char *idx4 = "tinta";
    unsigned long int valueia[LEN];
    if(Param_getIntArray(param, name, idx4, maxrows, &rows,
        valueia) != 0) {
        sprintf(stderr, "%s : Param_getIntArray() failed: %s",
            argv[0], Param_getErrStr(param));
        exit (-1);
    } else if(rows == 0) {
        sprintf(stderr, "%s(%s) not found.\n", name, idx4);
        exit (-1);
    } else {
        for (i = 0 ; i < maxrows ; i++) {
            printf("%s(%s %d) : %d\n", name, idx4, i, valueia[i]);
        }
    }
```

Glossar

ADL *A*SCII *D*isplay *L*ist

ALICE *A L*arge *I*on *C*ollider *E*xperiment

ANL *A*rgonne *N*ational *L*aboratory

ATM *A*synchronous *T*ransfer *M*ode: Ein Protokoll für breitbandige Glasfasernetze, das einzelnen Verbindungen Mindestbandbreiten garantiert werden können. ATM verzichtet auf Empfangsbestätigung der Pakete.

C Nach ANSI X3.159-1989 standardisierte Programmiersprache

CEBAF *C*ontinuous *E*lectron *Bea*m *F*acility: Das heutige Thomas Jefferson Laboratory.

CERN *C*onseil *E*uropéen pour la *R*echerche *N*ucléaire

CODA *C*EBAF *O*nline *DA*Q

COMPASS a *Co*mmom *M*uon *P*roton *A*pparatus for *S*tructure and *S*pectroscopy

CVS *C*oncurrent *V*ersions *S*ystem: Ein Werkzeug zur Verwaltung von Quellcode von Softwareprojekten, die von verschiedenen Programmierern an unterschiedlichen Orten gleichzeitig entwickelt werden sollen. Nähere Informationen dazu sind unter http://www.cvshome.org/ zu finden.

DAQ *D*ata *Acq*uisition

(R)DBMS (*R*elationales) *D*aten*b*ank *M*anagement *S*ystem

DELPHI *De*tector with *L*epton, *P*hoton and *H*adron *I*dentification

DESY *D*eutsches *E*lektronen-*Sy*nchrotron

DLS *Dil*epton *S*pectrometer — Ein Experiment, daß am BEVALAC in Berkeley, USA aufgebaut wurde, um Dileptonenpaare zu messen.

DSP *D*igital *S*ignal *P*rocessor

EB *E*vent *B*uilder

EPICS *E*xperimental *P*hysics and *I*ndustrial *C*ontrol *S*ystem

FPGA *F*ield *P*rogrammable *G*ate *A*rray

FSM *F*inite *S*tate *M*achines: Endliche Automaten

GSI *G*esellschaft für *S*chwer*i*onenforschung mbH

GUI *G*raphical *U*ser *I*nterfaces

HADES *H*igh *A*cceptance *Di*electron *S*pectrometer

HYDRA Auf ROOT basierende Analyseumgebung für HADES.

IPC *I*mage *P*rocessing *C*ard

IPU *I*mage *P*rocessing *U*nit: Eine Bildverarbeitungseinheit

MDC *M*ini *D*rift *C*hambers

OPI *Op*erator *I*nterface

POSIX POSIX ist die gängige Bezeichnung für den IEEE Std 1003.1-1990 Standard, der Schnittstellen zur Benutzung von und Programmierung für Betriebssysteme festlegt. Praktisch alle gängigen UNIX-Derivate halten sich in weiten Teilen an diesen Standard, um portierbare Programmierung von Anwendungen zu erleichtern.

Präcompiler Ein Präcompiler ist ein Programm, das Quellcode einer Programmiersprache in Quellcode einer anderen übersetzt. Im Gegensatz dazu erzeugt ein Compiler ausführbaren Code (also Maschinensprache). Allerdings ist diese Unterteilung nicht frei von Ausnahmen. Ein typischer C-Compiler zum Beispiel erzeugt Assembler-Code, der dann erst von einem Assembler ist Maschinensprache übersetzt wird.

PRC *P*attern *R*ecognition *C*ard

RICH *R*ing *I*maging *Ch*erenkov

ROOT Eine objektorientierte Klassenhierarchie, die von René Brun und Fons Rademakers am CERN zur Datenanalyse in der Teilchenphysik entwickelt wurde.

RPC *R*emote *P*rocedure *C*all; ein Modell zur Steuerung verteilter Prozesse.

SEB *S*ub *E*vent *B*uilder

SIS *S*chwer*i*onen*s*ynchrotron; Teilchenbeschleuniger an der GSI.

SMP *S*ymmetric *M*ulti*p*rocessing

SNC *S*tate *N*otation *C*ompiler

SNL *S*tate *N*otation *L*anguage

SQL *S*tructured *Q*uery *L*anguage: Eine im ISO-Standard 9075 definierte sogenannte Sprache der 4. Generation (4GL). Die aktuelle Definition stammt von 1997 und wird als SQL3 bezeichnet.

TIP *T*OF *I*mage *P*rocessing

TOF *T*ime *O*f *F*light

VME *V*ersa *M*odule *E*urocard: Ein Datenbussystem für Industrieanlagen, das auch in der Hochenergiephysik immer breitere Anwendung findet..

Cross-Compiler Ein Cross-Compiler erzeugt Maschinencode nicht für die Plattform und nicht für das Betriebssystem, auf der bzw. unter dem er läuft.

Literaturverzeichnis

[Ago98] C. Agodi, G. Bellia, R. Coniglione, et al., IEEE Trans. Nucl. Sci. 45 (1998) 665.

[All00] C. Allison, C/C++ Users Journal Sep. 2000.

[Blo92] J. Bloomer, *Power Programming with RPC*, O'Reilly & Associates, Inc., 1992.

[Böh00] M. Böhmer, *Das Auslesesystem für den Ringabbildenden Cherenkovdetektor im HADES Spektrometer*, Diplomarbeit, Technische Universität München, 2000.

[Bre99] T. Bretz, *Magnetfeldeigenschaften des Spektrometers HADES*, Diplomarbeit, Technische Universität München, 1999.

[Bru00] R. Brun and F. Rademakers, *ROOT User's Guide*, http://root.cern.ch/, 2000.

[Cas99] W. Cassing and E. L. Bratkovskaya, Phys. Rep. 308 (1999) 65.

[COM96] COMPASS Collaboration, *Common Muon and Proton Apparatus for Structure and Spectroscopy*, Technical Report, CERN, 1996.

[Dal00] B. Dalesio and M. Kraimer, *EPICS Homepage*, http://www.aps.anl.gov/epics/, 2000.

[Fab00] L. Fabbietti, priv. Mitteilung, 2000.

[Fri99] J. Friese et al., Progr. in Part. and Nucl. Physics 42 (1999) 235.

[Ger96] R. Gernhäuser et al., Nucl. Instrum. Meth. A371 (1996) 300.

[HAD94] HADES Collaboration, *Proposal for a High-Acceptance Di-Electron Spectrometer*, Technical Report, GSI, 1994.

[Hof00] J. Hoffmann, priv. Mitteilung, 2000.

[Hom99] J. Homolka and A. Kastenmüller, *DAQ basic principles*, http://www-hades.gsi.de/daq/, 1999.

[Hop79] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[Kaj98] M. Kajetanovicz, *Fanout Board — Trigger Distribution Board Manual*, 1998.

[Kaj00] M. Kajetanovicz, *Trigger Hub Manual*, 2000.

[Kas00] A. Kastenmüller, *Nachweis von e^+e^--Paaren aus Schwerionenstößen mit einem RICH Detektor*, Dissertation, Technische Universität München, 2000.

[Ker90] B. W. Kernighan und D. M. Ritchie, *Programmieren in C*, Carl Hanser Verlag, zweite Auflage, 1990.

[Kid00] L. Kidon, priv. Mitteilung, 2000.

[Küh00] W. Kühn, priv. Mitteilung, 2000.

[Kya98] O. Kyas, *ATM-Netzwerke*, International Thomson Publishing, 1998.

[Lan94] H. Langendörfer und B. Schnor, *Verteilte Systeme*, Carl Hanser Verlag, 1994.

[Leh00] J. Lehnert, *Echtzeit-Mustererkennung zum Elektronennachweis mit einem RICH-Detektor in relativistischen Schwerionenkollisionen*, Dissertation, Justus-Liebig-Universität Gießen, 2000.

[Lew91] D. Lewine, *POSIX Programmer's Guide*, O'Reilly & Associates, Inc., 1991.

[Lin98] E. Lins, *DTU — Detector Trigger Unit Manual*, 1998.

[Lin01] E. Lins, 2001, Dissertation, Justus-Liebig-Universität Gießen, noch nicht veröffentlicht.

[Lup98] W. Lupton, *EPICS Seq Module*, http://pipeline.keck.hawaii.edu:3636/realpublic/epics/seq/, 1998.

[Mac96] M. Macowicz, *Run Control System — Preliminary Note*, http://www.cern.ch/~macowicz/, 1996.

[Mat98] G. Matthiesen und M. Ullstein, *Relationale Datenbanken und SQL*, Addison-Wesley, 1998.

[Mat99] P. Mato, *Trigger and Data Acquisition*, Summer Student Lectures at CERN/EP-ACD, 1999.

[Mor99] A. Morsch, in *International Europhysics Conference on High-energy Physics*, 1999.

[Mün95] M. Münch, *Ein Datenaufnahmesystem mit Echtzeit-Bildverarbeitung für ringabbildende Cherenkovdetektoren*, Diplomarbeit, Technische Universität München, 1995.

[Mün01] M. Münch, 2001, Dissertation, Technische Universität München, noch nicht veröffentlicht.

[Ous94] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[Pet00] M. Petri, *Entwicklung eines kombinierten Auslese- und Echtzeit-Triggersystems zum Nachweis von Elektronen/Positronen-Signaturen in einem elektromagnetischen Schauerdetektor*, Dissertation, Justus-Liebig-Universität Gießen, 2000.

[Rap99] R. Rapp and J. Wambach, hep-ph/9909229 .

[Sal95] P. Salabura et al., Nucl. Phys. B 44 (1995) 195.

[Sán99] M. Sánchez, *HYDRA Manual*, 1999.

[Sán00] M. Sánchez, priv. Mitteilung, 2000.

[Sch95] H. Schön, *HADES — Ein Dileptonenspektrometer hoher Akzeptanz für relativistische Schwerionenkollisionen*, Dissertation, Universität Frankfurt, 1995.

[Sen93] P. Senger et al., Nucl. Instrum. Meth. A327 (1993) 393.

[Sen00] P. Senger, priv. Mitteilung, 2000.

[Slo89] M. Sloman und J. Kramer, *Verteilte Systeme und Rechnernetze*, Carl Hanser Verlag, 1989.

[Tlú00] P. Tlústy, priv. Mitteilung, 2000.

[Tra99] M. Traxler, *The Matching Unit for the HADES Trigger-System*, 1999.

[Wei94] W. Weise, in *Proceedings of the Workshop on Dilepton Production in Relativistic Heavy Ion Collisions*, 22, 1994.

Danksagung

Zunächst möchte ich mich ganz herzlich bei Prof. Körner für die freundliche Aufnahme an seinem Lehrstuhl bedanken. Ein so freundliches und hilfsbereites Klima wie hier ist mit Sicherheit nicht überall selbstverständlich.

Josef Homolka holte mich als Werkstudent zu E 12 und leitete so meine „Karriere“ an diesem Lehrstuhl ein. Die Tür zu seinem Zimmer stand mir in den Jahren immer und für jede Frage offen. Jürgen Friese, der großen Motivator, übertrug mir einen großen Teil der Faszination, die HADES nun für mich beinhaltet.

Mathias Münch ist es nicht nur zu verdanken, daß ich in diesem Jahr der Diplomarbeit eine Menge gelernt habe (auch wenn's manchmal etwas länger dauerte und ich insbesondere bei der Flugtauglichkeitsprüfung glatt durchfiel). Er war auch ein guter Freund, der sehr genau wußte, wann es mal nicht so gut lief und ich ein wenig „Aufbauhilfe“ benötigte.

Michael Böhmer, Anton Kastenmüller und Sönke Schröder komplettierten das Arbeitszimmer, in dem neben der Arbeit auch Gespräche über Physik und anderes (häufig anderes) ebensowenig zu kurz kamen wie der Humor. Auch hierfür Danke.

Bei Thomas Eberl und Andreas Stolz, mit denen ich nicht nur die Tücken der Systemadministration, sondern auch manch fernöstliche Teezeremonie teilte, möchte ich mich ebenso bedanken wie bei Laura Fabbietti, mit der mich seit meiner Diplomandenzeit eine gute Freundschaft verbindet.

Sie sind nur stellvertretend herausgegriffen für alle E 12er. Als einer von ihnen habe ich mich sehr wohl gefühlt. Besonders Frau Dreiseitl, unsere immer freundliche, hilfsbereite und kompetente Heldin des Alltags, möchte ich hier nennen, die für mich immer ein Symbol des guten Arbeitsklimas bei E 12 war.

Aber auch Leute außerhalb der TU mußte ich nie zweimal fragen, wenn ich einmal wieder Hilfe brauchte. Insbesondere Ilse und Wolfgang Koenig, das dynamische Duo der GSI, möchte ich hier erwähnen, aber auch Burkhard Kolb, Erik Lins, Michael Traxler und Jörn Wüstenfeld sowie all die anderen aus der großen HADES-Kollaboration.

Bei meiner Familie und meinen Freunden möchte ich mich dafür bedanken, daß sie mich immer unterstützt haben und meine Entscheidung, Physik zu studieren, ebenso trugen, wie sie mich ertrugen, wenn ich davon schwärmte oder darüber schimpfte.