Anwendung objektorientierter Methoden der Datenanalyse auf Endzustände mit Jets im H1-Detektor

von

Ingo Strauch

Diplomarbeit in Physik
vorgelegt der
Fakultät für Mathematik, Informatik und Naturwissenschaften
der
Rheinisch-Westfälischen Technischen Hochschule Aachen

im März 2001

angefertigt am
I. Physikalischen Institut, Lehrstuhl B
Prof. Dr. Ch. Berger

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Das H1-Experiment 1
1.2 Der H1-Detektor 2
1.3 Motivation neuer Software 4
1.3.1 Bisherige Umgebung 5
1.3.2 Das H1-OO-Projekt 6

2 Die Analyseumgebung bei H1 **7**
2.1 Datenmodell und technische Aspekte 7
2.2 Analyseschritte 9
2.3 Schwachstellen 10
2.4 Verbesserungsansätze 11

3 Entwicklung der Software **15**
3.1 Softwaredesign 15
3.1.1 Interface und Implementierung 15
3.1.2 Klassendesign nach der CRC-Methode 16
3.2 Konzepte im OO-Projekt 18
3.2.1 Die drei Ebenen der Datenspeicherung 18
3.2.2 Der Runkatalog 18
3.2.3 Das zentrale Objekt: der `H1Tree` 19
3.2.4 Persistente Zeiger: der `H1PointerManager` 20
3.3 Technische Aspekte 20
3.3.1 Programmierregeln 20
3.3.2 Versionsverwaltung 21
3.3.3 Release-Strategie 22
3.3.4 Weitere Hilfsmittel 23

4 Entwicklung der Teilchenklassen **25**
4.1 Bisheriges Teilchenmodell in H1PHAN 25
4.2 Entwicklung neuer Teilchenmodelle 26
4.2.1 Eine Klassenhierarchie ohne identifizierte Teilchen 27
4.2.2 Zwei unabhängige Hierarchien für allg. und identifizierte Teilchen . 29

4.2.3 Eine einzige Klassenhierarchie inklusive identifizierten Teilchen . . . 30
4.3 Das für diese Arbeit gewählte Modell . . . 32
4.3.1 Klassenhierarchie . . . 32
4.3.2 Die konkrete Analyseumgebung . . . 35
4.3.3 Jetfinder im OO-Framework . . . 36

5 Physikalische Fragestellung 39
5.1 Theorie . . . 39
5.1.1 Kinematik . . . 39
5.1.2 Jetalgorithmen . . . 42
5.2 Datensatz . . . 45
5.2.1 Vorselektion . . . 45
5.2.2 Reduktion des Untergrundes . . . 47
5.3 Erzielte Resultate . . . 50

6 Ergebnisse der Tests 59
6.1 Technische Resultate . . . 59
6.2 Bewertung . . . 61
6.2.1 Defizite in der Implementierung . . . 61
6.2.2 In der Entwicklung befindliche Konzepte . . . 62

7 Zusammenfassung und Ausblick 65

A Dokumentation 67
A.1 Glossar softwarespezifischer Begriffe . . . 67
A.2 Aufsetzen der Umgebung . . . 68
A.3 C++ Programming Guidelines . . . 69
A.4 Pakete und deren Verwendung . . . 73

B Werkzeuge zur Software-Entwicklung 75
B.1 CVS: Concurrent Versioning System . . . 75
B.1.1 CVS Glossar . . . 75
B.1.2 Die wichtigsten CVS Kommandos . . . 76
B.2 CRC: Classes, Responsibilities, Collaborators . . . 77

Abbildungsverzeichnis 79

Tabellenverzeichnis 81

Literaturhinweise 83

Danksagung 87

Kapitel 1

Einleitung

Diese Arbeit beschäftigt sich zum einen mit der Entwicklung einer neuen Analyseumgebung für das H1-Experiment am HERA-Speicherring des Deutschen Elektronen-Synchrotrons (DESY). Zum anderen geht es um die Anwendung dieser Software auf eine Analyse zur Produktion von Jets in ep-Streuung.

In diesem Kapitel soll das Experiment vorgestellt werden bevor Kapitel 2 auf die bisherige Situation der Datennahme und -verarbeitung eingeht. Deren Schwachstellen werden Ansätze für ein leistungsfähigeres und flexibleres System gegenübergestellt. In Kapitel 3 geht es um generelle Vorgehensweisen bei der Software-Entwicklung sowie Grundkonzepte der neuen Analyseumgebung. Kapitel 4 beschäftigt sich mit der Softwareseite dieser Arbeit, während in Kapitel 5 eine erste Version der Umgebung zum Einsatz gebracht wird. Die erzielten Ergebnisse zeigt Kapitel 6, worauf eine Zusammenfassung mit Ausblick in Kapitel 7 folgt.

Da sich diese Arbeit zu einem großen Teil mit Softwarefragen beschäftigt, wird auf die Übersetzung einiger englischen Fachbegriffe verzichtet, die sich als Quasi-Standard durchgesetzt haben. Im Anhang befindet sich ein Glossar der verwendeten Ausdrücke.

1.1 Das H1-Experiment

Das DESY wurde 1959 in Hamburg zur Erforschung fundamentaler Eigenschaften der Teilchenphysik gegründet und wird aus öffentlichen Mitteln finanziert. Der Speicherring HERA (Hadron-Elektron-Ring-Anlage; siehe Abb. 1.1) ist ein unterirdischer, nahezu kreisrunder Tunnel mit einem Umfang von 6,3 km. An vier Stellen des Ringes wurden Detektoren aufgebaut, die sich in zwei Kategorien aufteilen lassen: die Kollisionsexperimente H1 und Zeus sowie die Strahl-Target-Experimente Hera-B und Hermes. Im HERA-Ring werden Elementarteilchen durch Vorbeschleuniger eingespeist und während etlicher Umläufe auf sehr hohe Energien beschleunigt. Diese stoßen schließlich im H1-Detektor von entgegengesetzten Seiten zusammen und erzeugen viele andere Elementarteilchen, die vom Detektor

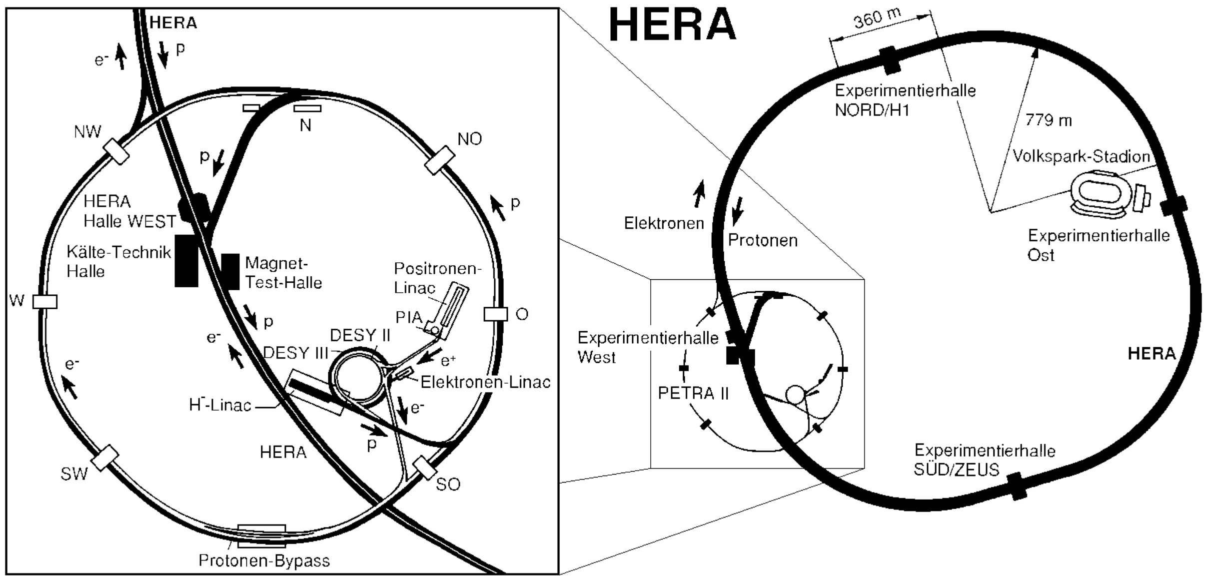

Abbildung 1.1: *Der HERA Speicherring und die Vorbeschleuniger*

gemessen werden. Zur Zeit wird bei HERA mit Protonen und Elektronen[1] einer Energie von 920 GeV bzw. 27,6 GeV experimentiert. Die zur Kollision kommenden Teilchen sind unpolarisiert, deshalb wird in der Ebene senkrecht zum Strahlrohr keine bevorzugte Richtung der produzierten Teilchen erwartet.

Die H1-Kollaboration besteht aus etwa 400 Wissenschaftlern, die aus 39 Instituten und 12 Ländern stammen. Der H1-Detektor ist während der Jahre 1988-1992 gebaut worden und begann 1992 seine Datennahme. Das Ziel ist die Bestimmung der Struktur des Protons, Untersuchung grundlegender Wechselwirkungen zwischen Elementarteilchen sowie das Standardmodell der Physik auf die Probe zu stellen.

1.2 Der H1-Detektor

Da sich die vorliegende Diplomarbeit im Schwerpunkt mit der Entwicklung von Software beschäftigt, soll der H1-Detektor nicht bis ins Detail vorgestellt werden. Ausführliche Informationen finden sich in [H1 93] bzw. in [H1 97]. An dieser Stelle wird lediglich eine Kurzvorstellung gegeben. Es handelt sich um einen 4π-Detektor, d.h. er umschließt den Wechselwirkungspunkt nahezu vollständig. In der azimuthalen Ebene ist der Detektor symmetrisch. Im Gegensatz dazu fällt in der Abbildung 1.2 der asymmetrische Aufbau in Richtung der Strahlachse auf. Grund dafür ist, daß die Protonen auf höhere Energien

[1] Seit 1994 (bis auf eine Periode von 1998 bis Anfang 1999) wurden die Elektronen durch Positronen ersetzt. Im nachfolgenden wird diesbezüglich kein Unterschied gemacht und beide werden stets als Elektronen bezeichnet.

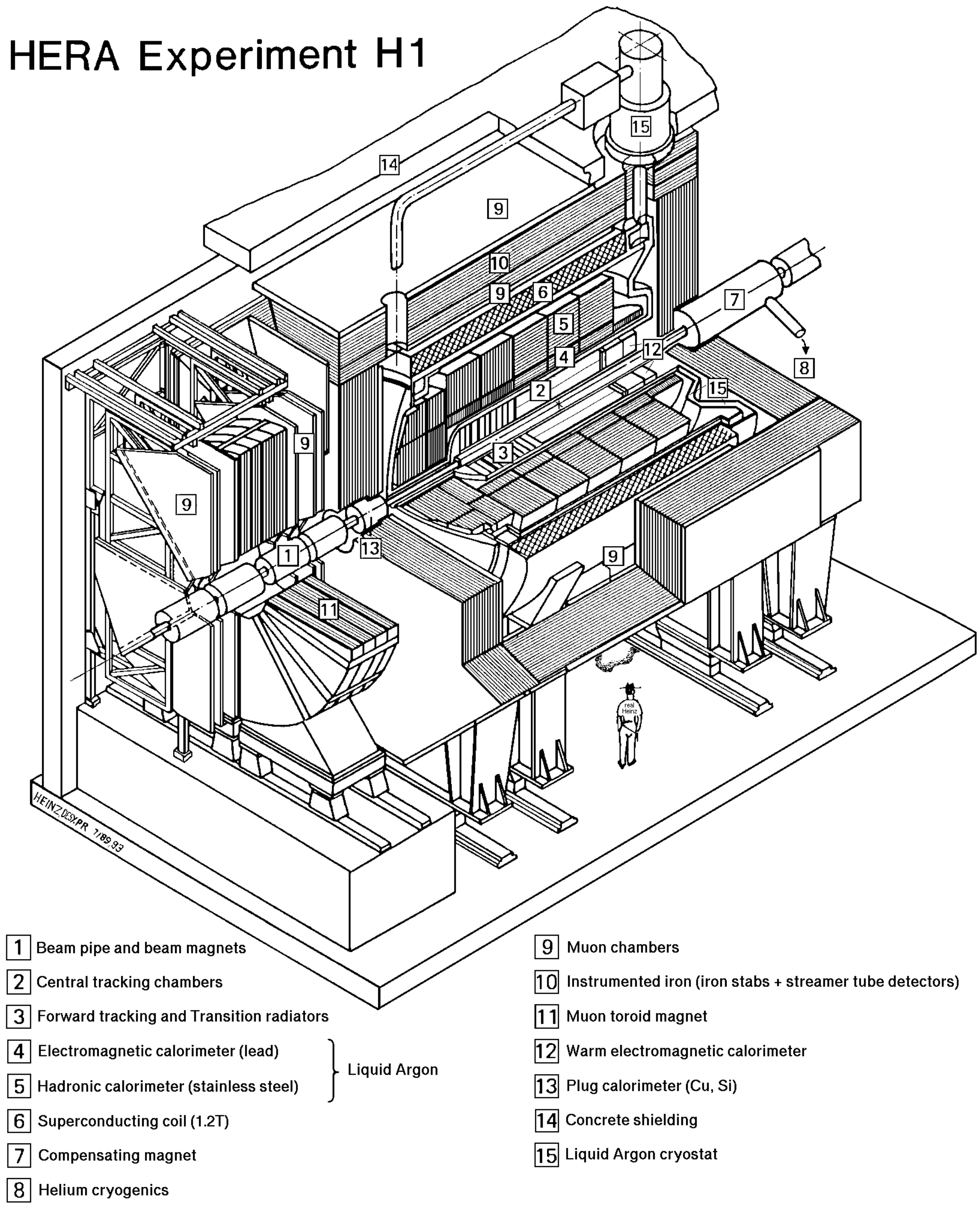

Abbildung 1.2: *Der H1-Detektor*

beschleunigt werden als die Elektronen. Daher bewegt sich der Schwerpunkt des gesamten Endzustandes in die ursprüngliche Protonrichtung, die ebenfalls die z-Richtung des H1-Koordinatensystems definiert. Dieses wird durch die Wahl der x-Richtung, die zur Beschleunigermitte zeigt, eindeutig festgelegt (rechtshändiges Koordinatensystem).

Die wichtigsten Detektorkomponenten sind – vom Strahlrohr nach außen hin betrachtet:

Spurkammern Direkt am Strahlrohr befindet sich sich das zentrale Spurkammersystem (**C**entral **T**racking **D**evice, CTD), das aus einem Siliziumdetektor sowie mehreren Jet- und Proportionalkammern besteht. In positiver z-Richtung wird es von einem Vorwärtskammersystem (**F**orward **T**racking **D**evice, FTD) aus Drift- und Proportionalkammern sowie einem Übergangsstrahlungsmodul unterstützt. Die Spurkammern dienen der Messung geladener Teilchen. In einem Magnetfeld lassen sich aus der Krümmung einer Spur ihr Impuls und Ladungsvorzeichen bestimmen. Aus der Rückverfolgung der Spuren ist der Wechselwirkungspunkt bestimmbar.

Kalorimeter Kalorimeter messen die Energie von Teilchen, indem sie durch Ionisation entstehende Ladungen aufsammeln. Im zentralen und Vorwärtsbereich des Detektors befindet sich das Flüssig-Argon-Kalorimeter (**L**iquid **Ar**gon, LAr), das aus einem elektromagnetischen und einem hadronischen Teil besteht. Im Rückwärtsbereich wurde bis 1994 das BEMC-Kalorimeter (**B**ackward **E**lectro**M**agnetic **C**alorimeter) eingesetzt. Es wurde während des Winters 1994/95 durch das SpaCal (**Spa**ghetti **Cal**orimeter) ersetzt, welches einen elektromagnetischen und einen hadronischen Teil besitzt.

Spule Die Kalorimeter werden von einer supraleitenden Spule umschlossen, die ein homogenes Magnetfeld von 1,2 Tesla erzeugt, welches die Spurkammern zur Bestimmung von Teilchenimpuls und -ladung benötigen.

Instrumentiertes Eisen Das Eisenjoch dient zur Rückführung des magnetischen Flusses. Es ist mit Streamerrohrkammern (10 Lagen) instrumentiert und kann daher ebenfalls Myonspuren identifizieren und als Kalorimeter für Teilchen benutzt werden, die nicht ihre gesamte Energie im LAr deponiert haben.

1.3 Motivation neuer Software

Im Jahr 2000 wurde ein Ausbau des Beschleunigers zur Erhöhung der Luminosität bei HERA begonnen. Unter Luminosität $\mathcal{L}$ versteht man den Proportionalitätsfaktor zwischen gemessener Ereignisrate und dem totalen Wirkungsquerschnitt: $dN/dt = \mathcal{L}\sigma$. Die geplante Spitzenluminosität soll von $1,5 \cdot 10^{31}$ cm^{-2} s^{-1} auf $7 \cdot 10^{31}$ cm^{-2} s^{-1} nahezu verfünffacht werden.

Dabei soll sich die Datenrate, mit der die Ereignisse nach dem Upgrade auf Band geschrieben werden, im Vergleich zu heute aber nicht erhöhen. Das Triggersystem wird dann mit härteren Kriterien betrieben, so daß für eine Vielzahl von Analysen mehr Ereignisse nach der Vorselektion übrig bleiben. Man könnte hier von einer Erhöhung der “effektiven” Datenmenge sprechen. Neben den zahlreichen Verbesserungsarbeiten am Detektor selbst sind also auch bei der Verarbeitung der Daten Änderungen notwendig.

Zu diesem Zweck wurde auf dem Kollaborationstreffen im Dezember 1999 in Zürich ein neues Analyse Framework vorgeschlagen. Dabei handelt es sich um eine zusammenhängende Einheit aus persistentem Datenformat, Event-Display und Umgebung sowohl für interaktive als auch batch-basierte Analyse. Das neue System soll insgesamt eine höhere Effizienz und niedrigere Einstiegsschwelle in den Analysen bei H1 bieten.

Bisher wurden die Daten im `FPACK/BOS` Format gespeichert und die Analysealgorithmen in `FORTRAN` geschrieben (siehe Abschnitt 1.3.1). Dies soll in den Stufen nach dem POT-Level durch ein einziges Framework ersetzt werden: ROOT/C++. Schematisch ergibt sich dabei ein Bild wie in Abb. 1.3.

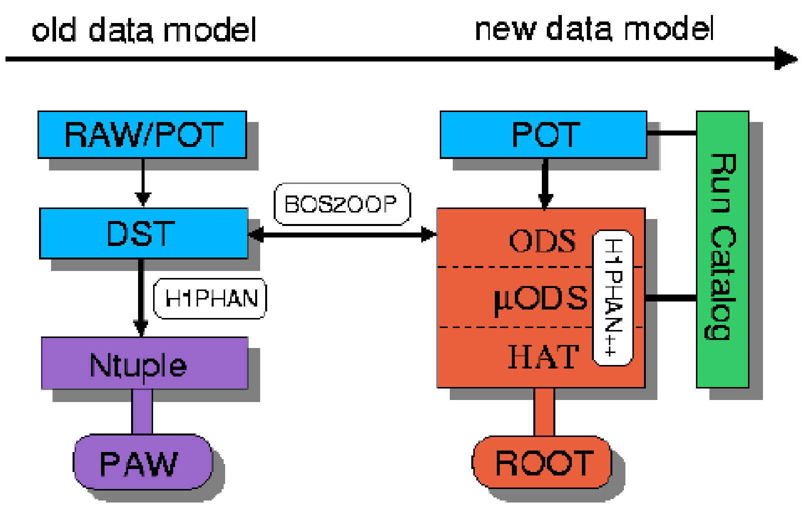

Abbildung 1.3: *Schematische Übersicht des Übergangs vom alten Analysemodell auf das neue Framework, welches auf objektorientierten Technologien basiert. Die ausführliche Erklärung ist Inhalt von Kapitel 2.*

1.3.1 Bisherige Umgebung

Derzeit werden die Daten auf POT (**P**roduction **O**utput **T**ape) und DST (**D**ata **S**ummary **T**ape, entgegen dem Namen inzwischen nicht mehr auf Band sondern auf Festplatte) im `FPACK` Format [Blo91] gespeichert. Dieses erlaubt Daten-Input/Output unabhängig von der darunterliegenden Rechnerarchitektur.

Analysen in H1 waren bisher überwiegend `FORTRAN` basiert, die Werkzeuge auf diesem Gebiet sind `HBOOK` und `PAW`, das die Makrosprache `KUIP` benutzt, welche ihrerseits Routinen aufrufen kann, die in weiteren Sprachen verfaßt sein können. Zur Visualisierung kommt das H1-Event-Display zum Einsatz, das sich dem Programmpaket `LOOK` bedient, welches eine eigene Makrosprache besitzt.

1.3.2 Das H1-OO-Projekt

Die Wahl für die neue Analyseumgebung fiel auf ROOT [BR96], welches als Komplettpaket für interaktive Analyse, Hochleistungsprogramme und das Event-Display dienen soll. Der entscheidende Vorteil ist hier, daß neue Studenten bei H1 für alle o.g. Disziplinen nur noch eine Programmiersprache lernen müssen, um ihre Analyse erfolgreich durchführen zu können: C++.

Durch das Einführen zweier zusätzlicher Ebenen der Datenspeicherung soll der Zugriff auf Teilinformationen des Gesamtdatensatzes effizienter gestaltet werden. Da die zusätzlichen Ebenen physikalische und analyseorientiertere Informationen enthalten sollen, ist es geplant, Standardberechnungen zentral durchzuführen und allen H1-Analysen in den neuen Datenebenen zur Verfügung zu stellen. Der Zugriff auf diese Informationen ist dabei transparent gestaltet, so daß ein Anwender der OO-Software nicht wissen muß, aus welcher der drei Ebenen (ODS, μODS, HAT) die von ihm angeforderten Daten kommen.

Kapitel 2

Die Analyseumgebung bei H1

An dieser Stelle soll ein Überblick geboten werden, der den derzeitigen Ablauf zeigt, welcher für das Erstellen einer Analyse bei H1 notwendig ist. Dies beinhaltet sowohl die Organisation der Daten, die aus dem Detektor anfallen, als auch die technischen Mittel, mit denen diese verarbeitet werden. Es werden die konkreten Schritte gezeigt, die bei der eigentlichen Analyse durchlaufen werden. Danach soll auf Schwachstellen eingegangen werden, an denen das heutige Modell zukünftigen Anforderungen nicht mehr gerecht wird. Ansätze zur Verbesserung dieser Punkte zeigt der letzte Abschnitt dieses Kapitels.

2.1 Datenmodell und technische Aspekte

Der Weg der Daten vom Detektor bis zu den Ergebnissen einer Analyse teilt sich bei H1 zunächst in zwei größere Teile: einen globalen Teil für die gesamte Kollaboration und einen Teil spezifisch für einzelne Arbeitsgruppen. Einen schematischen Überblick gibt Abbildung 2.1. Dieser Abschnitt beschäftigt sich mit den Aspekten, die für das gesamte Experiment gelten und bei den Signalen im Detektor beginnen. Abschnitt 2.2 geht auf die Schritte ein, die für eine konkrete Analyse wichtig sind.

Um beim H1-Detektor die Totzeiten – also Zeiten, in denen aufgrund des Signalauslesens keine Ereignisse festgehalten werden können – so gering wie möglich zu halten, gibt es ein vierlagiges Triggersystem [H1 97]. Die erste Stufe (L1) bezieht seine Eingabe von allen Subdetektoren, um innerhalb von $2,5\ \mu$s eine Entscheidung festzusetzen, ob das Ereignis behalten oder verworfen wird. In dieser Zeit werden die vollen Rohdaten in einem Puffersystem vorgehalten. Wurde das Ereignis von L1 angenommen, kommen die Trigger L2TT[1] und L2NN[2] zum Einsatz. Diese können schon kompliziertere Zusammenhänge und eine größere Signalmenge verarbeiten. Dies geschieht in einer Zeit von etwa 20 μs. Die geplante Triggerstufe L3 beginnt mit der Software-Auslese, in der Informationen aus den einzelnen Detektorkomponenten zusammengefaßt werden. Die vierte Stufe (L4) beginnt schließlich mit einer schnellen Rekonstruktion, die zum Teil aus speziell für L4 optimierten Algorithmen aber auch aus Teilen des Standard-Offline-Rekonstruktions-Programmes H1REC

[1]L2 Topological Trigger
[2]L2 Neural Network Trigger

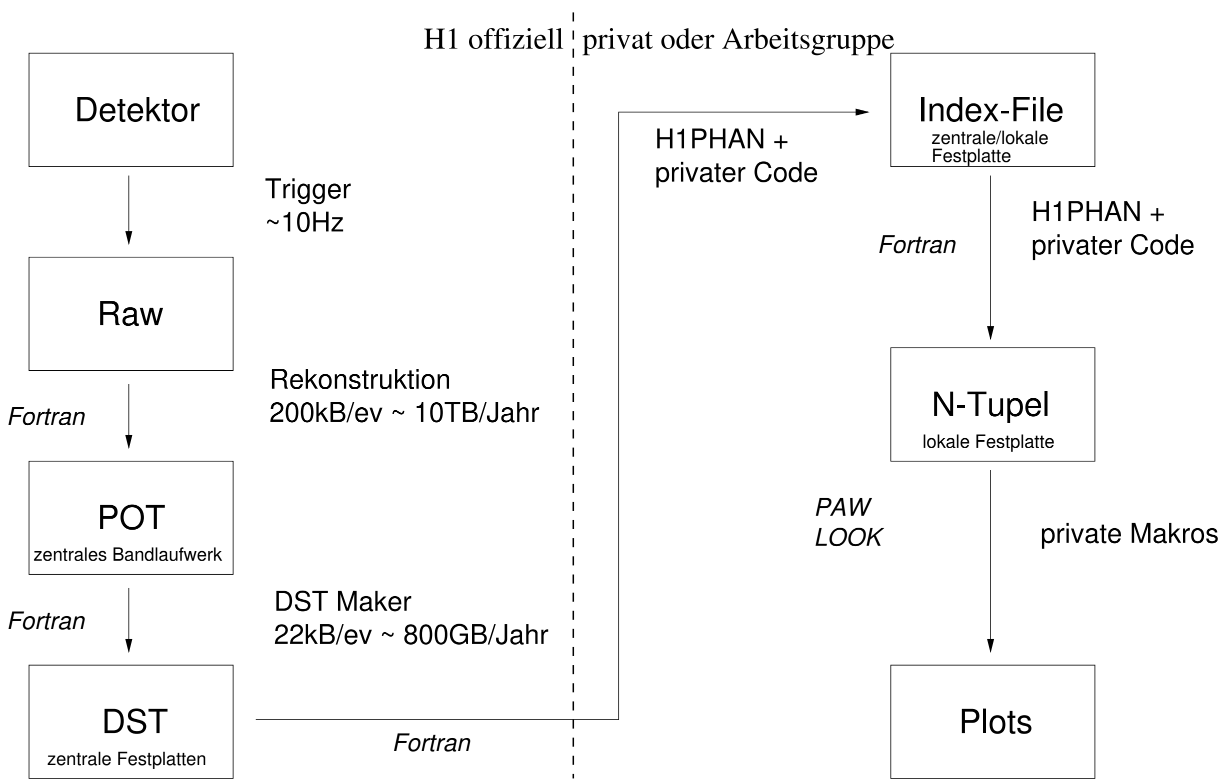

Abbildung 2.1: *Schematische Darstellung des Datenfluß bei H1. Die Erklärung der Begriffe findet sich in den Abschnitten 2.1 und 2.2. Die angegebenen Datenvolumen beziehen sich auf die Periode der Datennahme des Jahres 2000.*

[H1 95] besteht. Die Daten in diesem Stadium werden als “Raw” bezeichnet.

In einem nächsten Schritt folgt die Ereignisrekonstruktion. Dabei werden aus den Detektorsignalen physikalisch sinnvolle Einheiten zusammengestellt. Dies bedeutet zum einen, daß aus den Signalen der Drähte der Spurkammern eine Teilchenspur bzw. aus der Energiedeposition in benachbarten Kalorimeterzellen sogenannte “Cluster” gebildet werden. Ebenso wird z.B. aus gemessenen Ladungen auf die deponierte Energie zurückgerechnet. Die Summe von Raw-Daten und der Ausgabe der Rekonstruktion wird **P**hysics **O**utput **T**ape (POT) genannt und zentral auf Bändern gespeichert. Die Datenmenge beläuft sich an dieser Stelle auf etwa 200 kB pro Ereignis.

Aus den POT Daten werden sogenannte **D**ata **S**ummary **T**apes erstellt, die nicht mehr alle Daten enthalten sondern z.B. die für jede Analyse unabdingbaren Größen wie gefittete Spuren, Zellenergien der Kalorimeter und daraus gebildete Cluster, Trigger- und Detektorstatus-Informationen und einige Ereigniskennzeichen, die zur Klassifikation dienen. Die Datenmenge reduziert sich damit auf etwa 22 kB pro Ereignis. Der Name trifft heutzutage nicht mehr ganz zu, denn die DSTs werden zentral auf Festplatten gespeichert.

2.2 Analyseschritte

Der bisher übliche Weg einer Analyse besteht aus mehreren Schritten. Zu Beginn steht dabei eine Vorselektion, die der Art der zu erwartenden Analyse Rechnung trägt. Dabei werden von den DSTs die Ereignisse ermittelt, welche den losen Selektionskriterien genügen. Dieser Schritt wird i.a. wenige Male durchgeführt und dient mehreren Analysen als Grundlage. Das Ergebnis dieser Prozedur kann eine Kopie der passenden Ereignisse sein oder ein sogenanntes “Index-File”. Ein solches Index-File enthält für die Ereignisse, welche den Kriterien der Vorselektion entsprechen, Sprungadressen zu diesen Ereignissen in den dazu passenden DST Dateien. Erstere Option wird i.a. gewählt, wenn nur ein geringer Teil aller Ereignisse nach den Schnitten übrig bleibt. Wird die zu kopierende Datenmenge zu groß, wird auf die Index-Files zurückgegriffen. Im weiteren Verlauf dieser Arbeit wird angenommen, daß durch die Vorselektion ein Index-File erstellt wurde. Aussagen darüber gelten i.a. auch für eine Kopie der Ereignisse. Index-Files werden üblicherweise nicht mehr zentral für die gesamte Kollaboration gespeichert sondern für die betreffende Arbeitsgruppe.

Beginnt nun ein Physiker mit seiner Analyse, gilt es zunächst, die passende Vorselektion zu finden bzw. gegebenenfalls selbst zu erstellen. Aus diesem bereits reduzierten Datensatz wird als nächstes ein sogenanntes “N-Tupel” erstellt, das alle Informationen enthält, welche von der Analyse benutzt werden. In diesem Schritt werden weitere Schnitte angewandt, die dem speziellen Thema der Analyse angepaßt sind. Dem N-Tupel werden zusätzlich berechnete Größen hinzugefügt wie z.B. identifizierte Teilchen oder Jets. Auch dies ist ein iterativer Vorgang, der mehrfach durchlaufen wird. An seinem Ende steht ein Datensatz, der etwa 1 kB pro Ereignis enthält. N-Tupel werden typischerweise auf Festplatten gespeichert, die lokal zu einem Rechner gehören. Bis zu diesem Zeitpunkt wird i.a. mit `FORTRAN` gearbeitet.

In einem letzten Schritt wird ein N-Tupel mit dem Programm **P**hysics **A**nalysis **W**orkstation (`PAW`) bearbeitet. Es bedient sich der Makrosprache `KUIP`, die ihrerseits erlaubt,

Nachteile der bestehenden Analyseumgebung
• Langsamer Zugriff auf die große Datenmenge auf DST-Ebene • Jede Analyse durchläuft Standardalgorithmen selbst, wobei keine einheitliche Wahl der Parameter gewährleistet ist • Vom N-Tupel gibt es keine direkte Verbindung mehr zu den weiteren Informationen eines Ereignisses auf DST • In einer Analyse kommen drei Sprachen zum Einsatz, die nicht beliebig untereinander austauschbar sind: `FORTRAN`, `KUIP` und die Makrosprache von `LOOK` • Die Analyseumgebung ist sehr heterogen, da mehrere unterschiedliche Programmpakete für einzelne Teile einer Analyse zum Einsatz kommen

Tabelle 2.1: *Zusammenfassung der Nachteile der bestehenden Analyseumgebung.*

`COMIS`-Routinen aufzurufen, wobei `COMIS` stark an `FORTRAN` angelehnt ist. Die so geschriebenen Programme werden als **Ku**ip **mac**ros (Kumac) bezeichnet. Nach den endgültigen Selektionen und Schnitten erzeugen sie die finalen Ergebnisse einer Analyse. Dieser Schritt wird i.a. am häufigsten durchlaufen. Er beinhaltet auch das Betrachten einzelner Ereignisse mit dem Event-Display, falls diese mit ihren Daten stark von allen anderen abweichen. Dies geschieht nicht in derselben Umgebung, es sind Daten vom DST-Niveau nötig. Das H1-Event-Display benutzt das Programmpaket `LOOK`.

2.3 Schwachstellen

Das bestehende System hat im wesentlichen vier Schwachpunkte, die in Tabelle 2.1 kurz zusammengefaßt sind. Die nähere Erläuterung erfolgt nacheinander für die Schritte von DST zu den endgültigen Ergebnissen.

Das Erstellen eines Index-Files ist nötig, weil die offizielle Kennzeichnung der Ereignisse mittels weniger, binärer Variablen nicht ausreichend ist. Jede Analyse müßte sonst über den vollen Satz an DST Dateien laufen und die Größen berechnen, die eine Entscheidung zulassen, ob ein Ereignis für die betreffende Analyse brauchbar ist. Schon allein das Erstellen eines Index-Files konsumiert eine erhebliche Zeitmenge. Dies für jede einzelne Analyse durchzuführen ist ein enormer Aufwand.

Ist die Vorselektion vorhanden, muß beim Erstellen eines N-Tupel nicht mehr jedes Ereignis von DST gelesen werden. Es ist möglich, vor dem Start eines Programmes die Teile eines Ereignisses anzugeben, welche tatsächlich geladen werden sollen. Dies ist aber ein statisches Konzept, welches für jedes Ereignis die maximal benötigte Datenmenge überträgt, auch wenn es anschließend aufgrund simpler Schnitte direkt verworfen wird. Hier bremst die Datenübertragung die eigentlichen Berechnungen aus. Ebenfalls wichtig an dieser Stelle ist, daß hier eine große Menge Quelltext involviert ist, die nicht zentral verwaltet wird. Dies bedeutet einerseits, daß der Code üblicherweise verloren geht sobald der Betreffende die Kollaboration verläßt. Andererseits ist es sehr wahrscheinlich, daß viele Analysen dieselben Algorithmen verwenden, dabei aber Unterschiede in der Wahl der Parameter vorkommen. Dies bedeutet im Prinzip das Verschwenden von Rechenleistung und es kommt ein nachträglicher Aufwand hinzu, die Ergebnisse zu vergleichen. Zudem stellen private Implementierungen von häufig benutzten Algorithmen ein höheres Risiko bezüglich der Fehleranfälligkeit dar, weil gegenüber zentral verwalteter Software weniger Benutzer den Code einsetzen und somit testen.

Die im letzten Schritt kreierten N-Tupel sind sehr statische Gebilde. Sollte sich herausstellen, daß eine Größe nicht enthalten ist, gibt es keine Möglichkeit, diese nachträglich von DST nachzuladen bzw. aus DST-Informationen zu berechnen. Hier ist die komplette Neuerstellung des N-Tupels unumgänglich.

Für die drei grundsätzlichen Schritte einer physikalischen Analyse – Erstellen von N-Tupeln mittels `FORTRAN`-Programmen, Analyse der N-Tupel mittels `PAW`, Betrachten einzelner Ereignisse mittels `LOOK` – gilt, daß getesteter Code aufgrund verschiedener Sprachen und Programmpaketen nicht ohne weiteres als einfache Kopie von einem Teil in einen anderen transferiert werden kann. Zudem handelt es sich hier um drei komplett voneinander getrennte Arbeitsschritte, z.B. können in `PAW` selektierte Ereignisse nicht direkt betrachtet sondern müssen in `LOOK` erneut angewählt werden.

2.4 Verbesserungsansätze

Eine neue Analyseumgebung soll die in Abschnitt 2.3 dargestellten Mängel weitestgehend abstellen. Es ist sinnvoll, bei einem solchen Vorhaben die Situation an anderen bestehenden und geplanten Experimenten zu betrachten. Zudem ist es wichtig, sich an den allgemeinen Gegebenheiten der Software-Entwicklung zu orientieren, die nicht allein spezifisch für ein Experiment der Hochenergiephysik sind. Es geht um Werkzeuge, Methoden und nicht zuletzt um die eingesetzte Programmiersprache. So soll von den Erfahrungen aus anderen Bereichen profitiert und neuen Studenten eine moderne und attraktive Umgebung geboten werden.

Allgemein gilt heutzutage, daß **o**bjekt-**o**rientierte **P**rogrammierung (OOP) das adäquate Mittel zur Bewältigung sehr großer Softwareprojekte ist. Beispiele für objekt-orientierte Programmiersprachen sind `Java` und `C++`. Letztere wird in ROOT [BR96] eingesetzt, eine am CERN entwickelte Umgebung für Analysen in der Hochenergiephysik. Dieses Framework wird bereits für die meisten neuen Experimente wie Alice oder BaBar eingesetzt, andere existierende Experimente sind bereits migriert oder werden noch auf ROOT als Basis der Analysesoftware umstellen (z.B. CDF oder Minos). Auf dem Kollaborationstreffen im Dezember 1999 wurde beschlossen, auch für H1 eine auf ROOT basierende Analyseumgebung zu erproben.

Bei objekt-orientierter Programmierung werden Daten und die darauf definierten Funktionen zu Einheiten zusammengefaßt, welche Klassen genannt werden. ROOT bietet bereits eine Menge vordefinierter Klassen, die auf die Ansprüche der Hochenergiephysik abgestimmt sind. Darunter befinden sich Klassen für die Datenein- und -ausgabe, Histogramme und Funktionen, graphische Bedienelemente, geometrische Berechnungen und vieles mehr

Teilbereiche von ROOT
• 2-D und 3-D Grafikausgabe
• Auf hohen Durchsatz optimierte Ein- und Ausgabe von Daten
• Automatische Generierung von Dokumentation im HTML Format
• Elemente für das Erstellen eines **G**raphical **U**ser **I**nterface (GUI)
• Histogramme und Funktionen
• **R**un-**T**ime-**T**ype-**I**nformation (RTTI)
• Vektor- und Matrizenrechnung
• Verlustfreie Komprimierung von Dateien mittels `gzip`-Algorithmus
• Vierer-Vektoren und Lorentztransformationen
• Zufallszahlengenerator
• Zugriff auf eine Datenbank von Teilcheneigenschaften nach der Particle Data Group

Tabelle 2.2: *Übersicht der Bestandteile von* ROOT. *Die vorhandenen Klassen erlauben, die Kombination* ROOT/`C++` *sowohl für interaktive Analysetätigkeiten einzusetzen als auch für Hochleistungsprogramme und nicht zuletzt das Event-Display.*

(siehe auch Tabelle 2.2). Mit diesen Mitteln ist es möglich, interaktive Analysetätigkeiten, Hochleistungsprogramme und Event-Display in nur einer einzigen Umgebung durchzuführen, d.h. Quelltext kann direkt von einem dieser Einsatzfelder in eines der anderen übernommen werden. Makros für interaktive Analysen können leicht so erweitert werden, daß sie ebenfalls kompiliert und auf größere Datensätze angewandt werden können. Ebenso ist es möglich, während einer interaktiven Analysetätigkeit selektierte Ereignisse direkt zu betrachten, ohne ein weiteres Programm benutzen zu müssen.

Allein der Einsatz von ROOT und `C++` löst jedoch noch keine Probleme. Das gesamte Datenmodell muß optimiert werden. Von anderen Experimenten inspiriert, sollen in der zukünftigen Umgebung zusätzliche Ebenen der Datenspeicherung eingeführt werden. Das neue Schema besteht aus **O**bject **D**ata **S**tore (ODS), **Micro O**bject **D**ata **S**tore (μODS) und **H**1 **A**nalysis **T**ag (HAT). Diese verschiedenen Ebenen werden durch den sogenannten "Runkatalog" zusammengehalten. Unter "Run" versteht man die Zusammenfassung von Ereignissen, die unter denselben Voraussetzungen und Einstellungen aufgenommen wurden. Zusätzlich wird der Runkatalog dazu genutzt, daß ein Benutzer des neuen Schemas keine Details mehr über den Speicherort der Daten sowie die benutzten Dateinamen kennen muß.

Das ODS ist als Ersatz der DSTs konzipiert, d.h. es enthält im wesentlichen dieselben Informationen in einem anderen Datenformat. Auch wird das Ereignismodell nicht geändert, so daß zu jeder Zeit Rückwärtskompatibilität gewährleistet ist. Dies bedeutet, daß auch in Zukunft Benutzer ihre alte Software weiterverwenden können, wenn auch mit Geschwindigkeitseinbußen wegen der Rückkonvertierung. Das μODS enthält Informationen auf Teilchenniveau mit Referenzen auf die Informationen des ODS, aus denen sie konstruiert wurden. Hier werden Standardberechnungen durchgeführt, um der gesamten Kollaboration oft benötigte Informationen bereitzustellen. Neben der Zeitersparnis, daß

Verbesserungen durch das neue Schema
• Stark erweiterte, offizielle Ereignisklassifikation
• Standard-Algorithmen mit einheitlichen Parametern bereits durchgeführt
• Weniger Zugriff auf die vollen Daten eines Ereignisses nötig, da zusätzlich μODS und HAT zur Verfügung stehen
• Möglichkeit, nur Teile eines Events einzulesen
• Ein einziges Framework für interaktive und Hochleistungs-Analysen sowie Event-Display
• Nur noch eine einzige, moderne Programmiersprache

Tabelle 2.3: *Zusammenfassung der zu erwartenden Verbesserungen durch den Einsatz des neuen Schemas.*

nicht für jede Analyse immer wieder dieselben Algorithmen laufen müssen, ist ein wesentlicher Vorteil, daß unterschiedliche Analysen durch den Zugriff auf die zentralen Ergebnisse automatisch dieselben Parameter für Standardaufgaben benutzen. Der HAT beinhaltet eine Vielzahl von kinematischen Ereignisvariablen und Statusinformationen des Detektors. Über die HAT-Variablen ist ein sehr gezielter Zugriff auf Ereignisse möglich, welche komplexeren Bedingungen genügen, als mit den Klassifikationsmerkmalen der DSTs ausdrückbar ist. Die Verbindung zwischen Ereignissen auf POT, ODS, μODS und HAT wird durch eine relationale Datenbank bewerkstelligt. Den Dateinamen und Verzeichnissen werden die darin enthaltenen Run- und Eventnummern sowie die Gesamtzahl an Ereignissen zugeordnet. Dieses Konzept wird "Runkatalog" genannt und ermöglicht, Datensätze rein aufgrund von Jahresangaben oder Run/Event-Bereichen auszuwählen. Eine Übersicht der wesentlichen Merkmale des neuen Datenmodells zeigt Tabelle 2.4. Die durch das neue Schema zu erwartenden Vorteile sind in Tabelle 2.3 zusammengefaßt.

Das neue Datenmodell soll nun in folgender Weise eingesetzt werden:

- Selektieren des Datensatzes über Run/Event oder Jahresangaben
- Schnelle Selektion der zu verarbeitenden Ereignisse aufgrund von HAT Variablen
- Hauptsächlich Informationen auf dem μODS nutzen
- In wenigen Fällen auf die volle Information der ODS Dateien zugreifen
- Im interaktiven Fall direkt in der Analyse selektierte Ereignisse mittels Event-Display betrachten
- Neue oder verbesserte Algorithmen sollen zurück in das neue Framework fließen und somit der gesamten Kollaboration zur Verfügung gestellt werden

POT – Production Output Tape (`FPACK/BOS`)
• Enthält Raw- und rekonstruierte Daten wie bisher • Keine Änderung an der Software, die POT produziert
ODS – Object Data Store (ROOT)
• Enthält alle Standardobjekte für eine Analyse, z.B. Spuren, Cluster etc. • Korrespondiert zu den DSTs; das existente Ereignismodell wird nicht geändert • `BOS`-Bänke werden "eins zu eins" in Objekte umgesetzt • Rückwärtskompatibilität wird gewährleistet, so daß auch bestehende Analysesoftware genutzt werden kann (mit Geschwindigkeitseinbußen)
μODS – Micro Object Data Store (ROOT)
• Enthält Informationen auf Teilchenniveau, wie z.B. Vierervektoren, identifizierte Teilchen, Jets etc. • Referenzen auf die Objekte des ODS, aus denen die Teilchen konstruiert wurden • Die Expertise der physikalischen Arbeitsgruppen fließt an dieser Stelle ein • Standardberechnungen mit einheitlichen Parametern werden zentral für die gesamte Kollaboration durchgeführt
HAT – H1 Analysis Tag (ROOT)
• Wird zur schnellen Ereignisselektion benutzt • Enthält etwa 100 kinematische Ereignisvariablen, z.B. Q^2, x, y etc. sowie Statusinformationen des Detektors, z.B. Trigger, Zustand der Subdetektoren etc. • Zusätzlich wird eine Zusammenfassung des μODS gespeichert, z.B. Anzahl gefundener Elektronen, Myonen etc.
Runkatalog (`MySQL/Oracle`)
• Bindeglied zwischen den verschiedenen Ebenen der Datenspeicherung • Datensatz kann über Run/Event Bereiche oder Jahre ausgewählt werden, ohne Datei- oder Pfadnamen kennen zu müssen • Kopie an externen Instituten möglich durch untergeordneten Datenbankserver oder Export in eine ROOT-Datei

Tabelle 2.4: *Das neue Datenmodell. Für alle Ebenen, die auf* ROOT *basieren, ist es möglich, zu einem gegebenen Ereignis nur die Informationen zu laden, die wirklich benötigt werden. Zudem ist der Zugriff vollkommen transparent, d.h. der Benutzer muß nicht wissen, aus welcher Lage die von ihm angeforderten Daten stammen.*

Kapitel 3

Entwicklung der Software

Dieses Kapitel beschreibt in einem ersten Abschnitt die grundlegenden Überlegungen bei einem Softwareprojekt. Dazu gehört, welche Funktionalität für die zu bewältigenden Aufgaben benötigt wird, das Zusammenspiel der einzelnen Teilaufgaben und welche Konventionen beim Programmieren beachtet werden sollen. Der nächste Abschnitt beschäftigt sich mit der Organisation der Software-Entwicklung. Versionen und Entwicklungsverlauf des Quelltextes sind zu verwalten und Hilfsmittel einzusetzen, die über den direkten Umgang mit einem Texteditor und dem Compiler hinausgehen. Der letzte Abschnitt dieses Kapitels enthält die zentralen Konzepte, die das OO-Projekt von der bisherigen Analyseumgebung unterscheiden.

3.1 Softwaredesign

3.1.1 Interface und Implementierung

Ein wesentlicher Grundzug bei größeren Softwareprojekten ist die strikte Trennung von Interface und Implementierung. Zu Beginn des Projektes muß der Aufgabenbereich definiert werden, den die zu entwickelnde Software abdecken soll. Bei objektorientierter Programmierung sind Klassen zu modellieren, welche Teilaufgaben übernehmen und in der Summe die benötigte Funktionalität bieten. Während dieser Planungsphase richtet sich das Hauptaugenmerk auf die Schnittstellen der Klassen und Algorithmen. Sie definieren, wie die Objekte im späteren Einsatz miteinander interagieren, wie flexibel und erweiterbar das System sein wird. Es ist wesentlich zu verstehen, daß die Details, wie der Code endgültig geschrieben wird, erst zu einem späteren Zeitpunkt wichtig werden, die Methoden, die eine Klasse zur Verfügung stellt, aber sehr früh schon festgelegt und optimiert werden sollten. Idealerweise ist es dann möglich, die Implementierung zu einem beliebigen Zeitpunkt zu ändern und zu verbessern, ohne daß die Art, wie die Klasse benutzt wird, davon betroffen wird. Im OO-Projekt kommt die CRC-Methode für das Design der Klassen zum Einsatz, die in Abschnitt 3.1.2 vorgestellt wird. Eine kurze Einführung in die Thematik der Programmentwicklung zeigt z.B. [EH01].

3.1.2 Klassendesign nach der CRC-Methode

Ziel der CRC-Methode [BBSS97] (CRC steht für "**C**lasses, **R**esponsibilities, **C**ollaborators", d.h. Klassen, ihre Aufgabenbereiche und Partner) ist es, einen Überblick zu gewinnen, welche Funktionalität ein Projekt bieten soll. Davon ausgehend werden Kandidaten für Klassen und Algorithmen gesammelt. Diese werden anhand von Gemeinsamkeiten gruppiert, ihr Zusammenarbeiten in einigen Fällen durchgespielt. Am Ende steht dann ein Entwurf für die zu implementierende Paket- bzw. Klassenhierarchie sowie deren Interface.

Bei der CRC-Methode beginnt die Entwicklung von Software mit einer sogenannten "Brainstorming Session". Dort sitzt das Entwicklerteam beisammen und sammelt einige Begriffe, die im Zusammenhang mit dem Ziel des Projektes einen Sinn machen. Dabei werden die einzelnen Vorschläge nicht direkt diskutiert, sondern nur in beliebiger Form aufgezeichnet. Es spricht nichts dagegen, mehrere dieser Sitzungen durchzuführen, es zählt einzig die Tatsache, ob noch weitere Vorschläge gemacht werden oder nicht. Ein konkretes Beispiel aus dem OO-Projekt für solch eine Liste von Begriffen zeigt Tabelle 3.1.

Begriffe aus den CRC-Sitzungen	
Track	Particle Collection
Cluster	Loop
FourVector	Lorentz Frame
Vertex	Track Selector
Hit	Track Selection
Cell	Calibrator
Electron Finder	Calibration
Detector	Link
Particle (PDG Entry)	Energy Deposit
Finder	Particle Candidate (observed)
Trajectory	Jet Finder
Event	Boost
Jet	Track Collection
Scattered Electron	Combined Object

Tabelle 3.1: *Ungeordnete Liste der Begriffe, welche im Rahmen der CRC Brainstorming Session für das H1-OO-Projekt genannt wurden. Man erkennt hier schon eine grobe Möglichkeit der Einteilung in Informationen über den Detektor (Detector, Cell), gemessene Informationen (z.B. Track, FourVector) und Algorithmen bzw. Container (z.B. Jet Finder, Particle Collection).*

Die nächste Phase umfaßt das Sichten der gesammelten Stichwörter. Zum Beispiel können ähnliche Begriffe zu einem zusammengefaßt, andere komplett gestrichen oder durch aussagekräftigere Varianten ersetzt werden. Hier wird am Ende eine Liste von Kandidaten erstellt, die als Klassen oder Algorithmen in das Projekt eingehen (könnten). Dabei werden

Felder einer CRC-Karte	
Klassenname	Der Name des Klassenkandidaten
Superklasse	Eine mögliche Klasse, von der der Kandidat abgeleitet werden kann
Subklasse	Mögliche Klassen, die von dem Kandidaten abgeleitet werden können
Aufgaben	Die Funktionalität, die der Kandidat zur Verfügung stellen soll
Partner	Andere Klassen, mit denen der Kandidat zusammenarbeitet
Sonstiges	Für weitere Anmerkungen

Tabelle 3.2: *Die Felder und deren Zweck bei einer CRC-Karte. Eine Blankovorlage zeigt Abbildung Anhang B.1, ein Beispiel einer ausgefüllten Karte findet sich in Abbildung Anhang B.2*

auch schon Begriffe zu sinnvollen Gruppen zusammengefaßt. Hat man sich dann auf eine Liste von Klassenkandidaten geeinigt, werden die Begriffsgruppen an die teilnehmenden Personen verteilt, die dann für jeden Kandidaten eine CRC-Karte erstellen müssen.

Eine CRC-Karte besitzt im wesentlichen die Felder aus Tabelle 3.2. Natürlich muß nicht jedes Feld ausgefüllt sein, außerdem ist das Aussehen der Karten jeder Gruppe von Entwicklern freigestellt. Das H1-OO-Projektteam hat sich dabei auf eine Vorlage wie in Abbildung B.1 geeinigt.

Sind die CRC-Karten erstellt, werden sie von ihren Autoren vorgestellt und in der Gruppe diskutiert. An dieser Stelle des Entwicklungsprozesses soll herausgefunden werden, ob der Satz an Klassenkandidaten ausreicht und ob deren Zusammenspiel sinnvoll ist. Zu diesem Zweck werden bestimmte Situationen mit den Karten durchgespielt.

Ein einfaches Beispiel: es soll ein Massenplot erstellt werden.

- Von wo kommt die Masse? → Teilchen
- Von wo kommt das Teilchen? → Teilchenliste
- Wer erstellt diese Teilchenliste? → Teilchen-Selektor
- Welche Eingaben benötigt der Selektor? → Liste aller Teilchen und eine Selektion
- Woher kommt die Liste aller Teilchen? → Event
- Woher kommt die Selektion? → Vom Benutzer definierte Kriterien

An dieser Stelle ist es noch vollkommen uninteressant, wie z.B. die Masse in einem Teilchen steckt. Es könnte direkt ein Datenelement mit der Masse haben, aber genausogut könnte es einen Vierervektor besitzen, aus dem man die Masse noch berechnen muß. Dies sind jedoch nur Details der Implementierung, die zu diesem Zeitpunkt noch nicht wichtig sind.

Am Ende der Kette stehen dann CRC-Karten, die einerseits das Interface der Klassen beschreiben, die es zu implementieren gilt, andererseits in ihrer Gesamtheit ein Gerüst für eine Klassenhierarchie darstellen. Außerdem geben die CRC-Karten vielfach auch schon Hinweise auf die künftigen Dateninhalte der geplanten Klassen. Ein Beispiel für eine ausgefüllte Karte zeigt Abbildung Anhang B.2, es handelt sich dabei um einen Vorschlag für eine Klasse zur Verwaltung von verschiedenen Bezugssystemen.

3.2 Konzepte im OO-Projekt

3.2.1 Die drei Ebenen der Datenspeicherung

In der OO-Analyseumgebung werden die bisherigen DST Dateien durch ODS ersetzt, wobei im wesentlichen[1] eine 1 : 1 Korrespondenz besteht. Aufbauend auf ODS werden zusätzlich Informationen auf Teilchen-Niveau im μODS und einfache Ereignisvariablen im HAT gespeichert (siehe Abbildung 1.3).

Sinn des μODS ist es, bestimmte Standardberechnungen nicht mehr von jedem Physiker in seiner Analyse neu durchführen zu lassen. Stattdessen werden Ergebnisse bereitgestellt, welche auf Algorithmen basieren, die von den Experten der Physikarbeitsgruppen entwickelt wurden. Hierzu gehören z.B. der hadronische Endzustand, identifizierte Teilchen (insbesondere das gestreute Elektron) oder "gute" Spuren, die bestimmte Kriterien erfüllen. Als weitere Konsequenz werden auf diese Weise von verschiedenen Analysen automatisch dieselben Parameter benutzt. Konkreter gesagt ist vorgesehen, pro Ereignis eine globale Liste aller Teilchen zu speichern und daneben mehrere Listen für identifizierte Teilchen. Die identifizierten Teilchen besitzen wenige, für sie typische Eigenschaften und einen Verweis auf das ihnen entsprechende Teilchen in der globalen Liste. Auf diese Weise soll vermieden werden, die kinematischen Informationen mehrfach abzuspeichern.

Der HAT soll für schnelle Ereignisselektionen benutzt werden. Er enthält Ereignisvariablen wie z.B. Anzahl der verschiedenen Teilchentypen auf μODS, kinematische Variablen, Triggerinformation und Status des Hochspannungssystems etc. Darauf aufbauend lassen sich sogenannte "Eventlisten" erstellen, die vom Konzept her in etwa den Index-Files entsprechen. Verglichen mit diesen benötigt eine Eventlist deutlich weniger Zeit zur Erstellung, so daß eine Selektion auf dem HAT immer wieder "on-the-fly" gemacht werden kann.

3.2.2 Der Runkatalog

Der Runkatalog (siehe auch [Pro] Abschnitt 5) ist eine Datenbank, die zu offiziellen Dateien (d.h. deren Namen und Pfad) deren Run/Event Bereich, das Jahr und die Gesamtzahl enthaltener Ereignisse speichert. Auf diese Weise wird der Zusammenhang zwischen den drei Ebenen der Datenspeicherung hergestellt. Hinzu kommt, daß mit diesen Informationen gezielte Runranges einfach zu selektieren sind, z.B.

- Komplette Runs
- Run und Event für Beginn und Ende des Bereichs

[1] Es werden einige Track- und Cluster-Bänke durch vollständig neu modellierte C++ Klassen ersetzt

- Ganze Jahre
- Kombinationen dieser Möglichkeiten

Die Auswertung der vom Runkatalog zur Verfügung gestellten Daten obliegt dem `H1Tree`, welcher in Abschnitt 3.2.3 näher beschrieben wird.

Für den Runkatalog wird eine relationale Datenbank nach dem SQL-Standard (SQL = **S**tandard **Q**uery **L**anguage) benutzt. Zur Zeit ist dies `MySQL`, eine frei verfügbare Datenbank, die unter der GPL (**G**NU **P**ublic **L**icense) steht. Die benötigten Leistungsmerkmale sind verfügbar, das Plus dieser Wahl liegt darin, einen `MySQL`-Server ohne kostenpflichtige Lizenzen an externen Instituten aufsetzen zu können. Am DESY selbst gibt es einen "Master" und einen "Slave" Server auf verschiedenen Rechnern. Das Replizieren auf einem PC an externen Instituten ist ebenfalls getestet und durchführbar.

Die Alternative zu `MySQL` ist `Oracle`, für das das DESY bereits Lizenzen besitzt. Das Interface in ROOT unterstützt zur Zeit genau diese beiden Datenbanken, d.h. ein Übergang wäre transparent. Diese Umstellung auf `Oracle` ist dann zu erwägen, wenn sich herausstellt, daß an externen Instituten kein Bedarf für einen Slave-Server besteht, oder wenn mit `MySQL` nicht noch ein weiteres System gepflegt und administriert werden soll.

3.2.3 Das zentrale Objekt: der `H1Tree`

Bei dem `H1Tree` (siehe auch [Pro] Abschnitt 2) handelt es sich um eine Generalisierung des `TTree` von ROOT. Dieser wiederum ist eine Verallgemeinerung des N-Tupel-Prinzips und dient der Strukturierung und effizienten Speicherung von großen Datenmengen (siehe auch [BRP+00]). Die wesentlichen Unterschiede zu einem N-Tupel sind, daß nicht nur Zahlen und Arrays gespeichert werden können sondern auch Objekte, ein `TTree` kann auf mehrere Dateien verteilt und Teile – sogenannte Branche – unabhängig voneinander geladen werden. Genau genommen stellt ein `H1Tree` drei parallele Trees dar, die über ein und dasselbe Interface angesprochen werden.

Dabei ist es besonders hilfreich für den Benutzer, daß sich der `H1Tree` um die Zugriffe auf die Dateien kümmert. Ein Physiker muß selbst nur noch einen Run/Event Bereich angeben, wobei es für ihn nicht wichtig ist, in welchen Dateien die Informationen gespeichert sind und auf welchem Rechner und in welchen Verzeichnissen die Daten physikalisch liegen. Weiterhin werden nur dann Informationen in den Arbeitsspeicher gelesen, wenn sie auch wirklich angefordert werden, d.h. es wird nicht generell das gesamte Ereignis eingelesen. Folgendes Beispiel illustriert, wie dies ausgenutzt werden kann:

> Die Listen identifizierter Teilchen auf dem μODS enthalten nicht die kinematischen Informationen. Ein sogenannter "Smart Pointer" (siehe auch Abschnitt 3.2.4) verweist auf einen Eintrag in der Liste aller Teilchenkandidaten, von wo aus auf die kinematischen Eigenschaften zugegriffen werden kann.
>
> Will man nun weitere Schnitte anwenden, die ohne kinematische Information auskommen, so bietet es sich an, die (u.U. recht große) Liste aller Teilchenkandidaten erst dann zu laden, nachdem die zusätzliche Selektion stattgefunden

hat. Dies geschieht automatisch wenn die Smart Pointer dereferenziert werden, d.h. auf das Objekt zugegriffen werden soll, auf das sie verweisen. Die zugehörige Liste muß also nicht von Hand angefordert werden.

3.2.4 Persistente Zeiger: der `H1PointerManager`

Ein großes Problem stellen Zeiger auf Objekte dar, wenn sie als reine Verweise gespeichert werden sollen. ROOT erlaubt, persistente Zeiger in Klassen zu benutzen, doch werden beim Speichern eines solchen Objektes immer die kompletten Objekte, die sich hinter den Zeigern verbergen, mit herausgeschrieben. In vielen Fällen ist es aber gewünscht, nur die Referenzen beizubehalten und nicht die persistenten Objekte zu duplizieren. Ein solcher Mechanismus ist in ROOT nicht vorhanden und muß selbst implementiert werden.

Der gewählte Ansatz im OO-Projekt (siehe auch [Pro] Abschnitt 6.4) stützt sich auf die Annahme, daß ein Großteil von Objekten in Listen organisiert ist. Jedes dieser Objekte läßt sich eindeutig durch seine Position in einer Liste und diese Liste selbst identifizieren. Letztere kann durch ihre Position in der globalen Liste aller Listen in einem Ereignis spezifiziert werden. Somit benötigt man zwei Indizes für die eindeutige Kennzeichnung eines Objektes.

Im `H1Pointer` werden nun zwei vorzeichenlose Integerzahlen benutzt, um diese Relation festzuhalten. Zusätzlich besitzt die Klasse einen echten Zeiger, der nicht mit abgespeichert wird. Dies erspart das wiederholte Heraussuchen aus Listen bei weiteren Dereferenzierungen. Daneben gibt es noch weitere Klassen, z.B. den `H1ArrayPointer`, der auf eine komplette Liste verweist und somit nur eine Integerzahl enthält.

3.3 Technische Aspekte

3.3.1 Programmierregeln

Bei einem größeren Softwareprojekt ist es unumgänglich, Regeln aufzustellen, die jeder Entwickler bei seiner Arbeit zu beachten hat. Dabei reicht die Art der Regeln von Konventionen zur Namensgebung über Sprachmerkmale der zum Einsatz kommenden Programmiersprachen, bis hin zum Stil, der für den Quelltext benutzt werden sollte.

Die "C++ Programming Guidelines" [H1 00a] sind an die ROOT Konventionen [BR] angelehnt, die ihrerseits den "Taligent's Guide to Designing Programs" [Tal95] (online unter [Tal]) benutzen und erweitern.

Das konsequente Beachten dieser Regeln hat viele Vorteile. Zum einen wird der Code wesentlich übersichtlicher, wenn man für verschiedene Typen von Variablen unterschiedliche Präfixe benutzt (z.B. beginnen alle Konstanten mit "`k`", alle globalen Variablen mit "`g`" etc.). So ist schon beim Anblick des ersten Buchstabens ersichtlich, wie die Variable zu benutzen ist. Zum anderen ist auch eine einheitliche Groß-/Kleinschreibung sehr wertvoll. Beispielsweise beginnen Klassen und Methoden/Funktionen mit einem Großbuchstaben; besteht der Name aus mehreren Teilwörtern, so ist auch der Beginn eines jeden

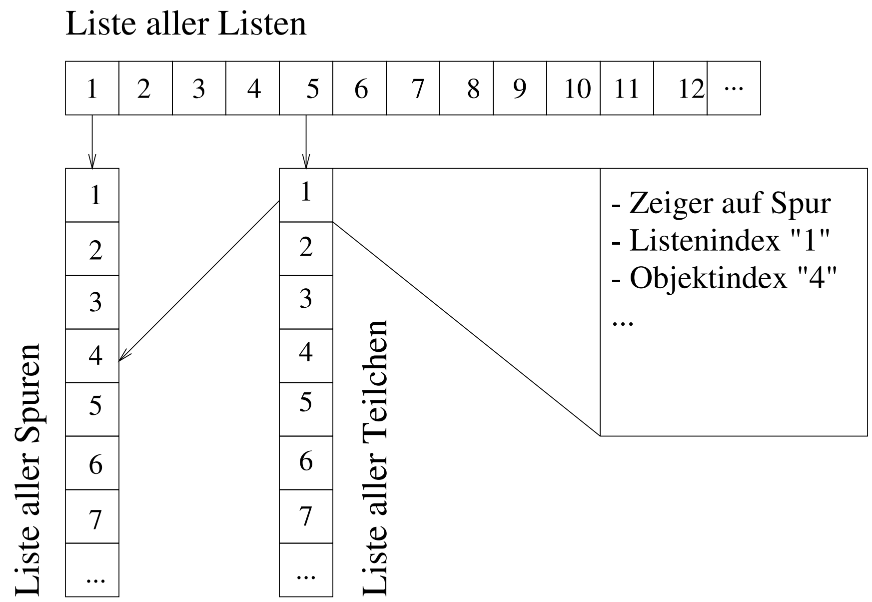

Abbildung 3.1: *Vereinfachte Funktionsweise eines* `H1Pointers`*: alle Objekte eines Ereignisses sind in Listen gespeichert, alle dieser Listen in einer globalen Liste. Ein Objekt läßt sich also durch die Nummer seiner Liste und seine Position darin mittels zweier ganzer Zahlen identifizieren. Diese werden gespeichert und beim Laden eines Ereignisses wieder in einen echten Zeiger umgewandelt.*

Teilworts groß zu schreiben. Weiterhin werden Präfixe benutzt, die den Zweck einer Methode/Funktion andeuten: “`Get`”, “`Set`” und “`Is`”.

Durch Ausschließen von Sprachmerkmalen, die nicht in allen eingesetzten Umgebungen benutzt werden können, wird erreicht, daß die Software auf allen vorhandenen Plattformen und für alle vorgesehenen Compiler verfügbar gemacht werden kann. Diese Regeln betreffen in erster Linie neue Standards, die noch nicht von allen Compilern unterstützt werden. Im Zweifelsfalle muß auf diese verzichtet werden.

Zusätzliche Konventionen ergeben sich durch die Funktionalität von ROOT, zur Laufzeit Typ-Informationen bereitzustellen oder auch z.B. automatisch HTML Dokumentation aus dem Quelltext zu generieren. Damit dies ausgenutzt werden kann, müssen die Kommentare im Code an bestimmten Stellen plaziert werden.

Die vollständigen H1 Programming Guidelines befinden sich im Anhang A.3.

3.3.2 Versionsverwaltung

Es gibt ein in der Hochenergiephysik entwickeltes Versionsverwaltungs-System namens CMZ (**C**ode **M**anagement with **Z**EBRA) [BBR], das am CERN entwickelt und von H1 als Standard benutzt wird. Es handelt sich um eine komplette Umgebung, in der Code verwaltet, verändert und kompiliert wird. Einen anderen Ansatz verfolgt CVS (**C**oncurrent **V**ersioning **S**ystem) [Ced], das auf RCS (**R**evision **C**ontrol **S**ystem) [Tic] aufsetzt und nur die Versionen des Codes verwaltet. Bearbeiten und Kompilieren geschieht dann mit beliebigen Mitteln (Editor, make etc.). Die Entscheidung, welches Werkzeug benutzt werden soll, fiel auf CVS, da es gegenüber CMZ flexibler ist und Funktionen beinhaltet, die CMZ nicht bietet. Zum Beispiel ist es in CMZ nicht möglich, daß Programmierer "gleichzeitig" am selben Code arbeiten.

Die Code Entwicklung im OO-Projekt wird mit CVS verwaltet. Mit diesem Werkzeug können verschiedene Programmierer gleichzeitig Code entwickeln und automatisch zusammenführen. In vielen Fällen ist dazu keine manuelle Intervention mehr nötig. Auch ist es möglich, die sogenannten Pakete oder Module in jeden beliebigen Zustand zurückzuversetzen. Dies geschieht entweder durch ein Datum oder eine Marke ("Tag" im CVS Jargon), die jeder Entwickler dem Code zu einem Zeitpunkt hinzufügen kann.

Eine Einführung in CVS bietet das Buch [Fog00], von dem auch ausgewählte Kapitel online [Fog] zur Verfügung stehen, unter anderem auch die sehr nützliche Befehlsreferenz. Ein Glossar der wichtigsten CVS Begriffe findet sich in Anhang B.1.1, häufig benutzte Kommandos in B.1.2.

3.3.3 Release-Strategie

Keine Software kann in nur einer Iteration der Programmierung fertiggestellt werden. Das H1-OO-Projekt schließt sich der Philosophie der Open Source Bewegung an, welche ebenfalls vom ROOT-Entwicklerteam angenommen wurde:

> "Release early and often, listen to the users." – "Stelle frühzeitig und häufig Versionen der Software zur Verfügung und höre darauf, was die Benutzer dazu sagen."

Auf diese Weise sollen Fehler schneller gefunden und Schwachstellen in der Handhabung früher aufgedeckt werden. Es soll erreicht werden, andere Mitglieder der Kollaboration für das Projekt zu interessieren, damit diese z.B. bestehende Algorithmen der alten Analyseumgebung in das neue Framework portieren und somit das Entwicklerteam erweitern.

Von der technischen Seite gesehen wird ein fester Rhythmus angestrebt, in dem neue Releases durchgeführt werden. Hier wird zwischen einem normalen Release und einem Bugfix-Release unterschieden. Ersterer ist i.d.R. dafür gedacht, neue Funktionalität bereitzustellen. Ein Bugfix-Release folgt dann i.a. kurz darauf, um Fehler zu beseitigen und Dokumentation nachzuliefern. Hier sollen keine neuen Merkmale eingeführt werden.

Die unterschiedlichen Releases werden durch Nummern gekennzeichnet, die folgendem Schema entsprechen <Major>.<Minor>.<Build>:

Major Meilenstein in der Entwicklung wie z.B. das Erreichen voller Funktionalität

Minor Größere Änderungen, möglicherweise nicht rückwärtskompatibel

Build Kleinere Änderungen, neue Funktionalität, Bereinigung von Fehlern

Zusätzlich wird zwischen dem letzten stabilen Release und dem Release für Entwickler unterschieden. Ersterer soll von der Kollaboration genutzt werden, um das neue Framework kennenzulernen und zu testen. Dieser sollte möglichst wenig schwerwiegende Fehler besitzen und sich ohne große Probleme benutzen lassen. Der Entwickler-Release bezeichnet i.a. den aktuellsten Stand, bei dem nicht gewährleistet sein muß, daß alle Programme absturzfrei laufen. Im Sinne eines Releases ist jedoch darauf zu achten, daß alle Pakete kompilieren (d.h. die zugehörigen Bibliotheken erstellt werden können) und sich möglichst alle Programme linken lassen (d.h. die ausführbare Datei erstellt werden kann).

Soll ein neuer Release erstellt werden, so ist bis zu einem bestimmten Zeitpunkt von jedem Verantwortlichen eines Paketes die Version ("Tag", siehe auch das CVS-Glossar in Anhang B.1.1) des Paketes zu benennen, die für den Release benutzt werden soll. Eine Versionstabelle führt Buch, aus welchen Paketen und deren Tags ein Release besteht.

Eine Release-Strategie mit kurzen Iterationszyklen ist auch ein Merkmal für die Software-Entwicklung nach der "Extreme Programming" Methode [Ble01]. Sie beinhaltet noch weitere, unkonventionelle Vorgehensweisen, die im OO-Projekt nicht zum Einsatz gekommen sind, z.B. das paarweise Arbeiten an Teilaufgaben nach dem Rotationsprinzip, so daß jeder Programmierer einmal an allen Teilen des Projektes gearbeitet hat.

3.3.4 Weitere Hilfsmittel

Damit nicht jeder Physiker bei H1, der sich für das OO-Projekt interessiert und Algorithmen selbst beisteuern möchte, sich intensiv mit CVS auseinander setzen muß, wurden Möglichkeiten gesucht, die Bedienung über ein Frontend zu erleichtern. Hier gibt es zwei prinzipielle Möglichkeiten: bereits vorhandene Frontends nutzen oder ein eigenes entwickeln.

In die erste Kategorie gehört `LinCVS` [RBDK], welches eine graphische Benutzerschnittstelle zu den CVS Kommandos bietet. Es läuft zwar stabil, hat jedoch drei Nachteile: nicht alle Funktionen sind voll ausgereift, man kommt nicht umhin, die CVS Kommandos zu lernen, und der Status der Dateien wird permanent überwacht, was insbesondere beim Kompilieren eine hohe Auslastung auf dem Rechner verursacht. Dies ist auch das schwerwiegendste Problem, da einige Programme nur auf Rechnern gestartet werden können, auf denen die Verzeichnisse sichtbar sind, in denen sich die Daten befinden. Diese sind im allgemeinen auch leistungsfähiger als der normale Desktop PC. Zusammen mit der Tatsache, daß es unbequem ist, auf verschiedenen Rechnern die Programme zu kompilieren und zu starten, führt das schnell dazu, daß dieses Tool unbenutzbar wird.

In die zweite Kategorie gehört das Perl-Skript `h1oo.pl`, welches speziell für den Anwender der H1-OO-Software geschrieben wurde (siehe Anhang A.2). Es versteckt die CVS Kommandos und sorgt durch eine Versionstabelle dafür, daß zueinander kompatible Versionen von verschiedenen Paketen ausgecheckt werden (siehe CVS Glossar in Anhang B.1.1). Dabei ist klar, daß nicht die volle Funktionalität von CVS bereitgestellt werden kann und soll. Das Skript ist hauptsächlich zu dem Zweck gedacht, eine lauffähige Umgebung (d.h. Anlegen benötigter Verzeichnisse, Generieren eines Makefiles für alle lokal ausgecheckten Pakete etc.) bereitzustellen, in der ein Benutzer typischerweise nur ein oder zwei Pakete selbst verändern will. Details finden sich in Anhang A.2.

Zuletzt bietet sich die Möglichkeit an, auf kommerzielle Produkte zur Softwareentwicklung zurückzugreifen, z.B. Rational Rose [Rat] und Sniff+ [Win].

Rose ist ein "visuelles Modellierungswerkzeug", d.h. es hilft bei der Darstellung von Zusammenhängen, Abläufen, Planspielen etc. Dabei wird die de facto Standardsprache für Modellierung eingesetzt: UML (**U**nified **M**odeling **L**anguage [Obj]). Es handelt sich also nicht um ein Werkzeug, mit dem man Code bearbeitet, sondern um ein Programm, das den eigentlichen Designprozeß vereinfacht. Da Rose nur auf Solaris zur Verfügung steht und zudem nur 10 Lizenzen bei DESY vorhanden sind, wird es nicht für das OO-Projekt eingesetzt.

Sniff+ dagegen ist eine sogenannte "integrierte Entwicklungsumgebung" (IDE = **I**ntegrated **D**evelopment **E**nvironment) für größere Projekte zwischen 100.000 und 5.000.000 Zeilen Code. Es bietet neben reinen Editierfunktionen mit Syntaxhervorhebung auch einen Klassenbrowser, CVS Frontend, automatische Kompiliermechanismen und Unterstützung für mehrere Sprachen und Plattformen. Zum jetzigen Zeitpunkt wird dieses Produkt getestet, Lizenzen zum allgemeinen Gebrauch sind aber noch nicht vorhanden.

Kapitel 4

Entwicklung der Teilchenklassen

Dieses Kapitel beschreibt den Fokus dieser Arbeit: Design und Implementierung von Teilchenklassen. Jets werden hier ebenfalls zu den Teilchen gezählt, da sie viele Eigenschaften gemeinsam haben und nur aus mehreren einzelnen Teilchen zusammengesetzt sind. Dies stellt sie – von der Softwareseite betrachtet – auf eine ähnliche Stufe wie die instabilen Teilchen, die man aus ihren Zerfallsprodukten zusammensetzt. Physikalisch gesehen liegen beiden "Teilchenarten" jedoch unterschiedliche Prozesse – Hadronisierung einerseits, Zerfall andererseits – zugrunde.

Information auf Teilchenniveau steht auf dem μODS zur Verfügung, wo es neben der Liste aller Teilchen auch weitere Listen gibt, die unter Annahme einer Identifizierung zusätzliche Informationen enthalten können (vgl. Abschnitt 3.2.1).

Was unter Teilchen verstanden werden kann und wie die Zusammenhänge zwischen verschiedenen Klassen sind, wird in Abschnitt 4.1 erörtert. Abschnitt 4.3 zeigt detaillierter, welcher der möglichen Ansätze gewählt wurde und auf welche Informationen die in Kapitel 5 vorgestellte Analyse zurückgreifen kann. Dies wird erweitert um einen Teil, der sich mit Implementierungen von Algorithmen zur Suche von Jets befaßt.

4.1 Bisheriges Teilchenmodell in H1PHAN

Analysen bei H1 benutzen üblicherweise das sogenannte "H1 physics analysis package" H1PHAN [H1 99b] und seine Standarderweiterungen wie z.B. QESCAT (für das Identifizieren des gestreuten Elektrons) oder HFS (für den hadronischen Endzustand). In H1PHAN werden Informationen über rekonstruierte und generierte Teilchen in einem "Q-Vektor" (QVEC) gespeichert. Dieser bietet einen ersten Ausgangspunkt für die Überlegungen, welche Eigenschaften spätere Teilchenklassen haben sollen. Dieser Abschnitt beschreibt das Teilchenmodell, wie es in H1PHAN benutzt wird. In 4.2.1 findet sich eine Umsetzung des H1PHAN-Konzeptes in objektorientierter Weise. In 4.2.2 wird der Entwurf erläutert, der in dieser Arbeit benutzt wird und eine Weiterentwicklung des ersten Konzeptes bedeutet. Zuletzt beschreibt Abschnitt 4.2.3 ein generalisiertes Design, das im weiteren Verlauf des Projektes implementiert werden soll.

Informationen im H1PHAN QVEC	
Kinematik	• Impuls
	• Energie
	• Masse
Eigenschaften	• Ladung (gemessen)
	• PDG Code
	• Typ (rekonstruiert, generiert)
	• Lock
	• Kovarianzmatrix
Relationen	• Entsprechender QVEC in anderem Lorentz System
	• Relation vom rekonstruierten (generierten) QVEC zum generierten (rekonstruierten)
	• Tochterteilchen und deren Anzahl
	• Mutterteilchen
aus PDG Tabelle	• Ladung
	• Lebensdauer
	• Masse
	• Zerfallsbreite
	• Antiteilchen

Tabelle 4.1: *Die Informationen im H1PHAN QVEC sortiert nach ihrem Typ.*

In Kapitel 10 von [H1 99b] – Working with 4-vectors – wird das Konzept des Q-Vektors erklärt. Q-Vektoren werden für verschiedene Arten von Objekten benutzt: gemessene Spuren/Cluster, daraus zusammengesetzte Objekte, Monte Carlo generierte Teilchen oder ganz allgemeine Vierervektoren. Nicht alle Eigenschaften machen dabei für jeden Typ Sinn bzw. sind für jeden Typ vorhanden. Ein QVEC enthält die Informationen aus Tabelle 4.1, Funktionen, die auf ihm operieren, befinden sich in Tabelle 4.2.

Diese Informationen und Funktionen sollen auch von künftigen Teilchenklassen beherrscht werden, damit diese sinnvoll eingesetzt werden können. Ebenso offensichtlich ist, daß es wenig Sinn macht, alle diese Eigenschaften in nur eine einzige Klasse zu zwängen. Das würde dem Grundgedanken objektorientierter Programmierung widersprechen. Besser ist es, die Dateninhalte und Funktionalitäten auf mehrere Klassen zu verteilen, d.h. eine Klassenhierarchie zu definieren.

4.2 Entwicklung neuer Teilchenmodelle

In Abschnitt 4.1 über das Modell, wie Teilchen in H1PHAN implementiert sind, wurden die Anforderungen an ein neues Modell aufgezeigt. In den weiteren Abschnitten von 4.2 sollen die einzelnen Schritte nachvollzogen werden, die zu verschiedenen Implementierungen eines

Funktionen für H1PHAN QVECs	
Erzeugen eines neuen QVEC	• als leerer QVEC
	• als Kopie eines anderen
	• als Summe zweier anderen
	• als Differenz zweier anderen
Lorentztransformationen	• ins Ruhesystem eines anderen QVECs
	• mittels Lorentzrotation
Berechnete Eigenschaften	• $\cos\theta$, θ, φ, β, γ
	• Transversalimpuls
	• Rapidität und Pseudorapidität
	• Invariante Masse von 2, 3 oder 4 QVECs
	• Skalarprodukt von Dreier- bzw. Vierervektoren
	• $\cos\angle$, $p_\perp$, $p_{\|\|}$ relativ zu anderem QVEC
Mit Massenhypothese	• Energie aus Impuls und angenommener Masse
	• Invariante Masse von 2 oder 3 QVECs unter Massenhypothese

Tabelle 4.2: *Vorhandene Funktionen, die auf H1PHAN QVECs operieren, sortiert nach ihrem Typ.*

Teilchenmodells geführt haben. Diese Entwicklung ist noch nicht abgeschlossen, mit der Implementierung eines Modelles, welches das in dieser Arbeit zur Analyse benutzte (siehe 4.3) ablösen wird, wurde bereits begonnen.

4.2.1 Eine Klassenhierarchie ohne identifizierte Teilchen

Ausgehend vom Konzept des H1PHAN QVECs enthält dieser Ansatz eine Basisklasse und davon abgeleitet drei konkrete Klassen für gemessene Teilchen, Jets und generierte Teilchen, wie in Abbildung 4.1 gezeigt.

Die Idee hinter diesem Konzept ist, daß die verschiedenen Arten von konkreten Teilchenklassen Gemeinsamkeiten besitzen, welche in der Basisklasse `H1Particle` implementiert sind. Dies reduziert sich auf die kinematischen Informationen, die in einem `TLorentzVector` von ROOT gespeichert werden; also p_x, p_y, p_z und E. Zusätzlich wird ein Interface zu weiteren Eigenschaften definiert, die zwar für jede der konkreten Klassen vorhanden sind, aber nicht auf dieselbe Art und Weise implementiert werden können. Dies sind Mutter/Tochter Relationen und die Relation zu einem Vertex.

Im `H1ParticleCand` wird zusätzlich abgespeichert, ob eine Ladung gemessen wurde und wenn ja, deren Wert. Dieses stellt die Standardklasse für generische Teilchen dar, über die noch keine Aussage gemacht wird, als was sie identifiziert werden können.

Das `H1MCParticle` ist gedacht für generierte Teilchen (inklusive Partonen), bei denen zusätzlich zu kinematischen Informationen auch die Eigenschaften aus den Tabellen der Particle Data Group zugänglich sind. Zu diesem Zweck wird in der Klasse selbst nur eine In-

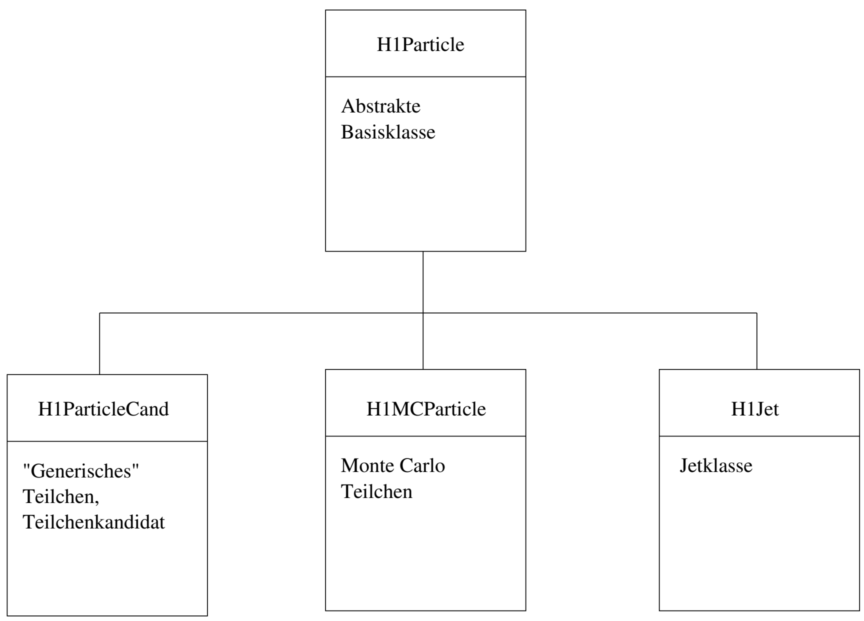

Abbildung 4.1: *Die Klassenhierarchie ohne identifizierte Teilchen. Das* `H1Particle` *enthält die kinematischen Informationen, die allen Teilchen gemeinsam sind.* `H1ParticleCand` *ist die Klasse für gemessene Teilchen (hat z.B. eine Ladung),* `H1MCParticle` *beschreibt Monte-Carlo-generierte Teilchen (besitzt einen PDG Code und die darüber verfügbaren Eigenschaften) und* `H1Jet` *ist die Summe mehrerer Teilchen (kennt den Gesamt-Vierervektor und die Teilchen, aus denen er aufgebaut ist).*

tegerzahl gespeichert, die den PDG Code enthält. ROOT besitzt eine Klasse `TDatabasePDG` über die anhand dieses Wertes alle weiteren Eigenschaften angesprochen werden können.

Der `H1Jet` stellt eine Zusammenfassung von einem oder mehreren Teilchen dar. Er wird als einziger nicht direkt mit Werten für seine Eigenschaften erzeugt sondern als leeres Objekt, dem dann Teilchen hinzugefügt werden. Diese werden beim Jet als Töchter bezeichnet und ihre Vierervektoren zum Gesamt-Vierervektor des Jets aufsummiert.

Das konkrete Design der Relationen war bei diesem Entwurf noch nicht vorhanden. Das H1-Pointermanagement wurde erst zu dem Zeitpunkt entwickelt, als klar wurde, daß in naher Zukunft keine zufriedenstellende Funktionalität in ROOT integriert werden würde, die erlaubt, Referenzen ohne vollständige Kopie des referenzierten Objektes abzuspeichern.

4.2.2 Zwei unabhängige Hierarchien für allgemeine und identifizierte Teilchen

Der Ansatz aus Abschnitt 4.2.1, welcher nur generische, gemessene Teilchen, Jets und generierte Teilchen vorsieht, wird dem Konzept des μODS nicht gerecht, da es keine Klassen gibt, die sich für die Listen der identifizierten Teilchen eignen. Somit wäre nicht möglich, beispielsweise allein aus den μODS-Informationen $\sum(E - p_z)$ aus dem gestreuten Elektron und dem HFS (**H**adronic **F**inal **S**tate = hadronischer Endzustand) zu berechnen, da die `H1ParticleCand`s keinerlei Information haben, aus was sie konstruiert wurden. Ausnahme wäre eine Situation, wo nur die Teilchen des hadronischen Endzustandes und das gestreute Elektron in dieser Liste gespeichert würden. Da dies nicht das angestrebte Ziel ist, muß ein allgemeingültigeres Konzept für die Behandlung von Teilchenidentifikation gewählt werden.

Vom `H1ParticleCand` abgeleitete Klassen kommen nicht in Frage, da mit diesem Ansatz Information dupliziert würde, die bereits in der globalen Liste aller Teilchen enthalten ist.

Möglich ist eine zweite Hierarchie von Teilchenklassen, die unabhängig von derjenigen aus Abbildung 4.1 ist. Auch hier gilt es wieder, Gemeinsamkeiten zu suchen, die in eine Basisklasse für identifizierte Teilchen (`H1IDPart`) integriert werden können. Es stellt sich heraus, daß als einzige gemeinsame Eigenschaft der Zusammenhang zwischen einem identifizierten Teilchen und seinem zugeordneten generischen Teilchen besteht. Alle weiteren Dinge sind zu speziell, um in der Basisklasse implementiert zu werden.

Eine konkrete Klasse für identifizierte Teilchen ist `H1IDScElec` für die gestreuten Elektronen. Sie enthält den Typ des Elektrons (im SpaCal oder LAr gemessen) und ein Qualitätskennzeichen. Letzteres gibt an, ob dieses Elektron engen oder losen Schnitten genügt.

Zwei abgeleitete Klassen – `H1IDTrack` und `H1HFS` – werden als reine "Zeigerklassen" benutzt, d.h. sie besitzen nur den `H1Pointer` aus `H1IDPart`, um einen Zugriff auf die zugeordneten generischen Teilchen zu ermöglichen. Bei den Spuren handelt es sich um solche, die besonderen Qualitätskriterien entsprechen.

Es ergibt sich ein Vererbungsbaum für die identifizierten Teilchen wie in Abbildung 4.2. Die Klassen aus diesem Baum werden im weiteren Verlauf dieser Arbeit als "Zeigerklassen" bezeichnet, da sie im wesentlichen nur wenig Informationen mehr enthalten als die generischen Teilchenkandidaten, auf die sie verweisen. Ihr Zweck ist hauptsächlich, den

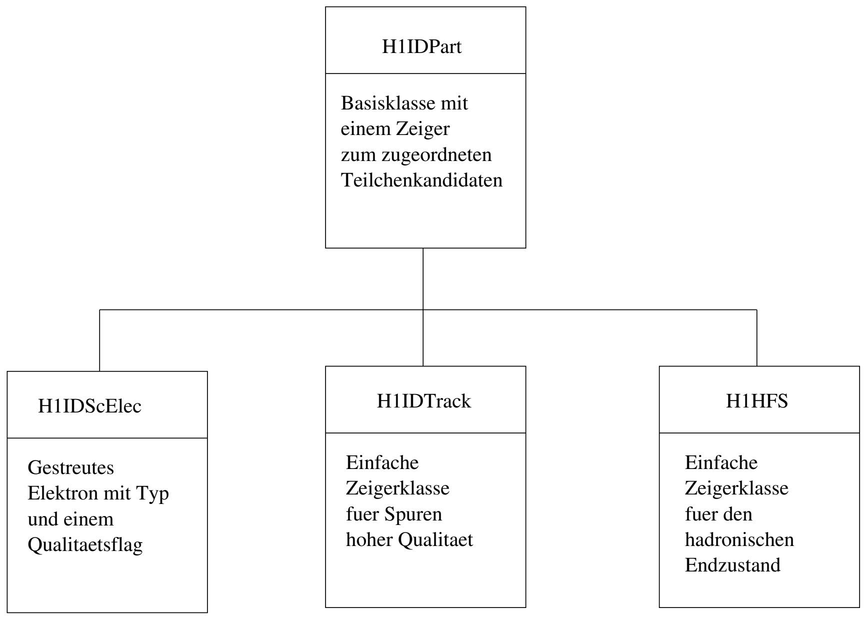

Abbildung 4.2: *Die Klassenhierarchie für separate identifizierte Teilchen.*

`H1ParticleCand`s eine Identität zu geben, wobei es auch erlaubt ist, von zwei unterschiedlichen identifizierten Teilchen auf denselben generischen Kandidaten zu verweisen. Auf diese Weise ist es z.B. möglich, einen Teilchenkandidaten je nach den an ihn gestellten Anforderungen als ein einfaches Objekt im Detektor, als Elektron oder als Myon anzusehen.

4.2.3 Eine einzige Klassenhierarchie inklusive identifizierten Teilchen

Auch das Design aus Abschnitt 4.2.2, in dem die allgemeinen Klassen von den Klassen für identifizierte Teilchen getrennt sind, ist nicht vollkommen zufriedenstellend. Das Vorhandensein von zwei getrennten Vererbungsbäumen suggeriert einen fundamentalen Unterschied zwischen den Klassen beider Seiten. Im Sinne einer transparenten Analyseumgebung sollte es aber keinen Unterschied machen, ob ein und dasselbe Teilchen als identifiziert angesehen wird oder nicht. Dies legt die Verwendung einer abstrakten Basisklasse nahe, d.h. einer Klasse, die selbst keine Daten enthält sondern nur ein Interface vorgibt.

Für alle Teilchen sollen z.B. kinematische Informationen verfügbar sein. Es muß in der Basisklasse also eine Methode bereitgestellt werden, welche diese zugänglich macht. Im Falle der identifizierten Teilchen ist diese Information nicht in den Klassen selbst gespeichert, sondern über den referenzierten Teilchenkandidaten abrufbar. Daneben macht es für alle Arten von Teilchen Sinn, Ladung, Mütter/Töchter und einen Vertex zu haben. Die Art der Implementierung wird aber von der konkreten Klasse abhängig sein, so daß diese Eigenschaften ebenfalls nur als Schnittstelle in die Basisklasse eingehen. Auf der anderen Seite gibt es Eigenschaften, die nicht allen Arten von Teilchen – im Sinne von Klassen –

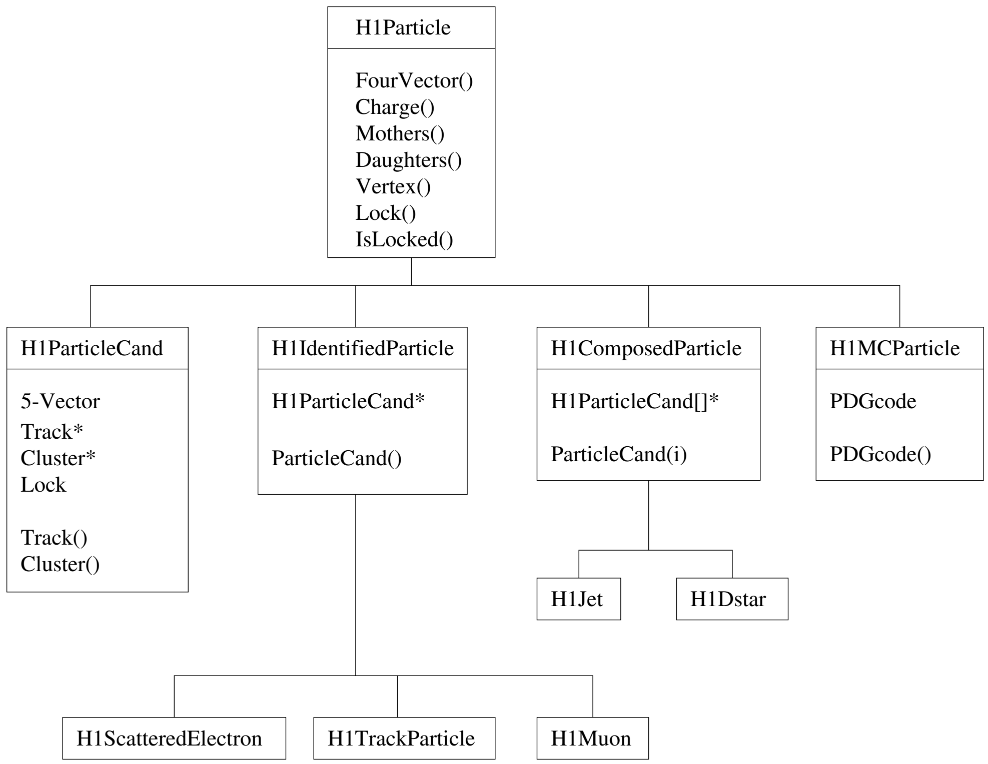

Abbildung 4.3: *Die Klassenhierarchie mit identifizierten Teilchen. Klammern hinter einem Begriff bedeuten, daß es sich um eine Methode der Klasse handelt, Begriffe mit einem Asterisk kennzeichnen Zeigervariablen und Begriffe ohne Zusatz stehen für normale Variablen. Unter "5-Vector" werden die drei Impulskomponenten, die aus Spurinformationen gewonnen werden, plus die Energien aus elektromagnetischem und hadronischem Kalorimeter verstanden. Dieses Design ist für noch folgende Updates vorgesehen und kommt in dieser Arbeit nicht zum Einsatz. [Kuh01]*

gemeinsam sind. Ein Beispiel hierfür ist der PDG Code, der nur für generierte Teilchen bekannt ist. Daher kann die Definition der Zugriffsmethode auf dieses Datum erst in einer abgeleiteten Klasse erfolgen.

Eine weitere Änderung gegenüber dem Modell aus Abbildung 4.2.2 ist die Einführung einer Klasse für zusammengesetzte Teilchen. In diese Kategorie gehören einerseits die Jets und andererseits Teilchen, die weiter zerfallen, z.B. D^* oder J/Ψ. Für beide Arten ist interessant, aus welchen ursprünglichen Teilchen sie zusammengesetzt wurden. Dieser Fall ist ähnlich zu den identifizierten Teilchen, die statt mehreren nur einen assoziierten Teilchenkandidaten besitzen.

Den Vererbungsbaum mit einer gemeinsamen Basisklasse für alle konkreten Teilchen zeigt Abbildung 4.3.

4.3 Das für diese Arbeit gewählte Modell

Während das Modell aus Abschnitt 4.2.3, bei dem alle Teilchenklassen eine gemeinsame Basisklasse besitzen, das langfristige Ziel des Projektes ist, steht zum Zeitpunkt dieser Arbeit ein auf Abschnitt 4.2.2 basierendes Modell zur Verfügung, in dem die allgemeinen und die Klassen für identifizierte Teilchen voneinander unabhängig sind. Weiterhin existiert noch kein Code, der μODS und HAT aus ODS erzeugt. Das heißt die zusätzlichen Ebenen müssen aus DSTs erzeugt werden, weshalb es noch keine Verbindung zu den Objekten auf ODS gibt. Um dies zu kompensieren, wurden weitere Listen von Objekten dem μODS hinzugefügt. Dies stellt eine temporäre Lösung dar und ist nicht Teil des eigentlichen Konzeptes. Das Schema dieses Ansatzes zeigt Abbildung 4.4.

4.3.1 Klassenhierarchie

In dieser hier im folgenden vorgestellten Analyse kommen die beiden Vererbungsbäume aus Abbildungen 4.1 und 4.2 zum Einsatz, d.h. der Baum für die generellen Klassen sowie derjenige für die Zeigerklassen. Zusätzlich stehen weitere Klassen zur Verfügung, die unter dem Oberbegriff "Detektorobjekte" zusammengefaßt werden. Sie enthalten Informationen über verschiedene Typen von Endzuständen, Teilchen aus dem elektromagnetischen Kalorimeter, Myonkandidaten und Spuren. Die Verbindung zu den anderen Klassen ist dergestalt, daß identifizierte Teilchen einen oder mehrere Zeiger auf Detektorobjekte besitzen. Ein Beispiel wäre das gestreute Elektron, dem ein Objekt der `H1LtEmParticle` Klasse zugeordnet ist. Diese Klasse speichert Informationen über Teilchen, welche Energie im elektromagnetischen Teil des Kalorimeters deponiert haben. Von dort sind unter anderem der Spur-Cluster Abstand sowie detailierte Informationen zu den ursprünglichen Spuren und/oder Clustern zugänglich.

Die Detektorobjekte stammen aus den Informationen, die der LOTUS Code bereitstellt. Dabei handelt es sich um ein Softwarepaket, das auf H1PHAN aufbaut. Zusätzlich führt es aber schon eine endgültige Kalibrierung durch, um die Daten zu verbessern. LOTUS stellt auf diese Weise einen Großteil der Variablen zur Verfügung, die bei H1-Analysen benötigt werden [Hei00]. Eine Liste aller LOTUS Variablen findet sich in [H1 99c].

Für das OO-Projekt wurde eine Wrapper-Bibliothek namens `lotus++` bereitgestellt, die die Informationen aus den `FORTRAN` Common Blöcken in C++ zugänglich macht. Diese wird ebenfalls benutzt, um μODS und HAT zu füllen solange es noch keinen Code für `H1PHAN++` gibt. Eine Übersicht der derzeitigen Produktionsweise der drei Datenlagen zeigt Abbildung 4.5. Die Konvertierung zwischen den Formaten auf DST und ODS in beide Richtungen erfolgt mittels des Paketes `bos2oop` (siehe auch die Paketübersicht in Anhang A.4), die Erzeugung von μODS und HAT geschieht mit Hilfe der `lotus++` Bibliothek. Dies soll in der Zukunft durch eine Produktionskette ersetzt werden, in der `H1PHAN++` auf ODS aufbauend die übergeordneten Ebenen erzeugt.

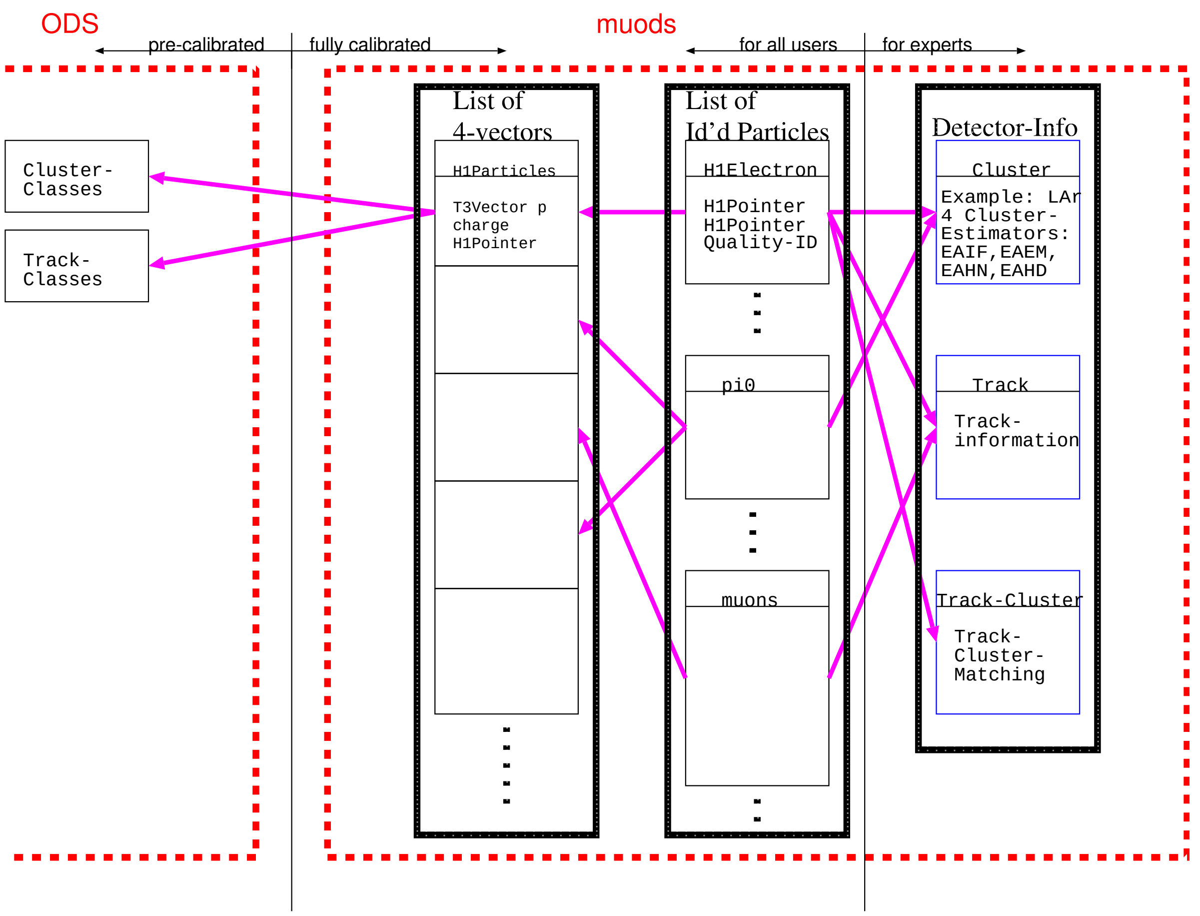

Abbildung 4.4: *Das Schema des temporären μODS Ansatzes. Da die Relationen von der Liste der Vierervektoren zu den Track und Cluster Klassen auf ODS noch nicht zur Verfügung stehen, wurde dem Modell ein zusätzlicher Teil mit Detektorinformationen hinzugefügt. In der vorläufigen Version arbeiten die Listen von Vierervektoren und identifizierten Teilchen mit den Detektorinformationen auf μODS zusammen. Später sollen diese Informationen teilweise über die Referenzen zu den ODS-Klassen zugänglich sein und zum Teil in die Klassen für identifizierte Teilchen verschoben werden. [Mey01]*

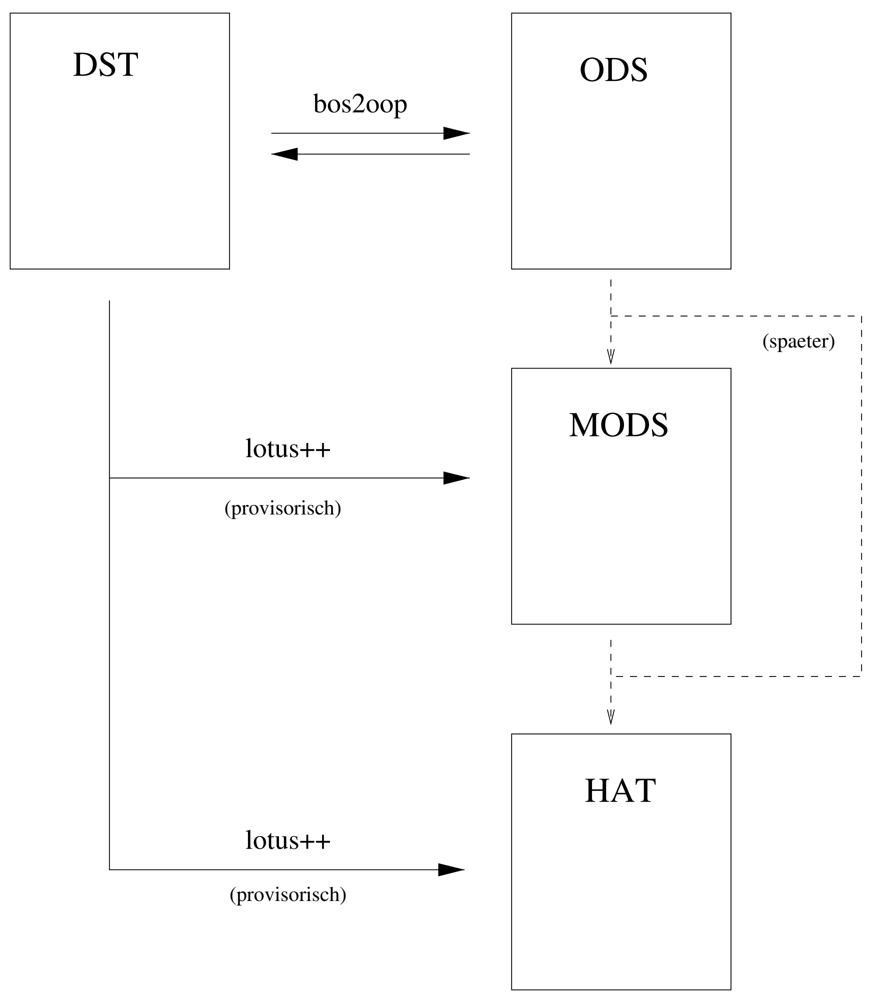

Abbildung 4.5: *Die Produktion der drei Datenlagen basiert z.Z. rein auf den alten DSTs. ODS wird vom* `bos2oop` *Paket erzeugt, μODS und HAT davon unabhängig aber parallel vom* `lotus++` *Paket. Dieser zweite Teil wird in der endgültigen Analyseumgebung durch* `H1PHAN++` *ersetzt, welches die beiden übergeordneten Ebenen von ODS ausgehend erzeugen wird.*

4.3.2 Die konkrete Analyseumgebung

In diesem Abschnitt wird näher auf die konkrete Implementierung der in 4.3.1 vorgestellten Klassenhierarchie eingegangen. Der Schwerpunkt soll dabei auf den tatsächlich benutzbaren Klassen liegen. Auf fehlende Funktionalität oder bisher nur unzureichend implementierte und somit noch nicht benutzbare Klassen wird zumindest hingewiesen.

Kernstück für die Datenanalyse auf dem Teilchenniveau ist die Klasse `H1ParticleCand`, die prinzipiell für jedes Objekt benutzt wird, das im Detektor gemessen wurde und bestimmte Mindestvoraussetzungen erfüllt. Sie erbt von `H1Particle` die kinematischen Informationen p_t, θ, φ, Energie im elektromagnetischen und hadronischen Kalorimeter sowie die gemessene Ladung. Ebenfalls in der Basisklasse befindet sich ein Datenelement H1Bits, das 32 binäre Zustände speichern kann. Dies wird z.B. benutzt um zu kennzeichnen, ob ein Teilchen gewissen Kriterien der Identifikation genügt. Die Klasse `H1ParticleCand` selbst beinhaltet zusätzlich die Relationen zu Müttern/Töchtern und seinem Vertex. Diese Funktionalität ist aber noch nicht implementiert. Die Behandlung von zugehörigen Spuren und Clustern fehlt noch ganz.

Als Klassen für identifizierte Teilchen kommen `H1IDScElec`, `H1HFS` und `H1IDTrack` zum Einsatz. Alle drei beinhalten einen `H1Pointer`, der auf den zugehörigen Teilchenkandidaten in der globalen Teilchenliste verweist. Dieses Datenelement wird von der Basisklasse `H1IDPart` geerbt. In der Klasse für gestreute Elektronen wird zusätzlich gespeichert, ob es im LAr oder im SpaCal gefunden wurde. Diesem zugeordnet ist die Detektorobjektklasse `H1LtEmParticle`. Sie enthält alle Details eines Teilchens, das im Kalorimeter Energie deponiert hat. Zu den "guten" Spuren und den HFS Objekten stehen keine weiteren Informationen in Form von Detektorobjekten zur Verfügung. Relationen zwischen identifizierten Teilchen und Detektorobjekten sind noch nicht implementiert. Weiterhin fehlt eine von `H1IDPart` abgeleitete Klasse für identifizierte Myonen. Es existiert aber eine Klasse `H1LtMuon`, die ähnlich dem `H1IDScElec` sehr detaillierte Informationen über ein identifiziertes Teilchen – hier ein Myon – enthält.

Zusammenfassend sei gesagt, daß das Konzept der Detektorobjekte stark an der alten Analysemethode mit N-Tupeln orientiert ist und in etwa die LOTUS Variablen darstellt, die nicht schon im HAT gespeichert sind, da sie sich auf Teilcheneigenschaften beziehen (zu LOTUS Variablen siehe [H1 99c]).

Damit ergibt sich eine Übersicht der nutzbaren Klassen wie in Tabelle 4.3.

Das Erzeugen von μODS und HAT erfolgt mit dem Paket `modshat` (für Paketbeschreibungen siehe Anhang A.4). Es benutzt `h1++` für den Zugriff auf die DST Dateien und `lotus++` für die physikalischen Algorithmen, welche die nötigen Informationen aus den Daten extrahieren.

Pro Ereignis wird die globale Liste aller Teilchen gefüllt. Sie enthält `H1ParticleCand`s aus gestreuten Elektronen, dem hadronischen Endzustand und Spuren hoher Qualität. Zu jedem dieser Typen gibt es eine separate Liste, die mit der entsprechenden Zeigerklasse gefüllt wird. In zwei weiteren Listen werden die detaillierten Myon- bzw. Elektroninformationen gespeichert. Dabei ist die Liste der `H1LtEmParticles` parallel zu derjenigen der `H1IDScElec`s, d.h. beide Klassen treten z.Z. paarweise auf. Auf diese Weise kann das Feh-

Klassen der konkreten Analyseumgebung	
Teilchenklassen	• `H1ParticleCand` mit Spurparametern, elektromagnetischer und hadronischer Energie, Ladung
	• `H1IDScElec` mit dem Detektor, in dem es gefunden wurde
	• `H1IDTrack` und `H1HFS` als reine Zeigerklassen
Detektorobjekte	• `H1LtEmParticle` mit Details zu den Spuren und/oder Clustern, aus denen das Teilchen erzeugt wurde
	• `H1LtMuon` mit detaillierten Informationen zu Myonkandidaten

Tabelle 4.3: *Übersicht über die Klassen, die in der Analyseumgebung zum Zeitpunkt dieser Arbeit verfügbar sind.*

len einer direkten Relation, die mit Zeigern realisiert würde, kompensiert werden. Die `H1LtMuon` Klasse hat keine Entsprechung bei den Zeigerklassen für identifizierte Teilchen.

Auf Seiten des HATs werden Ereignisvariablen gespeichert, die der Übersicht halber in fast allen Fällen in Klassen zusammengefaßt sind. Um nun doch jede einzelne Variable zur Selektion benutzen zu können, werden die Instanzen der Klassen im sogenannten "Splitmode" – einer Option zum Anlegen von ROOT-Dateien – gespeichert. In diesem Modus bekommt jedes Datenelement einer Klasse einen eigenen "Branch", der unabhängig von den anderen gelesen werden kann. Dieser Automatismus erspart viel Handarbeit gegenüber dem Anlegen von 200 einzelnen Branches. Details zum Tree-Konzept von ROOT finden sich in Kapitel 10 von [BRP+00]. Ausnahmen bilden Run-/Eventnummer, Runtyp/-qualität sowie Bunchnummer/-typ, die nicht in Klassen gekapselt sind. Auf diese Weise wird z.B. gewährleistet, daß das Programm zum Füllen des Runkataloges unabhängig von den weiteren OO-Bibliotheken ist. Es greift nur auf Run-/Eventnummer zu, die direkt als einfache Datentypen in der Datei vorliegen.

Näheres zu den Informationstypen auf HAT zeigt Tabelle 4.4, welche ebenfalls die Informationen auf ODS und μODS darstellt, wie sie zum Zeitpunkt dieser Arbeit zur Verfügung stehen.

4.3.3 Jetfinder im OO-Framework

Da sich Abschnitt 4.3.2 von der Softwareseite mit den Klassen und Algorithmen befaßt, welche für die in Kapitel 5 vorgestellte Analyse verfügbar sind, soll an dieser Stelle nur auf die technischen Aspekte der Implementierung eingegangen werden. Die physikalischen Aspekte der Jetidentifikation werden in Abschnitt 5.1.2 besprochen.

Die Identifizierung von Jets ist nicht in der Standard-Distribution von ROOT enthalten. Solche Algorithmen sind in der Hochenergiephysik aber von allgemeinem Interesse, so daß mit hoher Wahrscheinlichkeit bereits bei einem anderen Experiment an ihnen gearbeitet wurde. Auf der ROOT Webseite [BR96] findet sich ein Bereich für Beispielanwendungen. Unter anderem gibt es dort einen Verweis auf eine Bezugsquelle für Jetfinder/Thrustfinder Programme [Iwa00]. Diese konkrete Implementierung innerhalb des ROOT-Frameworks

Konkreter Inhalt der drei Datenlagen	
ODS	• Konvertierte BOS Bänke • Spuren und Cluster (unselektiert, keine endgültige Kalibrierung) • Keine Relationen zu μODS und HAT
μODS	• Globale Liste aller Teilchen bestehend aus gestreuten Elektronen, klassifizierten Myonen, HFS Objekten und "guten" Spuren Kann mehrere Einträge haben, die aus denselben Informationen konstruiert wurden • Listen identifizierter Teilchen für gestreute Elektronen, HFS Objekte und "gute" Spuren • Listen von "Detektorobjekten" für Teilchen aus dem elektromagnetischen Kalorimeter und Myonen
HAT	• Run/Event Nummer, Run Typ/Qualität, Bunch Nummer/Typ • Informationen zum Primärvertex • Ereigniskinematik • Taggerinformation • Triggerinformation • Status des Hochspannungssystems • Nicht-eP Hintergrund • Diffraktive Kinematik • Vorwärtsdetektorinformation • Weitere Details zum Vorwärtsdetektor • Hadronischer Endzustand und Energiesummen • Charged Current Selektionen • Hadronischer Nur-Cluster Endzustand • FSCOMB-Informationen zum hadronischen Endzustand • Extra FSCOMB Information für diffraktive Analysen • Gesamtzahlen der DTRA-Spuren im zentralen und Vorwärtsbereich • Zahl der Einträge in den Listen auf μODS

Tabelle 4.4: *Übersicht über den Inhalt der drei Datenlagen zum Zeitpunkt dieser Arbeit. Das geplante Datenvolumen pro Ereignis verhält sich von ODS absteigend wie* $30:6:1$.

wurde am SLAC entwickelt und enthält den Jade- [JAD86] und Durham-Algorithmus [CDW92].

Die Klassen dieser Implementierung arbeiten auf Listen, welche Objekte vom Typ `TVector3` und `TLorentzVector` enthalten dürfen. Intern wird eine neue Liste mit Vierervektoren gefüllt, im Falle eines `TVector3` wird die Pionmasse angenommen, um die Energie zu berechnen. Dieses Interface wurde so angepaßt, daß die Algorithmen Objekte vom Typ `H1ParticleCand`, `H1MCParticle` und `TLorentzVector` bearbeiten können. Es wird dazu jeweils die Vierervektor-Information umkopiert. Weitere Änderungen am Quelltext wurden nicht vorgenommen.

Desweiteren befindet sich eine objektorientierte Implementierung des k_t-Algorithmus für die neue Analyseumgebung in Arbeit [Dav01]. Diese kann zum Zeitpunkt dieser Arbeit aber noch nicht eingesetzt werden.

Kapitel 5

Physikalische Fragestellung

An dieser Stelle sollen die bereitgestellten Mittel des neuen Analyseframeworks auf eine Beispielanalyse angewendet werden. Zu diesem Zweck wird der Teil der Arbeit "Measurement and Pertubative QCD Fit of Dijet and Inclusive Cross Sections at HERA" [Had99] nachvollzogen, der sich mit der Messung beschäftigt. Dabei handelt es sich um die Bestimmung von Wirkungsquerschnitten für die Produktion von Jets in der *ep*-Streuung anhand der Daten, die das H1-Experiment in den Jahren 1995 bis 1997 geliefert hat. Diese Arbeit beschäftigt sich mit den Daten des Jahres 2000.

5.1 Theorie

5.1.1 Kinematik

Da sich der Schwerpunkt des *ep*-Systems aufgrund der unterschiedlichen Energie von Elektron und Proton relativ zum Detektor in Bewegung befindet, ist es sinnvoll, alle kinematischen Variablen zur Beschreibung der Reaktion in lorentzinvarianter Form zu schreiben. Damit werden Transformationen zwischen Schwerpunkt- und Laborsystem vermieden. Zudem vereinfacht eine solche Notation die Vergleiche mit anderen Experimenten und der

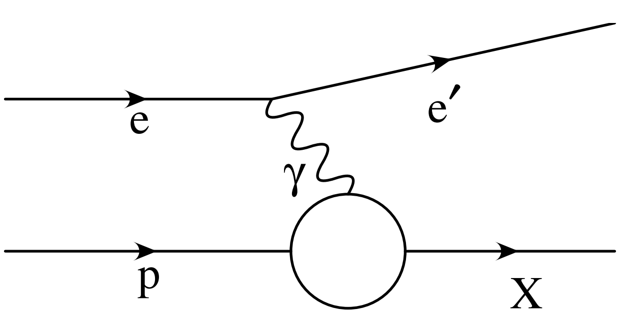

Abbildung 5.1: *Ein einfacher Graph zur ep-Streuung unter Austausch eines Photons.*

theoretischen Vorhersagen. Weiterhin wird ein Maßsystem benutzt, in dem $\hbar = 1$ und $c = 1$ gesetzt wird. So erhalten Energie, Impuls und Masse die gleiche Dimension und werden in GeV angegeben. Eine Umrechnung erfolgt anhand der Relation

$$197,33 \text{ MeV} = 1 \text{ fm}^{-1}. \tag{5.1}$$

Zudem können aufgrund der hohen Energien von Elektron und Proton deren Massen vernachlässigt werden.

Die Einführung der Invarianten soll anhand der inklusiven ep-Streuung geschehen. Eine graphische Darstellung des Prozesses ist in Abbildung 5.1 zu sehen, die zugehörige Gleichung ist

$$e(k) + p(P) \longrightarrow e'(k') + X(h). \tag{5.2}$$

Die Teilchen werden dabei durch die relativistischen Vierervektoren

$$p \equiv p^\mu \equiv \begin{pmatrix} E \\ \vec{p} \end{pmatrix} \tag{5.3}$$

beschrieben, deren Quadrat definiert ist als

$$p_\mu p^\mu = p^\mu p_\mu \equiv E^2 - \vec{p}^2 = m^2 \tag{5.4}$$

und dem Quadrat der Teilchenmasse entspricht.

Im Falle, daß nur das gestreute Elektron e' gemessen und der hadronische Endzustand nicht aufgelöst wird, ist der Vierervektor des gesamten hadronischen Endzustandes X durch Energie- und Impulserhaltung berechenbar:

$$h = k + P - k'. \tag{5.5}$$

Die Vierervektoren der einlaufenden Teilchen sind bekannt:

$$k := \begin{pmatrix} E_e \\ 0 \\ 0 \\ -E_e \end{pmatrix} \tag{5.6a}$$

$$P := \begin{pmatrix} E_p \\ 0 \\ 0 \\ E_p \end{pmatrix} \tag{5.6b}$$

so daß in Gleichung 5.5 drei voneinander unabhängige Unbekannte übrig bleiben, da bereits eine Komponente des Viererimpulses über Gleichung 5.4 durch die restlichen ausgedrückt werden kann. Dies reduziert sich auf zwei unabhängige Variablen zur Beschreibung der vollständigen Kinematik, da die Elektron- und Protonstrahlen unpolarisiert sind und somit keine Abhängigkeit von φ besteht.

Eine naheliegende Wahl für lorentzinvariante Größen zur Beschreibung der Kinematik sind die sogenannten Mandelstam-Variablen, die über

$$\begin{aligned} s &:= (k+P)^2 = (h+k')^2 && (5.7a)\\ t &:= (k-k')^2 = (h-P)^2 && (5.7b)\\ u &:= (P-k')^2 = (h-k)^2 && (5.7c)\\ m_X^2 \approx m_X^2 + m_p^2 + 2m_e^2 &= s+t+u, && (5.7d) \end{aligned}$$

definiert sind. Für die Summe der drei Variablen ergibt sich die Summe über die Massenquadrate aller beteiligten Teilchen. Die Massen von Elektron und Proton werden hierbei als klein gegenüeber der invarianten Masse des hadronischen Endzustandes X angenommen. Die Variable t kann als invariantes Massenquadrat des Austausches zwischen Elektron und Proton angesehen werden, d.h. als Virtualität des ausgetauschten Bosons ($t = q^2$). Nun ist s jedoch schon durch die einlaufenden Teilchen zu $4E_pE_e$ festgelegt und da die vierte Gleichung eine Beziehung zwischen den drei Größen herstellt, reichen sie nicht zur vollständigen Beschreibung aus.

Dazu behilft man sich mit den beiden lorentzinvarianten Skalenvariablen x und y, die definiert sind als

$$\begin{aligned} x &:= \frac{-t}{2P\cdot(k-k')} && (5.8a)\\ y &:= \frac{P\cdot q}{P\cdot k} = \frac{2P\cdot(k-k')}{s} && (5.8b)\\ Q^2 &:= -t = sxy, && (5.8c) \end{aligned}$$

wobei die letzte Gleichung die Skalenvariablen mit mit den Mandelstam-Variablen s und t verknüpft. Somit stehen zwei unabhängige Größen zur Verfügung, z.B. Q^2 und x.

y wird auch Inelastizität genannt. Man sieht dies nach einer Lorentztransformation ins Ruhesystem des Protons. Dort ist

$$P^{\mathrm{prs}} = \begin{pmatrix} m_p \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad (5.9)$$

so daß y dem relativen Energieverlust des Elektrons entspricht:

$$y = \frac{P\cdot(k-k')}{p\cdot k} = \frac{m_p(E_e^{\mathrm{prs}} - E_{e'}^{\mathrm{prs}})}{m_p E_e^{\mathrm{prs}}} = \frac{E_e^{\mathrm{prs}} - E_{e'}^{\mathrm{prs}}}{E_e^{\mathrm{prs}}}. \qquad (5.10)$$

Die invarianten Größen lassen sich bei einem Prozeß der neutralen Ströme auf unterschiedliche Arten berechnen. Zunächst bietet es sich an, als Observablen den Winkel $\theta_{e'}$ und die Energie $E_{e'}$ des gestreuten Elektrons zu benutzen:

$$Q_e^2 = 4E_eE_{e'}\cos^2\frac{\theta_{e'}}{2} \qquad (5.11a)$$

$$x_e = \frac{Q_e^2}{4E_p\left(E_e - E_{e'}\sin^2\frac{\theta_{e'}}{2}\right)} \tag{5.11b}$$

Die Bestimmung von Q und x über den hadronischen Endzustand wäre nach Gleichung 5.5 ebenso möglich, doch ist hier die Messung schwieriger. Teilchen des Endzustandes können aus verschiedenen Gründen – z.B. nicht-instrumentierte oder ineffiziente Bereiche des Detektors – unentdeckt bleiben. Beim Prozeß des geladenen Stroms bleibt jedoch nur die Möglichkeit, die Kinematik über die hadronische Methode zu rekonstruieren. Hier wird unter Austausch eines $W^{\pm}$ Bosons das Elektron in ein Neutrino umgewandelt, welches den Detektor ohne Signal verläßt. Man nennt die hadronische auch die Jaquet-Blondel Methode.

Es existieren weitere Rekonstruktionsmethoden, welche Kombinationen aus den Observablen von Elektron und hadronischem Endzustand benutzen. An dieser Stelle soll nicht weiter auf diese Methoden eingegangen werden; detailliertere Informationen finden sich in [BB95].

Im weiteren Verlauf dieser Arbeit wird die sogenannte $e\Sigma$-Methode angewendet. Die Größe Σ (siehe Gleichung 5.12a) ist unabhängig vom Energieverlust durch Teilchen, die den Detektor ungemessen durch das Strahlrohr in Vorwärtsrichtung verlassen. Dies hat zur Folge, daß die Kinematik auch berechnet werden kann, falls das Elektron zusätzlich Bremsstrahlung unterliegt. Die zugehörigen Gleichungen sind:

$$\Sigma = \sum_i (E_i - p_{z,i}) \tag{5.12a}$$

$$y_\Sigma = \frac{\Sigma}{\Sigma + E_{e'}(1-\cos\theta_{e'})} \tag{5.12b}$$

$$Q_\Sigma^2 = \frac{E_{e'}^2 \sin^2\theta_{e'}}{1-y_\Sigma} \tag{5.12c}$$

$$Q_{e\Sigma}^2 \equiv Q_e^2 = 4E_e E_{e'}\cos^2\frac{\theta_{e'}}{2} \tag{5.12d}$$

$$x_{e\Sigma} \equiv x_\Sigma = \frac{Q_\Sigma^2}{s y_\Sigma}. \tag{5.12e}$$

5.1.2 Jetalgorithmen

In Abschnitt 4.3.3 wurde der Software-Aspekt von Algorithmen zur Identifikation von Jets angesprochen. Es ging dabei um Verfügbarkeit und die Schnittstelle zu den verwendeten Datenstrukturen, ohne näher auf die physikalischen Eigenschaften von Jets einzugehen. In diesem Abschnitt soll kurz der Begriff Jet erklärt und die verfügbaren Algorithmen vorgestellt werden. Für eine ausführlichere Diskussion der Algorithmen sei auf [Nie97] bzw. [Had99] und die darin enthaltenen Referenzen verwiesen.

Der harte Subprozeß in der ep-Streuung läuft auf Partonebene ab, d.h. unter Beteiligung von quasifreien Quarks und Gluonen sowie dem mit dem Elektron ausgetauschten

Boson. Aufgrund der Eigenschaften der starken Wechselwirkung müssen sich im Endzustand Hadronen bilden. Auf diese Weise ergeben sich aus den wenigen, am harten Subprozeß beteiligten Partonen, eine ganze Reihe von Teilchen, welche im Detektor gemessen werden. Diese sind nicht gleichmäßig im Raum verteilt, sondern gruppieren sich zu Teilchenbündeln, Jets genannt. Die Teilchen werden durch Vierervektoren beschrieben und der Jetalgorithmus berechnet daraus neue Vierervektoren, welche die Teilchenbündel charakterisieren. Das Ziel ist, eine Korrelation zwischen den Jets des Endzustandes und den Partonen des harten Subprozesses herzustellen. Dabei muß ein Jetalgorithmus eine Antwort auf die folgenden Fragen finden:

1. Welche Objekte müssen im nächsten Schritt zusammengefaßt werden?
2. Auf welche Weise werden zwei Objekte zusammengefaßt (rekombiniert)?
3. Bei welcher Bedingung ist der Algorithmus beendet?

In der neuen Analyseumgebung stehen bisher objektorientierte Implementierungen des Jade- [JAD86] und Durham-Algorithmus [CDW92] zur Verfügung. Beide basieren auf der unterschiedlichen Berechnung der invarianten Masse aller möglichen Paarbildungen zweier Teilchen. Den Jade-Jetfinder gibt es dabei in zwei Varianten.

$$\begin{aligned}
\texttt{Jade}: \quad m_{ij}^2 &= 2E_iE_j\,(1-\cos\theta_{ij}) && (5.13a)\\
\texttt{JadeE}: \quad m_{ij}^2 &= (E_i+E_j)^2-(\vec{p_i}+\vec{p_j})^2 && (5.13b)\\
\texttt{Durham}: \quad m_{ij}^2 &= 2\min{(E_i,E_j)}^2\,(1-\cos\theta_{ij}) && (5.13c)
\end{aligned}$$

Die drei Algorithmen berechnen in einer Schleife die Massenquadrate für alle Paare von Teilchen. Als Referenzskala dient das Quadrat der Energiesumme des hadronischen Endzustandes: $M_{ref}^2 = E_{\mathrm{HFS}}^2$. Die beiden Teilchen mit dem kleinsten m_{ij} werden solange kombiniert, bis folgende Abbruchbedingung gilt:

$$y_{\mathrm{cut}} < \frac{m_{ij}^2}{E_{\mathrm{HFS}}^2}. \tag{5.14}$$

Hierbei ist y_{cut} ein Parameter der Jetdefinition. Dies bedeutet, daß weitere Resultate, die von den Jeteigenschaften abgeleitet werden, ebenfalls von der Wahl dieses Parameters abhängig sind. Gleiches gilt für die Wahl des Rekombinationsschemas. Dieses ist bei allen drei Varianten gleich, die Vierervektoren beider Teilchen werden einfach addiert. Andere Möglichkeiten schließen das explizite Zu-Null-Setzen der Masse ein, wobei entweder Energie- oder Impulserhaltung verletzt wird.

Um die Ergebnisse dieser Arbeit mit denen von [Had99] vergleichen zu können, ist der sogenannte “inklusive k_t-Algorithmus” notwendig, welcher im OO-Framework z.Z. nur durch einen Wrapper zugänglich ist. D.h. die bestehende `FORTRAN`-Routine wird durch eine Wrapper-Klasse benutzt, die für die Konvertierung zwischen den `FORTRAN`-Datentypen und den Klassen der neuen Analyseumgebung verantwortlich ist.

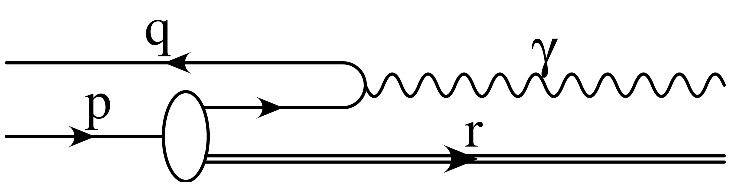

Abbildung 5.2: *Diagramm zum Prozeß des Quark-Parton-Modells im Breit-System.*

Der inklusive k_t-Algorithmus [Sey98] beruht ebenso wie Jade- und Durham-Algorithmus auf der iterativen Zusammenfassung von Teilchenpaaren. Die Entscheidung, welches Paar zusammengefaßt werden soll, wird jedoch rein aufgrund relativer Messungen von Transversalimpulsen gefällt. Dem liegt die Überlegung zugrunde, daß die Partonen, welche durch weiche Abstrahlung nach dem harten Subprozeß entstehen, eine geringe Energie senkrecht zum abstrahlenden Parton besitzen. Demzufolge sollten alle Hadronen, die aus demselben harten Parton entstanden sind, im Raum nahe beieinander liegen.

Das Abstandsmaß und Rekombinationsschema des inklusive k_t-Algorithmus lautet wie folgt:

$$R_{ij} = \sqrt{(\Delta\eta_{ij})^2 + (\Delta\varphi_{ij})^2} \tag{5.15a}$$

$$E_t = E_{t,i} + E_{t,j} \tag{5.15b}$$

$$\eta = (E_{t,i} + E_{t,j}) \left(\frac{\eta_i}{E_{t,i}} + \frac{\eta_j}{E_{t,j}} \right) \tag{5.15c}$$

$$\varphi = (E_{t,i} + E_{t,j}) \left(\frac{\varphi_i}{E_{t,i}} + \frac{\varphi_j}{E_{t,j}} \right) . \tag{5.15d}$$

Hierbei wird solange das Teilchenpaar mit dem geringsten R_{ij} anhand der Vorschriften aus Gleichungen 5.15b bis 5.15d zusammengefaßt, wie R_{ij} kleiner als 1 ist. Dieser Algorithmus kann zu mehreren niederenergetischen Jets führen, so daß eine minimale transversale Energie aller Jets gefordert wird. Auf diese Weise wird auch der Jet entfernt, welcher den Protonrest beinhaltet, da dieser keine Transversalenergie besitzt.

Die Identifizierung der Jets wird im sogenannten "Breit"-System durchgeführt. Dieses ist dadurch definiert, daß das am harten Subprozeß beteiligte Parton am ausgetauschten Boson um 180° reflektiert wird (siehe Abbildung 5.2).

Die Definition ist also dergestalt, daß

$$2xp^* + q^* = \begin{pmatrix} Q \\ 0 \\ 0 \\ 0 \end{pmatrix} . \tag{5.16}$$

Für die Vierervektoren des ausgetauschten Bosons q und ein- bzw. auslaufenden Partons

f bzw. f' bedeutet dies:

$$q^* = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -Q \end{pmatrix} \tag{5.17a}$$

$$f^* = \begin{pmatrix} \frac{Q}{2} \\ 0 \\ 0 \\ \frac{Q}{2} \end{pmatrix} \tag{5.17b}$$

$$f'^* = \begin{pmatrix} \frac{Q}{2} \\ 0 \\ 0 \\ -\frac{Q}{2} \end{pmatrix}. \tag{5.17c}$$

Größen im Breit-System werden mit einem $*$ markiert.

5.2 Datensatz

Neben kleineren Testproduktionen wurden für die gesamten Daten des Jahres 2000 die entsprechenden μODS und HAT Dateien erstellt. Sie entsprechen den DST Dateien von `CDST1.C0000001` bis `CDST1.C0001962` und stellen ein Datenvolumen von $65,9$GB bzw. $34,3$GB dar. Diesem liegt eine integrierte Luminosität von $56,8\ \mathrm{pb}^{-1}$ (unkorrigiert) zugrunde. Zum Vergleich dazu die Werte für die Jahre 1995 bis 1997: $3,8\ \mathrm{pb}^{-1}$, $7,9\ \mathrm{pb}^{-1}$ bzw. $21,3\ \mathrm{pb}^{-1}$ (korrigiert).

Ein vollständiger Satz von ODS Dateien für diese Menge an Daten ist bisher nicht vorhanden, die Analyse muß und kann sich rein auf Informationen aus μODS und HAT stützen.

Für das Jahr 2000 existieren noch keine Monte Carlo Simulationen, so daß auf die MC-Datensätze vergangener Jahre zurückgegriffen wird. Dies ist gerechtfertigt, da sich der Detektor in dem entsprechenden Zeitraum nicht gravierend verändert hat. Der Monte Carlo Aspekt des OO-Frameworks ist aber noch nicht ausreichend modelliert und somit nicht einsatzfähig. Da die im folgenden vorgestellte Analyse eine Wiederholung von Kapitel 5 aus [Had99] darstellt, werden die dort ermittelten Korrekturfaktoren für den inklusiven, differentiellen Zweijet-Wirkungsquerschnitt übernommen.

5.2.1 Vorselektion

Unter Vorselektion ist im weiteren Verlauf dieser Arbeit im allgemeinen das Selektieren von Ereignissen anhand der HAT-Variablen zu verstehen. Dies unterscheidet sich insbesondere

Größe	Kriterium
Trigger	S67 oder S75
Elektron	Detektiert
Vertex	Ein zentraler Vertex gefordert
Runqualität	"gut" oder "mittel"
Elektronenergie	$E_{e'} > 11$ GeV
Elektronwinkel	$5° < \theta_{e'} < 153°$
Elektronspur	$r_{\mathrm{DCA}} < 12$ cm
Photonvirtualität	$150\ \mathrm{GeV}^2 \leq Q^2_{e\Sigma} \leq 5000\ \mathrm{GeV}^2$

Tabelle 5.1: *Übersicht über die Schnitte der Vorselektion. An dieser Stelle ist Vorselektion in der herkömmlichen Weise zu verstehen, da die Eigenschaften des gestreuten Elektrons nicht auf dem HAT zur Verfügung stehen.*

von der Interpretation des Begriffes, die in den Analysen des alten Modells verwendet wird. Siehe dazu auch Abschnitt 2.2.

Die Messung tiefinelastischer Ereignisse wird unter anderem durch das Auffinden des gestreuten Elektrons in den Kalorimetern des H1-Detektors durchgeführt. In dieser Analyse werden nur die Ereignisse betrachtet, bei denen das Elektron im LAr Kalorimeter detektiert wurde, was einem Gebiet des Polarwinkels zwischen 5° und 153° entspricht. Zudem wird gefordert, daß die Trigger S67 oder S75 gesetzt sein müssen. Dies bedingt im wesentlichen eine Energiedeposition im elektromagnetischen Teil des Flüssig-Argon-Kalorimeters unter hohen Q^2 und E_t. Für Elektronen einer Energie größer als 11 GeV betrug die Triggereffizienz $\gtrsim 99.5\%$ in der Datennahme von 1994 bis 1997 [H1 99a]. Die Effizienz für die Daten des Jahres 2000 war zum Zeitpunkt dieser Arbeit noch nicht bestimmt. Da Detektor und Trigger im wesentlichen unverändert sind, wird hier keine Änderung erwartet und o.g. Wert übernommen. Zusätzlich soll für das gestreute Elektron der Abstand zwischen seinem Cluster und einer zugeordneten Spur nicht größer sein als 12 cm. Dieser Abstand wird als "**d**istance of **c**losest **a**pproach" (DCA) bezeichnet. Das Vorhandensein eines rekonstruierten Primärvertex wird gefordert, um Strahl-Untergrund zu unterdrücken. Im übrigen werden nur solche Runs akzeptiert, die als "gut" oder "mittel" eingestuft wurden. Dies bedeutet, daß die wichtigsten Detektorkomponenten zum jeweiligen Zeitpunkt eingeschaltet waren. Um die Ergebnisse dieser Arbeit mit denen von [Had99] vergleichen zu können, wird der Bereich der Photonvirtualität auf das Intervall $150\ \mathrm{GeV}^2 \leq Q^2 \leq 5000\ \mathrm{GeV}^2$ beschränkt. Die Verteilung von Q^2 nach der Selektion auf dem HAT zeigt Abbildung 5.3.

Nicht berücksichtigt werden Schnitte auf Bereiche des Detektors, an denen sich Segmente des Kalorimeters treffen. Dies liegt in der Tatsache begründet, daß in der derzeitigen Version der Umgebung keine Geometrieinformationen verfügbar sind.

Die Schnitte, die als Vorselektion auf dem HAT gewählt wurden, sowie weitere technische Schnitte sind in Tabelle 5.1 zusammengefaßt.

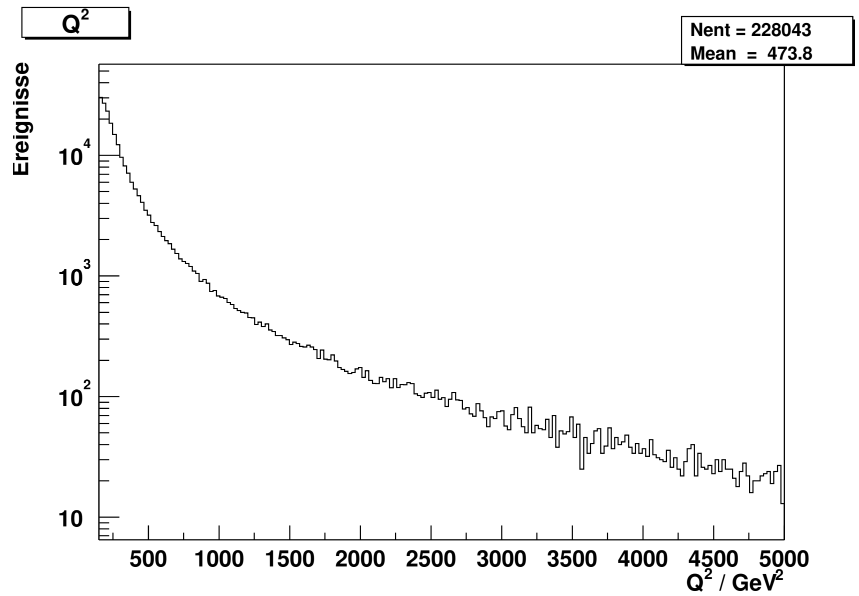

Abbildung 5.3: *Die Verteilung von Q^2 nach der Vorselektion auf HAT, d.h. alle Schnitte der Tabelle 5.1 außer denjenigen auf die Elektroneigenschaften wurden angewandt.*

5.2.2 Reduktion des Untergrundes

Strahl-Untergrund Ereignisse sollen unterdrückt werden. Sie kommen z.B. zustande durch Reaktionen eines Strahlteilchens mit dem Restgas im Strahlrohr. Um deren Anteil niedrig zu halten werden die Ereignisse verworfen, deren rekonstruierter Vertex außerhalb eines Bereiches von 35 cm in z-Richtung um den Ursprung des H1 Koordinatensystems liegt. Die zugehörige Verteilung zeigt Abbildung 5.4.

Weiterhin bilden solche Ereignisse einen Untergrund, die nicht aus dem Zusammentreffen zweier Teilchenbündel stammen. Die zentrale Jetkammer (CJC = **C**entral **J**et **C**hamber) mißt auch das Timing ihrer Hits. ep-Kollisionen finden bei HERA alle 96 ns statt, was 500 Einheiten der CJC-Auflösung entspricht. Mit dem Wissen, daß eine Teilchenkollision alle 500 Einheiten vorkommt, lassen sich die Zeitfenster für "gute" Ereignisse bestimmen. In Abbildung 5.4 ist zu sehen, daß der größte Teil aller Ereignisse beim Wert 450 liegt. Weitere Spitzen befinden sich im erwarteten Abstand bei -50 bzw. 950, der einer Kollision Unterschied entspricht.

Für die einlaufenden Teilchen gilt bei Vernachlässigung der Masse folgende Vierervektor-Summe

$$\begin{pmatrix} E_p \\ 0 \\ 0 \\ E_p \end{pmatrix} + \begin{pmatrix} E_e \\ 0 \\ 0 \\ -E_e \end{pmatrix} = \begin{pmatrix} E_p + E_e \\ 0 \\ 0 \\ E_p - E_e \end{pmatrix}. \tag{5.18}$$

Anders ausgedrückt ergibt sich als Summe über $E-p_z$ ein Wert von $2E_e = 55$ GeV, der wegen Erhaltung von Energie und Impuls ebenso für den Endzustand erwartet wird. Wenn

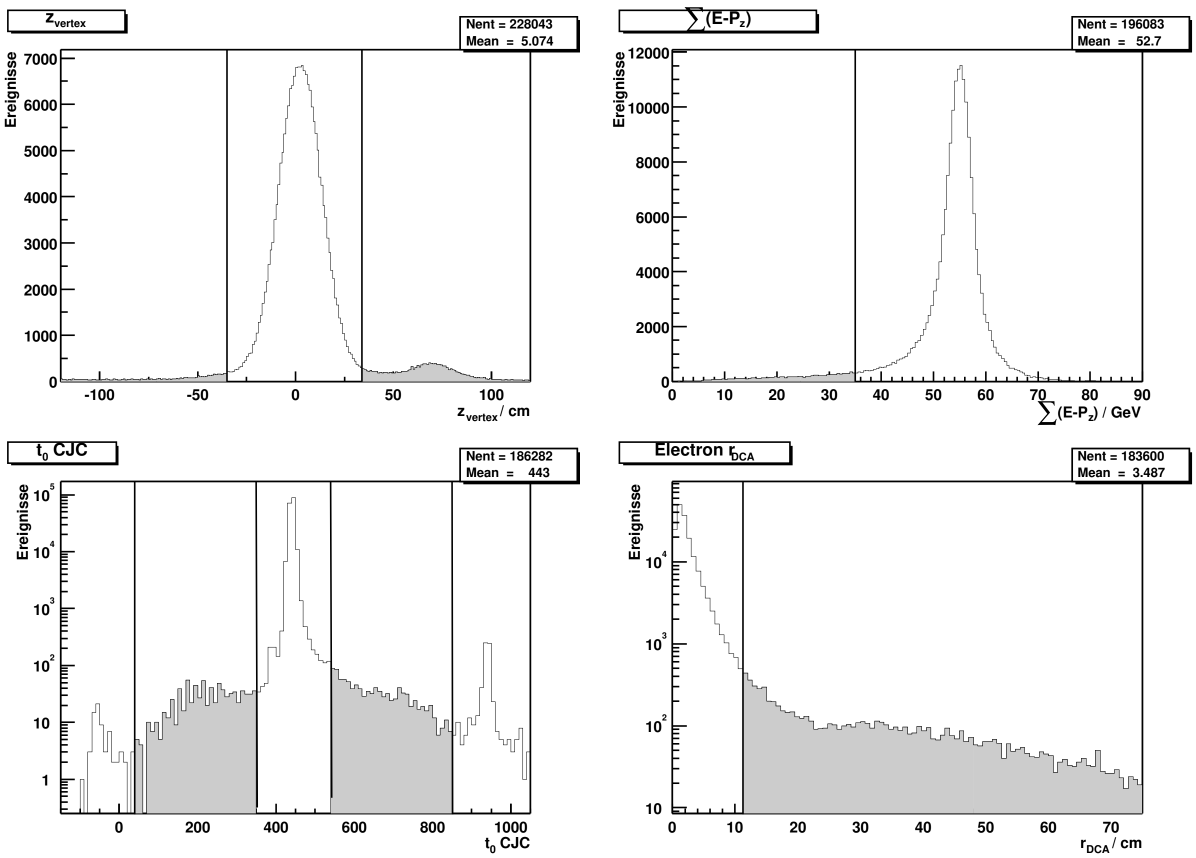

Abbildung 5.4: *Die grundlegenden Größen der Selektion. Oben links: die z-Position des Primärvertex, oben rechts: $E - p_z$ des Endzustandes, unten links: in der zentralen Jetkammer gemessenes Timing, unten rechts: Abstand zwischen Spur und Cluster des gestreuten Elektrons. Allen Verteilungen liegen folgende Schnitte zugrunde: Runqualität "gut" oder "mittel", Trigger S67 oder S75, vorhandener Primärvertex,* $150\ GeV^2 \leq Q^2_{e\Sigma} \leq 5000\ GeV^2$, $y_{e\Sigma} < 0,9$ *und ein identifiziertes Elektron. Zusätzlich wird von links oben nach rechts unten jeweils auf die vorangegangenen Größen geschnitten. Die Übersicht dieser Schnitte zur Reduktion des Untergrundes zeigt Tabelle 5.2.*

Größe	Kriterium	Bedingung
Inelastizität	$y_{e\Sigma} < 0,9$	
Inelastizität	$0,2 < y_{e\Sigma} < 0,7$	nur Zweijet-Ereignisse
z-Position des Vertex	-35 cm $< z_{\text{vertex}} <$ 35 cm	
Timing in CJC	$T_{0,\text{CJC}} = 450 \pm 100 \pm 1$ BC	(ein Bunchcrossing $\equiv 500$)
$E - p_z$ des Endzustandes	$\sum(E - p_z) > 35$ GeV	

Tabelle 5.2: *Zusammenfassung der Schnitte zur Reduktion von Untergrundereignissen.*

nun das Elektron im Strahlrohr bzw. toten Bereichen des Detektors verschwindet oder ein Photon im Anfangszustand abgestrahlt wird, erwartet man einen niedrigeren Wert. Durch die Forderung eines minimalen $E - p_z$ des Endzustandes werden solche Ereignisse entfernt, in denen das gestreute Elektron nicht detektiert wurde, ein anderes Teilchen aber ein elektronähnliches Signal produzierte. Die Verteilung von $E - p_z$ zeigt Abbildung 5.4.

Die Rekonstruktion der kinematischen Größen verschlechtert sich bei kleinen und großen Werten der Inelastizität y. Für Zweijet-Ereignisse ist dies besonders wichtig, da die kinematischen Informationen dort für eine Lorentztransformation in das Breit-System benutzt werden. Bei der Vorselektion auf HAT werden nur Ereignisse mit $y_{e\Sigma} < 0,9$ akzeptiert. Im Fall von Zweijet-Ereignissen wird der Bereich in $y_{e\Sigma}$ zusätzlich eingeschränkt, wie weiter unten beschrieben wird.

Die aus diesen Überlegungen stammenden Schnitte sind in Tabelle 5.2 zusammengefaßt.

Die Reihenfolge der Schnitte entspricht nicht dem, was in der alten Analyseumgebung angewendet würde. Wie in 2.2 beschrieben, werden grundlegende Schnitte beim Erstellen der Index-Files angewandt. Analysenspezifischere Schnitte kommen für die Erstellung der N-Tupel zum Einsatz. Das eigentliche Analyseprogramm liest anschließend zu jedem Ereignis alle Variablen ein. Dies ist im neuen Schema anders. Jetzt können Variablen, Objekte und Listen von Objekten einzeln gelesen werden. Um dies auszunutzen, kann nach jedem Lesevorgang auf die entsprechende Größe geschnitten werden. Wird das Ereignis aufgrund dieses Kriteriums verworfen, müssen die Daten späterer Schnitte nicht mehr gelesen werden. Das ist auch der Grund, warum in Abbildung 5.4 nicht alle Größen nach denselben Schnitten dargestellt sind. Sinnvoll ist es, solche Schnitte vorzuziehen, die bereits einen großen Teil der Ereignisse verwerfen. Die exakte Reihenfolge, in der die Schnitte vorgenommen wurden, zeigt Tabelle 5.3.

Abbildung 5.5 zeigt die Verteilung von Energie und Winkel des gestreuten Elektrons nach den Schnitten aus den Tabellen 5.1 und 5.2. Die Spitze der Energieverteilung wird bei einem Wert von $E_{e'} \approx E_e = 27,5$ GeV erwartet, da die Anzahl der Ereignisse in einem Q^2-Intervall mit $\sim 1/Q^4$ abnimmt. Der dominierende Anteil der Verteilung sind also Ereignisse mit kleinem Q^2, wo das Elektron wenig Energie überträgt. Ein kleines Q^2 bedeutet geringe Ablenkung von der ursprünglichen Richtung, ein großes Q^2 eine starke Ablenkung, was man auch besonders gut an der Verteilung des Polarwinkels sehen kann.

Ein Vergleich der Messung des Transversalimpulses des gestreuten Elektrons und des hadronischen Endzustandes ist in Abbildung 5.6 zu sehen. Im Anfangszustand laufen Elektron und Proton entlang der z-Achse, so daß auch im Endzustand eine verschwindende

Reihenfolge der Schnitte	
Runqualität	HAT
Vertex vorhanden	HAT
Elektron vorhanden	HAT
Trigger gesetzt	HAT
Photonvirtualität	HAT
Inelastizität	HAT
Vertexposition	HAT
$E - p_z$	HAT
Timing CJC	HAT
r_{DCA}	μODS
Energie und Winkel des Elektrons	μODS
Jetanzahl	Algorithmus

Tabelle 5.3: *Die Reihenfolge, in der die Schnitte durchgeführt wurden, unterteilt in Vorselektion (d.h. Erstellen einer Eventliste; hier oberhalb der Trennlinie) und Analyseprogramm. Dies unterscheidet sich wesentlich von der Vorgehensweise im alten Schema, z.B. werden Daten, die in späteren Schnitten abgefragt werden, gar nicht erst geladen, wenn ein Ereignis schon aufgrund eines früheren Kriteriums verworfen wurde. Die Parameter der Schnitte finden sich in Tabelle 5.1, 5.2 und 5.5.*

Summe über alle Transversalimpulse erwartet wird.

5.3 Erzielte Resultate

Abbildung 5.4 zeigt einige Verteilungen von Größen auf die aus technischen Gründen geschnitten wird (vor dem Schnitt). Die Position in z für den Primärvertex entspricht einem Gaußprofil, an welchem ein Fit durchgeführt wird. Tabelle 5.4 zeigt die Lage des Maximums und seine Breite. Das $E - p_z$ des Endzustandes ist nicht gauß-verteilt sondern besitzt ein abflachendes Verhalten zu kleinen Werten hin. Aus diesem Grund beschränkt sich der Fit auf einen engeren Bereich um das Maximum. Es herrscht eine gute Übereinstimmung mit dem erwarteten Wert von der zweifachen Elektronenergie (siehe auch Abschnitt 5.2.2). Die entsprechenden Zahlen zeigt Tabelle 5.4. Der Test von $E - p_z$ macht nur Aussagen über den longitudinalen Impuls des Endzustandes. Um den Transversalimpuls zu prüfen wird die relative Abweichung von $p_{t,\mathrm{HFS}}$ und $p_{t,e}$ betrachtet. Es wird erwartet, daß beide Anteile entgegengesetzt gleich groß sind. Die zugehörige Verteilung ist in Abbildung 5.6 zu sehen, das Ergebnis der Anpassungen von Gaußverteilungen an die o.g. Größen ist in Tabelle 5.4 zu finden.

An dieser Stelle ist es möglich, den inklusiven, differentiellen Wirkungsquerschnitt $\mathrm{d}\sigma(e^+p \to e^+X)/\mathrm{d}Q^2$ in dem gewählten Datensatz anzugeben. Dieser ist in Abbildung 5.7 zu sehen und stimmt gut mit dem in [H1 00b] bestimmten überein.

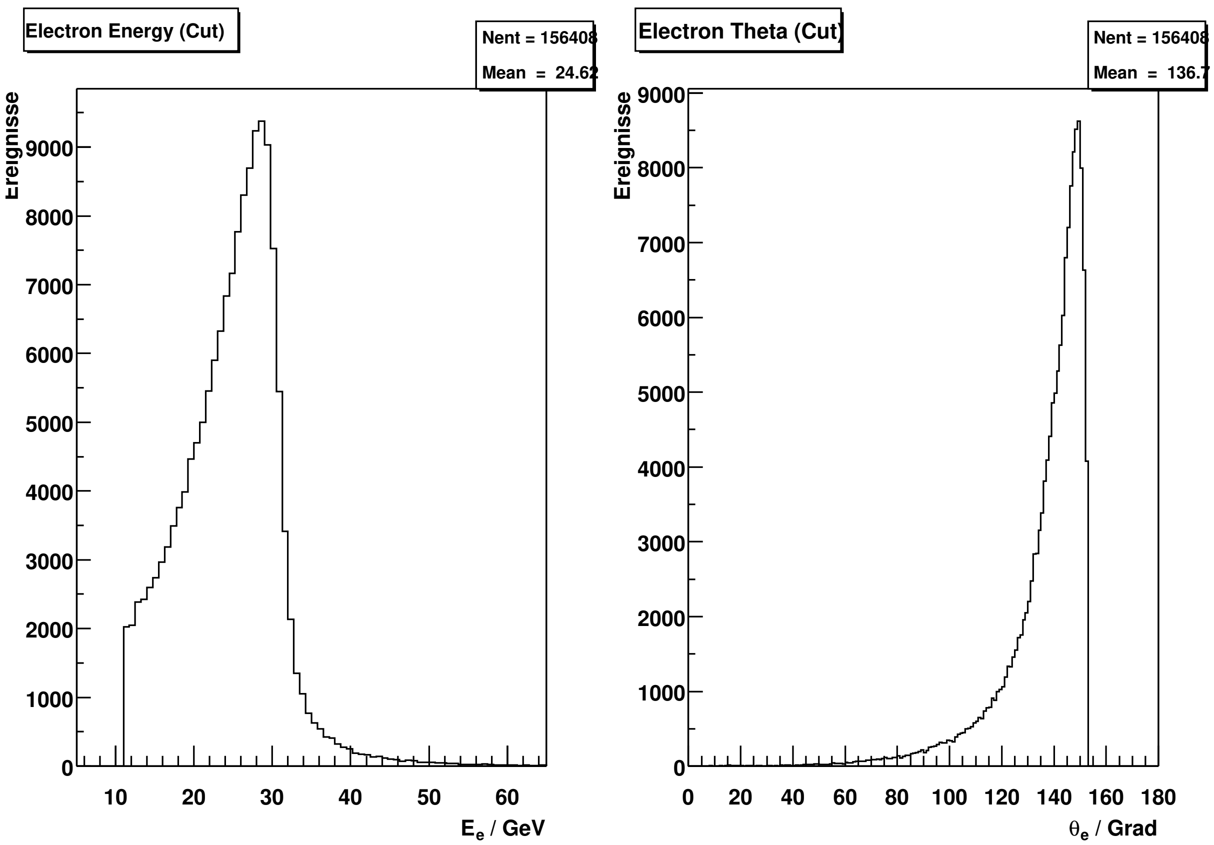

Abbildung 5.5: *Die Verteilungen der Elektroneigenschaften. Links: Energie, rechts: Polarwinkel. Das Maximum der Energieverteilung wird bei einem Wert von $E_{e'} \approx E_e = 27,5\ GeV$ erwartet, da die Anzahl Ereignisse in einem Q^2-Intervall mit $\sim 1/Q^4$ abnimmt. Die Verteilung wird also von Ereignissen mit kleinem Q^2 dominiert, wo wenig Energie des Elektrons übertragen wird. Dies sieht man auch besonders gut an der Verteilung des Polarwinkels: kleines Q^2 bedeutet wenig Ablenkung von der ursprünglichen Richtung, großes Q^2 eine starke Ablenkung. Die Schnitte der Tabellen 5.1 und 5.2 wurden für diese Abbildung angewandt.*

Größe	Bereich	Position des Maximums	Breite des Maximums
z_{vertex} [cm]	-30...30	1.96 ± 0.08	11.04 ± 0.07
$\sum(E - p_z)$ [GeV]	50...60	55.24 ± 0.03	3.13 ± 0.04
$p_{t,\text{bal}}$ [‰]	-400...400	-0.32 ± 0.01	1.86 ± 0.01

Tabelle 5.4: *Ergebnisse der Anpassungen von Gaußverteilungen an z_{vertex}, $\sum(E - p_z)$ und $p_{t,bal}$.*

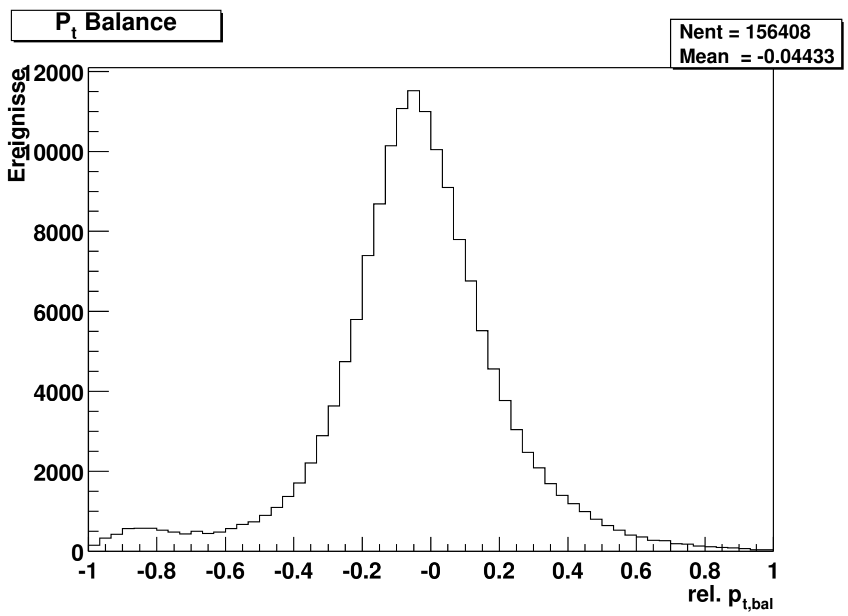

Abbildung 5.6: *Relative Abweichung des Transversalimpulses von Elektron und hadronischem Endzustand:* $(p_{t,HFS} - p_{t,e})/p_{t,e}$.

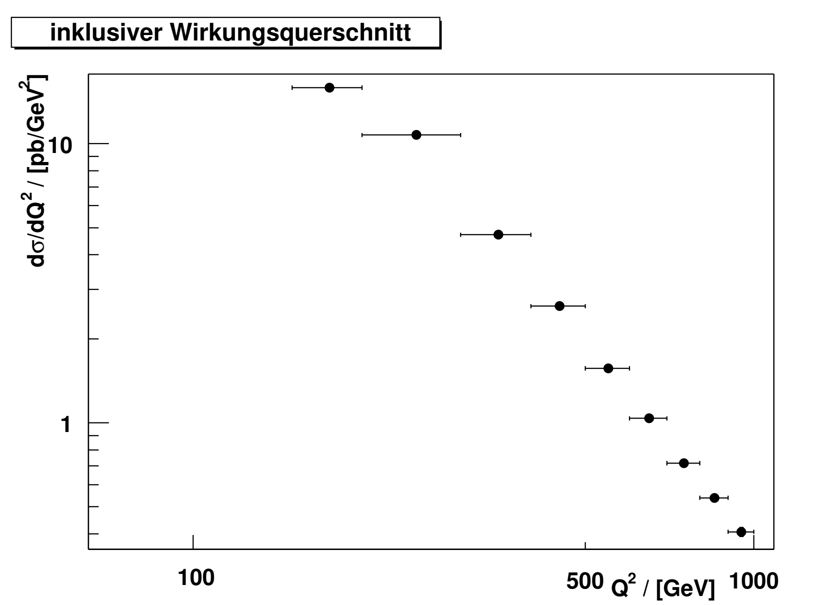

Abbildung 5.7: *Der inklusive Wirkungsquerschnitt* $d\sigma(e^+p \to e^+X)/dQ^2$ *für die Daten des Jahres 2000. Fehlerbalken für den statistischen Fehler und den Fehler in der Luminositätsbestimmung sind in dieser Auftragung nicht zu erkennen.*

Größe	Kriterium
$E^*_{t,\,\mathrm{jet}}$	> 5 GeV
E_{jet}	> 4 GeV
η_{jet}	$-1\ldots 2,5$
Anzahl der Jets	mindestens zwei
$E^*_{t,\,\mathrm{jet1}} + E^*_{t,\,\mathrm{jet2}}$	> 17 GeV

Tabelle 5.5: *Übersicht der Schnitte, die auf Zweijet-Ereignisse angewandt wurden.*

Datensatz	Ereignisse	Luminosität / [pb^{-1}]	Ereignisse/ pb^{-1}
2000er Daten	12973	48,389	268,1 $\pm$ 2.4 $\pm$ 4,0

Tabelle 5.6: *Anzahl der Zweijet-Ereignisse nach allen Schnitten in den Q^2 und x Bins, in denen der inklusive, differentielle Zweijet-Wirkungsquerschnitt angegeben wird. Für die normierte Zahl der Ereignisse ist der statistische Fehler angegeben. Die zweite Fehlerangabe entspricht einer Unsicherheit in der Luminositätsbestimmung von 1,5%.*

Für die Messung des inklusiven, differentiellen Zweijet-Wirkungsquerschnittes wurde der inklusive k_t-Algorithmus benutzt, der in Abschnitt 5.1.2 beschrieben wird. Die geforderte minimale Transversalenergie E^*_t im Breit-System wurde auf 5 GeV gesetzt. Die Suche nach Jets wurde auf das LAr Kalorimeter beschränkt, so daß die Pseudo-Rapidität der Jets in einem Bereich von $-1 < \eta_{\mathrm{jet}} < 2,5$ liegen muß. Dies entspricht einem Bereich des Polarwinkels zwischen 10° und 140°. Aus technischen Gründen, die in [Had99] beschrieben werden, wird noch gefordert, daß die Inelastizität $y_{e\Sigma}$ in einem Intervall $0,2 < y_{e\Sigma} < 0,7$ und die Jetenergie E im Laborsystem über einer Grenze von 4 GeV liegt. Anschließend muß für die beiden Jets mit dem höchsten E^*_t gelten, daß ihre Summe der Transversalenergie mindestens $E^*_{t,\,\mathrm{jet1}} + E^*_{t,\,\mathrm{jet2}} = 17$ GeV beträgt. Eine Übersicht der Schnitte, die auf Zweijet-Ereignisse angewandt wurden zeigt Tabelle 5.5, Effizienzen wurden nicht neu bestimmt.

In Abbildung 5.8 sind die Verteilungen der Jetgrößen E^*_t und η zu sehen. An dieser Stelle wurde noch nicht auf die Pseudo-Rapidität geschnitten, die übrigen Schnitte aus Tabelle 5.5 wurden jedoch angewandt. Die Unterteilung in ersten und zweiten Jet geschieht anhand der transversalen Energie E^*_t im Breit-System, wobei der erste Jet die höhere Transversalenergie besitzt.

Die Verteilungen von $E^*_{t,\,\mathrm{jet}}$ und η_{jet} nach allen Schnitten der Tabellen 5.1, 5.2 und 5.5 zeigt die Abbildung 5.9. Die Zahl der übrigbleibenden Ereignisse für diese Einstellungen zeigt Tabelle 5.6. Dabei werden nur die Zweijet-Ereignisse gezählt, welche auch innerhalb der Bereiche in Q^2 und x liegen, die für die Berechnung des inklusiven, differentiellen Zweijet-Wirkungsquerschnittes gewählt wurden. Wie sich zeigt, liegt die Zahl der Ereignisse pro pb^{-1} um einen Faktor 2,4 höher als in [Had99]. Dies bedeutet, daß die Jetidentifikation mit identischen Schnitten nicht zum gleichen Ergebnis kommt. An dieser Stelle wäre eine Überprüfung mittels Monte Carlo Simulationen nötig, die aber aus den in Abschnitt 5.2 angeführten Gründen nicht durchgeführt werden kann.

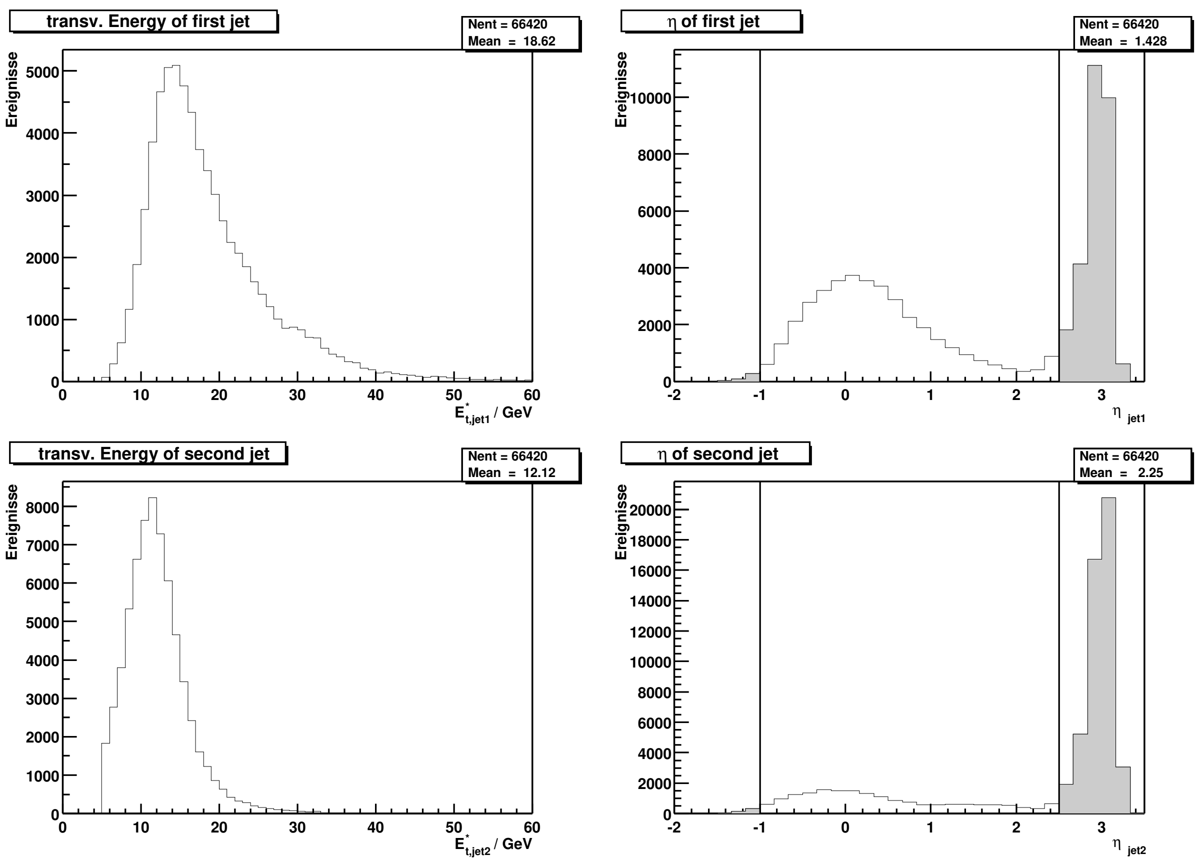

Abbildung 5.8: *Verteilung von E_t^* und η für die beiden energiereichsten Jets. An dieser Stelle wurden alle Schnitte der Tabellen 5.1 und 5.2 angewandt. Von den Zweijet-Schnitten der Tabelle 5.5 kamen alle bis auf den η-Schnitt zum Einsatz.*

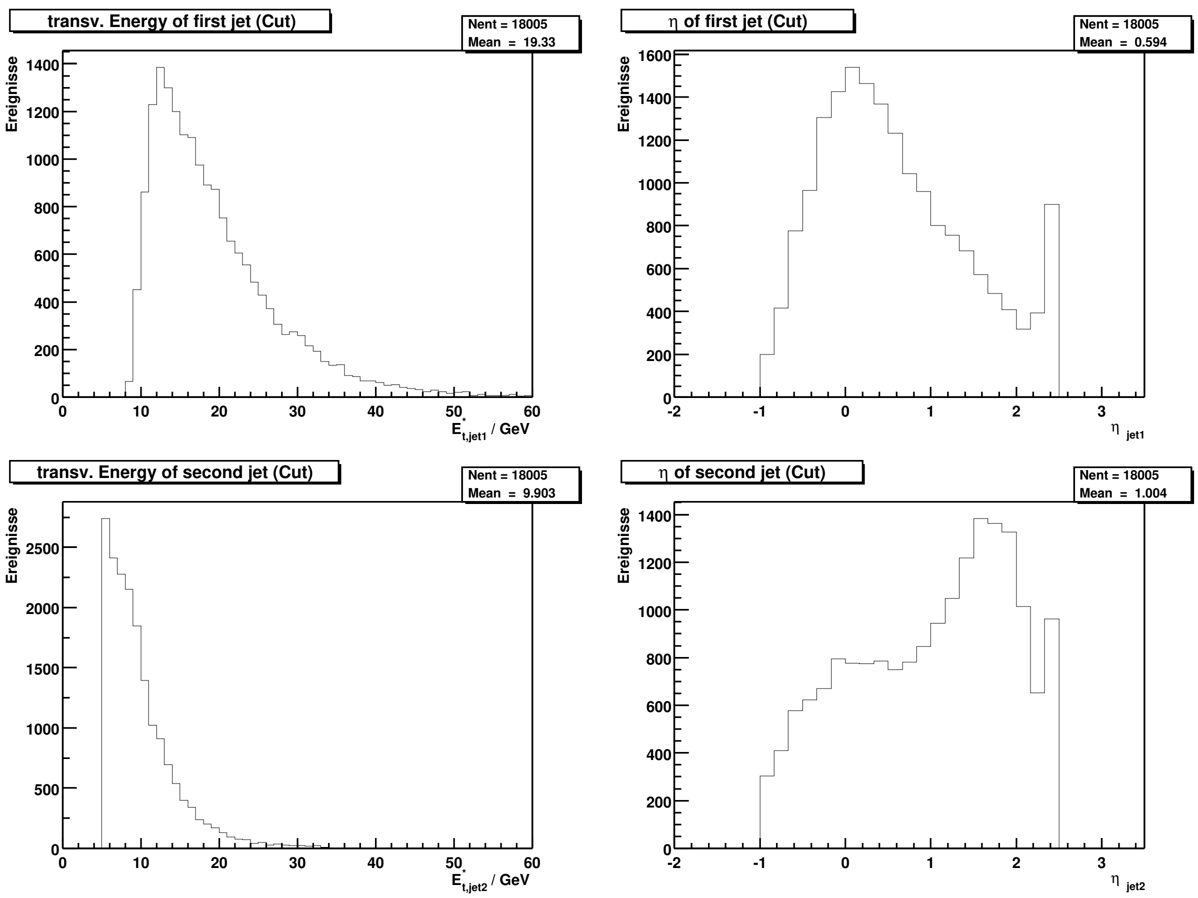

Abbildung 5.9: *Verteilung von E_t^* und η für die beiden energiereichsten Jets. An dieser Stelle wurden alle Schnitte der Tabellen 5.1, 5.2 und 5.5 angewandt.*

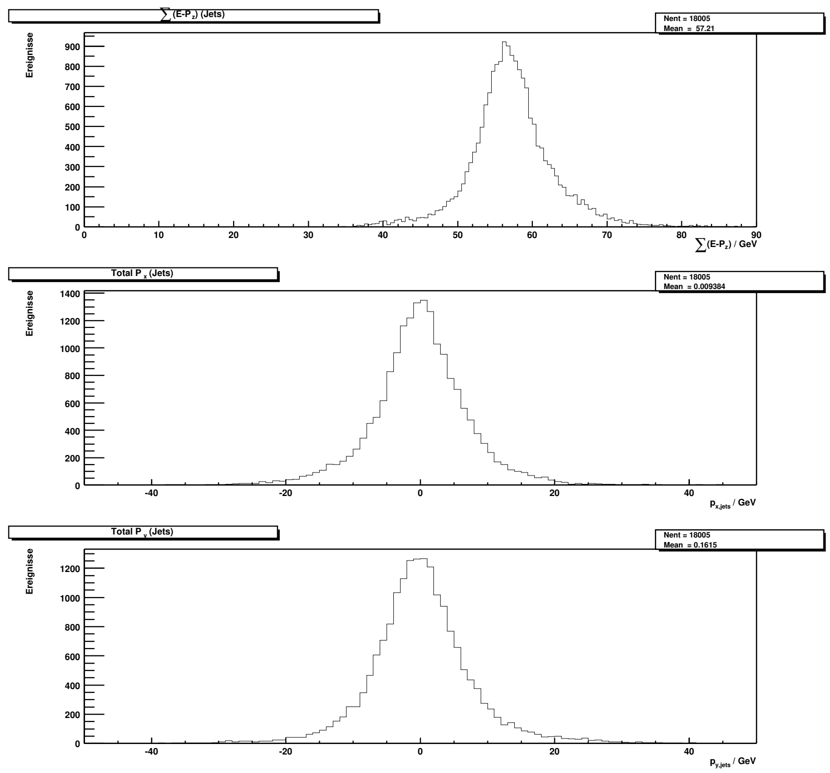

Abbildung 5.10: *Die Kinematik der gefundenen Jets, oben:* $\sum(E-p_z)$, *mitte:* p_x, *unten:* p_y.

Als zusätzlicher Test soll die Kinematik der gefundenen Jets betrachtet werden. In Abbildung 5.10 ist u.a. die Verteilung von $\sum(E-p_z)$ des gestreuten Elektrons und aller Jets zu sehen. Wie in Abschnitt 5.2.2 erklärt, wird ein Mittelwert von der zweifachen Energie des einlaufenden Elektrons erwartet. Das Maximum liegt hier leicht darüber. Die weiteren Verteilungen aus Abbildung 5.10 zeigen den Gesamtimpuls des gestreuten Elektrons und aller Jets in x- bzw. y-Richtung. Da die einlaufenden Teilchen keinen Transversalimpuls besitzen, wird hier in beiden Fällen eine Verteilung um Null erwartet. Dies ist hier der Fall. Aus diesem Grund kann ausgeschlossen werden, daß die Routinen zur Transformation zwischen Labor- und Breitsystem sowie der Jetalgorithmus grundlegende Fehler enthalten.

Nach allen Schnitten wird der Zweijet-Wirkungsquerschnitt in Bins von Q^2 und x bestimmt. Die Aufteilung erfolgt in drei Q^2-Bins, von denen jedes vier x-Bins enthält. Die unkorrigierten Daten sind in Abbildung 5.11 zu sehen, die mit den Faktoren aus [Had99] versehenen, korrigierten Daten zeigt Abbildung 5.12. Der berechnete Wirkungsquerschnitt liegt um einen Faktor $\approx 2,4$ oberhalb der Messung aus [Had99].

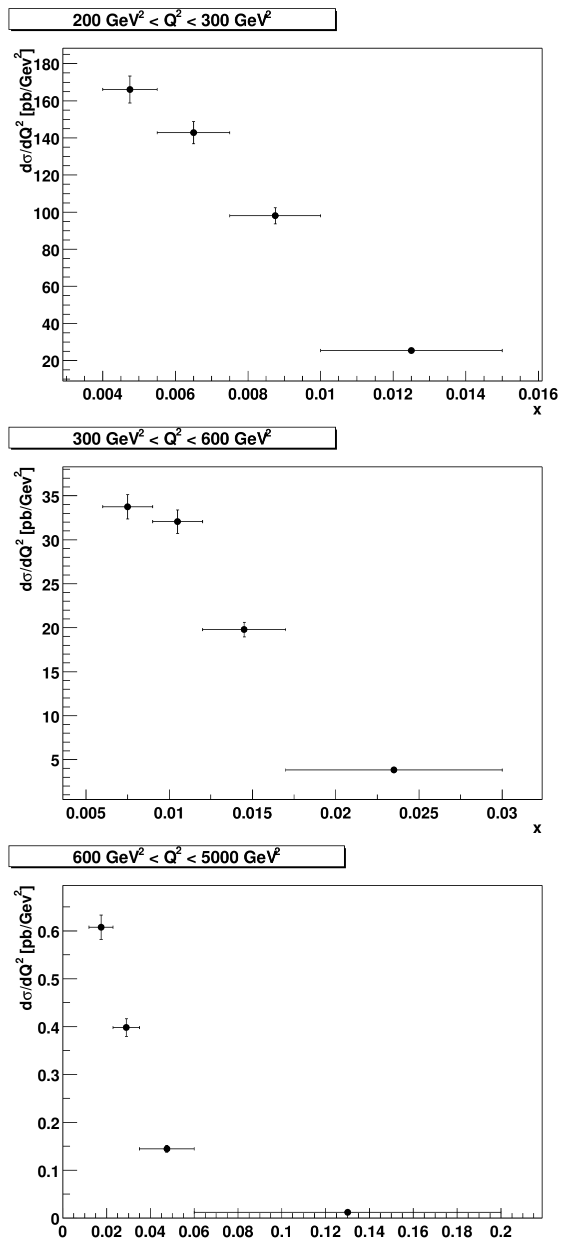

Abbildung 5.11: *Unkorrigierter Zweijet-Wirkungsquerschnitt für die Daten des Jahres 2000 in Bins von Q^2 und x. Die Fehlerbalken geben den statistischen Fehler und den Fehler in der Luminositätsbestimmung an.*

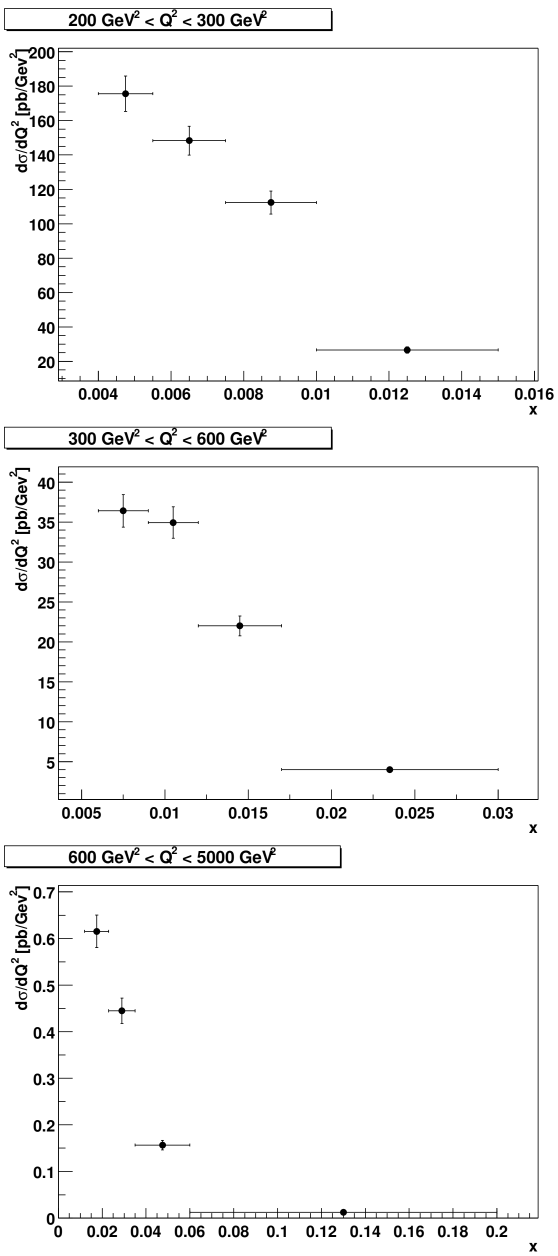

Abbildung 5.12: *Korrigierter Zweijet-Wirkungsquerschnitt für die Daten des Jahres 2000 in Bins von Q^2 und x. Die Fehlerbalken geben den statistischen Fehler und den Fehler in der Luminositätsbestimmung an.*

Kapitel 6

Ergebnisse der Tests

Dieses Kapitel soll in einem ersten Abschnitt die technischen Resultate zusammenfassen, d.h. eine Übersicht bieten, wieviel Zeit und Plattenplatz für die Erstellung von μODS und HAT benötigt wird, wie lange Vorselektionen auf HAT dauern und in welcher Zeit eine Analyse wie in Kapitel 5 vorgestellt über den Datensatz eines gesamten Jahres laufen kann. In einem anschließenden Abschnitt wird auf die Mängel eingegangen, die das neue Modell noch hat, d.h. welche Funktionalität nicht implementiert ist, an welchen Stellen geplante Ziele noch nicht erreicht sind und wo die Bedienung vereinfacht werden muß.

6.1 Technische Resultate

Auch wenn durch die zusätzlichen Datenlagen einzelne Analysen einen Zeitgewinn davontragen werden, bedeutet das Erstellen von μODS und HAT einen Mehraufwand gegenüber dem alten Schema, in dem nur DSTs existieren. Tabelle 6.1 zeigt den Zeitaufwand, den das Erstellen eines Paares von μODS und HAT Dateien benötigt. Bei insgesamt 356 Dateien pro Lage für das Jahr 2000 summiert sich dies auf ca. 96 Tage Gesamtrechenzeit. Durch paralleles Verarbeiten läßt sich dies im Idealfall unter Auslastung der kompletten Rechnerfarm auf ca. einen Tag real verstrichener Zeit reduzieren.

Menge	Bedarf an Rechenzeit (ein PC)
Ein Paar μODS/HAT	$\approx 6\frac{1}{4}$h
Gesamte Daten 2000	$\approx$ 96d

Tabelle 6.1: *Zeitbedarf zum Erstellen von μODS/HAT für ein Paar von Dateien und den gesamten Datensatz des Jahres 2000. Mehrere Dateien können gleichzeitig bearbeitet werden, so daß sich der reale Zeitverbrauch auf etwa einen Tag für die gesamten 2000er μODS/HAT Dateien reduziert. Die Zahlen beziehen sich auf die mittlere Rechenleistung eines PCs der H1-Rechnerfarm, die aus insgesamt 43 Maschinen besteht. Diese sind mit jeweils zwei CPUs zwischen Pentium II 266 MHz und Pentium III 800 MHz sowie 128 MB oder 256 MB RAM ausgestattet. Die genaue Rechnerzusammensetzung der Farm zeigt Tabelle 6.2.*

Menge	CPU Typ	Taktfrequenz / [MHz]	Hauptspeicher / [MB]
11	2 × Pentium II	266	128
1	2 × Pentium II	400	256
21	2 × Pentium III	500	256
10	2 × Pentium III	800	256

Tabelle 6.2: *Übersicht über die zur Verfügung stehenden Ressourcen der Rechnerfarm. Diese liegen den Werten der Tabelle 6.1 zugrunde. An die Rechnerfarm geschickte Programme werden abhängig von der Auslastung auf diese Maschinen verteilt.*

Anzahl Dateien	Zeit/[s], interpretiert	Zeit/[s], kompiliert	Faktor
4	145,74	142,41	1,023
20	396,59	379,78	1,044
342	—	6357,20	—

Tabelle 6.3: *Vergleich der Zeiten, welche die Vorselektion aus Kapitel 5 im interpretierten und im kompilierten Fall benötigt.*

Dieser Bedarf an Rechenleistung entspricht laut [Had01] in etwa dem Zeitaufwand für die Selektion und Erstellung eines N-Tupels der Arbeit [Had99]. Der Unterschied ist, daß für μODS und HAT die Verantwortung nicht mehr nur bei einer Person liegt. Dies bedeutet, daß der Nutzen der gesamten Kollaboration zugute kommt und das Expertenwissen an einer Stelle zusammenfließt. Jede Analyse im neuen Schema muß zwar immer noch eine eigene Vorselektion auf dem HAT machen. Doch wird in diesem Schritt nicht mehr der gesamte Datensatz angefaßt, wie das bei der Erstellung von Index-Files bzw. N-Tupeln der Fall ist, wo in beiden Fällen auf DSTs zugegriffen wird. Tabelle 6.3 zeigt einen Vergleich des Zeitbedarfs, den die Selektion aus Kapitel 5 im interpretierten und im kompilierten Modus benötigt. Der Unterschied ist sehr gering, da der entscheidende Faktor das Lesen der Dateien ist. Der Wert der dritten Spalte entspricht in etwa der Selektion über die gesamten Daten des Jahres 2000. Es ist also möglich, innerhalb von weniger als zwei Stunden alle Ereignisse eines Jahres aufzufinden, die Schnitten auf fünf verschiedenen Größen genügen. Diese Selektion kann jedesmal neu gemacht werden, aber wie die Zeiten in Tabelle 6.3 andeuten, ist es schon bei einem geringen Datenvolumen sinnvoll, die Selektion einmal zu machen und dann abzuspeichern.

Von der technischen Seite ist auch interessant, welchen Platzbedarf die zusätzlichen Datenlagen besitzen. Wie bereits in Kapitel 2 erwähnt, ist die durchschnittliche Größe eines Ereignisses auf DST 22 kB. Die Dateien der neuen Analyseumgebung werden standardmäßig mit der "Stärke" 2 komprimiert, wobei es sich um einen verlustfreien Algorithmus handelt. Damit ergeben sich für die Größe eines Ereignisses in den drei Ebenen die Werte wie in Tabelle 6.4. Die Designwerte für die die durchschnittliche Größe pro Ereignis werden bis auf ODS sogar von den Spitzenwerten unterboten. Im Vergleich mit den DSTs zeigt sich, daß ein Ereignis auf ODS im Mittel nur etwa 3/4 der Größe benötigt. Für μODS und

Typ	geplante Größe (⌀)	reale Größe (max)
ODS	15kB/ev	$< 15,8$kB/ev
μODS	3kB/ev	$< 1,3$kB/ev
HAT	0,5kB/ev	$< 0,4$kB/ev

Tabelle 6.4: *Geplante und reale Größen von ODS, μODS und HAT pro Ereignis.*

Typ	Gesamtgröße	Größe pro Datei (⌀)
μODS	65,9GB	184,9MB
HAT	34,4GB	96,4MB

Tabelle 6.5: *Gesamtdatenvolumen, das die μODS und HAT Dateien für das Jahr 2000 in Anspruch nehmen.*

HAT läßt sich auch der gesamte Platzbedarf für das Jahr 2000 angeben (siehe Tabelle 6.5). Das heißt, die Analyse dieser Arbeit stützt sich auf ein Gesamt-Datenvolumen von etwa 100GB.

Das Analyseprogramm, welches für die Ergebnisse in Kapitel 5 benutzt wurde, benötigte auf einem Farmrechner mit Dual-Pentium III 500 MHz und 256 MB RAM für die kompletten Daten des Jahres 2000 ca. 8 Stunden realer Zeit. Dies schließt alle Schritte ein, beginnend bei der Vorselektion auf HAT.

Es soll nicht unerwähnt bleiben, daß mit dem neuen Analyseframework alle Schritte einer Analyse auch auf Desktop PCs durchgeführt werden können. Deren Zahl hat sich in den letzten Jahren drastisch erhöht und in den meisten Fällen die X-Terminals ersetzt. Dies bedeutet ein Plus an Rechenkapazität, die nicht vergeudet wird, weil im Gegensatz zu früher alle Maschinen nun vollen Zugriff auf die zentralen Festplatten haben.

6.2 Bewertung

Wie in Kapitel 5 gesehen, ist es bereits jetzt möglich, grundlegende Aspekte einer Datenanalyse bei H1 mit dem OO-Framework durchzuführen. Dieser Abschnitt soll auf die Stellen eingehen, an denen in der für diese Arbeit zur Verfügung stehenden Umgebung noch Mängel oder Lücken bestehen.

6.2.1 Defizite in der Implementierung

Das H1-OO-Framework befindet sich zur Zeit dieser Arbeit in einem Misch-Stadium, d.h. die zur Verfügung stehende Funktionalität wird nicht allein durch neu programmierte Software erreicht, sondern zu einem großen Teil durch den Aufruf alter `FORTRAN`-Routinen. Dies betrifft vor allem das Füllen der neuen Strukturen mit physikalischen Inhalten. Nahezu alle Berechnungen zur Erzeugung von μODS und HAT laufen über bestehende H1-Software.

Implementierungs-Defizite
• Die Struktur der zusätzlichen Datenlagen μODS und HAT ist noch nicht im endgültigen Zustand. • Die Trennung des Quelltextes in Pakete für Datenklassen, Struktur der Datenspeicherung und Füllen der Datenstrukturen ist noch nicht ausreichend. • Ein großer Teil der Algorithmen zum Füllen des physikalischen Inhalts ist noch nicht auf `C++/OO` umgestellt.

Tabelle 6.6: *Defizite in der Implementierung, die für diese Arbeit benutzt wurde.*

Das hat auch zur Folge, daß noch nicht die endgültige Produktionskette läuft, in der die höheren Ebenen der Datenspeicherung auf den Informationen des ODS aufbauen. Relationen zwischen ODS und μODS sind aus diesem Grunde ebenfalls noch nicht möglich.

Was die Strukturen auf μODS und HAT angeht, besteht noch Änderungsbedarf. Der jetzige Aufbau ähnelt zu einem Teil – hauptsächlich den Detektorobjekten – noch zu sehr dem Konzept der N-Tupel, in denen alle Information für eine Analyse enthalten sind. Dies bedeutet, daß einerseits Informationen an Stellen stehen, wo sie vom Konzept her nicht hinpassen (z.B. Vierervektoren auf HAT, komplette Spur- bzw. Clusterinformation auf μODS). Zum zweiten werden fehlende Relationen temporär durch Duplizieren von Information kompensiert. In diesem Rahmen ist auch zu erwähnen, daß die Teilchenklassen einer weiteren Überarbeitung unterzogen werden, um allen Anforderungen der physikalischen Arbeitsgruppen gerecht zu werden.

Für Physiker, die nicht schon länger Teil des OO-Projekts sind, ist es zum jetzigen Zeitpunkt noch schwer, eigene Beiträge zu leisten. Die Paketstruktur wie sie jetzt besteht trennt nicht sauber genug zwischen den Klassen, welche die physikalischen Informationen beinhalten sollen, denen, die die Struktur der Datenspeicherung festlegen, und den Algorithmen/Programmen, welche die Daten in die Klassen füllen. Ziel der näheren Zukunft ist es, die Paketstruktur zu vereinfachen und ein klares Interface zu definieren, welches das Hinzufügen neuer Funktionalität auch für Nicht-Experten der Arbeitsgruppen erleichtert.

Die vollständige Bereitstellung aller Funktionalität durch reine OO-Mittel ist eine längerfristige Aufgabe. Im neuesten Release für Entwickler wurde zu diesem Zweck mit der Neustrukturierung der μODS- und HAT-Klassen sowie der Aufteilung der Software in besser definierte Pakete begonnen. Ein einheitliches Schema für das Erzeugen aller drei Datenlagen wurde modelliert, um eine stärkere Trennung zwischen Algorithmen und Datenstrukturen zu erreichen. Sobald hier ein stabiler Zustand erreicht ist, sind die Probleme, die in den letzten beiden Absätzen beschrieben wurden, weitestgehend ausgeräumt.

6.2.2 In der Entwicklung befindliche Konzepte

Ein wesentlicher Teil des OO-Frameworks fehlt z.Z. noch: die Einbeziehung von generierten und simulierten Monte Carlo Ereignissen. Diese werden bisher ebenfalls im DST Format

In der Entwicklung befindliche Konzepte
• Die Einbindung von Monte Carlo Simulationen ist noch nicht ausreichend modelliert. • Es ist noch kein Mechanismus zur Steuerung ausführbarer Programme verfügbar. • Es existiert noch kein generisches Analyse-Gerüst. • Das Benutzen eines zu den offiziellen Datenlagen parallelen User-Trees ist noch nicht integriert.

Tabelle 6.7: *Zum Zeitpunkt dieser Arbeit in der Entwicklung befindliche Konzepte.*

gespeichert und können prinzipiell analog zu den Daten-DSTs in ODS, μODS und HAT konvertiert werden. Doch fehlen an dieser Stelle im jetzigen Design von μODS und HAT die Klassen für Generatorinformationen. Der wesentliche Unterschied zu den Daten – zusätzliche Information auf Hadron- und Parton-Niveau – wäre momentan nicht zugänglich. Durch das Fehlen von Monte Carlo Informationen im reinen OO-Framework ist es z.Z. nicht möglich, Korrekturfaktoren für die Daten zu berechnen. Somit sind nur unkorrigierte Ergebnisse möglich. Es bleibt auch noch zu klären, ob eine Produktion von HAT im Falle von Monte Carlos überhaupt sinnvoll ist, da dort meist keine Selektionen nötig sind, was den Nutzen des HAT-Konzeptes in diesem Fall auf ein Minimum reduziert. Ein Weglassen würde aber bedeuten, daß in einer Analyse zwei Programme benutzt werden müssen: je eines für Daten und Monte Carlo. Dies erhöht die Fehleranfälligkeit und widerspricht dem Konzept, die Analyseumgebung bei H1 zu vereinheitlichen.

Bei der Nutzung der neuen Analyseumgebung gibt es noch mehrere Schwachstellen, welche die Bedienung erschweren. So gibt es z.B. noch kein generelles Konzept, ausführbare Programme zu steuern. Bei den bisherigen Analysen wurde dies durch Textdateien erreicht, welche beim Start eines Programmes als dessen Standardeingabe übergeben wurden. Dies benutzt die `FORTRAN`-Routine FPARM(5). Eine Möglichkeit wäre, dies weiter zu benutzen und an das OO-Framework anzupassen. Es bleibt zu testen, ob die dafür nötige Arbeit nicht besser in ein komplett neues Schema gesteckt werden sollte. So ist es z.B. möglich, aus einem kompilierten ROOT-Programm normale `C++`-Makros aufzurufen. Eine weitere Alternative wäre, auf Mittel zurückzugreifen, die sich "normaler" `C/C++`-Standardfunktionen bedienen, wie z.B. dem `getopt`-Paket zum Parsen von Kommandozeilen-Optionen. Eine Entscheidung für eines dieser Systeme steht noch aus. Im derzeitigen Zustand muß sich ein Benutzer selbst um solche Dinge kümmern.

Weiterhin gibt es noch kein allgemeines Gerüst für eine Analyse. Das heißt jeder Benutzer schreibt von Grund auf eine Schleife, die über alle Ereignisse läuft, welche dem `H1Tree` übergeben wurden. Diese wird mit spezifischen Anweisungen gefüllt. Dabei wurde der Zugriff auf die Daten im Laufe der Zeit vielfach komfortabler und transparenter gestaltet. Neuere Releases bieten bereits ein einfaches Konzept, wie in einer Analyse auf die Daten zugegriffen werden kann. Ein Erzeugen eines sogenannten "User-Trees", der parallel zu ODS, μODS und HAT für vom Benutzer berechnete Größen genutzt werden kann, ist dagegen noch nicht in einem einsatzfähigen Zustand.

Das H1-OO-Projekt ist noch relativ jung verglichen mit der bisher benutzten Software. Neue Funktionalität wird häufig eingeführt, bestehende Teile werden permanent überarbeit. In solch einer Situation ist es zwangsläufig so, daß die vorhandene Dokumentation nicht mit den Weiterentwicklungen Schritt hält. Trotzdem wurde im H1-OO-Projekt von Beginn an Wert auf zügige Bereitstellung von Dokumentation gelegt. Es macht ebenso wenig Sinn, die Software in diesem Stadium immer wieder auf Geschwindigkeit zu optimieren, so daß auch noch keine Aussagen über einen Leistungsvergleich von alter und neuer Software gemacht werden können. Zuletzt gibt es noch wenig Erfahrung mit der Rückwärtskompatibilität und dem Geschwindigkeitsverlust, wenn alte Software auf zurückkonvertierten `BOS`-Bänken eingesetzt wird.

Kapitel 7

Zusammenfassung und Ausblick

Diese Arbeit ist im Rahmen des H1-OO-Projektes entstanden, dessen Ziel es ist, eine neue, auf C++ und ROOT basierende Analyseumgebung bereitzustellen. In dieser kommt ein überarbeitetes Datenmodell zum Einsatz, welches den gesteigerten Anforderungen heutiger Analysen Rechnung trägt. Dem alten Datenmodell werden zusätzliche Ebenen höherer Abstraktion und geringeren Platzbedarfs pro Ereignis hinzugefügt. Ausgehend vom Object Data Store (ODS), das den bisher gebräuchlichen DSTs entspricht, werden Informationen auf Teilchenniveau im Micro Object Data Store (μODS) soviel eine Vielzahl von Ereignisvariablen im H1 Analysis Tag (HAT) gespeichert. In diese zusätzlichen Ebenen fließt das Expertenwissen der physikalischen Arbeitsgruppen ein, so daß Standardberechnungen mit einheitlichen Parametern zentral ausgeführt und verwaltet werden können. Diese Ergebnisse stehen dann der gesamten Kollaboration zur Verfügung. Der HAT kann für schnelle Ereignisselektion genutzt werden, was das Konzept der Index-Files des alten Frameworks ersetzt. Zusammen mit den bereits vorberechneten Informationen auf μODS wird es in der neuen Analyseumgebung weit weniger häufig nötig sein, auf den vollen Datensatz eines Ereignisses zuzugreifen.

Von der Softwareseite gesehen werden die verschiedenen Programmiersprachen, die bisher zum Einsatz gekommen sind, durch nur noch eine einzige, moderne Sprache ersetzt: C++. Auf diese Weise läßt sich Quelltext leichter wiederverwenden und von Programmen einzelner Benutzer in offizielle H1-Software integrieren. Damit fällt auch die Barriere zwischen den privaten N-Tupeln und den H1-weiten Daten, denn bisher war es nicht möglich, einem einmal erzeugten N-Tupel weitere Informationen hinzuzufügen, ohne erneut über den vollen Datensatz auf DST zu laufen. Im neuen Schema dagegen kann jede Information aus allen drei Datenlagen einzeln eingelesen werden. Dies bedeutet, daß nur die Teile eines Ereignisses transferiert werden müssen, die wirklich gebraucht werden. Und weil eine Verbindung zwischen den Objekten auf ODS, μODS und HAT besteht, kann bei Bedarf zu jedem Zeitpunkt direkt auf die weniger abstrakten Informationen unterer Ebenen zugegriffen werden.

Zu Beginn dieser Arbeit befand sich das H1-OO-Projekt noch in der Planungsphase. Klassen, die über eine "eins zu eins" Kopie der bisherigen BOS-Bänke hinausgehen, wurden mit der in Kapitel 3 beschriebenen CRC-Methode entwickelt. Dabei handelt es sich um Spuren, Cluster und Teilchen, wobei letztere den Schwerpunkt dieser Arbeit darstellen.

Im weiteren Verlauf des Projekts wurde die technische Struktur von μODS und HAT festgelegt sowie das Programm geschrieben, das diese beiden Datenlagen – noch aus DST-Informationen – füllt. Hilfsmittel wie eine H1Pointer Managementklasse mußten eingeführt werden, um die geplanten Merkmale bereitstellen zu können. Erst als die technischen Aspekte im Griff waren, wurde die neue Software für H1 offiziell zur Benutzung freigegeben und die Arbeitsgruppen zu Rückmeldungen aufgefordert. Diese beinhalteten sowohl physikalische Verbesserungen wie auch Anregungen zur Benutzerfreundlichkeit, welche in den weiteren Entwicklungsverlauf eingeflossen sind und einfließen.

Der in Kapitel 5 vorgestellte, physikalische Teil dieser Arbeit ist eine konkrete Anwendung der zu diesem Zeitpunkt verfügbaren OO-Mittel. Das Projekt zum Ende dieser Arbeit ist noch nicht in dem Zustand, in dem alle Analysen wie gewohnt durchgeführt werden können, wenn auf die Verwendung bestehender H1-Software verzichtet werden soll. Vor allem die noch nicht vollständige Integration von Monte Carlo Simulationen in der neuen Analyseumgebung lassen keine echten Ergebnisse zu, da keine Korrekturen an die Daten angebracht werden können.

Es konnte jedoch gezeigt werden, daß die prinzipiellen Funktionen vorhanden sind, wobei das Hauptaugenmerk darauf zu richten ist, mehr physikalische Algorithmen in `C++/OO` bereitzustellen. In einer nächsten Phase des Projektes muß dazu übergegangen werden, die zusätzlichen Datenlagen schrittweise aus ODS zu erzeugen. Nur so ist es möglich, Relationen zwischen μODS und ODS herzustellen. Dies ist ein wichtiger Punkt zum Erreichen voller Funktionalität. Ein Schritt in diese Richtung wurde mit dem neuesten Entwickler-Release getan, in dem die OO-Software in klarer definierte Pakete aufgeteilt wurde und eine logischere Trennung zwischen Datenstrukturen und Füllalgorithmen stattgefunden hat. Erste Tests des Erzeugens von μODS und HAT aus ODS haben den technischen Grundstein für den Übergang zu rein objektorientierter Software gelegt.

Von Seiten der Software-Entwicklung gesehen hat das Projekt mit der Einführung der in Kapitel 3 beschriebenen Release-Strategie in Verbindung mit CVS einen großen Schritt nach vorne getan. Die Verwaltung mittels CVS läßt mehrere Entwickler am selben Quelltext arbeiten und führt deren Änderungen konsistent zusammen. Behobene Fehler und neue Funktionalität verbreiten sich so automatisch. Hinzu kommt die Kommunikation über das bei H1 recht neue Hypernews-System, welches die bisherigen Newsgroups und Mailinglisten ersetzt hat. Es vereinigt die beiden bisherigen Kommunikationsmittel, so daß Nachrichten einerseits mit geringer Zeitverzögerung bei jedem Abonnenten eines Forums im Postfach ankommen, andererseits aber auch später noch über das World Wide Web für alle Mitglieder von H1 verfügbar sind.

Die Designwerte für den Speicherverbrauch pro Ereignis scheinen in allen drei Ebenen realistisch abgeschätzt worden zu sein. Bei den derzeitig verfügbaren Datensätzen bewegen sich die Spitzenwerte noch unter den angestrebten Zahlen für die durchschnittliche Größe eines Ereignisses. Die verfügbare Rechenleistung reicht aus, μODS und HAT für ein Jahr innerhalb weniger Tage zu produzieren. Selektionen auf dem HAT sind wesentlich schneller als das Erstellen von Index-Files in der alten Umgebung, sie bewegen sich in der Größenordnung von einigen Stunden für die Daten eines ganzen Jahres, wobei nicht auf den vollen Datensatz jedes Ereignisses zugegriffen werden muß.

Anhang A

Dokumentation

A.1 Glossar softwarespezifischer Begriffe

Basisklasse Jede *Klasse*, von der mindestens eine andere Klasse abgeleitet wird.

Datamember Zu deutsch: Datenelement. Eine zu einer *Klasse* gehörende Variable.

Dereferenzierung Bei der Dereferenzierung eines Zeigers wird auf das Objekt im Speicher zugegriffen, auf das der Zeiger verweist.

Framework Zu deutsch: Umgebung. Eine Sammlung bereitgestellter Mittel für einen bestimmten Aufgabenbereich.

Frontend Programm, daß die Bedienung eines oder mehrerer anderer Programme erleichtert, indem es die Funktionalität auf das wesentliche reduziert und/oder eine leichter nachvollziehbare Schnittstelle bereitstellt.

Inheritance Siehe *Vererbung*

Inheritance Tree Siehe *Vererbungsbaum*

Instanz Ein *Objekt*, das nach den Definitionen einer *Klasse* konstruiert wurde.

Klasse Zusammenfassung von Daten und den darauf definierten Operationen.

Methode Funktion einer *Klasse*, die auf ihren Datenelementen operiert.

Objekt Während die *Klasse* eine Art "Bauplan" darstellt, ist ein Objekt sozusagen deren Ausführung.

persistent Auch nach Beendigung eines Programms durch Speicherung fortbestehend, abspeicherbar.

RTTI **R**un-**T**ime-**T**ype-**I**nformation, also Informationen über den Typ eines Objektes zu Laufzeit.

Smart Pointer Klassen, die sich nach außen wie echte Zeiger verhalten, intern aber mit einem anderen Mechanismus arbeiten.

transient Nur während der Laufzeit eines Programmes existent, nicht abspeicherbar.

Vererbung Wenn eine *Klasse* von einer anderen erbt, kann sie deren Funktionalität wiederverwenden und durch zusätzliche erweitern.

Vererbungsbaum Graphische Darstellung der *Vererbungs*-Hierarchie

Wrapper Funktion, die zwei Schnittstellen aufeinander abbildet, z.B. um den Aufruf einer `FORTRAN`-Funktion in eine `C++`-Syntax umzusetzen.

A.2 Aufsetzen der Umgebung

Zum Zeitpunkt der Arbeit waren folgende Schritte nötig, um das OO-Framework benutzen zu können:

```
mkdir <workingdirectory>
cd <workingdirectory>
h1oo [-n <releasenummer>]
```

Hier ist `h1oo` eine Funktion, die intern andere Skripte aufruft und einige Shell Variablen ändert. Im wesentlichen werden durch dieses Tool vier Dinge erreicht:

- Anlegen einiger Verzeichnisse und Verstecken von CVS Kommandos
- Setzen der zu benutzenden Release Nummer (falls nicht angegeben, wird die Nummer gesetzt, die dem "current" zu diesem Zeitpunkt entspricht)
- Konsistenz in den zu benutzenden Paketen
- Anpassungen der Shell Variablen `$PATH` und `$LD_LIBRARY_PATH`

Auf diese Weise können dann schon die ausführbaren Programme eines Releases benutzt werden. Um nun selbst Änderungen vorzunehmen, können Pakete dem lokalen Arbeitsverzeichnis hinzugefügt und anschließend auch wieder daraus entfernt werden. Nach Abschluß der Arbeit kann das Arbeitsverzeichnis auch wieder komplett freigegeben werden. Dazu dienen folgende Kommandos:

```
h1oo addmodules [Paket1] [Paket2] [...]
h1oo removemodules [Paket1] [Paket2] [...]
h1oo removeall
```

Es existieren Kurzformen der Befehle: `am` für `addmodules`, `rm` für `removemodules` und `ra` für `removeall`.

A.3 C++ Programming Guidelines

1. New C++ Features
   (a) The support of new C++ features is still very limited and compiler dependent. **Therefore templates, exceptions, RTTI (runtime type information) and namespaces must not be used!**
   (b) **The usage of the STL (Standard Template Library) is not allowed.** But it is recommended to be coherent with STL conventions if they are widely used, e.g. for iterators.
2. Naming Conventions
   (a) For naming conventions it is obligatory to use the Taligent rules together with their ROOT extensions, as is recommended by the ROOT developers in their ROOT C++ coding conventions.
   (b) Use self-explanatory English names for code readability. Use first-letter capitalization for constructing names, avoid the use of underscore characters (e.g. GetTotalEnergy() instead of get_total_energy()).
   (c) Adherence to the C programming rule to start preprocessor defines with one or two underscores is also obligatory, e.g. #define __FILE_CXX.
3. Constants
   (a) Avoid global constants in header files.
   (b) Avoid constants at file scope if possible.
4. Classes
   (a) For easier maintainance each class implementation should go into a single source code file.
   (b) The public, protected and private keywords must be used explicitly in the class declaration.
   (c) All data members must be private. If you want to grant direct access to an inherited class, declare the data member as protected.
   (d) All inline function implementations should go into the header files, all non-inline into the implementation file. Simple getters and setters for primitive types can be implemented in the declaration. For debugging purposes use the following schema:

```
class Event {

public:
```

```
    double GetTotalEnergy();
    ...

  private:
    ...

  }

  INLINE double Event::GetTotalEnergy() {

    // do some calculation

    return e;

  }
```

INLINE will be defined in a global include file.

(e) We only use public inheritance. We don't allow protected inheritance. Use aggregation instead of private inheritance, that is, replace

```
class C : private B {
}
```

with

```
class C {
private:
  B b;
}
```

We also don't recommend to use multiple inheritance.

(f) Try to avoid using the friends concept whenever possible.

(g) Methods should be declared const, if they don't alter the state of any data member.

5. Files

   (a) C++ header files have the file name extension ".h".

   (b) C++ implementation files have the file name extension ".C".

   (c) Each file starts with a header, which includes author, updates, date, etc.

   (d) Use the directive #include "filename.h" for user defined include files.

   (e) Use the directive #include <filename> for C++ standard library include files. Don't forget to add using namespace std; in your source file.

(f) Use the directive #include <filename.h> for C standard library include files.

(g) Never use any relative or absolute pathnames (like ../directory/filename.h) in #include directives. An allowed exception is if you need to include a header file of a different package. Then, include the release directory in an -I option and use #include <packagename/file.h>.

(h) Each implementation file must include as first include file its corresponding header file.

(i) Every include file must contain a mechanism that prevents multiple inclusions of the file:

```
#ifndef __FILE_H
#define __FILE_H

// some code

#endif
```

(j) All H1 specific include directives must be surrounded by an additional external inclusion guard:

```
#ifndef __FILENAME_H
#include "filename.h"
#endif
```

(k) If only pointers or references to a class (e.g. ClassA) are used in the header file of another class (e.g. ClassB), then do NOT include the header file of ClassA in classb.h:

```
#ifndef CLASSB_H
#define CLASSB_H

class ClassA; // #include "classa.h" not necessary here!

class ClassB {
public:
  ClassA *getSomething();
  int     getAnInt();
private:
  ClassA *somePointer;
  ClassA& aReference;
}
```

Imagine another file, classc.C:

```
#include "classb.h"
```

```
void ClassC::dosomething (ClassB: anObject) {
  //...
  int i = anObject.getAnInt();
}
```

Although ClassB holds a pointer to an ClassA object, this is not used in ClassC. Therefore, if classa.h is changed, classc.C does not have to be recompiled.

6. Coding Style

   (a) **Use comments to document your code!** We recommend to use the ROOT coding style for automatic documentation.

   (b) Use indentation (>=2 characters) for better readability.

   (c) There are a lot of different styles for placing braces. When maintaining someone else's code, always use the style used in that code. We recommend to use the TOOTBS (The Original One True Brace Style) style, which is a slight variant of the K&R (Kernighan and Ritchie) style. With this style, the else part of an if-else statement and the while part of a do-while statement appear on the same line as the close brace.

```
if (x == y) {
   ..
} else if (x > y) {
   ...
} else {
   ....
}
```

7. Miscellaneous

   (a) The use of C-style I/O functions (e.g. printf()) is forbidden, use C++ stream I/O (e.g. cout) instead. C++ and C (and therefore Fortran) I/O can be mixed on a per-character basis. A call of ios::sync_with_stdio(); before the first stream I/O operation in the execution of a program guarantees that the C-style and C++-style I/O operations share buffers. Also, according to the ANSI standard cout << endl; flushes the buffer.

   (b) C like memory allocation (malloc/free) is forbidden, use operators new and delete instead.

   (c) The more powerful cfortran.h should be preferred with respect to minicf.h.

   (d) It is strongly recommended to compile with g++ -Wall or equivalent and to eliminate all compiler warnings.

A.4 Pakete und deren Verwendung

Das Projekt unterteilt sich zum Zeitpunkt dieser Arbeit in die folgenden Pakete:

bos2oop Programm, das aus der `DDL`-Beschreibung (`DDL` = **D**ata **D**efinition **L**anguage) von `BOS`-Bänken entsprechende Klassendefinitionen erzeugt bzw. aus den Klassen die `BOS`-Bänke wiederherstellt.

clusters Definition der Klassen für Kalorimeterzellen und Cluster sowie der Vorschrift, diese entweder aus `BOS`-Bänken oder ODS-Klassen zu erzeugen.

ddl_adds Korrekturen und Zusätze zu den offiziellen `ddl`-Beschreibungen.

examples Beispiele, wie man das OO-Framework benutzt. In diesem Paket müssen die Quelltexte kompiliert werden.

h1++ Sammlung von `C++`-Klassen, um auf die Daten der DSTs zuzugreifen.

jetfinder Klassen zur Jetidentifikation.

lotus++ Wrapper, um in `C++` auf die LOTUS-Variablen zuzugreifen.

macros Beispiele für die interaktive Nutzung des OO-Frameworks.

modsdetobj Definition der Klassen, welche die spezifischen Informationen zu identifizierten Teilchen speichern.

modshat Definiert die Klassen und die Struktur von μODS und HAT und beinhaltet das Programm zum Schreiben der beiden Datenlagen.

mysql Bibliothek zu Anbindung an die `MySQL`-Datenbank.

ods Definiert die Klassen, welche auf ODS gespeichert werden sollen, und enthält das Programm, das ODS aus DST erzeugt.

oo_tools Enthält Hilfswerkzeuge zur Benutzung des OO-Frameworks.

particles Definiert die Klassenhierarchie für Teilchen.

pointers Enthält spezifische Klassen für Smart Pointer.

runcatalog Programm, das den Runkatalog mit den Informationen über die bestehenden Dateien füllt.

selections Enthält Klassen, um aus einer Menge von Objekten einzelne auszuwählen, die gegebenen Anforderungen erfüllen.

skeleton Definiert die Schnittstelle zwischen den Daten im neuen Format und dem Benutzer.

tracks Definiert die Klassenhierarchie für Spuren und die Vorschrift, diese aus `BOS`-Bänken zu erzeugen.

wrapper Definiert ein Schema, wie allgemeine `FORTRAN` Funktionen und Datenstrukturen einfach aus `C++` aufzurufen sind.

Anhang B

Werkzeuge zur Software-Entwicklung

B.1 CVS: Concurrent Versioning System

B.1.1 CVS Glossar

Dieses Glossar beschreibt nicht nur Begriffe, die aus dem allgemeinen CVS Jargon stammen, sondern auch zusätzliche, die sich innerhalb des OO-Projektes im Zusammenhang mit Versionsverwaltung gebildet haben. Reine CVS Begriffe werden klein geschrieben, H1-OO-spezifische Begriffe groß und Programmnamen, Shellvariablen etc. im Schreibmaschinenstil. Begriffe, die selbst wieder in diesem Glossar erklärt werden, erscheinen kursiv.

attic Platz in jedem *Paket*, in den Dateien kopiert werden, wenn sie nicht weiter von CVS verwaltet werden sollen. Von dort aus können sie wieder reaktiviert bzw. in einen alten Zustand zurückversetzt werden.

`$H1CURRENT` Bezeichnet die *Release* Version, die ein Anwender benutzen will. Der Wert '`current`' bezeichnet dabei den letzten stabilen *Release* für Benutzer.

`$H1DIST` Beinhaltet das Hauptverzeichnis für *Pakete* und *Releases*.

head Die neueste Version eines *Paketes*. Es existiert ein spezieller *tag* namens "HEAD", der immer auf den Zustand mit der höchsten Versionsnummer verweist.

Paket Logische Einheit des Quelltextes, die im Idealfall nur eine einzige Aufgabe bewältigt. Wird auch als "Modul" bezeichnet.

Release Sammlung von mehreren *Paketen* in zueinander kompatiblen Versionen (*tags*). Im allgemeinen gibt es eine stabile Serie und eine für Entwickler.

repository Das zentrale Verzeichnis, an dem CVS den Code verwaltet. Es dürfen keine Änderungen direkt im repository gemacht werden.

revision Versionsnummer einer Datei, wird mit jeder Änderung am *repository* inkrementiert. Bei einem *tag* wird der Zustand eines Paketes durch die zugehörigen Dateien und deren *revision* gekennzeichnet.

tag Ein symbolischer Name, der dem Zustand eines *Paketes* zu einem gewissen Zeitpunkt zugeordnet wird.

B.1.2 Die wichtigsten CVS Kommandos

Dies ist eine Kurzzusammenfassung der CVS-Befehle, welche am häufigsten benötigt werden. Der Abschnitt soll keine Befehlsreferenz darstellen, sondern nur die generellen Möglichkeiten vorstellen. Kursiv gedruckte Begriffe werden im Glossar in Abschnitt B.1.1 erklärt.

add Fügt die angegebenen Dateien einem *Paket* hinzu, damit diese mit CVS verwaltet werden.

checkout Legt auf dem Rechner des Benutzers zu den angegebenen *Paketen* entsprechende Verzeichnisse an und kopiert deren Quelltext, welcher nun beliebig bearbeitet werden kann. Standardmäßig wird die aktuellste Version kopiert, es kann aber auch ein *tag*, eine *revision* oder ein Datum angegeben werden.

commit Schreibt die Änderungen, die ein Benutzer auf seinem Rechner gemacht hat, zurück ins *repository*. Dabei besteht die Möglichkeit, die Änderungen zu kommentieren.

diff Vergleicht die lokale Kopie mit dem *repository*. Standardmäßig wird mit der aktuellsten Version verglichen, es können aber auch ein *tag*, eine *revision* oder ein Datum angegeben werden. Auch ist es möglich, verschiedene Versionen miteinander zu vergleichen, ohne sich auf die lokale Kopie zu beziehen.

log Zeigt alle Änderungen von Dateien mit Datum, Kommentar und Benutzer an, zudem wird aufgelistet, welche *tags* existieren.

release CVS führt Buch, welche Pakete mittels `checkout` zur Bearbeitung kopiert wurden. Mit diesem Befehl hebt man diesen Status auf.

remove Entfernt eine Datei aus einem *Paket*, eine Kopie befindet sich aber weiter im *attic* dieses Paketes.

status Zeigt zu einer Datei an, welche *revision* sie besitzt, ob sie zu einem *tag* gehört und ob sie gegenüber dem *repository* verändert wurde.

tag Markiert in einem *Paket* die verschiedenen *revisions* der Dateien als zusammengehörig. Auf diese Weise wird dem Zustand des *Paketes* zu einem gewissen Zeitpunkt ein symbolischer Name zugeordnet.

update Gleicht die lokale Arbeitskopie von Dateien mit dem Zustand im *repository* ab. Wurde eine Datei von anderen Benutzern ebenfalls verändert, werden lokal alle Änderungen zusammengeführt. Sind diese nicht miteinander vereinbar, wird eine Fehlermeldung ausgegeben und der Benutzer muß den Konflikt von Hand bereinigen.

Die empfohlene Vorgehensweise, eigene Änderungen ins Repository zurückzuschreiben, umfaßt die folgenden Schritte:

`cvs -n update -A`	Zeigt an, was bei einem update zum HEAD passieren würde.
`cvs update -A`	Holt die letzten Änderungen anderer Autoren am Repository in die Arbeitskopie.
`cvs commit`	Schreibt alle eigenen Änderungen zurück ins Repository.
`cvs tag <name>`	Gibt dem aktuellen Zustand des Paketes einen symbolischen Namen.

B.2 CRC: Classes, Responsibilities, Collaborators

Class:	**Superclass:**
Subclasses:	
Responsibilities	**Collaborations**

Anything else:

Abbildung B.1: *Vorlage für eine CRC Karte*

Class: H1Frame	**Superclass:** TObject
Subclasses:	
Responsibilities	**Collaborations**
knows the boostvector for this frame knows the rotation for this frame	H1Particle (everything which has a 4vector)

Anything else:

- user should create a lorentz-frame once and can then forget about the actual boost parameters
- example: `a_particle→GetFourVector(breitframe);` where `breitframe` is an instance of H1Frame initialised to the parameters of the breit frame

Abbildung B.2: *Beispiel für eine ausgefüllte CRC Karte*

Abbildungsverzeichnis

1.1 HERA und Vorbeschleuniger 2
1.2 Der H1-Detektor 3
1.3 Altes und neues Analysemodell 5

2.1 Schematischer Datenfluß bei H1 8

3.1 Funktionsweise eines `H1Pointers` 21

4.1 Klassenhierarchie ohne identifizierte Teilchen 28
4.2 Klassenhierarchie für separate identifizierte Teilchen 30
4.3 Klassenhierarchie mit identifizierten Teilchen 31
4.4 Schema des temporären μODS Ansatzes 33
4.5 Produktion der drei Datenlagen 34

5.1 Einfacher Graph für ep-Streuung 39
5.2 Diagramm zum QPM-Prozeß im Breit-System 44
5.3 Q^2-Verteilung nach der Vorselektion auf HAT 47
5.4 Grundlegende Größen der Selektion 48
5.5 Eigenschaften des Elektrons 51
5.6 Balance des Transversalimpulses 52
5.7 Inklusiver, differentieller Wirkungsquerschnitt 52
5.8 E_t^*- und η-Verteilung der beiden energiereichsten Jets 54
5.9 E_t^*- und η-Verteilung der beiden energiereichsten Jets 55
5.10 Kinematik der gefundenen Jets 56
5.11 Unkorrigierter Zweijet-Wirkungsquerschnitt für die Daten des Jahres 2000 57
5.12 Korrigierter Zweijet-Wirkungsquerschnitt für die Daten des Jahres 2000 . . 58

B.1 CRC Kartenvorlage 77
B.2 Beispiel CRC Karte 78

Tabellenverzeichnis

2.1 Nachteile der bestehenden Analyseumgebung 10
2.2 Einsatzgebiete von ROOT 12
2.3 Verbesserungen durch das neue Schema 13
2.4 Das neue Datenmodell 14

3.1 Ergebnis der CRC Brainstorming Session 16
3.2 Felder einer CRC-Karte 17

4.1 Informationen im H1PHAN QVEC 26
4.2 Funktionen für H1PHAN QVECs 27
4.3 Klassen der konkreten Analyseumgebung 36
4.4 Der konkrete Inhalt der drei Datenlagen 37

5.1 Schnitte der Vorselektion 46
5.2 Schnitte zur Untergrundreduktion 49
5.3 Reihenfolge der Schnitte 50
5.4 Ergebnisse der Fits an z_{vertex}, $\sum(E - p_z)$ und $p_{t,\mathrm{bal}}$ 51
5.5 Übersicht der Schnitte auf Zweijet-Ereignisse 53
5.6 Anzahl der Zweijet-Ereignisse nach allen Schnitten 53

6.1 Zeitbedarf zum Erstellen von μODS/HAT Paaren 59
6.2 Übersicht der Rechnerfarm-Ressourcen 60
6.3 Geschwindigkeitsvergleich von interpretiertem und kompilierten Code . . . 60
6.4 Größen von ODS, μODS und HAT 61
6.5 Datenvolumen für μODS und HAT des Jahres 2000 61
6.6 Derzeitige Defizite in der Implementierung 62
6.7 In der Entwicklung befindliche Konzepte 63

Literaturverzeichnis

[BB95] U. Bassler and G. Bernardi. On the kinematic reconstruction of deep inelastic scattering at HERA: the Σ method. *Nucl. Instrum. Meth.*, A361:197, 1995. 5.1.1

[BBR] M. Brun, R. Brun, and F. Rademakers. *CMZ On-Line Documentation*. World Wide Web: `http://wwwinfo.cern.ch/cmz/`. 3.3.2

[BBSS97] D. Bellin, G. Booch, and S. Suchman-Simone. *The CRC Card Book*. Addison-Wesley, 1997. ISBN 0-20-189535-8. 3.1.2

[Ble01] A. Bleul. Programmier-Extremisten – Softwareprojekte auf den Kern reduziert. *c't – magazin für computer technik*, 3:182–185, 2001. 3.3.3

[Blo91] V. Blobel. *Fortran package for input/output, FPACK Manual V1.00/00*, 1991. `http://www-h1.desy.de/icas/imanuals/fpack/fpack.v10000.manuals.html`. 1.3.1

[BR] R. Brun and F. Rademakers. *C++ Coding Conventions*. World Wide Web: `http://root.cern.ch/root/Conventions.html`. 3.3.1

[BR96] R. Brun and F. Rademakers. `ROOT` - an object oriented data analysis framework. *Paper presented at the AIHENP conference in Laussanne*, 1996. World Wide Web: `http://root.cern.ch`. 1.3.2, 2.4, 4.3.3

[BRP+00] R. Brun, F. Rademakers, S. Panacek, D. Buskulic, J. Adamczewski, and M. Hemberger. `ROOT` *User's Guide*, 2000. World Wide Web: `http://root.cern.ch/root/RootDoc.html`. 3.2.3, 4.3.2

[CDW92] S. Catani, Yu.L. Dokshitzer, and B.R. Webber. The $k_\perp$ clustering algorithms for jets in deep inelastic scattering and hadron collisions. *Physics Letters B*, 285:291–299, 1992. 4.3.3, 5.1.2

[Ced] P. Cederqvist. *Open Source-Projekte mit CVS*. World Wide Web: `http://www.cvshome.org/docs/manual/cvs.html`. 3.3.2

[Dav01] M. Davidsson, 2001. Private Kommunikation. 4.3.3

[EH01] A. Elting and W. Huber. Immer im Plan? Programmieren zwischen Chaos und Planwirtschaft. *c't – magazin für computer technik*, 2:184–191, 2001. 3.1.1

[Fog] K. Fogel. *Open Source-Projekte mit CVS (Online).* World Wide Web: `http://cvsbook.red-bean.com/translations/german/`. 3.3.2

[Fog00] K. Fogel. *Open Source-Projekte mit CVS.* MITP-Verlag GmbH, 2000. ISBN 3-8266-0628-0. 3.3.2

[H1 93] H1 Collaboration. The H1 detector at HERA. Technical Report 93-103, DESY, Juli 1993. 1.2

[H1 95] H1 Collaboration. `H1REC` - *H1 reconstruction program*, 1995. H1 Internal. 3

[H1 97] H1 Collaboration. The H1 detector at HERA. (Updated version). *Nucl. Instrum. Meth.*, A386:310, 1997. 1.2, 2.1

[H1 99a] H1 Collaboration. Measurement of neutral and charged current cross-sections in positron-proton collisions at large momentum transfer. *submitted to Eur. Phys. J.*, C, 1999. DESY-99-107. 5.2.1

[H1 99b] H1 Collaboration. `H1PHAN` - *function library for H1*, 1999. H1 Internal, Version 3.00/04. 4.1, 4.1

[H1 99c] H1 HIP Group. *Overview of the LOTUS ntuple variables*, 1999. World Wide Web: `http://www-h1.desy.de/iwork/ihip/software_tools/nt.doc.html`. 4.3.1, 4.3.2

[H1 00a] H1 Collaboration. *C++ Programming Guidelines*, 2000. World Wide Web: `http://www-h1.desy.de/icas/oop/cxx_programming_guidelines.html`. 3.3.1

[H1 00b] H1 Collaboration. Search for compositeness, leptoquarks and large extra dimensions in eq contact interaction at HERA. *Phys. Lett.*, B479:358–370, 2000. 5.3

[Had99] T. Hadig. *Measurement and Pertubative QCD Fit of Dijet and Inclusive Cross Sections at HERA*. PhD thesis, RWTH Aachen, Dezember 1999. PITHA-99-41. 5, 5.1.2, 5.1.2, 5.2, 5.2.1, 5.3, 5.3, 5.3, 5.3, 6.1

[Had01] T. Hadig, 2001. Private Kommunikation. 6.1

[Hei00] B. Heinemann, 2000. Private Kommunikation. 4.3.1

[Iwa00] M. Iwasaki. Jetfinder/thrustfinder programs, 2000. Download: `ftp://ftp.slac.stanford.edu/groups/lcd/Physics_tools/`. 4.3.3

[JAD86] JADE Collaboration. Experimental studies on multi–jet production in e^+e^- annihilation at PETRA energies. *Z.Phys.*, C33:23f, 1986. 4.3.3, 5.1.2

[Kuh01] T. Kuhr, 2001. Private Kommunikation. 4.3

[Mey01] A. Meyer, 2001. Private Kommunikation. 4.4

[Nie97] Ch. Niedzballa. *Determination of the strong coupling constant from jet rates at the H1 experiment (in German)*. PhD thesis, RWTH Aachen, November 1997. PITHA-97-36. 5.1.2

[Obj] Object Management Group, Inc. *OMG Unified Modeling Language Specification*. `ftp://ftp.omg.org/pub/docs/formal/00-03-01.ps.gz`. 3.3.4

[Pro] H1 OO Project. *Documentation for H1 OO Project*. World Wide Web: `http://www-h1.desy.de/icas/oop/documents/oo.ps.gz`. 3.2.2, 3.2.3, 3.2.4

[Rat] Rational Software Corporation. *Rational Rose – A visual modeling tool*. World Wide Web: `http://www.rational.com/products/rose/index.jsp`. 3.3.4

[RBDK] T. Riemer, F. Brettschneider, W. Delvaux, and H. Koll. *LinCVS*. World Wide Web: `http://www.lincvs.de/`. 3.3.4

[Sey98] M.H. Seymour. Jets in hadron collisions in QCD. Technical report, RAL, 1998. 5.1.2

[Tal] Taligent Inc. *Taligent's Guide to Designing Programs (Online)*. World Wide Web: `http://pcroot.cern.ch/TaligentDocs/TaligentOnline/DocumentRoot/1.0/Docs/books/WM/WM_1.html`. 3.3.1

[Tal95] Taligent Inc. *Taligent's Guide to Designing Programs*. Addison-Wesley, 1995. ISBN 0-201-40888-0. 3.3.1

[Tic] W. Tichy. *Official RCS Homepage*. World Wide Web: `http://www.cs.purdue.edu/homes/trinkle/RCS/`. 3.3.2

[Win] Wind River Systems, Inc. *Sniff+ – An embedded development tool for C, C++, Java or ADA*. World Wide Web: `http://www.windriver.com/products/html/sniff.html`. 3.3.4

Danksagung

Zunächst möchte ich mich bei Prof. Dr. Christoph Berger bedanken, der mich an diese interessante Thematik herangeführt hat. Sein Vertrauen in meinen Beitrag für das Projekt sowie seine Hilfestellungen waren mir sehr wichtig.

Herrn Prof. Dr. Stefan Schael danke ich, daß er das Co-Referat übernommen hat.

Den Mitarbeitern der Aachener Institute sei herzlich für Unterstützung und das gute Arbeitsklima gedankt: Roman Adolphi, Adil Aktas, Sascha Caron, Carlo Duprel, Gilles Frising, Thomas Hadig, Thomas Kluge, Boris Leißner, Peer-Oliver Meyer, Jan Olzem, Klaus Rabbertz, Jürgen Scheins, Anja Vest, Martin Wessels und Markus Wobisch.

Allen Mitarbeitern des H1-OO-Projektes, mit denen ich während meiner Arbeit in Kontakt gekommen bin, möchte ich danken für zahlreiche Anregungen und die gute Kommunikation, die wesentlich zum Spaß an der Arbeit beigetragen hat.

Bedanken möchte ich mich bei allen, die diese Arbeit Korrektur gelesen haben: Ralf Gerhards, Thomas Hadig, Thomas Kluge, Andreas Meyer, Jan Olzem und Peter Schleper. Ihre verschiedenen Sichtweisen haben dazu beigetragen, dieser Arbeit eine klarere Struktur zu verleihen.

Besonderer Dank gilt auch Thomas Kuhr und Andreas Meyer für das Bereitstellen einiger Abbildungen.

Ich danke allen Mitarbeitern bei HERA und H1, die durch ihre Arbeit die Datennahme erst ermöglicht haben. Darüber hinaus haben sie ihre Maschinen stetig zu höheren Leistungen verholfen, was das Projekt, an dem ich teilhatte, erst notwendig werden ließ.

Nicht zuletzt bedanke ich mich bei meinen Eltern, die von Anfang an mein Studium der Physik voll unterstützt haben. Dieser besondere Abschnitt meines Lebens wäre ohne sie nicht möglich gewesen.