Entwicklung einer neuen HADES Online Monitoring Software

Bachelorarbeit von Julian Thomas

vorgelegt am
Fachbereich Physik
der Goethe-Universität
in Frankfurt am Main

durchgeführt am
GSI Helmholtzzentrum für Schwerionenforschung
in Darmstadt

Frankfurt am Main, den 16.06.2010

Inhaltsverzeichnis

Inhaltsverzeichnis **III**

Abbildungsverzeichnis **IV**

Listings **V**

1 Einleitung **1**
1.1 Das HADES-Experiment 1
1.1.1 Die Komponenten 2

2 Das HADES Online-Monitoring **4**
2.1 Prinzip des Online-Monitorings 4
2.2 Die Monitoring-Histogramme 6

3 Anforderungen an die neue Online-Monitoring Software **8**
3.1 Serverprogramm 9
3.2 Clientprogramm 9
3.3 Client-Server-Kommunikation 11
3.4 Konfiguration 11

4 Implementierung der Online-Monitoring Software **13**
4.1 Server 13
4.2 Client 15
4.3 Konfiguration 23

5 Zusammenfassung und Ausblick **26**
5.1 Zusammenfassung 26
5.2 Ausblick 26

A XML-Beispielkonfigurationsdatei **27**

Literaturverzeichnis **31**

Abbildungsverzeichnis

1.1 Das HADES-Experiment 3

2.1 Schematische Darstellung des Online-Monitoring-Prozesses 5
2.2 Beispiel eines 2-dimensionalen Säulenhistogramms aus Daten der MDCs. 7
2.3 Beispiel eines Trendhistogramms 7

3.1 Use-Case-Diagramm des Clientprogrammes 10

4.1 Zustandsdiagramm für das Clientprogramm 16
4.2 Klassendiagramm des Clientprogramms 17
4.3 Screenshot des Hauptfensters 18
4.4 Einfaches Fenster des Clientprogramms 19
4.5 Einfaches Fenster und Tabfenster des Clientprogramms mit einem in mehrerer Segmente unterteilten Canvas 20
4.6 Baumstruktur der XML-Konfigurationsdatei des Clientprogramms 25

A.1 Screenshot des Programms 30

Listings

4.1 Signatur der HMonClientMain Init-Methode 17
4.2 Aufruf der HMonClientMain Init-Methode zur Übertragung eines Kommandos . 17
4.3 Aufruf der HMonClientMain Init-Methode für das Online-Monitoring 18
4.4 Aufruf des Clientprogramms zur Übertragung des list-Kommandos 23
4.5 Aufruf des Clientprogramms zur Verwendung für das Online-Monitoring 23
A.1 XML Beispielkonfiguration 27

Kapitel 1

Einleitung

Schießt man beschleunigte Ionen mit hoher Energie auf ein Target, entsteht unmittelbar nach der Kollision eines Ions mit einem Atom des Targetmaterials für einen extrem kurzen Augenblick ($10^{-22}s$) ein Zustand heißer und dichter Kernmaterie. Dieser Zustand wird auch als Feuerball bezeichnet. Aufgrund des daraus folgenden Expansionsdranges fliegt der Feuerball auseinander (Fragmentation). Durch die kurze Reaktionszeit ist eine direkte Beobachtung dieses Vorgangs unmöglich. Um dennoch Aussagen über den Reaktionsablauf treffen zu können, ist eine genaue Untersuchung der Reaktionsprodukte (Fragmente) nötig. Aus den entstandenen Reaktionsprodukten und deren Eigenschaften kann der Ablauf der Reaktion anschließend rekonstruiert werden. Daher ist es wichtig, die Fragmente des Feuerballs nachzuweisen, sowie deren Eigenschaften zu messen.

1.1 Das HADES-Experiment

Das HADES-Experiment (**H**igh **A**cceptance **DiEl**ectron **S**pectrometer) ist ein Dielektronen-Spektrometer an der GSI Gesellschaft für Schwerionenforschung in Darmstadt. Es dient im Wesentlichen der Untersuchung der leichten Vektormesonen (ρ und ω). Diese Vektormesonen entstehen aus dem Zustand dichter Kernmaterie unmittelbar nach der Kollision von zwei Atomkernen. Aufgrund ihrer kurzen Lebenszeit zerfallen die Vektormesonen unmittelbar nach ihrer Entstehung in e^+e^--Paare (Dileptonen). Diese e^+e^--Paare können den Feuerball ungestört verlassen, da sie nicht der starken Wechselwirkung unterliegen und daher keine Reaktion mit der umgebenden Kernmaterie stattfinden kann. Die Eigenschaften der e^+e^--Paare geben somit direkt Aufschluss über den Zustand des Systems unmittelbar nach der Kollision.
Mit dem HADES-Experiment werden diese Elektron-Positron-Paare hinsichtlich ihrer Eigenschaften wie beispielsweise Energie oder Impuls untersucht. Der experimentelle Aufbau zeichnet sich durch seine hohe geometrische Akzeptanz und eine hohe Massenauflösung aus. Das HADES-Experiment besitzt eine Reihe unterschiedlicher Detektorsysteme. Sie dienen unter anderem zum Nachweis und zur Identifikation von Teilchen, zur Bestimmung der Flugbahn der Teilchen und zur Impulsbestimmung. Eine umfassende Beschreibung des Experimentes ist unter [Aga09] zu finden.

1.1.1 Die Komponenten

Da das HADES-Experiment primär zur Untersuchung von Dileptonen gedacht ist, besitzt das System einige Detektoren, die ausschließlich zum Nachweis von Leptonen und Messung von deren Eigenschaften konzipiert worden sind. Aber auch Hadronen lassen sich mit dem HADES-Experiment untersuchen, wie beispielsweise die Arbeiten von [Lor08] und [Sch08] zeigen.

Die Start-/Veto-Detektoren

Mit dem Start-Detektor lässt sich die Startzeit der Reaktion messen. Diese dient als Referenz für die Flugzeitmessung und zur Berechnung der Driftzeit in den Vieldrahtdriftkammern. Mit dem Veto-Detektor können Ereignisse, bei der keine Kollision im Target stattgefunden hat, von der Datenaufnahme ausgeschlossen werden. Dies reduziert die zu verarbeitende Datenmenge und die Totzeit der Detektoren.

RICH-Detektor

Der RICH-Detektor (**R**ing **I**maging **CH**erenkov) ist ein Detektor zur Identifizierung von Elektronen und Positronen. Die Funktionsweise basiert auf dem Cherenkov-Effekt. Hochenergetische Teilchen emittieren beim Durchqueren eines Dielektrikums Photonen unter einem festen Winkel bezogen auf die Teilchenbahn, wenn ihre Geschwindigkeit größer ist als die In-Medium-Phasengeschwindigkeit. Dadurch entsteht ein Kegel von Photonen. Da Leptonen leichter und damit schneller sind als Hadronen, produzieren nur die Leptonen bei Verwendung eines geeigneten Dielektrikums einen Photonenkegel. Anhand des Kegels lassen sich daher Leptonen nachweisen.

Das Magnetspektrometer

Das Magnetspektrometer besteht aus einem supraleitenden Magneten und mehreren Ebenen von Vieldrahtdriftkammern (MDC: **M**ulti-wire **D**rift **C**hamber). Es dient in erster Linie zur Impulsbestimmung und Spurverfolgung der geladenen Teilchen. Jeweils zwei Ebenen von Driftkammern befinden sich vor und hinter dem Magneten. Die Impulsbestimmung basiert auf der Messung von Spurpunkten vor und hinter dem Magnetfeld mit Hilfe der Driftkammern. Anhand der Spurpunkte lässt sich die Flugbahn rekonstruieren. Aus der Flugbahn wird der Ablenkwinkel des Teilchens im Magnetfeld bestimmt. Daraus kann anschließend der Impuls der Teilchen berechnet werden.

Der META-Detektor

Der META-Detektor (**M**ultiplicity and **E**lectron **T**rigger **A**rray) dient zur Flugzeitmessung, Leptonenidentifikation und zur Messung der Multiplizität. Er besteht aus TOF- (**T**ime **O**f **F**light), TOFino- und SHOWER-Detektor. TOF- und TOFino-Detektor sind Detektoren zur Flugzeitmessung. Der TOF-Detektor deckt dabei den äußeren Winkelbereich, der TOFino-Detektor den inneren Winkelbereich ab. Der Shower-Detektor dient zur Leptonenidentifikation.

Das gesamte HADES-Experiment befindet sich derzeit in einer Umbau- und Erweiterungsphase.

Unter anderem wird der TOFino-Detektor durch RPCs (**R**esistive **P**late **C**hamber) ersetzt. Parallel dazu wird das Datenaufnahmesystem grundlegend umgebaut, um größere Datenmengen als bisher zu verarbeiten.

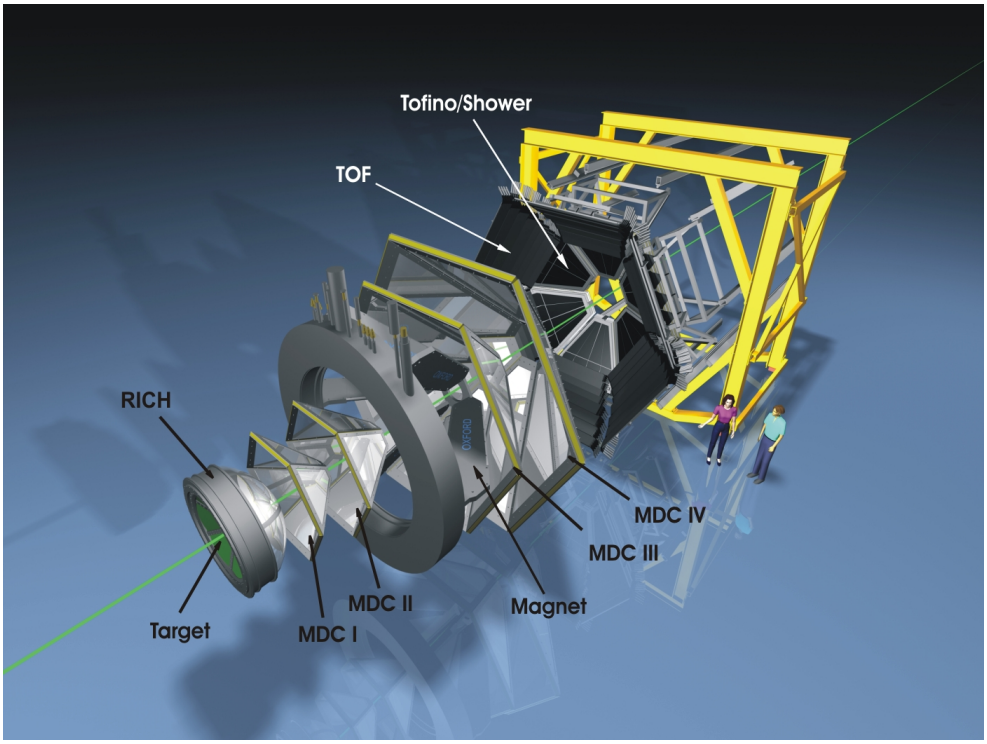

Abbildung 1.1: Explosionszeichnung des HADES-Experimentes

Kapitel 2

Das HADES Online-Monitoring

2.1 Prinzip des Online-Monitorings

Das HADES-Experiment besteht aus einem komplexen System von unterschiedlichen Detektoren. Für den laufenden Betrieb während einer Strahlzeit ist es unerlässlich, die Funktionsfähigkeit der einzelnen Detektoren kontinuierlich zu überwachen, um etwaige Fehlfunktionen unmittelbar erkennen zu können. Im Fehlerfall können so zeitnah geeignete Gegenmaßnahmen vorgenommen werden. Daher ist der Einsatz eines effizienten Online-Monitorings nötig. Das Monitoring ist die Beobachtung und Überwachung der Funktionsfähigkeit eines Systems. Wird das Monitoring im laufenden Betrieb durchgeführt, bezeichnet man es auch als Online-Monitoring. Im Gegensatz dazu existiert auch die Möglichkeit eines Offline-Monitorings. Dabei wird die Funktionsfähigkeit kontinuierlich aufgezeichnet und im Nachhinein analysiert. Der Vorteil des Online-Monitorings ist, dass im Fehlerfall direkt Maßnahmen zur Behebung des Fehlers eingeleitet werden können. Allerdings ist es bei großen oder komplexen Systemen nicht immer möglich, alle Betriebsparameter online zu überwachen.
Das HADES-Experiment verwendet eine Kombination aus Online- und Offline-Monitoring. Es können nicht alle Betriebsparameter online analysiert werden, da es zu viele sind und die Aufbereitung, Darstellung und Beurteilung zu lange dauern würde. Aus diesem Grund ist eine Kombination aus Offline- und Online-Monitoring nötig. Ein Teil der Betriebsparameter wird für das Online-Monitoring ausgewählt. Die ausgewählten Betriebsdaten werden graphisch aufbereitet. In der Regel werden Histogramme für die graphische Darstellung verwendet. Die Experimentatoren und Operateure können diese dann beurteilen und mögliche Fehlfunktionen des Systems anhand der Histogramme erkennen. Zusätzlich werden die Daten der Betriebsparameter aufgezeichnet, um sie bei der späteren Auswertung der physikalischen Sachverhalte berücksichtigen zu können. Die Auswahl der Parameter, die für das Online-Monitoring verwendet werden, hat entscheidenden Einfluss auf die Qualität und Effizienz des Online-Monitorings und bestimmt somit maßgeblich die Erkennungswahrscheinlichkeit von Fehlern.

Der Online-Monitoring-Prozess des HADES-Experimentes besteht aus verschiedenen Schritten, die teilweise von unterschiedlichen Systemen durchgeführt werden. Die Daten, die die Detektoren produzieren, gelangen zunächst in die zentrale Datenakquisition (DAQ: **D**ata **Ac**q**u**isition)

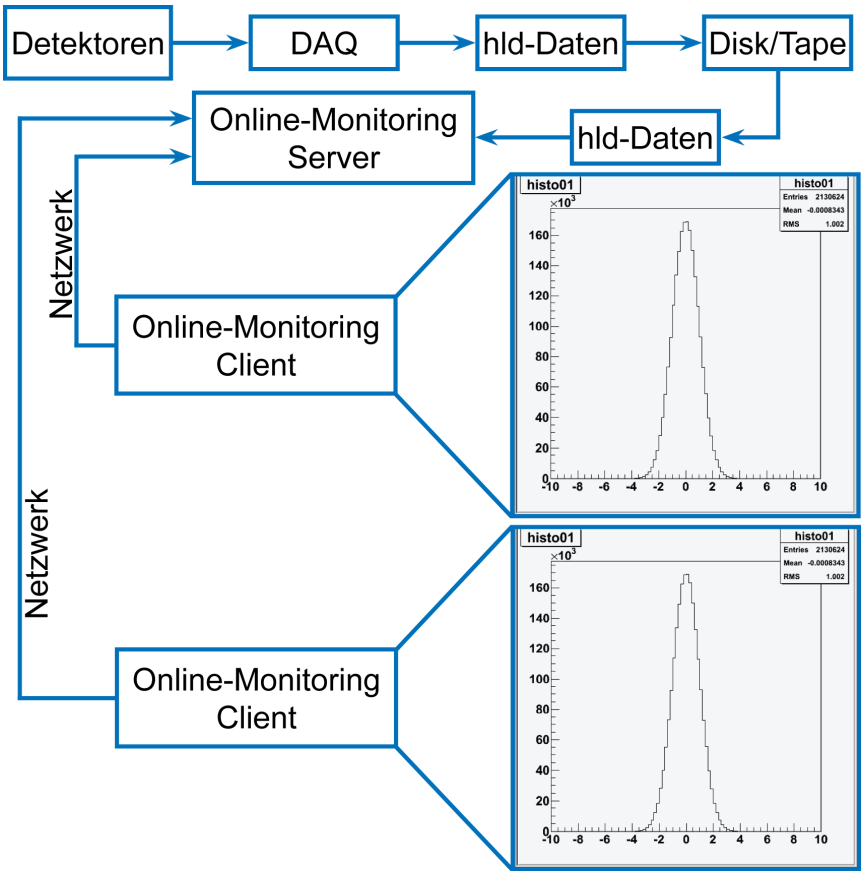

Abbildung 2.1: Schematische Darstellung des Online-Monitoring-Prozesses

des HADES-Experimentes. Dieses Datenaufnahmesystem rekonstruiert die stattgefundenen Reaktionen (Events) und sorgt für die Digitalisierung der Detektorsignale. Die Detektorsignale werden den stattgefundenen Reaktionen zugeordnet und in sogenannte hld-Dateien geschrieben. Jede hld-Datei enthält die Detektordaten von 200 000 Events und hat eine Größe von 2.0 GByte. Aus den Daten der hld-Dateien werden verschiedene Histogramme für das Online-Monitoring generiert und laufend aktualisiert. Die Experimentatoren und Operateure beurteilen dann die erzeugten Histogramme am Bildschirm und können anhand der Histogramme mögliche Fehler erkennen.

Darüber hinaus ist für so ein komplexes System wie das HADES-Experiment der Einsatz eines automatisierten Online-Monitoring-Systems sinnvoll. Beim Online-Monitoring ist man auf die manuelle Beobachtung angewiesen. Eventuelle Fehlfunktionen müssen von den Operateuren entdeckt und wahrgenommen werden. Dabei besteht grundsätzlich die Gefahr des Übersehens von Fehlern. Die Wahrscheinlichkeit des Übersehens hängt von vielen verschiedenen Faktoren ab. Es ist zu erwarten, dass nachts aufgrund von Ermüdungserscheinungen eher Fehler übersehen werden, als tagsüber. Zusätzlich ist die Erkennung eines Fehlers oftmals personenabhängig. Daher ist es sinnvoll, die Beurteilung der Daten und damit den Online-Monitoring-Prozess zu automatisieren. Dabei analysiert das System selbständig seine Betriebsparameter. Es kann seine Funk-

tionsfähigkeit dadurch beurteilen, dass die aktuell ermittelten Betriebsparameter mit gespeicherten Referenz- und Grenzwertdaten verglichen werden. Im Fehlerfall benachrichtigt das System den Benutzer/Operateur. Zusätzlich sind Trendanalysen (vorherige Daten werden mit den aktuell ermittelten verglichen, um spontan auftretende Abweichung erkennen zu können) denkbar. Bei einem solchen automatisierten Online-Monitoring ist die Erkennung von Fehlern nicht personen- oder tageszeitabhängig. Es besteht allerdings die Gefahr, dass das automatisierte System auf bestimmte Fehler nicht sensitiv ist und diese daher systematisch übersehen werden. Ausführliche Softwaretests können die Wahrscheinlichkeit einer Nichterkennung bestimmter Fehler jedoch stark reduzieren.
Im Idealfall sollte eine Kombination von beiden Verfahren verwendet werden. Manuelle Begutachtung der Monitoring-Daten durch die Operateure, sowie eine Analyse durch das System. Dies gewährleistet eine hohe Erkennungsrate von Fehlern.

2.2 Die Monitoring-Histogramme

Histogramme bilden die Grundlage des Online-Monitorings. Jeder Detektor besitzt Betriebsparameter und produziert Messdaten. Aus diesen Daten werden einige ausgewählt und als Histogramme graphisch aufbereitet. Die so erzeugten Histogramme können anschließend beurteilt werden. Für jeden Detektor werden Histogramme für das Online-Monitoring erzeugt. Sie sind dem jeweiligen Detektor eindeutig zugeordnet.
Für das Online-Monitoring existieren unterschiedliche Arten von Histogrammen, um die Daten der Detektoren anhand dieser Histogramme optimal beurteilen zu können. Grundsätzlich gibt es einfache Histogramme und sogenannte Trendhistogramme. Bei einfachen Histogrammen werden die Daten kumuliert, d.h. die neuen Daten werden den alten hinzugefügt. Trendhistogramme stellen einen zeitlichen Verlauf von Messdaten dar. Die neuesten Daten werden an der rechten Seite des Histogramms hinzugefügt, die ältesten Daten, die sich am linken Rand der Kurve befinden, werden entfernt. Die Anzahl der Messpunkte im Histogramm bleibt daher konstant. Daher ergibt sich über einen längeren Zeitraum hinweg betrachtet eine Kurve, die das Histogramm „durchwandert“.
Sowohl einfache als auch Trendhistogramme gibt es in ein- und zweidimensionalen Varianten. Für jedes Histogramm können verschiedene Eigenschaften wie zum Beispiel Darstellungsart der Daten (Balken, Punkte, Linien, etc.) eingestellt werden. Zusätzlich kann für jedes Histogramm ein Reset-Intervall festgelegt werden. Das Reset-Intervall gibt an, in welchen Abständen das Histogramm wieder auf null zurückgesetzt werden soll.
Für die einfachere Handhabung können Histogramme zu Histogrammarrays zusammengefasst werden. In einem Histogrammarray werden Histogramme verwaltet, die vom experimentellen Aufbau her gesehen inhaltlich zusammengehören.
Die Eigenschaften der Histogramme in einem Array werden nicht für jedes Histogramm einzeln festgelegt, sondern nur einmal für das gesamte Array. Dies vereinfacht die Handhabung der Histogramme erheblich.

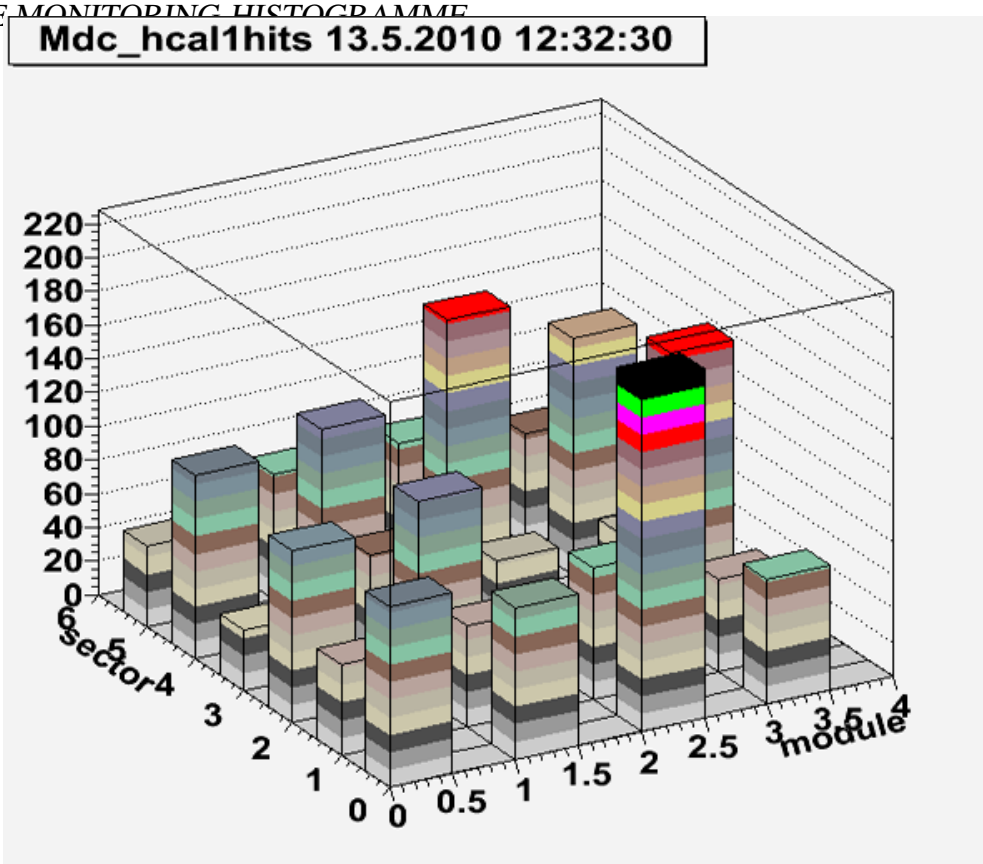

Abbildung 2.2: Beispiel eines 2-dimensionalen Säulenhistogramms. Das Histogramm beein-

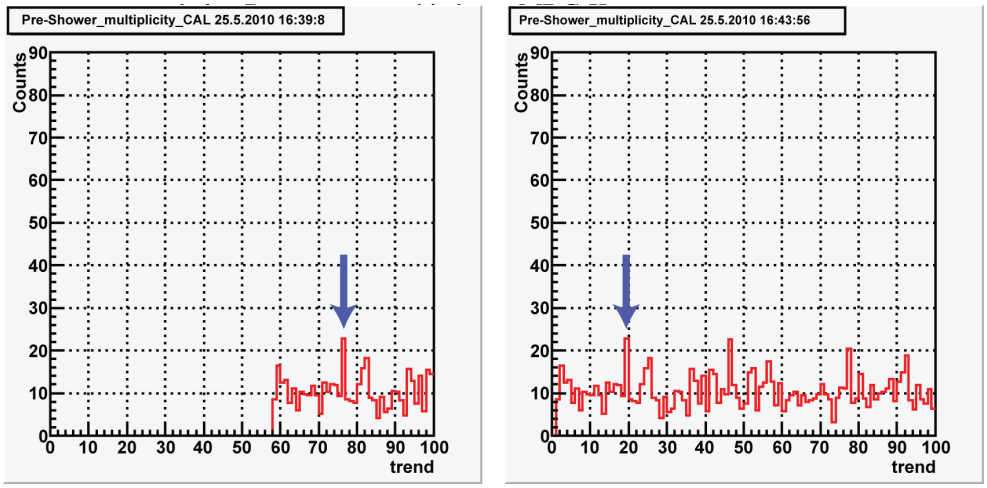

Abbildung 2.3: Beispiel eines Trendhistogramms. Dargestellt sind zwei Darstellung des gleichen Histogramms, die zeitlich versetzt sind. Der Pfeil im linken Histogramm kennzeichnet einen beliebigen Messwert, der Pfeil im rechten Histogramm zeigt auf denselben Messwert. Das Durchwandern der Kurve ist deutlich zu erkennen.

Kapitel 3

Anforderungen an die neue Online-Monitoring Software

Gegenstand dieser Arbeit ist die Entwicklung einer **neuen** Online Monitoring Software für das HADES-Experiment. Ziel ist es, ein Basis-Softwaresystem zu schaffen, das den Anforderungen an das Monitoring der Detektoren gerecht wird, um Störungen und Ausfälle im laufenden Betrieb erkennen zu können. Ausgehend von der Funktionalität der vorhandenen Software soll eine neue Software entwickelt werden, die stabil und schnell arbeitet. Zusätzlich sind flexible Konfigurationsmöglichkeiten erforderlich, um zukünftige Änderungen mit minimalem Aufwand durchführen zu können. Ausgehend von diesesm Basissystem können zukünftig Erweiterungen entwickelt werden, um den Online-Monitoring-Prozess zu erweitern und zu vervollständigen. Unter anderem ist eine automatisierte Analyse der Detektordaten nötig, um zu verhindern, dass Fehler übersehen werden.
Die Online-Monitoring-Software muss aus einem 2-Komponenten-System bestehen. Eine Komponente ist das Serverprogramm, die andere ist das Clientprogramm. Das Serverprogramm muss auf einem zentralen Server laufen. Es greift auf die hld-Daten zu, die das DAQ-System erzeugt, und produziert daraus Histogramme. Die produzierten Histogramme werden fortlaufend mit neuen Daten aus den hld-Dateien aktualisiert. Darüber hinaus muss das Serverprogramm den Clients die Histogramme über das Netzwerk zur Verfügung stellen.
Das Clientprogramm soll auf beliebigen Clientcomputern laufen und dient dem Abrufen der Histogramme vom Server sowie dem Zeichnen der Histogramme auf dem Bildschirm. Dazu verbindet sich das Clientprogramm mit dem Serverprogramm über das Netzwerk. Über diese Verbindung können die Histogramme vom Server zum Client übertragen werden. Das 2-Komponenten-Modell (auch Client-Server-Architektur) bietet die Möglichkeit, dass viele Personen (Experimentatoren, Operateure) die Histogramme zeitgleich an verschiedenen Orten abrufen und betrachten können.
Sowohl Client- als auch Serverprogramm müssen flexibel konfigurierbar sein.

3.1 Serverprogramm

Zu Beginn des Online-Monitoring-Prozesses muss das Serverprogramm auf einem zentralen Server gestartet und initialisiert werden. Dabei muss es Definitionen der Histogramme aus einer Konfigurationsdatei lesen. Mit den Histogrammdefinitionen wird festgelegt, welche Histogramme generiert werden sollen und welche Eigenschaften die einzelnen Histogramme haben sollen. Mit Hilfe der Histogrammdefinitionen kann die Software anschließend die Histogrammobjekte anlegen.
Danach muss das Serverprogramm die Events aus den hld-Daten durchlaufen. Dazu muss eine Schleife (Eventloop) gestartet werden. Diese läuft über die Events/Reaktionen aus den hld-Dateien und füllt die erzeugten Histogrammobjekte mit den Daten aus den Events auf. In jedem Durchgang werden die Histogramme aktualisiert und zusätzlich mögliche Client-Anfragen abgearbeitet. Dazu wird dem Client das angeforderte Histogramm gesendet oder im Fehlerfall eine Fehlermeldung in Textform übermittelt.
Darüber hinaus muss es eine Möglichkeit geben, das Serverprogramm vom Client aus zu steuern. Dazu muss es im Clientprogramm die Möglichkeit geben, Steuerungskommandos an den Server zu senden. Dieser kann auf ankommende Steuerungskommandos entsprechend reagieren. Vorläufig sind die Kommandos `start`, `stop` und `reset` geplant.

3.2 Clientprogramm

Die Experimentatoren und Operateure müssen die Histogramme von beliebigen Rechnern (Clients) inner- und außerhalb des GSI-Netzwerkes mit dem Clientprogramm vom Histogrammserver abrufen und ansehen können. Die abgerufenen Histogramme auf den Client-Computern müssen in einem gewissen Zeitintervall automatisch aktualisiert werden. Dabei erhält jedes Histogramm einen Zeitstempel. Der Benutzer des Clientprogramms kann sein Programm individuell konfigurieren. Jeder Benutzer kann für sein Clientprogramm Detektoren konfigurieren, deren Histogramme er beobachten möchte. Dazu können innerhalb des Programms Detektoren definiert werden. Für jeden Detektor müssen die graphische Oberfläche (Anzahl und Art der Fenster), sowie die zu zeichnenden Histogramme festgelegt werden. Einige Einstellungen, wie z.B. die Struktur der graphischen Oberfläche, werden vor dem Programmstart über eine Konfigurationsdatei festgelegt. Diese Einstellungen können zur Laufzeit nicht mehr verändert werden, andere Einstellungen, wie das Aktualisierungintervall der Histogramme, können zur Laufzeit geändert werden. Das Clientprogramm liest beim Start die Konfigurationsdatei ein und verarbeitet die darin gespeicherten Einstellungen. Einstellungen, die zur Laufzeit verändert werden können, kann der Benutzer in einem separaten Kontrollfenster (Hauptfenster des Clientprogramms) vornehmen. Beispielsweise können zur Laufzeit die Detetektoren im Clientprogramm an- und abgeschaltet werden. Beim Anschalten werden alle Fenster, die zu diesem Detektor gehören, geöffnet und die darin befindlichen Histogramme gezeichnet und fortlaufend aktualisiert. Bei Abschalten eines Detektors werden alle dazugehörigen Fenster geschlossen. Die Konfiguration mit der Konfigurationsdatei muss in einer Art und Weise geschehen, dass auch nicht so versierte Anwender Konfigurationsänderungen vornehmen können.

Des weiteren muss das Clientprogramm einige wichtige Zusatzfunktionen bieten. Beispielsweise muss es eine Funktion geben, die alle momentan geöffneten Histogramme auf der Festplatte speichert (Snapshot-Funktion).
Darüber hinaus muss es mit dem Clientprogramm möglich sein, das Serverprogramm zu kontrollieren (Starten, Stoppen und Zurücksetzen des Serverprogramms).
Beim Programmstart wird die Konfigurationsdatei eingelesen und die Software entsprechend der Vorgaben aus der Konfigurationsdatei initialisiert. Dabei müssen die einzelnen Elemente der Konfigurationsdatei sofort im Programm abgebildet werden. Anschließend baut der Client eine Verbindung zum Histogrammserver auf. Danach kann der Benutzer die konfigurierten Detektoren anschalten und das Online-Monitoring beginnen.

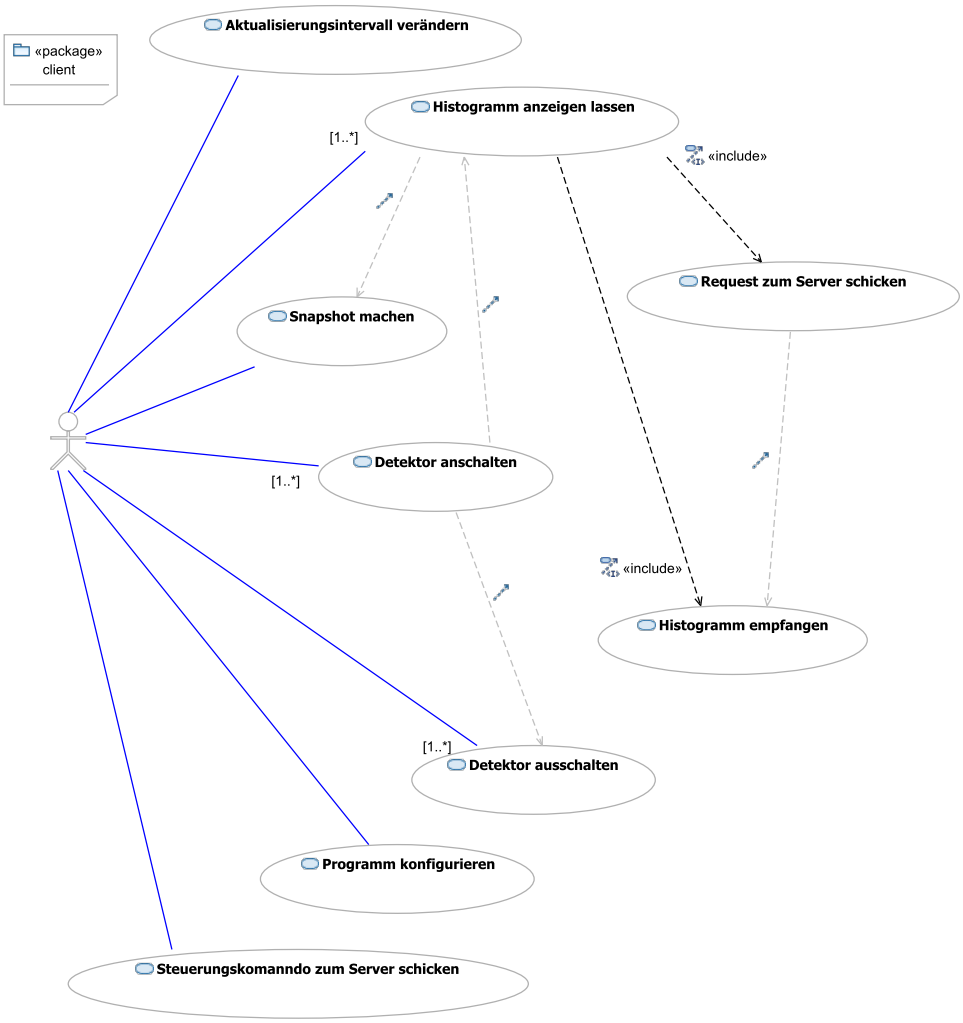

Abbildung 3.1: Use-Case-Diagramm des Clientprogrammes. Das Strichmännchen stellt den Programmbenutzer dar.

3.3 Client-Server-Kommunikation

Das Clientprogramm läuft auf beliebigen Computern im Netzwerk. Es kommuniziert über das Netzwerk mit dem Server. Dabei werden auf Client- und Serverseite TCP/IP-Sockets verwendet. Über diese Verbindung werden die Histogramme vom Server zum Client übertragen. Der Client kann dann die empfangenen Histogramme anzeigen. Für die Übertragung kann entweder die **Push-** oder die **Pull-Technologie** verwendet werden. Bei der **Push-Technologie** sendet der Server von sich aus die Histogramme zum Client, sobald diese auf dem Server neu generiert bzw. aktualisiert worden sind. Der Server initiiert damit die Histogrammübertragung und entscheidet somit darüber, wann der Client aktualisierte Histogramme erhält.
Bei der **Pull-Technologie** entscheidet der Client, wann er Histogramme vom Server empfangen möchte. Dazu sendet der Client eine Anfrage für ein neues Histogramm zum Server. Der Server beantwortet diese Anfrage mit dem geforderten Histogramm. Die Kontrolle über die Histogrammübertragung und damit über die Aktualisierung der Histogramme liegt damit vollständig beim Client.
Im Gegensatz zur alten Software wird in der neuen Software die **Pull-Technologie** verwendet. Dies hat gegenüber der Push-Technologie in der vorliegenden Situation gewisse Vorteile. Im Fall der Push-Technologie kann es passieren, dass der Server in so kurzen Zeitabständen Histogramme an den Client sendet, dass dieser nicht ausreichend Zeit hat, ein empfangenes Histogramm zu zeichnen, bevor das nächste Histogramm eintrifft. Die Folge ist ein Absturz/Einfrieren der Client-Software, da der Client mit dem Zeichnen der Histogramme nicht nachkommt. Bei Verwendung der Pull-Technologie ist diese Situation ausgeschlossen. Da der Client auf die Beantwortung oder ein Timeout einer Anfrage wartet, bevor er eine neue Anfrage zum Server sendet, passen sich die „Geschwindigkeiten “des Client und des Servers an. Daraus entsteht jedoch direkt ein Nachteil, der in einer zukünftigen Version der Software durch den Einbau eines Bandbreitenmanagements behoben werden kann. Da sich die „Geschwindigkeit “des Client an die des Servers anpasst, kann es bei vielen gleichzeitig verwendeten Clients passieren, dass die Antwortzeit des Servers stark nachlässt. Dann werden gleichzeitig alle Clients ausgebremst, da diese sich an die Bearbeitungsgeschwindigkeit des Servers anpassen. Durch ein intelligentes Bandbreitenmanagement (Priorisierung der Client-Requests nach beispielsweise Herkunft/IP-Adresse) lassen sich solche Situationen steuern. In der gegenwärtigen Version beeinhaltet die Software keinerlei Bandbreitenmanagement, was zur gleichen Problematik führen kann, wie die aus dem Internet bekannten Denial-of-Service-Angriffe.

3.4 Konfiguration

Eine Konfigurationsmöglichkeit dient der flexiblen Anpassung der Software an die verschiedenen Einsatzsituationen. Damit ist es möglich die Software mit verschiedenen Einstellungen laufen zu lassen, ohne den Quelltext des Programms zu ändern und neu kompilieren zu müssen. Alle Benutzer können somit eine identische Version der Software benutzen, ohne auf eigene Anpassungen verzichten zu müssen. Sowohl Client- als auch Serverprogramm sollen jeweils separat konfigurierbar sein. Konfiguriert werden sowohl Server- als auch Clientprogramm über Konfigu-

rationsdateien. Beim Serverprogramm müssen die Histogrammdefinitionen konfigurierbar sein. Die Konfiguration des Serverprogramms wird an zentraler Stelle vorgenommen und wird in der Regel nur von spezialisiertem Personal durchgeführt. Das Clientprogramm dagegen kann von seinen Benutzern angepasst werden. Die Konfiguration sollte möglichst flexibel und komfortable durchführbar sein. Einfache Betriebsparameter wie der Hostname/die IP-Adresse und der Port des Histogrammservers können eingestellt werden. Darüber hinaus sollen auch komplexe Einstellungen wie die Struktur der graphischen Oberfläche vorgenommen werden können.
In der Konfigurationsdatei des Clientprogramms wird unter anderem festgelegt, welche Detektoren beobachtet werden sollen, wie viele Fenster die Detektoren besitzen, wie viele Tabs die einzelnen Fenster haben, wie viele Canvases die Tabs/Fenster haben und in wie viele Segmente die Canvases unterteilt werden sollen. Zusätzlich wird festgelegt, welches Histogramm in welches Canvas/Canvassegment gezeichnet werden soll. Außerdem muss in der Datei ein Hauptfenster definiert sein. Dieses dient u.a. zur Steuerung der Fenster der Detektoren. Dadurch ist es möglich, die gesamte graphische Oberfläche mit Hilfe der Konfigurationsdatei festzulegen. Des weiteren werden die Betriebsparameter wie z.B. die IP-Adresse/der Hostname des Histogrammservers in der Konfigurationsdatei angegeben.
Die Vorteile dieses Verfahrens liegen auf der Hand: Es existiert nur eine Version des Clientprogramms. Auf verschiedenen Computern können mit verschiedenen Konfigurationsdateien unterschiedliche Benutzeroberflächen des Clientprogramms verwendet werden. Auch die Wahl der Histogramme, die angezeigt werden sollen, lässt sich für jede Instanz des Clientprogramms festlegen.

Kapitel 4

Implementierung der Online-Monitoring Software

Vorgabe für die Implementierung der beschriebenen Software war die Verwendung des Frameworks ROOT für die Netzwerkkommunikation, die Threads und die Histogramme unter dem Betriebssystem Linux. Daneben sollten keine weiteren Softwarebibliotheken oder Frameworks verwendet werden. Aus diesem Grund wurden auch für die graphischen Oberflächen die entsprechenden GUI-Klassen von ROOT verwendet.

4.1 Server

Das Serverprogramm wurde zum größten Teil aus der alten Online-Monitoring-Software übernommen. Es wurden an einigen Stellen kleinere Anpassungen vorgenommen und eine **neue** Komponente entwickelt, die die Kommunikation von Server und Client über das Netzwerk bereitstellt. Die alte Netzwerk-Komponente wurde entfernt. Die Funktionen zur Konfiguration des Servers, sowie zur Definition, Erzeugung und Aktualisierung der Histogramme wurden aus der alten Software nahezu unverändert übernommen. Daher werden diese Funktionen hier nicht weiter beschrieben.

HOnlineClientServerCom

Die **neue** Klasse `HOnlineClientServerCom` beinhaltet alle Client-Server-Komponenten, d.h. alle Funktionen zur Kommunikation von Client und Server. Sie ist in die vorhandene Online-Monitoring-Bibliothek libOnline integriert.
Es wurden Funktionen zum Übertragen von Text, Histogrammen und einer Liste vom Server zum Client erstellt: `sendHistToClient`, `sendListToClient`, `sendTextToClient`. Gedacht sind die Funktionen zum Senden eines Histogramms, zum Senden einer Liste mit den Namen aller auf dem Server verfügbaren und damit vom Client abrufbaren Histogramme, und zur Übermittlung eines Textes - beispielsweise einer Fehlermeldung.
Die Funktion `init` dient der Initialisierung der Client-Server-Kommunikation. Sie erstellt den

Serversocket und das Monitor-Objekt. Das Monitor-Objekt beobachtet die Aktivität von Sockets. Mit Hilfe des Monitor-Objektes kann man abfragen, welche Sockets gerade auf Beantwortung von Anfragen warten.
Eine Funktion `processClientRequests` kategorisiert anstehende Client-Anfragen. Dazu wird mit Hilfe des Monitor-Objektes eine Liste mit allen auf Antwort wartenden Sockets generiert und anschließend jede Anfrage in der Liste bearbeitet. Dabei muss unterschieden werden zwischen Histogramm-Anfragen und Verbindungsanfragen. Verbindungsanfragen entstehen immer beim Aufbau und beim Abbau einer Netzwerkverbindung zwischen Clientprogramm und Serverprogramm. Verbindungsanfragen werden direkt beantwortet und die Verbindung wird entsprechend auf- oder abgebaut. Besteht die Client-Anfrage aus einem String, wird die Anfrage als Histogrammanfrage oder als Übertragung eines Steuerungskommandos an den Server gewertet. In beiden Fällen wird die Anfrage in die Map `clientRequests` eingetragen. Der Schlüssel der Map ist dabei entweder das Steuerungskommando oder der Name des angeforderten Histogramms, der Wert ist der entsprechende Socket des Client, an den die Antwort gesendet werden soll. Eine Unterscheidung zwischen Steuerungskommando und Histogrammanfrage findet zu diesem Zeitpunkt nicht statt. Auch die Beantwortung der Steuerungskommandos und Anfragen in der Map `clientRequest` werden an dieser Stelle noch nicht vorgenommen.

HOnlineServer

Die HOnlineServer-Klasse ist weitgehend unverändert aus der alten Software übernommen worden. Lediglich die `eventLoop`-Funktion ist angepasst worden. Die Klasse ist ebenfalls in die vorhandene Online-Monitoring-Bibliothek libOnline integriert. Die `eventLoop`-Funktion holt bei jedem Aufruf das nächste Event aus den hld-Daten. Falls es kein Event mehr gibt, wird die `eventLoop`-Funktion beendet und der Wert -1 zurückgegegeben. Andernfalls werden die Histogramme aktualisiert und jedes verfügbare Histogramm zusätzlich in einen Pool eingetragen (`histpool`). Anschließend werden die zu dem Zeitpunkt vorliegenden Client-Anfragen abgearbeitet. Dazu wird die Funktion `HOnlineClientServerCom::processClientRequests` aufgerufen. Diese bearbeitet alle Verbindungsanfragen (Auf- und Abbau) und fügt alle anderweitigen Anfragen (Histogramm- und Steuerungsanfragen) in die dafür vorgesehene Map ein. In der `eventLoop`-Funktion werden die einzelnen Elemente der Map in einer Schleife abgearbeitet. Beginnt der Schlüssel eines Elements mit `CMD`, handelt es sich um ein Steuerungskommando für den Server. Gegenwärtig sind nur die Kommandos `stop` und `list` implementiert. Das `stop`-Kommando bricht die gesamte Event-Schleife ab und stoppt den Server. Das `list`-Kommando sendet eine Liste mit den Namen aller auf dem Server verfügbaren Histogramme zu dem Client, der das Kommando gesendet hat. Dazu wird der Histogrammpool, in dem sich alle Histogramme befinden, komplett durchlaufen und die Namen der Histogramme werden extrahiert. Falls der Schlüssel des Map-Elementes nicht mit `CMD` beginnt, wird die Anfrage als Histogrammanfrage behandelt. Der Schlüssel ist dann der Histogrammname. Aus dem Histogrammpool wird das Histogramm mit dem dem Schlüssel entsprechenden Namen herausgesucht und zum Client übertragen. Falls kein passendes Histogramm gefunden werden kann, wird eine Fehlermeldung zum Client gesendet. Die Übertragung der Histogramm-Namensliste, des Histogramms oder einer Fehlermeldung übernehmen die send-Funktionen der Klasse `HOnlineClientServerCom`.

hadesonlineserver

`hadesonlineserver` ist das Rahmenprogramm für das Serverprogramm. Beim Aufruf müssen 4 Argumente angegeben werden. Argument 1 ist ein frei wählbarer Name des Servers, Argument 2 ist der Hostname des Servers, Argument 3 ist der Port, auf dem das Serverprogramm laufen soll und Argument 4 ist die Konfigurationsdatei für den Server. Nach einer Initialisierung beginnt eine Schleife, die so oft die `eventLoop` der Klasse `HOnlineServer` aufruft, bis diese den Wert -1 zurückgibt. In diesem Fall wird die Schleife abgebrochen und anschließend das Programm beendet.

4.2 Client

Das Clientprogramm dient der benutzerseitigen Aufbereitung der Dienste, die der Server anbietet. Dabei können die angebotenen Dienste benutzerspezifisch konfiguriert werden. Realisiert werden die Clientfunktionen in mehreren Klassen, die hierarchisch angeordnet sind.
Die Netzwerkverbindung basiert auf TCP/IP-Sockets, die die entsprechenden ROOT-Klassen zur Verfügung stellen. Die Hauptklasse besitzt drei parallele Handlungsstränge:

1. Fortlaufende Aktualisierung der Histogramme
2. Fortlaufende Aktualisierung der Canvases
3. Reaktion auf Benutzer-Ereignisse

Daraus ergibt sich die zwingende Verwendung von Threads. Die Aktualisierung der Histogramme wird in einen separaten Thread ausgelagert, die Aktualisierung der Canvases geschieht im Hauptthread in einer Endlos-Schleife, die auch gleichzeitig die Bearbeitung möglicher Benutzerer-Eignisse anstößt. Die weitere Bearbeitung der Benutzerereignisse übernehmen anschließend root-interne Funktionen.
Sowohl die Histogramme, als auch die Canvases müssen in der Hauptklasse registriert werden durch die Eintragung in eine entsprechende Liste. Bei der Aktualisierung der Histogramme und der Canvases wird die entsprechende Liste durchlaufen und alle Elemente in der Liste werden aktualisiert.

Für die Abbildung der Struktur aus der Konfigurationsdatei in das Programm wurden sogenannte Wrapper-Klassen entwickelt. Die Wrapper-Klassen bilden die Struktur der Fenster, Tabs, Canvases und Histogramme aus der Konfigurationsdatei im Programm ab. Die einzelnen ROOT-GUI-Elemente werden innerhalb der Wrapper-Klassen verwaltet. Beim Schließen eines Fensters werden alle ROOT-GUI-Elemente des Fensters gelöscht, die eigentliche Struktur des Fensters bleibt jedoch erhalten, da die Instanz der Wrapper-Klasse erhalten bleibt. Es existieren GUI-Wrapper-Klassen für das Hauptfenster (`HMonClientMainWin`), einfache Fenster (`HMonClientSimpleWin`), Fenster mit Tabs (`HMonClientTabWin`), für die Tabs in einem Fenster mit Tabs (`HMonClientTab`) und für die Canvases (`HMonClientCanvas`). Für die Histogramme existiert ebenfalls eine Wrapper-Klasse (`HMonClientHist`). Sie beinhaltet das eigentliche Histogrammobjekt vom Typ `HOnlineMonHistAddon`.

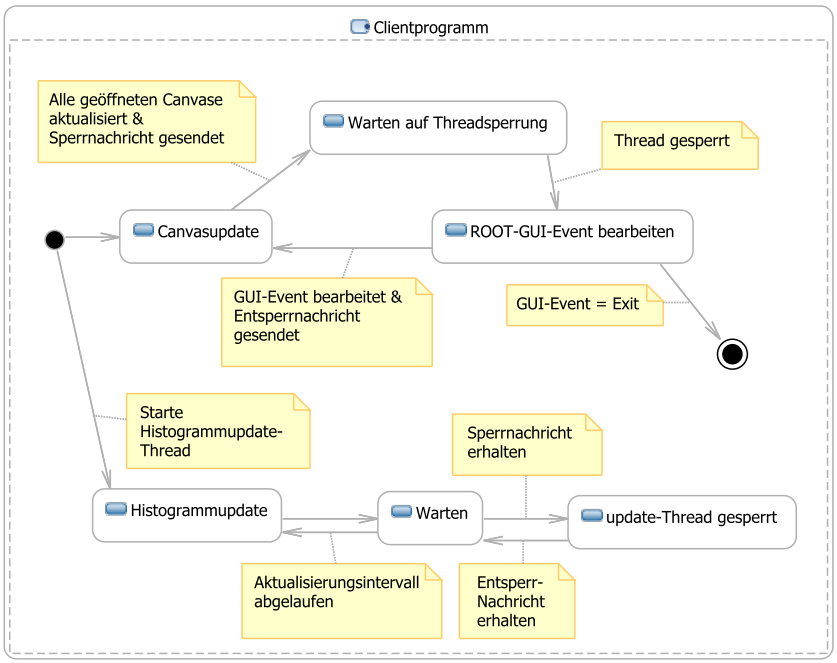

Abbildung 4.1: Vereinfachtes Zustandsdiagramm für das Clientprogramm. Dargestellt sind nur die beiden Update-Prozesse und die GUI-Event-Behandlung.

Die Instanzen der Wrapper-Klassen werden unmittelbar nach der Verarbeitung der Konfigurationsdatei nach Programmstart angelegt und erst beim Schließen des gesamten Programms wieder gelöscht.
Im Folgenden sind die einzelnen Klassen jeweils in einer Kurzbeschreibung dargestellt.

HMonClientMain

Die Klasse `HMonClientMain` ist die Hauptklasse des Clientprogramms. Sie wird vom Rahmenprogramm `hmonclient` instanziert und initialisiert. Die Klasse hat folgende Aufgaben:

- Verwaltung der Liste von Histogrammen, die fortlaufend aktualisiert werden
- Verwaltung der Liste von Canvases, die fortlaufend aktualisiert werden müssen
- Verwaltung der Liste der Detektoren
- Möglichkeiten zum An- und Abschalten von Detektoren im Programm
- Netzwerkverbindung auf- und abbauen
- Senden von Anfragen an den Server, Empfangen von Histogrammen vom Server
- Bereitstellung der Event-Handler, um auf Benutzerereignisse reagieren zu können

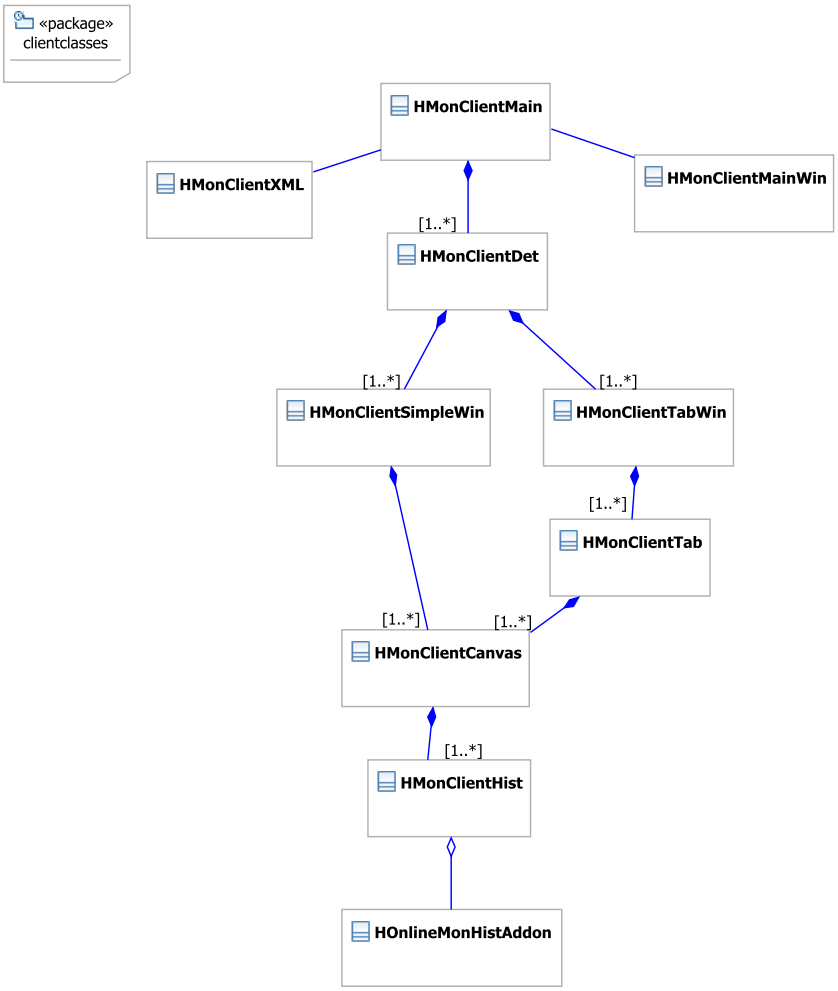

Abbildung 4.2: Klassendiagramm des Clientprogramms

Dem Konstruktor der Klasse wird der Name der XML-Konfigurationsdatei übergeben. Anschließend muss die init-Methode aufgerufen werden. Diese unterscheidet anhand der übergebenen Parameter zwischen den zwei Arten der Benutzung. Die erste Art ist die Übertragung eines Steuerungskommandos zum Server, die zweite ist die Verwendung für das Online-Monitoring. Die Parameter `host`, `port` und `cmd` sind nur für die Übertragung von Steuerungskommandos relevant.

```
1 void HMonClientMain::Init(Bool_t sendCmdOnly, TString host, Int_t port, TString cmd)
```

Listing 4.1: Signatur der HMonClientMain Init-Methode

```
1 HMonClientMain *clientmain = new HMonClientMain();
2 clientmain->Init(kTRUE, "histoserver.gsi.de", 9876, "stop");
```

Listing 4.2: Aufruf der HMonClientMain Init-Methode zur Übertragung eines Kommandos

```
1 HMonClientMain *clientmain = new HMonClientMain("Config.xml");
2 clientmain->Init();
```

Listing 4.3: Aufruf der HMonClientMain Init-Methode für das Online-Monitoring

Beim Aufruf der Init-Methode zur Übertragung eines Kommandos wird ein Socket für die Netzwerk-Kommunikation erzeugt und das Kommando mit Hilfe der Funktion `SendCmdToServer` übertragen. Daran anschließend wird der Socket wieder geschlossen und das Programm beendet. Wenn die `Init`-Methode für das Online-Monitoring aufgerufen wird, erzeugt sie eine Instanz der Klasse `HMonClientMainWin` für das Hauptfenster der Anwendung. Anschließend wird die Funktion `CreateClientConfig` aufgerufen. Diese instanziert die `HMonClientXML`-Klasse und verarbeitet die XML-Konfigurationsdatei. Danach wird das Hauptfenster der Anwendung erzeugt. Dazu wird die Methode `CreateMainWin` der Klasse `HMonClientMainWin` aufgerufen. Durch den Aufruf einer Connect-Funktion wird die Netzwerk-Verbindung zum Server hergestellt und nach erfolgreichem Verbindungsaufbau wird der update-Thread gestartet. Dieser Thread ruft beim Start die Methode `ThreadUpdateHists` auf. Sie ist für die fortlaufende Aktualisierung der Histogramme zuständig und beinhaltet zu diesem Zweck eine Endlos-Schleife. In dieser wird die Funktion `UpdateHists` aufgerufen. Sie durchläuft die Liste der Histogramme und ruft ihrerseits für jedes Histogramm die Funktion `UpdateHist` auf und übergibt dieser das Histogramm aus der Liste. `UpdateHist` ermittelt den realen Namen des übergebenen Histogramms und fordert beim Server eine aktuelle Version dieses Histogramms an. Das neue Histogramm wird nach dem Empfan vom Client mit einem Zeitstempel versehen und in den entsprechenden Histogrammtyp umgewandelt. Das konvertierte Histogramm wird anschließend im dazugehörigen Canvas gezeichnet.
Nachdem der Thread gestartet wurde, tritt das Programm in eine Endlos-Schleife ein. In der Schleife wird zunächst die Bearbeitung möglicher Benutzerereignisse aus der ROOT-Event-Schleife angestoßen (`gSystem->ProcessEvents()`). Nach der Sperrung des update-Threads werden die Canvases und Subpads aus der Liste aktualisiert. Anschließend wird der update-Thread wieder freigegeben. Diese Endlos-Schleife läuft parallel zum update-Thread und wird erst bei Programmende verlassen.

HMonClientMainWin

Die Klasse `HMonClientMainWin` bildet das Hauptfenster zur Steuerung des gesamten Clientprogramms. Das Hauptfenster ist in mehrere Bereiche unterteilt. Jeder Bereich fasst die inhaltlich zusammengehörigen Bedienelemente zusammen. Im oberen Bereich gibt es einen Abschnitt „detectors“. Er beinhaltet die Steuerung für die konfigurierten Detektoren. Für jeden Detektor existiert eine Checkbox. Sie dient zum An- und Abschalten des Detektors. Darunter befindet sich der Bereich zum Einstellen der Aktualisierungsrate der Histogramme. Der dritte und bisher letzte Bereich dieses Fensters beinhaltet einen Button zur Erzeugung von Snapshots. Das Hauptfenster wird nach der Initialisierung des

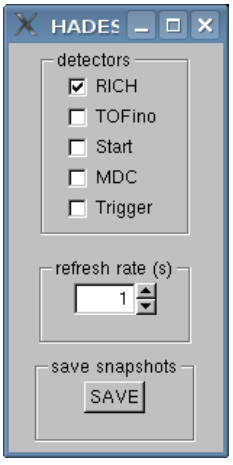

Abbildung 4.3: Hauptfenster

Client-Programms erzeugt und auf dem Bildschirm angezeigt.
Beim Schließen des Hauptfensters wird das gesamte Programm beendet. Die Klasse besteht im Wesentlichen aus den Elementen für die graphische Oberfläche und deren Verwaltung. Die Methode `CreateMainWin` erzeugt das Hauptfenster und zeigt es auf dem Bildschirm an. Für jeden Detektor in der Detektorliste der Hauptklasse wird im Detektor-Bereich eine Checkbox angelegt. Anschließend werden die Elemente für die Aktualisierungsrate und die Snapshot-Funktion erzeugt. Die Elemente der graphischen Oberfläche sind teilweise mit Event-Handlern verknüpft, um auf Benutzerereignisse reagieren zu können. Alle Event-Handler befinden sich in der Hauptklasse. Die Checkboxes der Detektoren sind mit dem Event-Handler `HandleControlButtonClick` verknüpft, die Box zum Einstellen der Aktualisierungsrate mit dem Event-Handler `HandleRefreshRateChange` und der Snapshot-Button mit `HandleSnapshotSave`.

HMonClientDet

Die Klasse `HMonClientDet` dient der Abbildung von Detektoren in der Programmstruktur. Für jeden in der Konfigurationsdatei definierten Detektor wird eine Instanz von `HMonClientDet` erzeugt. Die Instanz wird beim Parsen der Konfigurationsdatei angelegt und erst bei Programmende wieder gelöscht. Jeder Detektor besitzt eine Liste von Fenstern für die Darstellung der Histogramme. Diese Fenster werden für jeden Detektor in der Liste `listWindows` verwaltet. Zum An- und Abschalten der Detektoren besitzt die Klasse die Funktionen `CreateDet` und `DestroyDet`. Diese durchlaufen die Liste der Fenster und rufen ihrerseits für jedes Fenster in der Liste die Funktion `CreateWindow` bzw. `DestroyWindow` auf.

HMonClientSimpleWin

`HMonClientSimpleWin` ist eine GUI-Wrapper-Klasse für ein einfaches Fenster mit einer beliebigen Anzahl von Canvases.
Die Canvases der Fenster sind Instanzen der Klasse `HMonClientCanvas` und werden in der Liste `listCanvases` verwaltet. Die Variable `mainFrame` beinhaltet das eigentliche Fenster-Objekt aus der ROOT-GUI-Klasse. Das Fenster-Objekt repräsentiert das Fenster, das auf dem Bildschirm erscheint.
Die Klasse hat außerdem eine Methode zum Erzeugen (CreateWindow) und Zerstören (DestroyWindow) des Fensters. Diese Methoden werden zum Öffnen bzw. Schließen des mainFrame-

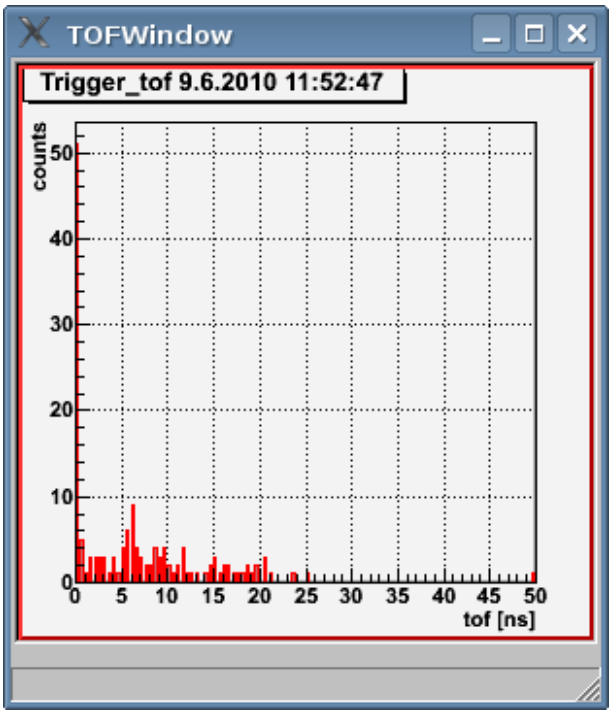

Abbildung 4.4: Einfaches Fenster des Clientprogramms zur Darstellung eines Histogramms.

Fensters aufgerufen.
Die Aufgabe dieser Methoden ist das Anlegen bzw. Zerstören des mainFrame-Objektes. Die Create- und Destroy-Methoden rufen ihrerseits für jedes Canvas aus der Liste die Create- und Destroy-Methoden des Canvas auf. Die Canvases in der Liste werden erst bei Programmende gelöscht.

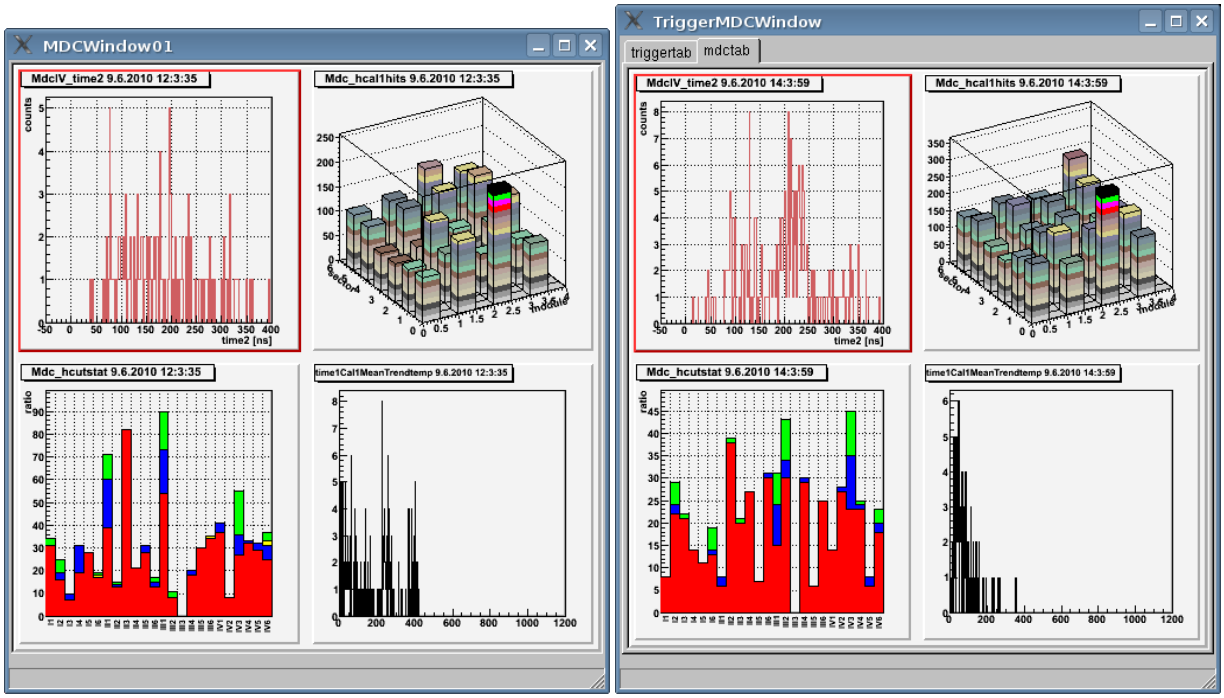

Abbildung 4.5: Links: Einfaches Fenster des Clientprogramms mit einem in mehrerer Segmente unterteilten Canvas. Rechts: Ein Tabfenster mit 2 Tabs. In jedem einzelnen Segment kann ein Histogramm dargestellt werden.

HMonClientTabWin

`HMonClientTabWin` ist eine GUI-Wrapper-Klasse für ein Fenster mit Tabs. Die einzelnen Tabs sind vom Typ `HMonClientTab`. Sie werden in der Liste `listTabs` verwaltet. Die Variable `mainFrame` beinhaltet das eigentliche Fenster-Objekt aus der ROOT-GUI-Klasse. Das Fenster-Objekt repräsentiert das Fenster, das auf dem Bildschirm erscheint. Zusätzlich existieren ein TabContainer und ein TabContainer-Frame, die für die graphische Darstellung der Tabs im Fenster zuständig sind.
Die Klasse besitzt eine Methode zum Erzeugen (`CreateWindow`) und Zerstören des Fensters (`DestroyWindow`). Die Methode `CreateWindow` durchläuft die Liste der Tabs und ruft für jeden Tab die Methode `CreateTab` auf. Zusätzlich wird das `mainFrame`-Objekt angelegt, sowie ein TabContainer- und ein TabContainer-Frame-Objekt, um die Tabs im Fenster auf dem

Bildschirm darstellen zu können. `DestroyWindow` ruft entsprechend für jedes Canvas die Methode `DestroyTab` auf und löscht die Container-Elemente und das mainFrame.

HMonClientTab

`HMonClientTab` repräsentiert einen einzelnen Tab innerhalb eines `HMonClientTabWin`-Fensters. Jeder Tab kann beliebig viele Canvases enthalten. Sie werden in der Liste `listCanvases` verwaltet. Die Klasse besitzt eine Methode zum Erzeugen (`CreateTab`) und Zerstören des Tabs (`DestroyTab`). Die Methode `CreateTab` durchläuft die Liste der Canvases und ruft für jedes Canvas die Methode `CreateCanvas` auf. `DestroyTab` ruft entsprechend für jedes Canvas die Methode `DestroyCanvas` auf.

HMonClientCanvas

Die `HMonClientCanvas`-Klasse stellt ein Canvas für die graphische Oberfläche des Client dar. Jede Instanz von `HMonClientCanvas` kann beliebig viele Histogramme besitzen. Sie werden in der Liste `listHists` verwaltet. Die Klasse beinhaltet das eigentliche Canvas-Objekt, das auf dem Bildschirm erscheint. Die Eigenschaften des Canvas können über entsprechende Get- und Set-Methoden gelesen und verändert werden. Die Werte der Eigenschaften nx bzw. ny sind ganze Zahlen und stellen die Unterteilung (split) des Canvas in Segmente in x- bzw. y-Richtung dar.

HMonClientHist

Die Klasse `HMonClientHist` ist eine Histogramm-Wrapper-Klasse für das eigentliche Histogrammobjekt. Die Klasse hat die Aufgabe, Histogrammdefinitionen aus der Client-Konfigurationsdatei in eine Klasse abzubilden. Für jede Histogrammeigenschaft, die in der Konfigurationsdatei angegeben werden kann, existiert in der Klasse eine entsprechende Eigenschaft. Das eigentliche Histogrammobjekt ist in der Variablen `histo` vom Typ `HOnlineMonHistAddon` untergebracht. Es existieren jeweils Get- und Set-Methoden für die verschiedenen Eigenschaften der Klasse. Alle `HMonClientHist`-Objekte im Clientprogramm, die in der Konfigurationsdatei angegeben sind, werden für die Histogrammaktualisierung in der Hauptklasse `HMonClientMain` in eine Liste eingetragen. Diese Liste verwaltet die zu aktualisierenden Histogramme. Es ist erforderlich eindeutige Namen für jedes Objekt in der Liste zu verwenden. Andernfalls kann man die Objekte in der Liste nicht unterscheiden. Wenn sich ein Benutzer das gleiche Histogramm in zwei verschiedenen Fenstern anzeigen lassen möchte, wird das Histogramm zweimal in die Liste eingetragen, da beide Histogramme aktualisiert werden müssen. Hätte das Histogramm jeweils den gleichen Namen, könnte man die Objekte in der Liste nicht mehr voneinander unterscheiden. Um diese Problematik zu vermeiden, hat jedes Histogramm zwei verschiedene Namen. Die Variable `realname` beinhaltet den realen Namen des `HOnlineMonHistAddon`-Histogramms. Der Name entspricht genau dem Namen, der in der Histogrammdefinition auf dem Server angegeben ist. Dieser Name muss auch in der Client-Konfigurationsdatei als Name des Histogramms angegeben werden. Darüber hinaus hat jede Instanz der Klasse eine Namensvariable

aus der Veerbung von `TNamed`. Der Wert dieser Namensvariable ist eine Kombination aus dem Canvasnamen und dem realen Histogrammnamen. Die Kombination hat immer die Form *canvasname_subpadnumber_realerHistogrammname*. Da die Namen der Canvases per definitionem eindeutig sein müssen, ist damit die Eindeutigkeit des Histogrammnamens gewährleistet. Die Hauptklasse verwendet daher bei der Eintragung eines Histogrammes in die Liste der zu aktualisierenden Histogramme diesen Namen. Für die Übermittlung einer Anfrage zum Server zur Aktualisierung eines Histogrammes wird der reale Histogrammname verwendet, damit der Server das richtige Histogramm zurücksenden kann. Der Server hat keinerlei Kenntnis über den aus Canvasnamen und realem Namen kombinierten Histogrammnamen.

HMonClientXML

Die Klasse `HMonClientXML` stellt alle benötigten Methoden zur Verfügung, um die XML-Konfigurationsdatei einzulesen und zu verarbeiten. Neben dem Konstruktor existiert nur eine weitere öffentliche Funktion. Sie heißt `ParseXMLFile` und erwartet als Argument den Dateinamen der XML-Datei und einen Zeiger auf die Instanz der Hauptklasse. Für jeden Knoten in der baumartigen Struktur der XML-Datei existiert jeweils eine private Funktion zum Parsen des entsprechenden Knotens. Es gibt folgende Knoten in der XML-Datei: `config`, `server`, `MainWindow`, `detector`, `window`, `tab`, `canvas`, `histogram`. Die Parse-Funktionen haben die Aufgabe, die Informationen aus der XML-Datei zu extrahieren und diese im Programm abzubilden. Beim Aufruf der Funktion `ParseXMLFile` wird mit Hilfe eines XML-Parser-Objekts vom Typ `TDOMParser` die XML-Datei verarbeitet und ein XML-Document-Objekt vom Typ `TXMLDocument` erzeugt. Das XML-Document-Objekt beeinhaltet die Informationen aus der XML-Datei als Baumstruktur. Dabei wird jeder Knoten in eine Instanz vom Typ `TXMLNode` überführt. Mit den Methoden `GetChildren` und `GetNextNode` der Klasse `TXMLNode` kann das XML-Document-Objekt durchlaufen werden. Um den Namen und den Wert eines Knotens zu ermitteln, existieren in der Klasse `TXMLNode` die Funktionen `GetNodeName` und `GetText`. Für die Verarbeitung der XML-Datei wird das XML-Document-Objekt vollständig durchlaufen. Startpunkt ist der Root-Knoten `<client>`. Anschließend werden die Knoten der nächsten Ebene (child nodes) ermittelt und durchlaufen. Die Ebene besteht aus dem `config`-Knoten, dem `MainWindow`-Knoten und beliebig vielen `detector`-Knoten. Für jeden ermittelten Knoten wird die entsprechende Parse-Funktion aufgerufen (`ParseConfigNode`, `ParseMainWindo` `ParseDetectorNode`), je nachdem welchen Namen der Knoten hat. Der Name des Knotens wird mit der Funktion `GetNodeName` der Klasse `TXMLNode` ermittelt. Zusätzlich wird für jeden so gefundenen `detector`-Knoten ein Detektor-Objekt vom Typ `HMonClientDet` angelegt und in die Liste der Detektoren `listDetectors` der Hauptklasse eingefügt. Die drei aufgerufenen Parse-Funktionen ermitteln ihrerseits wiederum ihre nächste Ebene von Knoten und rufen die entsprechenden Parse-Funktionen auf. So ermittelt die Funktion `ParseDetectorNode` die Knoten `name`, `title` und `window`. `name` und `title` sind Endknoten, d.h. Knoten ohne eine nächste Ebene. Bei einem Endknoten muss der Wert, der zwischen Anfangs- und Endtag steht, extrahiert und der entsprechenden Eigenschaft des Objektes zugewiesen werden. In diesem Fall müssen den Eigenschaften `name` und `title` des Detektor-Objektes die aus den Endknoten `name` und `title` extrahierten Werte zugewiesen werden. Der `window`-Knoten ist kein End-

knoten, daher wird die Funktion `ParseWindowNode` aufgerufen. Dieser Vorgang setzt sich fort, bis die Baumstruktur des XML-Document-Objektes vollständig durchlaufen ist. Nach erfolgtem Durchlauf sind alle Instanzen der jeweiligen Klassen erzeugt und in die entsprechenden Listen der Klassen eingetragen worden.

Das Rahmenprogramm hmonclient

`hmonclient` ist das Rahmenprogramm für das Clientprogramm. Es kann auf zwei verschiedene Arten genutzt werden. Wenn das Clientprogramm zum Senden eines Steuerungskommandos an den Server genutzt werden soll, muss das Clientprogramm mit 3 Argumenten aufgerufen werden. Das erste Argument ist der Hostname des Servers, der zweite ist der Port und der dritte ist das Steuerungskommando. Derzeit sind nur die beiden Kommandos `list` und `stop` implementiert. Das Kommando `list` gibt eine Liste von Namen aller auf dem Server verfügbaren Histogramme auf der Kommandozeile aus, das Kommando `stop` stoppt den Server. Bei dieser Art der Nutzung wird keine graphische Oberfläche erzeugt. Nach Ausführung des Kommandos wird das gesamte Programm beendet.
Der Standardfall ist die Verwendung der graphischen Oberfläche zur Betrachtung der Histogramme. Dazu wird das Programm mit nur einem Argument aufgerufen. Das Argument ist der Name der XML-Konfigurationsdatei.

```
1 $ hmonclient.exe histoserver.gsi.de 9876 list
```

Listing 4.4: Aufruf des Clientprogramms zur Übertragung des list-Kommandos

```
1 $ hmonclient.exe ClientConfig.xml
```

Listing 4.5: Aufruf des Clientprogramms zur Verwendung für das Online-Monitoring

In beiden Fällen wird eine Instanz der Klasse `HMonClientMain` erzeugt. Der Konstruktor erwartet eigentlich als Parameter den Dateinamen der Konfigurationsdatei als String. Da jedoch nicht bei beiden Nutzungsarten eine Konfigurationsdatei angegeben wird, ist der Standardwert des Parameters ein leerer String. Nach dem Aufruf des Konstruktors wird die `Init`-Methode der Instanz von `HMonClientMain` aufgerufen, um das Programm zu initialisieren. Dabei werden je nach Fall unterschiedliche Parameter übergeben. Der erste Parameter der `Init`-Methode ist ein boolscher Wert, der angibt, ob ein Steuerungskommando zum Server gesendet werden soll. Falls der Parameter den Wert `kTRUE` hat, müssen 3 weitere Parameter (Hostname, Port und das eigentliche Kommando) mit übergeben werden. Wird die `Init`-Methode gänzlich ohne Parameter aufgerufen, startet das Programm die graphische Oberfläche und erzeugt das Hauptfenster. Wird das Hauptfenster geschlossen, wird das gesamte Programm beendet.

4.3 Konfiguration

Die Konfiguration des Clientprogramms wird mit einer XML-Datei vorgenommen. Die Verwendung von XML bietet folgende Vorteile: Es ist einfach zu handhaben, da es ein textbasiertes Format ist und besitzt im Gegensatz zu einer ASCII-Konfigurationsdatei eine festen Struktur.

XML hat eine baumartige Struktur zur Organisation des Inhaltes. Für jeden Knoten in dieser Baumstruktur wird in XML ein sogenanntes Tag verwendet. In XML kennzeichnet man Tags durch spitze Klammern, in denen der Name des Tags steht. Die Verwendung von Tags zur Organisation des Inhaltes verbessert die Übersichtlichkeit und dient dem schnelleren Verständnis der inhaltlichen Struktur.
Das root-Tag ist das `<client>`-Tag. Sämtliche weiteren Tags stehen innerhalb des root-Tags. Die XML-Datei besitzt verschiedene weitere Tags innerhalb des `<client>`-Tags, die die einzelnen Abschnitte kennzeichnen. Es gibt einen `<config>`-Abschnitt, einen `<MainWindow>`-Abschnitt und beliebig viele `<detector>`-Abschnitte. Im `<config>`-Abschnitt werden allgemeine Parameter für das Programm festgelegt. Zur Zeit sind dies nur die beiden Parameter `<host>` und `<port>`, die den Hostnamen und den Port des Histogrammservers beinhalten. Weitere Parameter sind denkbar und sollten in diesem Abschnitt untergebracht werden. Der nächste Abschnitt ist der `<MainWindow>`-Abschnitt. Er legt fest, wie das Hauptfenster des Client aussehen soll. Derzeit können Name, Titel, Höhe und Breite (Angabe jeweils in Pixeln) angegeben werden. Danach können beliebig viele Detektoren mit Hilfe der `<detector>`-Tags definiert werden. Es gibt keine Limitierung für die Anzahl der Detektoren. Für jeden Detektor muss mindestens ein Fenster (`<window>`) angegeben werden. Jedes Fenster hat einen Namen, einen Titel und ein Tag `<tabbed>`. `<tabbed>` ist ein boolscher Wert, der festlegt, ob das Fenster ein einfaches Fenster oder ein Fenster mit Tabs sein soll. Dementsprechend sind hier die gültigen Werte `true` oder `false`. Darüber hinaus kann jedes `<window>` Canvases oder Tabs besitzen. Diese beiden Optionen schließen sich gegenseitig aus. Einfache Fenster besitzen ein oder mehrere Canvases, Tabfenster ein oder mehrere Tabs. Es ist darauf zu achten, dass der Wert des `<tabbed>`-Tags konsistent ist mit der Angabe von Tabs oder Canvases. Bei tabbed=true müssen ein oder mehrere Tab-Definitionen folgen, bei tabbed=false müssen ein oder mehrere Canvas-Definitionen folgen. Ein Tab wiederum hat einen Namen, einen Titel und beliebig viele Canvases. Ein Canvas hat einen Namen, Höhe und Breite (Angabe in Pixeln), die Eigenschaft `splitted` mit gültigen Werten true oder false, die Anzahl der Unterteilungen (splits) in x- und y-Richtung (nx, ny) und beliebig viele Histogramme. Ein Histogramm hat einen Namen, einen Typ und ein Canvassegment (subpad), in das es gezeichnet werden soll. Bei einem Canvas, das nicht in Segemente unterteilt ist, muss die Eigenschaft subpad den Wert 0 haben, bei einem unterteilten Canvas werden die einzelnen Segmente beginnend mit 1 durchnummeriert. Die Nummerierung erfolgt zeilenweise. Subpad 1 ist das Segment links oben, Subpad 2 ist das Segment rechts daneben. Der Histogrammtyp gibt an, ob es sich um ein einfaches Histogramm oder um ein Histogramm aus einem Histogrammarray handelt. Bei einem Histogramm aus einem Array kann entweder das einzelne Histogamm aus dem Array gezeichnet werden (bei beiden Array-Indizes müssen gültige Werte angeben werden) oder alle Histogramme aus dem Array (bei beiden Indizes muss -1 angeben werden).
Die verschiedenen Tags und Eigenschaften müssen entsprechend der Vorgaben geschachtelt werden.

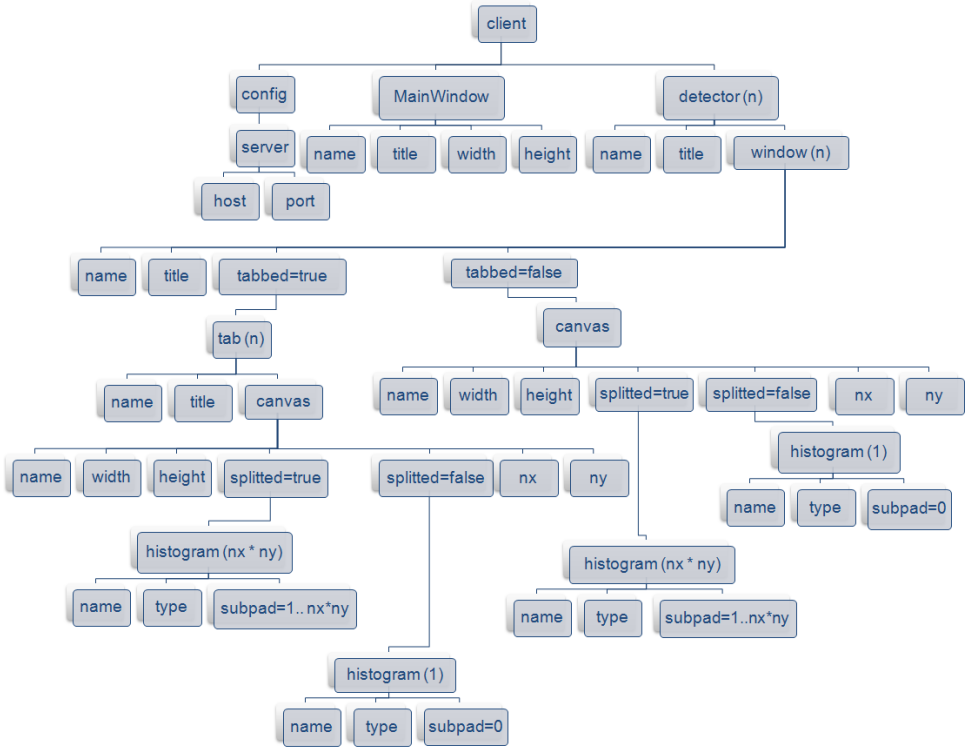

Abbildung 4.6: Baumstruktur der XML-Konfigurationsdatei des Clientprogramms. Die Angaben in Klammern geben an, wie häufig das Element vorkommen kann (n=beliebig). Die Angabe beim subpad-Element bedeutet: Ein Wert zwischen 1 und nx*ny.

Kapitel 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

Die Entwicklung einer multi-threaded Client-Server-Anwendung mit ROOT hat sich als realisierbar herausgestellt. Das ROOT-Framework ist ein universelles Framework und deckt neben den Anforderungen an mathematische und physikalische Phänomene auch nahezu alle Systemfunktionen, wie Sockets, Threads und graphische Oberflächen ab. Die ROOT-Klassen bieten oftmals leider nicht die Qualität und Funktionsvielfalt anderer Frameworks, die sich jedoch nur auf gewisse Teilaspekte spezialisiert haben. Die ROOT-Dokumentation in Form eines Users Guide ([RoG10]) reicht für Softwareprojekte der vorliegenden Art nicht aus, da viele Sachverhalte nicht detailliert genug erklärt werden. Daher ist oftmals ein Blick in die Klassenreferenz ([RoR08]) und den Quellcode vonnöten.
Mit dieser Arbeit ist eine neue Software für das Online-Monitoring entstanden. Sie hat im Vergleich zur alten Software weniger Quellcode und besitzt eine übersichtliche Klassen-Struktur. Eine Weiterentwicklung der Anwendung wird dadurch vereinfacht.

5.2 Ausblick

Die Software ist nicht „fertig“. Es ergab sich eine zeitliche Begrenzung dadurch, dass es ein Thema für eine Bachelor-Arbeit ist. Einige Funktionen **müssen** noch überarbeitet werden (u.a. auch Anpassungen nach Erfahrungen in der Praxis), viele Erweiterungen **können** noch entwickelt werden. Die clientseitigen Steuerungsmechanismen des Servers müssen besser implementiert und noch vervollständigt werden. Dazu gehört auch die Integration der Steuerung in die graphische Oberfläche des Client. Darüber hinaus **muss** ein Bandbreitenmanagement für den Server entwickelt werden, mit Optionen zur Optimierung und Priorisierung der Client-Anfragen. Für die Zukunft ist auch ein automatisiertes Online-Monitoring denkbar. Dazu ist eine Histogrammanalyse innerhalb des Systems erforderlich. Die Software bietet Raum für viele weitere Ideen, Features und Zusätze. Von einem User- und Gruppenmanagement mit Authentifizierung und Berechtigungsrollen bis zur Steuerung der Bandbreite je nach User-Recht und User-Standort und der Steuerung des Servers nur für bestimmte User ist alles denkbar.

Anhang A

XML-Beispielkonfigurationsdatei

```
<?xml version="1.0"?>

<!DOCTYPE client [

    <!-- genau ein config und ein MainWindow, mindestens ein detector -->
    <!ELEMENT client (config, MainWindow, detector+)>

    <!-- config hat genau ein server -->
    <!ELEMENT config (server)>
    <!ELEMENT server (host, port)>
    <!ELEMENT host (#PCDATA)>
    <!ELEMENT port (#PCDATA)>

    <!-- MainWindow must contain name, title, width, height -->
    <!ELEMENT MainWindow (name,title,width,height)>

    <!-- mindestens ein window pro detector -->
    <!ELEMENT detector (name,title,window+)>

    <!-- window must contain name, title, tabbed -->
    <!ELEMENT window (name, title, tabbed, canvas*, tab*)>

    <!-- tab hat mindestens ein Canvas -->
    <!ELEMENT tab (name, title, canvas+)>

    <!-- canvas must contain name, title, width, height, splitted, nx, ny -->
    <!-- mindestens ein Histogramm -->
    <!ELEMENT canvas (name, width, height, splitted, nx, ny, histogram+)>

    <!-- histogram must contain name, title, subpad -->
    <!-- mindestens ein Canvas -->
    <!ELEMENT histogram (name, type, subpad)>


    <!ELEMENT name (#PCDATA)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT width (#PCDATA)>
    <!ELEMENT height (#PCDATA)>
    <!ELEMENT tabbed (#PCDATA)>
    <!ELEMENT splitted (#PCDATA)>
    <!ELEMENT nx (#PCDATA)>
    <!ELEMENT ny (#PCDATA)>
    <!ELEMENT type (#PCDATA)>
    <!ELEMENT subpad (#PCDATA)>
]>
```

```
46
47 <client>
48   <config>
49     <server>
50         <host>histoserver.gsi.de</host>
51         <port>9876</port>
52     </server>
53   </config>
54   <MainWindow>
55     <name>HADESMonitoring</name>
56     <title>HADES Monitoring</title>
57     <width>200</width>
58     <height>400</height>
59   </MainWindow>
60
61   <!-- TOF-Detektor mit 2 einfachen Fenstern ohne Canvas-Unterteilung -->
62   <detector>
63     <name>TOF</name>
64     <title>TOF</title>
65     <window>
66       <name>TOFWIN01</name>
67       <title>TOFWIN01</title>
68       <tabbed>false</tabbed>
69       <canvas>
70         <name>TOFWIN01C01</name>
71         <width>200</width>
72         <height>200</height>
73         <splitted>false</splitted>
74         <nx>0</nx>
75         <ny>0</ny>
76         <histogram>
77           <name>hTriggertof</name>
78           <type>single</type>
79           <subpad>0</subpad>
80         </histogram>
81       </canvas>
82     </window>
83     <window>
84       <name>TOFWIN02</name>
85       <title>TOFWIN02</title>
86       <tabbed>false</tabbed>
87       <canvas>
88         <name>TOFWIN02C01</name>
89         <width>200</width>
90         <height>200</height>
91         <splitted>false</splitted>
92         <nx>0</nx>
93         <ny>0</ny>
94         <histogram>
95           <name>hTriggerTofNumbers</name>
96           <type>single</type>
97           <subpad>0</subpad>
98         </histogram>
99       </canvas>
100    </window>
101  </detector>
102
103 <!-- MDC-Detektor mit einem Tabfenster mit zwei Tabs. Im ersten Tab befindet sich ein in 4
       Segmente unterteiltes Canvas. -->
104  <detector>
105    <name>MDC</name>
106    <title>MDC</title>
107    <window>
108      <name>MDCWIN01</name>
109      <title>MDCWIN01</title>
```

```
110       <tabbed>true</tabbed>
111       <tab>
112         <name>MDCWIN01TAB01</name>
113         <title>MDCWIN01TAB01</title>
114         <canvas>
115           <name>MDCWIN01TAB01C01</name>
116           <width>200</width>
117           <height>200</height>
118           <splitted>true</splitted>
119           <nx>2</nx>
120           <ny>2</ny>
121           <histogram>
122             <name>hMdccutstat</name>
123             <type>single</type>
124             <subpad>1</subpad>
125           </histogram>
126           <histogram>
127             <name>hMdctime2Mod3</name>
128             <type>array:-1:-1</type>
129             <subpad>2</subpad>
130           </histogram>
131           <histogram>
132             <name>hMdccal1hits</name>
133             <type>single</type>
134             <subpad>3</subpad>
135           </histogram>
136           <histogram>
137             <name>hMdcrawRoc_Subev</name>
138             <type>single</type>
139             <subpad>4</subpad>
140           </histogram>
141         </canvas>
142       </tab>
143       <tab>
144         <name>MDCWIN01TAB02</name>
145         <title>MDCWIN01TAB02</title>
146         <canvas>
147           <name>MDCWIN01TAB02C01</name>
148           <width>200</width>
149           <height>200</height>
150           <splitted>false</splitted>
151           <nx>0</nx>
152           <ny>0</ny>
153           <histogram>
154             <name>hMdctime1Cal1MeanTrendtemp</name>
155             <type>single</type>
156             <subpad>0</subpad>
157           </histogram>
158         </canvas>
159       </tab>
160     </window>
161   </detector>
162
163 </client>
```

Listing A.1: XML Beispielkonfiguration

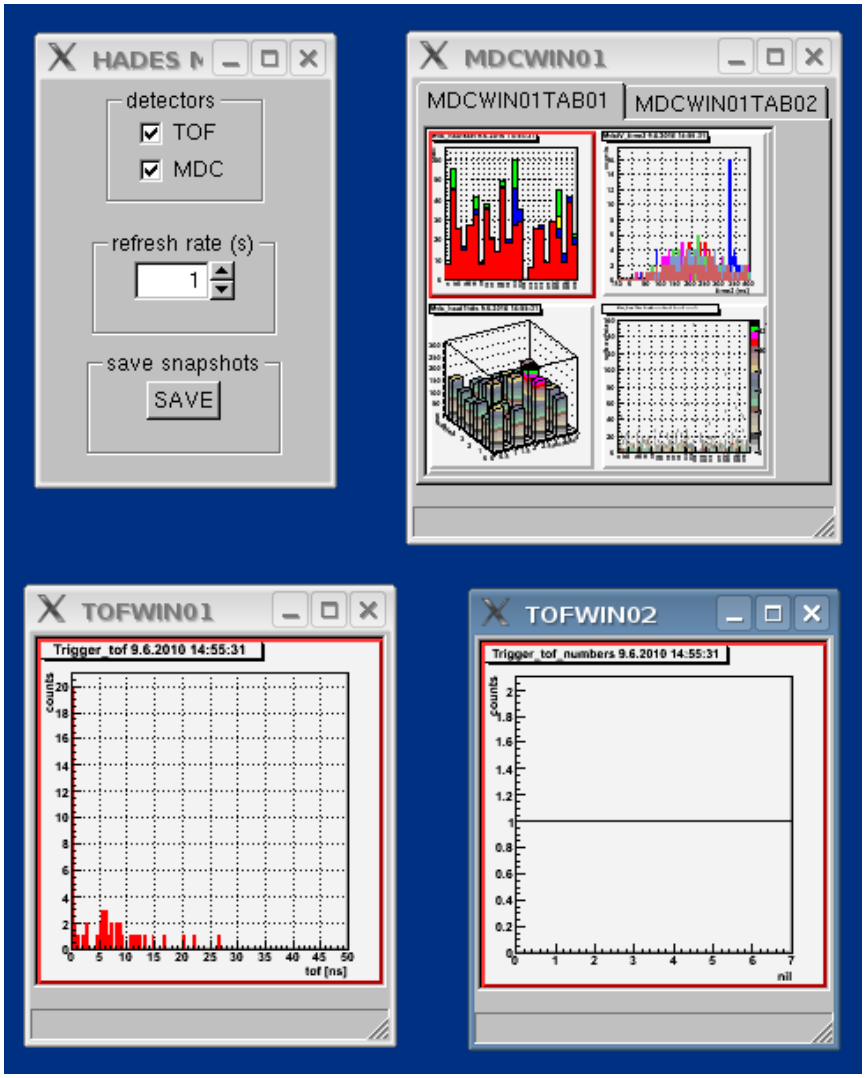

Abbildung A.1: Screenshot des Programmes bei Verwendung der obigen XML-Konfiguration

Literaturverzeichnis

[Aga09] G. et al. Agakishiev. The High-Acceptance Dielectron Spectrometer HADES. *Eur. Phys. J.*, A41:243–277, 2009. 1

[Lor08] M. Lorenz. *Geladene Kaonen Produktion in Ar+KCl Reaktionen bei 1.756 AGeV*. Diplomarbeit, Institut für Kernphysik, Johann Wolfgang Goethe-Universität, Frankfurt am Main, 2008. 2

[RoG10] *ROOT User's Guide, http://root.cern.ch/drupal/content/users-guide*. Onlinedokumentation, CERN, European Organization for Nuclear Research, Genève, 2010. 26

[RoR08] *ROOT Klassenreferenz, http://root.cern.ch/root/html522/ClassIndex.html*. Onlinedokumentation, CERN, European Organization for Nuclear Research, Genève, 2008. 26

[Sch08] A. Schmah. *Produktion von Seltsamkeit in Ar+KCl Reaktionen bei 1.756 AGeV*. Dissertation, Fachbereich Physik, Technische Universität Darmstadt, Darmstadt, 2008. 2