RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

Gabriel Anders

Die Fernüberwachung des ATLAS Level-1 Kalorimeter Triggers

Diplomarbeit

HD-KIP 10-61

KIRCHHOFF-INSTITUT FÜR PHYSIK

Fakultät für Physik und Astronomie
Ruprecht-Karls-Universität Heidelberg

Diplomarbeit
im Studiengang Physik

vorgelegt von
Gabriel Anders
aus Waldbröl
2010

Die Fernüberwachung des ATLAS Level-1 Kalorimeter Triggers

Die Diplomarbeit wurde von Gabriel Anders ausgeführt am
Kirchhoff-Institut für Physik
unter der Betreuung von
Prof. Dr. Hans-Christian Schultz-Coulon

Zusammenfassung

Der störungsfreie Betrieb des Level-1 Kalorimeter Triggers ist wesentlich für die Datennahme des ATLAS-Experiments. Die für die *Online*-Überwachung des ATLAS-Experiments relevanten Daten befinden sich innerhalb des streng abgeschirmten ATLAS-Netzerks im sogenannten *Information Service*. Im Rahmen dieser Arbeit wird die Fernüberwachungsanwendung *L1CaloDisplay* entwickelt, mit der über das Internet der Status des Level-1 Kalorimeter Triggers beobachtet und Fehleranalysen durchgeführt werden könnnen. Als Basis des *L1CaloDisplay* wird ein neues Fernüberwachungssystem konzipiert, das die Abfrage von *Information Service*-Daten über das Internet ermöglicht und auf der Übermittlung der Daten in XML-Darstellung basiert. Die Funktionsfähigkeit des Systems wird anhand eines Prototypen demonstriert. Als bekannt wurde, dass zeitgleich am CERN ein ähnliches Fernüberwachungssystem namens *Web-IS Interface* entstand, wurde dieses als Basis des *L1CaloDisplay* gewählt und mit dessen Entwickler zusammengearbeitet. Das *L1CaloDisplay* ist plattformunabhängig, leicht erweiterbar und kann aus einem *Webbrowser* heraus gestartet werden. Es besitzt einen optimierten XML-Parser und Klassen, die die vollständige Funktionalität des *Web-IS Interface* zur Verfügung stellen. Das *L1CaloDisplay* ist zurzeit die einzige Anwendung mit graphischer Benutzeroberfläche, mit der über das Internet alle im *Information Service* vorliegenden Daten und Histogramme dargestellt und durchsucht werden können und die für alle ATLAS-Physiker zur Verfügung steht.

Abstract

The failure-free operation of the Level-1 Calorimeter Trigger is essential for the data acquisition of the ATLAS experiment. The data relevant for monitoring is located inside the strictly shielded ATLAS network and provided by the *Information Service*. This thesis is concerned with the developement of an application named *L1CaloDisplay*, which allows experts to remotely monitor and diagnose errors in the Level-1 Calorimeter Trigger via the Internet. As the basis of the *L1CaloDisplay* a new remote monitoring framework is designed which permits querying *Information Service* data via the Internet and is based on encoding the trasferred data in XML. The functional ability of the framework is demonstrated by a prototype. During the thesis it emerged that at the same time a similar framework called *Web-IS interface* was developed at CERN which has been used later on as the basis of the *L1CaloDisplay*. The *L1CaloDisplay* is platform independent, easily expandable and can be started using a web browser. It possesses an optimized XML parser and classes which provides all *Web-IS Interface* functionality. At this time the *L1CaloDisplay* is the only application with a graphical user interface which allows a remote user to browse all existent *Information Service* data and histograms and is available to all ATLAS physicists.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Am europäischen Kernforschungszentrum CERN[1] befindet sich der ringförmige Teilchenbeschleuniger *Large Hadron Collider* (LHC), mit dessen Hilfe Antworten auf einige der fundamentalen Fragen der Teilchenphysik gefunden werden sollen. Hierfür bringt er Protonen oder Blei-Kerne mit bisher unerreichten Energien von mehreren TeV zur Kollision. Einer der vier großen Detektoren des LHC ist der Vielzweckdetektor ATLAS, der auf die Untersuchung von Protonen-Kollisionen optimiert ist. Eine seiner Hauptaufgaben ist die Suche nach dem Higgsboson über einen großen Massenbereich und die Entdeckung von neuer Physik jenseits des Standardmodells.
Im LHC kollidieren alle 25 ns zwei Protonenbündel, in jedem dieser sogenannten Ereignisse gibt es ungefähr 20 inelastische Proton-Proton-Wechselwirkungen. Das Speichern und anschließende Verarbeiten aller zugehörigen Informationen ist weder technisch möglich noch sinnvoll, da die meisten Ereignisse physikalisch uninteressant sind. Um die seltenen interessanten Physikprozesse trotz der kleinen Wirkungsquerschnitte mit hoher Effizienz herauszufiltern, wurde ein dreistufiges System entworfen, das ATLAS Trigger- und Datennahmesystem. Die erste Stufe wird als Level-1-Trigger bezeichnet, die zweite und dritte Stufe als *High Level Trigger*. Der Level-1-Trigger reduziert die Ereignisrate auf Basis von Informationen des Myon- und Kalorimetersystems von 40 MHz auf 75 kHz. Innerhalb des Level-1-Triggers ist der Level-1 Kalorimeter Trigger (L1Calo) für das Auslesen und Analysieren der analogen Signale des Kalorimetersystems zuständig.
Das Ziel dieser Arbeit ist die Entwicklung einer Anwendung zur Fernüberwachung des Level-1 Kalorimeter Triggers, die es Experten über das Internet ermöglicht, den aktuellen Status des L1Calo-Systems zu beobachten und bei Auftreten von Problemen deren Ursache zu analysieren.
Die für die *Online*-Überwachung des ATLAS-Experiments relevanten Daten befinden sich innerhalb eines streng abgeschirmten Netzwerks im sogenannten *Information Service*. Da die existierenden ATLAS-Fernüberwachungssysteme in ihrer Funktionalität limitiert waren, wurde als Basis der Fernüberwachungsanwendung ein neues Fernüberwachungssystem konzipiert, das über das Internet lesenden Zugriff auf die Daten des *Information Service* ermöglicht. Im Laufe der Arbeit wurde bekannt, dass zeitgleich am CERN ein ähnliches Fernüberwachungssystem namens *Web-IS Interface* entwickelt wurde, das ab dann als Basis der Fernüberwachungsanwendung diente.
Das Kapitel 2 gibt einen Überblick über den LHC sowie den Aufbau und die Eigenschaften des ATLAS-Detektors. Es wird das Standardmodell der Teilchenphysik vorgestellt und die Suche nach dem Higgsboson mit dem ATLAS-Experiment erläutert.

[1] Die Abkürzung CERN leitet sich von dem ehemaligen franz. Namen *Conseil Européen pour la Recherche Nucléaire* ab.

Das Kapitel 3 umfasst eine Beschreibung des Level-1-Triggers und der *High Level Trigger*. Detaillierter werden der technische Aufbau und die Arbeitsweise des Level-1 Kalorimeter Triggers dargestellt.
In Kapitel 4 werden die Infrastruktur der *Online*-Überwachung vorgestellt sowie die bereitgestellten Systeme zur Fernüberwachung charakterisiert und verglichen.
In Kapitel 5 werden Anforderungen an das eigene Fernüberwachungssystem formuliert. Darauf aufbauend wird ein Prototyp entwickelt, dessen Funktionsfähigkeit mit einer *Client*-Anwendung demonstriert wird.
In Kapitel 6 wird die Anwendung *L1CaloDisplay* zur Fernüberwachung des Level-1 Kalorimeter Triggers vorgestellt. Es werden die eingesetzten Technologien beschrieben und Besonderheiten und Herausforderungen bei der Umsetzung aufgezeigt. Augenmerk wird auf die Authentifizierung mittels *CERN Single Sign-On* gelegt. Nach einem Überblick des Aufbaus und der wichtigsten Klassen wird die Benutzung der Anwendung beschrieben sowie die Bedeutung der dargestellten Informationen erläutert.
Das Kapitel 7 fasst die Ergebnisse der Arbeit zusammen.

Kapitel 2

Physik mit dem ATLAS-Experiment

2.1 Der „Large Hadron Collider“

Der *Large Hadron Collider* (LHC) am europäischen Kernforschungszentrum CERN, an der französisch-schweizerischen Grenze in der Nähe von Genf, ist ein ringförmiger Teilchenbeschleuniger[1]. Um Antworten auf einige der fundamentalen Fragen der Teilchenphysik zu finden, bringt er Protonen oder Blei-Kerne mit bisher unerreichten Energien von mehreren TeV zur Kollision. Eine wichtige Eigenschaft des LHC ist seine hohe Luminosität. Dank dieser werden die interessanten und seltenen Wechselwirkungen zwischen den Quarks und Gluonen der Hadronen statistisch relevant. Der LHC erreicht im Protonen-Betrieb eine Schwerpunktsenergie von 14 TeV und eine instantane Luminosität von $\mathcal{L}_{inst} = 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, im Schwerionen-Betrieb 5,5 TeV pro Nukleon-Paar und eine instantane Luminosität von $\mathcal{L}_{inst} = 10^{27}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$.
Erste Protonen zirkulierten im LHC am 10. September 2008, nach 10 Tagen wurde er jedoch aufgrund schwerer technischer Probleme, welche zur Beschädigung des Kühlsystems führten, wieder abgeschaltet. Nach einjährigen Reparaturarbeiten und der Installation von zusätzlichen Sicherheitssystemen wurde er am 20. November 2009 erneut in Betrieb genommen. Die Protonenenergie wurde seitdem in mehreren Schritten von 450 GeV auf 3,5 TeV gesteigert. Damit hat der LHC das *Tevatron*[2] als bisher stärksten Teilchenbeschleuniger abgelöst. In den Jahren 2010 und 2011 sind Proton-Proton-Kollisionen mit einer Schwerpunktsenergie von 7 TeV geplant, anschließend wird diese auf 14 TeV erhöht.
Der LHC befindet sich 50 bis 175 Meter unter der Erde in einem ringförmigen Tunnel, in dem sich zuvor der *Large Electron-Positron Collider*[3] befand, und hat einen Umfang von ungefähr 27 Kilometern. Die Protonen kreisen in zwei parallelen Strahlröhren in entgegengesetzten Richtungen. Sie werden mit einer Energie von 450 GeV in den LHC injiziert, nachdem sie eine Vorbeschleunigerkette durchlaufen haben. Im LHC werden sie anschließend von 450 GeV auf die Maximal-Energie beschleunigt.
Die zwei Teilchenstrahlen bestehen aus Bündeln (*Bunches*) von jeweils $1,15 \times 10^{11}$ Teilchen. Im Regelbetrieb kreisen gleichzeitig 2808 dieser Bündel im LHC. Sie haben jeweils

[1] Die Entscheidung zu dessen Bau wurde im Jahr 1994 getroffen. Damals war vorgesehen, den Beschleuniger zu Beginn mit einer Protonen-Schwerpunktsenergie von 10 TeV zu betreiben und ihn in einer zweiten Stufe auf 14 TeV aufzurüsten. Nach Zusagen verschiedener Geldgeber und Änderungen der Pläne wurde im Dezember 1996 entschieden den Beschleuniger direkt mit einer Schwerpunktsenergie von 14 TeV zu betreiben.

[2] Das *Tevatron* ist ein ringförmiger Proton-Antiproton-Beschleuniger des Fermilab in Batavia (Illinois, USA) und erreicht eine Schwerpunktsenergie von 1,96 TeV. Im Jahr 1995 wurden hier erstmals Top-Quark-Paare nachgewiesen [1].

[3] Der *Large Electron-Positron Collider* war von 1989 bis 2000 in Betrieb und wurde unter anderem für Präzisionsmessungen der W- und Z-Bosonen Masse genutzt.

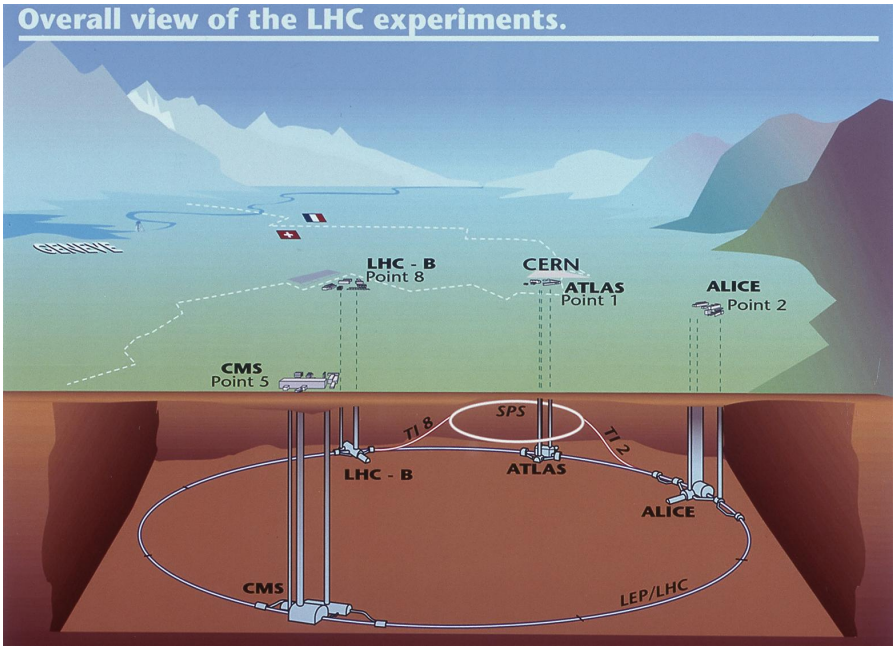


Abbildung 2.1: Der *Large Hadron Collider* und seine vier großen Experimente [2]

eine Länge von ungefähr 7,5 cm in Strahlrichtung und einen maximalen Durchmesser von 16,7 µm orthogonal zur Strahlrichtung. Der Abstand zwischen zwei benachbarten Teilchenbündeln beträgt 7,5 m. Die Bündel kreuzen sich in den vier Kollisionspunkten mit einer Frequenz von 40 MHz. Dies entspricht einer Zeitspanne von 25 ns zwischen den Strahlkreuzungen (*Bunch Crossings*). Bei jeder Strahlkreuzung finden ungefähr 20 inelastische Proton-Proton-Wechselwirkungen statt, welche von den Experimenten erfasst und ausgewertet werden, die an an den Kollisionspunkten aufgebaut sind.

1232 Dipolmagnete halten die Teilchenstrahlen auf der gewünschten Kreisbahn und weitere 392 Quadrupolmagnete fokussieren sie. Zusätzlich werden einige Tausend Korrekturmagneten eingesetzt. Um die nötigen Magnetfelder von nahezu 9 T zu erzeugen, werden supraleitende Magnete bei einer Arbeitstemperatur von 1,9 K verwendet. Die Kühlung der Magnete macht den LHC weltweit zur größten Tieftemperatureinrichtung auf Flüssig-Helium-Temperatur. Für detaillierte technische Informationen zum LHC wird auf [3] und [4] verwiesen.

In Abb. 2.1 erkennt man die Standorte der vier großen LHC-Experimente[4], die jeweils verschiedene Ziele verfolgen:

ALICE ist optimiert auf die Untersuchung von Blei-Kollisionen. Es soll nach Anzei-

[4]Bedeutung der Namen der Experimente:

ALICE	**A** **L**arge **I**on **C**ollider **E**xperiment
ATLAS	**A** **T**oroidaL **L**HC **A**pparatu**S**
CMS	**C**ompact **M**uon **S**olenoid
LHCb	**L**arge **H**adron **C**ollider **b**eauty experiment

chen des Quark-Gluon-Plasmas suchen und dessen Eigenschaften vermessen. Man nimmt an, dass dies der Zustand der Materie kurz nach dem *Big Bang* war.

ATLAS ist ein Vielzweckdetektor, der zur Suche nach dem Higgsboson und neuer Physik eingesetzt wird.

CMS ist ebenfalls ein Vielzweckdetektor und hat die gleichen Ziele wie ATLAS. CMS und ATLAS sollen unabhängige Messungen liefern.

LHCb studiert B-Zerfälle[5] mit dem Ziel die Parameter der CP-Verletzung detailliert zu vermessen.

2.2 Das ATLAS-Experiment

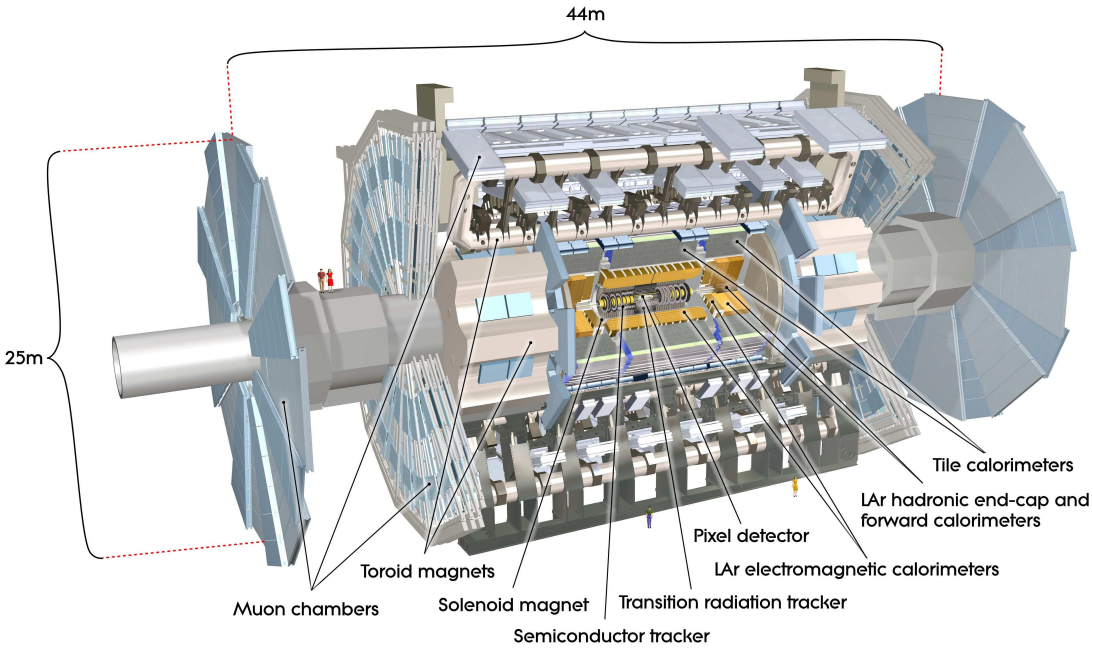


Abbildung 2.2: Querschnitt des ATLAS-Detektors [5]. Im Zentrum befindet sich der innere Detektor, der von der Kalorimetrie umschlossen wird. Weiter außen befindet sich das Myonspektrometer. Die Maße von ATLAS betragen 25 m in der Höhe und 44 m in der Länge. Das Gesamtgewicht des Detektors beträgt ungefähr 7000 t.

Der ATLAS-Detektor (siehe Abb. 2.2) ist ein Vielzweckdetektor zur Untersuchung von Protonen- und Schwerionen-Kollisionen. Er wurde über fünfzehn Jahre hinweg in einer großen Kollaboration von etwa 2000 Physikern, Ingenieuren, Technikern und Studenten

[5]Dies sind Zerfälle von Hadronen, die ein b-Quark enthalten.

entwickelt und gebaut.
Eine der Hauptaufgaben von ATLAS ist die Suche nach dem Higgsboson über einen großen Massenbereich. Eine weitere Aufgabe ist die Suche nach neuer Physik jenseits des Standardmodells, zu nennen sind hier mögliche Substrukturen von Quarks[6], supersymmetrische Teilchen oder Extradimensionen sowie die mögliche Produktion von mikroskopischen schwarzen Löchern.
Die hohe Luminosität des LHC ermöglicht weitere präzise Überprüfungen der Quantenchromodynamik, der elektroschwachen Wechselwirkung und der Flavourphysik. Da das Top-Quark mit einer Rate von einigen zehn Hertz erzeugt wird, können dessen Spin und Kopplungen genau vermessen werden.

Koordinatensystem

Der Ursprung des rechtshändigen ATLAS-Koordinatensystems ist der nominale Kollisionspunkt. Die Strahlachse definiert die z-Achse, die hierzu orthogonale Ebene ist die x-y-Ebene. Die positive x-Achse zeigt von dem Kollisionspunkt zum Zentrum des LHC-Rings und die positive y-Achse zeigt von dem Kollisionspunkt nach oben. Der Azimutwinkel ϕ wird senkrecht zur Strahlachse gemessen und der Polarwinkel θ beschreibt den Winkel relativ zur Strahlachse. Eine häufig benutzte Größe ist die Pseudorapidität $\eta = -\ln\tan(\frac{\theta}{2})$. Der Vorteil dieser Größe ist die Invarianz von Abständen $\Delta\eta$ unter Lorentz-Transformation. Der Transversalimpuls p_T, die Transversalenergie E_T und die fehlende Transversalenergie E_T^{miss} beziehen sich auf die x-y-Ebene.

Anforderungen

Die aus den Aufgaben abgeleiteten Anforderungen an ATLAS lassen sich wie folgt zusammenfassen [5] :

- **Großer Akzeptanzbereich** von Teilchen in **Pseudorapidität und Azimutwinkel**.
- Eine **gute Impulsauflösung** von geladenen Teilchen und eine **hohe Rekonstruktionseffizienz im inneren Spurdetektor**. Um τ-Leptonen und b-Jets in der *Offline*-Analyse zu identifizieren, sind *Vertex*-Detektoren nahe am Kollisionspunkt erforderlich, die sekundäre Wechselwirkungspunkte bestimmen können.
- Voraussetzung für die Identifikation und Messung von Elektronen und Photonen sowie Jets und E_T^{miss} ist ein **Kalorimetersystem** bestehend aus **elektromagnetischen und hadronischen Komponenten**, die eine **hohe Auflösung** und **Raumwinkelabdeckung** besitzen.
- Eine gute **Identifikation und Impulsauflösung von Myonen** über einen großen Impulsbereich sowie die **Ladungsunterscheidung von Myonen selbst bei großem** p_T.

[6] Bekannt als *Compositeness*.

- Ein **sehr effizienter Trigger** mit guter Untergrundunterdrückung auch bei Objekten mit niedrigem p_T. So können viele interessante physikalische Prozesse mit einer akzeptablen Triggerrate aufgezeichnet werden.
- Aufgrund der experimentellen Bedingungen am LHC benötigt der Detektor schnelle und vor allem **strahlungsresistente Elektronik und Sensoren**. Um den hohen Teilchenfluss verarbeiten zu können und um den Einfluss von überlappenden Ereignissen zu minimieren, wird zudem eine **feine Granularität** gefordert.

Die Anforderungen an ATLAS bezüglich Energieauflösung und η-Abdeckung sind in Tabelle 2.1 aufgelistet. Für Myonen mit großem p_T ist die Leistungsfähigkeit des Myonspektrometers unabhängig von dem inneren Detektor.

Komponente	**Auflösung**	**η-Abdeckung**
Innerer Detektor	$\sigma_{p_T}/p_T = 0.05\% p_T \oplus 1\%$	± 2.5
EM Kalorimetrie	$\sigma_E/E = 10\%/\sqrt{E} \oplus 0.7\%$	± 3.2
Hadronische Kalorimetrie (Jets)		
Zentralbereich und Endkappen	$\sigma_E/E = 50\%/\sqrt{E} \oplus 3\%$	± 3.2
Vorwärts	$\sigma_E/E = 100\%/\sqrt{E} \oplus 10\%$	$3.1 < \lvert\eta\rvert < 4.9$
Myonspektrometer	$\sigma_{p_T}/p_T = 10\%$ bei $p_T = 1\,\mathrm{TeV}$	± 2.7

Tabelle 2.1: Geplante Leistungsfähigkeit des ATLAS-Detektors. E und p_T sind in Einheiten von GeV gegeben. [5]

Das Magnetsystem

ATLAS besitzt ein hybrides Magnetsystem mit einem Durchmesser von 22 m und einer Länge von 26 m. Sein Feld speichert eine Energie von 1,6 GJ. Es setzt sich aus den folgenden vier supraleitenden Magneten zusammen:

- Ein Solenoid, der parallel zur Strahlachse ausgerichtet ist und ein zur Strahlachse paralleles, 2 T starkes Magnetfeld im inneren Spurdetektor erzeugt.
- Ein Zentral-Toroid und zwei Endkappen-Toroide, die im Myondetektor ein Magnetfeld von ungefähr 0.5 T (Zentralbereich) beziehungsweise 1 T (Endkappen) erzeugen. Die Größe der Toroide definiert die Gesamtgröße des ATLAS-Detektors.

Der innere Detektor

Bei voller Luminosität des LHC kommen im Bereich $|\eta| < 2.5$ alle 25 ns annähernd 1000 Teilchen von dem Kollisionspunkt. Der innere Detektor (ID) rekonstruiert die Spuren dieser Teilchen und bestimmt primäre und sekundäre Wechselwirkungspunkte. Um trotz der hohen Spurdichte fast gleichzeitige und örtlich überlappende Ereignisse zu unterscheiden, besitzt der innere Detektor eine sehr feine Granularität und schnelle Elektronik zur

Signalverarbeitung. Er wurde aufgrund seiner Nähe zum Kollisionspunkt strahlungshart konstruiert.
Eine Darstellung des Aufbaus des inneren Detektors findet man in Abb. 2.3. Der innere Detektor besteht aus drei unabhängigen Teilen: Im Zentrum befindet sich der hochauflösende Pixeldetektor, der von dem *Semiconductor Tracker* (SCT) umgeben wird. Beide haben je eine Komponente, die den Vorwärts-Bereich abdeckt, und eine zweite, die den Zentralbereich abdeckt. Der SCT wird von dem ähnlich aufgebauten *Transition Radiation Tracker* (TRT) umgeben. Die Präzisions-Spurdetektoren (Pixel und SCT) decken den Bereich $|\eta| < 2.5$ über den gesamten ϕ-Bereich ab. Um den transversalen Impuls der Teilchen zu bestimmen, wird der innere Detektor von einem zur Strahlachse parallelen, 2 T starken Magnetfeld durchdrungen.

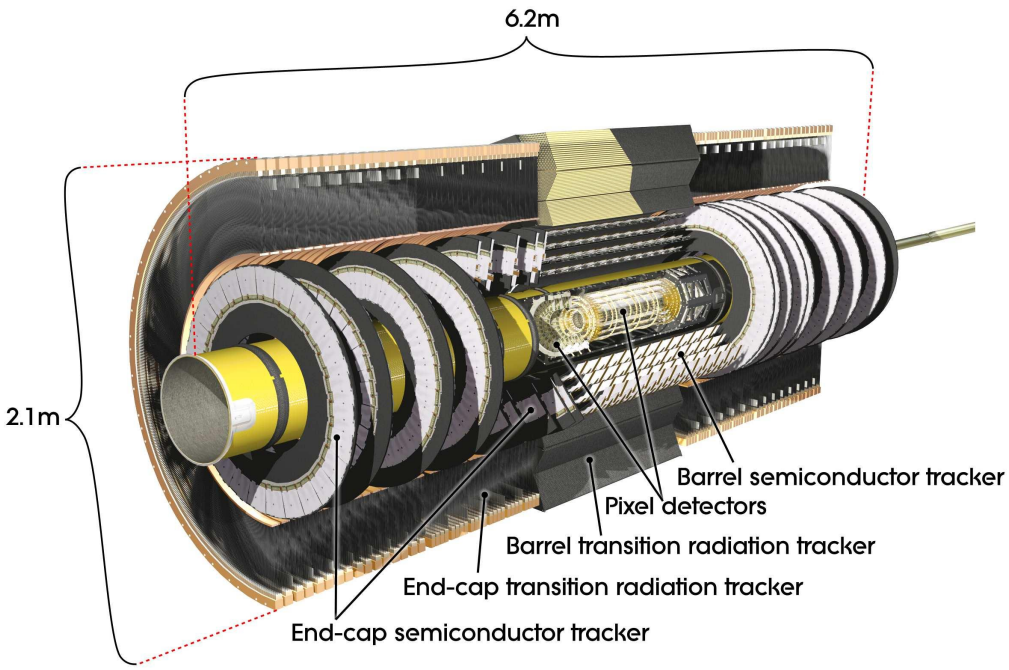


Abbildung 2.3: Querschnitt des inneren ATLAS-Detektors [5]

Der Pixeldetektor besteht aus drei konzentrisch um die Strahlachse angeordneten Trommeln und drei senkrecht zur Strahlachse angeordneten Scheiben auf jeder Seite. Mit einer Auflösung von 10 µm in $R - \phi$ und 115 µm in z-Richtung in den Trommeln und 10 µm in $R - \phi$ und 115 µm in R in den Scheiben ist diese Komponente die höchstauflösende des ID. Die Anzahl der Auslesekanäle beträgt 80,4 Millionen.

Der *Semiconductor Tracker* besteht aus vier konzentrisch um die Strahlachse angeordneten Trommeln und neun senkrecht zur Strahlachse angeordneten Scheiben auf jeder Seite. Er liefert pro Spur vier Raumpunkte, die jeweils durch zwei um

90 ° verdrehte Lagen von Siliziumstreifen erzeugt wird. Er besitzt eine Auflösung von 17 µm in $R-\phi$ und 580 µm in z-Richtung in den Trommeln und 17 µm in $R-\phi$ und 580 µm in R in den Scheiben. Die Anzahl der Auslesekanäle beträgt 6,3 Millionen.

Der *Transition Radiation Tracker* besteht aus Driftröhren mit jeweils 4 mm Durchmesser und erfasst Spuren im Bereich $|\eta| < 2$. Pro Spur werden typischerweise um die 30 Treffer registriert. Die $R-\phi$ Informationen der Treffer mit einer Driftröhrenauflösung von 130 µm ermöglichen eine sehr genaue Impulsbestimmung. Durch die Messung der Anzahl von Übergangsstrahlungsphotonen, die beim Durchqueren der Driftröhren entstehen, wird die Identifikation von geladenen Teilchen stark verbessert. Der TRT besitzt 351.000 Auslesekanäle.

Weitere Informationen zur erwarteten Leistungsfähigkeit des inneren Detektors finden sich in [6].

Das Kalorimetersystem

Die Aufgaben des Kalorimetersystems sind die Energiemessung und Identifikation von Leptonen, Photonen, Hadronen und Jets sowie die Bestimmung der fehlenden transversalen Energie E_T^{miss}. Die Energie, der in den Kollisionen entstandenen neutralen und geladenen Teilchen, wird im ATLAS-Detektor mit Hilfe verschiedener *Sampling*-Kalorimeter gemessen.
Sampling-Kalorimeter bestehen aus sich abwechselnden Absorber- und Nachweisschichten. Die Absorberschichten bestehen aus einem Material hoher Dichte und sorgen dafür, dass die gesamte Energie der zu messenden Teilchen in verschiedenen Wechselwirkungsprozessen auf Sekundärteilchen niedrigerer Energie übertragen wird. Während Elektronen und Photonen Energie hauptsächlich durch Bremsstrahlung und Elektron-Positron-Paarbildung verlieren, geben Hadronen ihre Energie zum größten Teil durch inelastische Streuungen mit den Nukleonen der Atome ab. Die Schauer der Sekundärteilchen erzeugen in den Nachweisschichten durch Ionisations- und Anregungsprozesse je nach Material freie Ladungen oder Szintillationsphotonen, deren Anzahl direkt proportional zur ursprünglichen Energie ist. Die Ladungen werden dann mit Hilfe einer anliegenden Spannung, die Szintillationsphotonen mit Hilfe von Photomultipliern, in ein elektronisches Signal umgewandelt.
Da Hadronen eine höhere Durchschlagskraft als Elektronen und Photonen haben, werden im ATLAS-Detektor sowohl elektromagnetische (EM) als auch hadronische Kalorimeter (HAD) eingesetzt. Ein Querschnitt des Kalorimetersystems findet sich in Abb. 2.4. Es umschließt den inneren Detektor samt Solenoidspule und deckt den Bereich $|\eta| < 4.9$ ab. Im mit dem inneren Detektor übereinstimmenden η-Bereich sind die Messungen im feingranularen EM-Kalorimeter sehr präzise. Im verbleibenden η-Bereich hat das Kalorimetersystem den physikalischen Anforderungen an Jet-Rekonstruktion und E_T^{miss}-Genauigkeit entsprechend eine gröbere Granularität.
Die Kalorimeter sollen elektromagnetische und hadronische Schauer vollständig absorbieren, um Durchschläge ins Myonsystem zu minimieren. Daher ist die Dicke der Kalori-

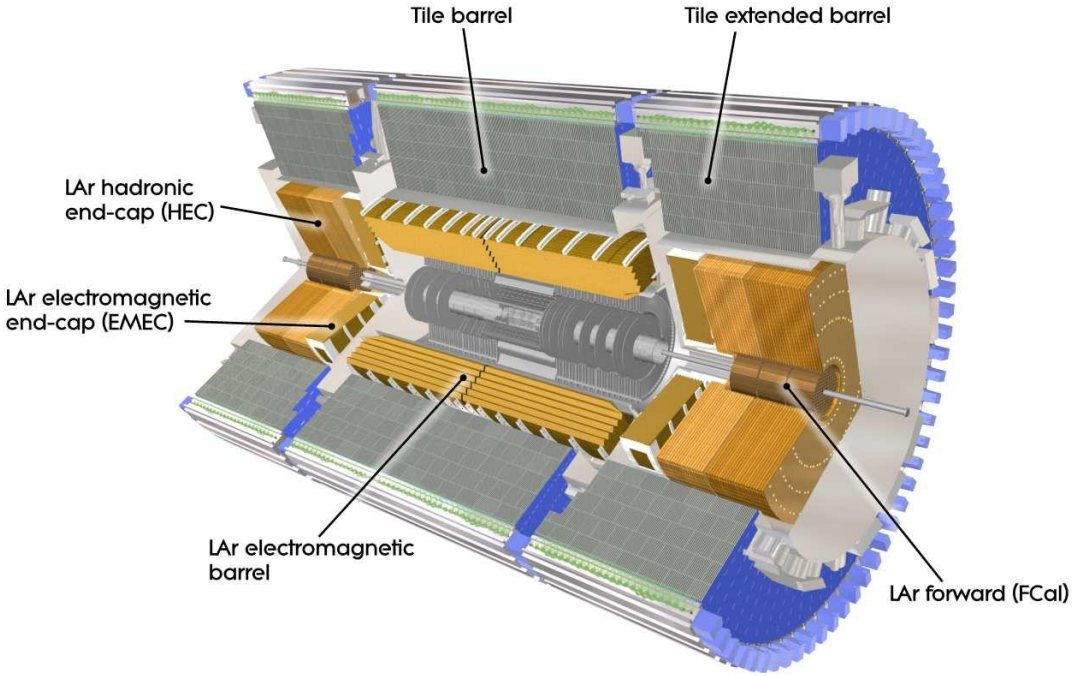


Abbildung 2.4: Querschnitt des ATLAS Kalorimetersystems [5]

meter eine wichtige Eigenschaft. Diese wird in Strahlungslängen[7] X_0 bzw. mitleren freien Weglängen[8] λ angegeben. Die komplette Dicke des EM-Kalorimeters im Zentralbereich ist größer als 22 X_0, in den Endkappen größer als 24 X_0. Die Dicke des Kalorimeters für hochenergetische Jets beträgt im Zentralbereich ungefähr 9.7 λ und in den Endkappen ungefähr 10 λ. In Messungen und Simulationen wurde gezeigt, dass die gewählte Tiefe für eine genaue E_T-Messung geeignet ist und Durchschläge auf ein Maß reduziert, das viel niedriger als das von prompten Myonen oder Zerfallsmyonen ist.
Die Energieauflösung eines *Sampling*-Kalorimeters kann folgendermaßen parametrisiert werden:

$$\frac{\Delta E}{E} = \frac{a}{\sqrt{E}} \oplus \frac{b}{E} \oplus c$$

In dieser Formel bezeichnet E die Energie, $\frac{a}{\sqrt{E}}$ den *sampling*-Term, $\frac{b}{E}$ den *noise*-Term und c einen konstanten Term. Die Proportionalität des *sampling*-Terms zu $\frac{1}{\sqrt{E}}$ erklärt sich aus statistischen Fluktuationen der Messung der sekundären Teilchen, der Faktor

[7] Die Strahlungslänge X_0 ist definiert als die Länge, nach der ein hochenergetisches Elektron aufgrund von Bremsstrahlung im Mittel nur noch $1/e$ seiner Anfangsenergie besitzt, beziehungsweise als 7/9 der mittleren freien Weglänge für Paarbildung von hochenergetischen Photonen.
[8] Die mittlere freie Weglänge λ ist definiert als die Länge, die ein Teilchen im Mittel zurücklegen kann ohne mit anderen Teilchen zu wechselwirken.

a repräsentiert die Detektor-Geometrie. Der Einfluss des *noise*-Terms auf den absoluten Fehler der Energiemessung ist konstant und kann daher bei hohen Energien vernachlässigt werden. Der Term c dominiert die Auflösung von Objekten hoher Energie und beschreibt den Einfluss von Kalibrationsfehlern, Ungleichförmigkeiten in der Signalantwort oder toten Detektorbereichen sowie von Sekundärteilchen, die nicht absorbiert werden. Die Auflösungen der verschiedenen im Folgenden beschriebenen Kalorimeterkomponenten finden sich in Tabelle 2.1.

Das elektromagnetische Kalorimeter

Das EM-Kalorimeter absorbiert elektromagnetische Schauer vollständig und ungefähr 60 % der Energie der hadronischen Schauer. Die Absorberschichten des EM-Kalorimeters bestehen aus Blei, die Nachweisschichten aus Flüssig-Argon. Da die Anordnung der Schichten einem Akkordeon ähnelt, wird eine kontinuierliche Abdeckung in ϕ und η gewährleistet (siehe Abb. 2.5).

Das EM-Kalorimeter ist in eine Zentral-Komponente ($|\eta| < 1.475$) und zwei Endkappen-

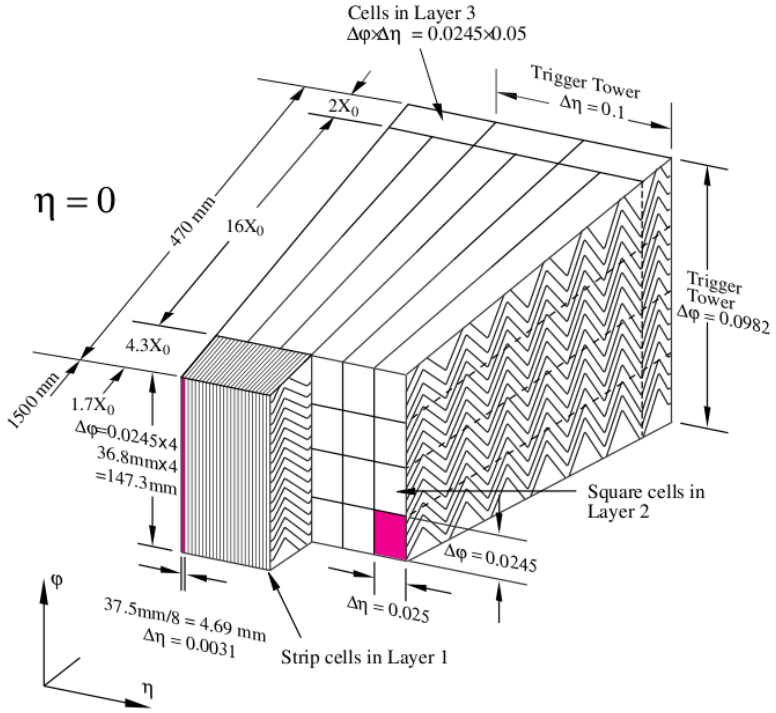


Abbildung 2.5: Skizze eines Zentralbereich-Moduls des EM-Kalorimeters. Gut zu erkennen ist die Akkordeonstruktur sowie die drei verschiedenen Detektorlagen. [5]

Komponenten ($1.375 < |\eta| < 3.2$) unterteilt, die jeweils in eigenen Kryostaten untergebracht sind. Die Zentral-Komponente besteht aus zwei identischen Halbtrommeln, die durch einen 4 mm breiten Spalt bei $z = 0$ getrennt sind. Die Endkappen-Komponenten (EMEC) bestehen aus einer äußeren und einer inneren Scheibe, die den Bereich $1.375 <$

$|\eta| < 2.5$ bzw. $2.5 < |\eta| < 3.3$ abdecken. Um im Bereich $|\eta| < 2.5$ Präzisionsmessungen durchführen zu können, besteht das EM-Kalorimeter hier aus drei übereinanderliegenden Schichten, deren Auflösung von innen nach außen abnimmt. Die innerste der drei Schichten kann neutrale Pionen durch das Auflösen beider Zerfallsphotonen identifizieren. Die Endkappen-Scheibe im Bereich $2.5 < |\eta| < 3.3$ besteht aus zwei Schichten gröberer Granularität.
Um den Energieverlust von Elektronen und Photonen vor dem EM-Kalorimeter zu bestimmen, existiert im Bereich $|\eta| < 1.8$ zusätzlich ein dünner *presampler*-Detektor mit einer Flüssig-Argon Nachweisschicht.

Das „Tile"-Kalorimeter

Das hadronische *Tile*-Kalorimeter (TileCal) grenzt, von dem Wechselwirkungspunkt aus gesehen, außen an das EM-Kalorimeter an. Es ist ebenfalls ein *Sampling*-Kalorimeter und verwendet Stahl in den Absorberschichten sowie szintillierende Kacheln in den Nachweisschichten. Die Zentral-Komponente befindet sich im Bereich $|\eta| < 1.0$, die erweiterten Zentral-Komponenten liegen im Bereich $0.8 < |\eta| < 1.7$. Azimutal ist das TileCal in 64 Module untergliedert.
Der innere Radius des TileCals beträgt 2.28 m, der äußere 4.25 m. Es ist radial aus drei Lagen zusammengesetzt, die in der Zentral-Komponente eine Dicke von 1.5, 4.1 and 1.8 λ haben, in den erweiterten Zentral-Komponenten eine Dicke von 1.5, 2.6, und 3.3 λ. Die Auflösung beträgt 0.1×0.1 in den zwei inneren Lagen und 0.2×0.1 in $\Delta\eta \times \Delta\phi$ in der äußeren Lage.
Die Photonen werden auf zwei Seiten der szintillierenden Kacheln mit Hilfe von Wellenlängen schiebenden Fasern in Photomultiplier-Röhren geführt (siehe Abb. 2.6). Die durch das Bündeln von Fasern definierten Auslesezellen sind in η pseudo-projektiv im Hinblick auf den Kollisionspunkt.

Das hadronische Endkappen-Kalorimeter

Das hadronische Endkappen-Kalorimeter (HEC) besteht auf jeder Seite aus zwei unabhängigen Scheiben, die sich direkt hinter dem EMEC befinden. Es ist ein *Sampling*-Kalorimeter, welches in den Nachweisschichten Flüssig-Argon verwendet und in den Absorberschichten Platten aus Kupfer. Es deckt den Bereich $1.5 < |\eta| < 3.2$ ab und überlappt für kleine $|\eta|$ mit dem TileCal und für große $|\eta|$ mit dem FCal. Diese Überschneidungen stellen eine volle Akeptanz in η sicher. Jede der insgesamt vier HEC-Scheiben setzt sich aus 32 keilförmigen Modulen zusammen.

Das Vorwärts-Kalorimeter

Das Vorwärts-Kalorimeter (FCal) ist wie das HEC ein Endkappen-*Sampling*-Kalorimeter und deckt den Bereich $3.1 < |\eta| < 4.9$ ab. Es hat eine Dicke von ungefähr 10 λ und besteht aus drei hintereinander liegenden Modulen. Das dem Kollisionspunkt am nächsten gelegene Modul verwendet als Absorbermaterial Kupfer und wurde für elektromagnetische Schauer optimiert, die zwei Module weiter außen verwenden Wolfram als Absorber-

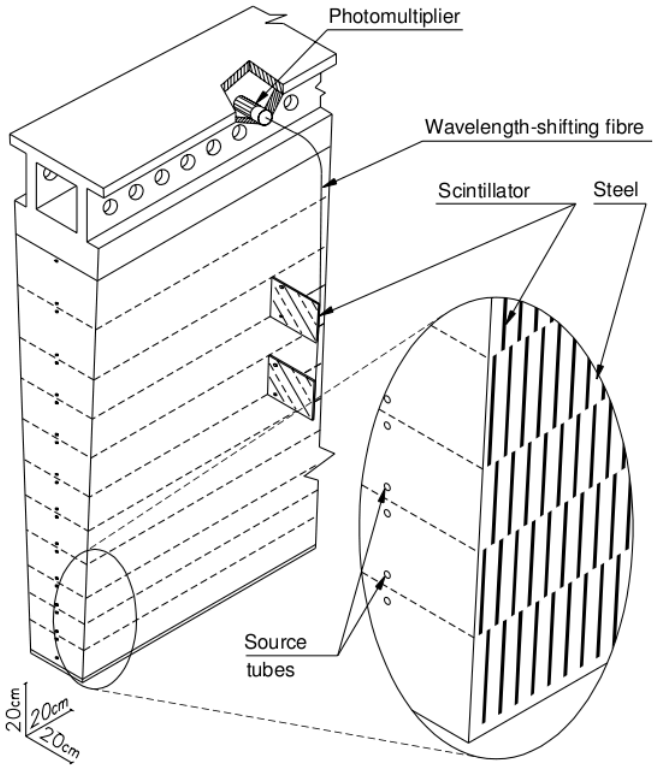


Abbildung 2.6: Skizze eines TileCal-Moduls. Das Auslesesystem, bestehend aus szintillierenden Kacheln, Fasern und Photomultipliern ist hervorgehoben. [5]

material und messen hadronische Schauer. Als Nachweismaterial wird jeweils Flüssig-Argon genutzt.

Das Myonsystem

Myonen sind minimal ionisierende Teilchen und wechselwirken nur schwach mit dem Detektormaterial. Sie werden daher weder im inneren Spurdetektor noch in der Kalorimetrie gestoppt. Da Endzustände mit Myonen vielversprechende und robuste Signaturen in der Physikanalyse sind, wurde außerhalb des Kalorimetersystems ein Myonsystem installiert. Dieses misst die Spur der Myonen im Magnetfeld und bestimmt aus der Krümmung den transversalen Impuls mit hoher Genauigkeit. Im Zentral- und Endkappen-Bereich messen drei hintereinander liegende Lagen von Kammerdetektoren die Myonspuren. Im Zentralbereich befinden sich die Kammern in Modulen parallel zur Strahlrichtung, in der Übergangsregion und den Endkappen befinden sie sich auf Scheiben senkrecht zur Strahlrichtung.

In Abb. 2.7 findet sich eine Skizze des Aufbaus, in Tabelle 2.2 werden die wichtigsten Parameter der verschiedenen eingesetzten Kammertechnologien aufgelistet. Über einen großen η-Bereich können die *Monitored Drift Tubes* (MDTs) sehr präzise die Spurkoor-

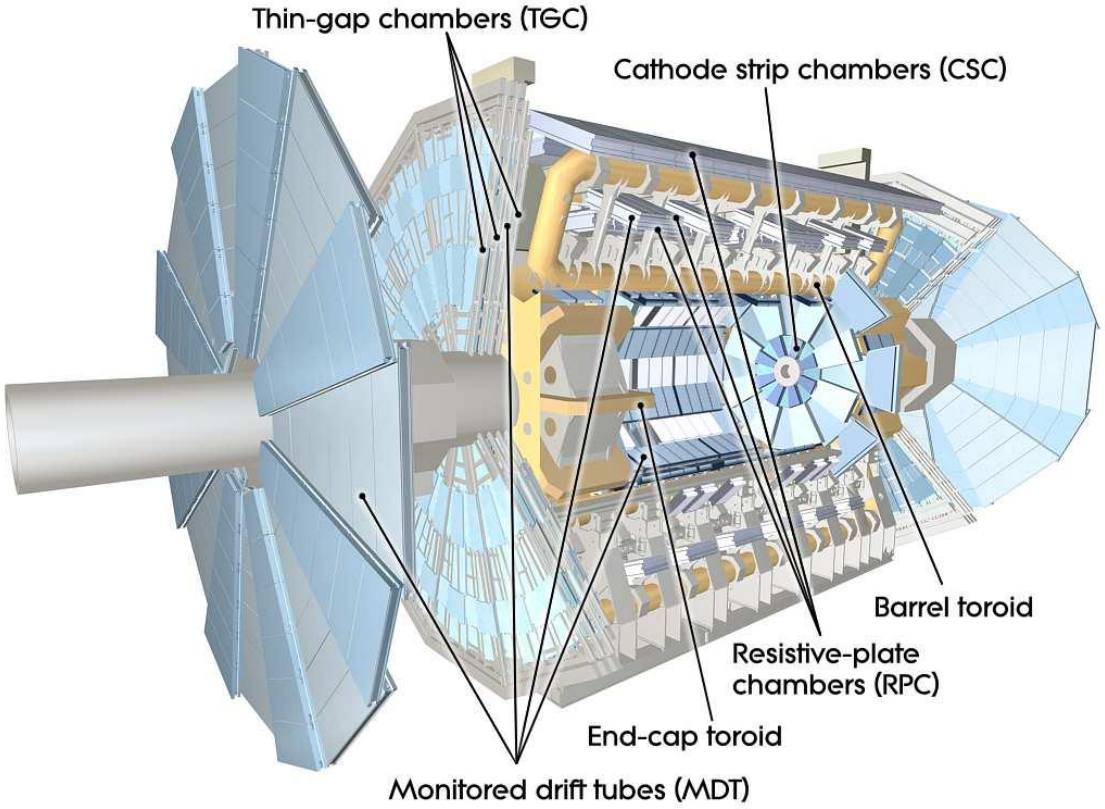


Abbildung 2.7: Querschnitt des ATLAS Myonsystems [5]

dinaten erfassen. Die *Cathode Strip Chambers* (CSCs) arbeiten mit einer gröberen Granularität, um mit den hohen Raten und den Untergrundbedingungen in der Vorwärts-Richtung umgehen zu können. Schnelle *Resistive Plate Chambers* (RPCs) im Zentralbereich und *Thin Gap Chambers* (TGCs) in den Endkappen werden im Bereich $|\eta| < 2.4$ zum Triggern eingesetzt. Sie liefern zudem eine zweite Spurkoordinate, die orthogonal zu denen der Präzisionskammern ist.
Um die genaue Ausrichtung der MDTs sicherzustellen, wurde in und zwischen den Kammern ein optisches Justierungssystem mit ungefähr 12.000 Sensoren installiert.

2.3 Physik mit dem ATLAS-Experiment

Nach einem kurzen Überblick über das Standardmodell der Teilchenphysik wird im folgenden Abschnitt die Suche nach dem Higgsboson am ATLAS-Experiment erläutert.

Monitored drift tubes	**MDTs**
- Abdeckung	$\|\eta\| < 2.7$ (innerste Lage: $\|\eta\| < 2.0$)
- Anzahl Kammern	1150
- Anzahl Kanäle	354 000
- Funktion	Präzisionsspurerfassung
Cathode strip chambers	**CSCs**
- Abdeckung	$2.0 < \|\eta\| < 2.7$
- Anzahl Kammern	32
- Anzahl Kanäle	31 000
- Funktion	Präzisionsspurerfassung
Resistive plate chambers	**RPCs**
- Abdeckung	$\|\eta\| < 1.05$
- Anzahl Kammern	606
- Anzahl Kanäle	373 000
- Funktion	Triggern, zweite Koordinate
Thin gap chambers	**TGCs**
- Abdeckung	$1.05 < \|\eta\| < 2.7$ (2.4 für das Triggern)
- Anzahl Kammern	3588
- Anzahl Kanäle	318 000
- Funktion	Triggern, zweite Koordinate

Tabelle 2.2: Die wichtigsten Parameter des Myonspektrometers [5]

Das Standardmodell der Teilchenphysik

Das Standardmodell der Teilchenphysik (SM) ist eine Theorie, die die fundamentalen Teilchen beschreibt sowie drei der vier fundamentalen Kräfte, die zwischen den Teilchen wirken. Die Teilchen im Standardmodell heißen Leptonen und Quarks und machen die sichtbare Materie im Universum aus. Sie sind Fermionen[9] und werden aufgrund ihrer Eigenschaften in drei Familien eingeteilt (siehe Tabelle 2.3). Jedem Fermion wird ein Antifermion zugeordnet, welches die gleiche Masse, jedoch eine entgegengesetzte elektrische Ladung, Farbe sowie dritte Komponente des schwachen Isospins besitzt.
Die fundamentalen Wechselwirkungen (stark, elektromagnetisch und schwach) sind in ihrer Struktur sehr ähnlich und werden durch den Austausch von Vektorbosonen[10] vermittelt (siehe Tabelle 2.4). Die Austauschbosonen der starken Kraft (Gluonen) tragen selbst Farbe und koppeln daher untereinander. Auch die Bosonen der schwachen Wechselwirkung ($W^{\pm}, Z^0$) tragen selbst eine elektroschwache Ladung und wechselwirken untereinander.

[9] Fermionen sind Teilchen mit halbzahligem Spin.
[10] Bosonen sind Teilchen mit ganzzahligem Spin.

Fermionen	Familie			elektr.	Farbe	schwacher Isospin		Spin
	1	2	3	Ladung		linkshdg.	rechtshdg.	
Leptonen	ν_e	ν_μ	ν_τ	0	–	1/2	–	1/2
	e	μ	τ	-1	–	1/2	0	1/2
Quarks	u	c	t	$+2/3$	r,b,g	1/2	0	1/2
	d	s	b	$-1/3$	r,b,g	1/2	0	1/2

Tabelle 2.3: Anordung der Fermionen in aufsteigender Masse in drei Familien oder Generationen. Fermionen sind wie die Vektorbosonen fundamentale Teilchen. [7]

Die Reichweite der elektromagnetischen Kraft ist unendlich, da ihre Austauschteilchen (Photonen) keine Masse besitzen. Die große Masse der Bosonen $W^\pm$ und Z^0 begrenzt die Reichweite der schwachen Wechselwirkung auf ungefähr 10^{-3} fm. Obwohl Gluonen ebenfalls keine Masse besitzen, ist die Reichweite der starken Kraft sehr gering. Dies liegt daran, dass Gluonen untereinander wechselwirken und freie Teilchen immer farbneutral sein müssen. Bei Abständen größer als 1 fm ist die Energie des Farbfeldes groß genug, um aus dem Vakuum reale Quark-Antiquark-Paare zu erzeugen. Farbneutrale Teilchen, die sich aus Quarks oder Antiquarks zusammensetzen, nennt man Hadronen. Zu den bekanntesten Vertretern der Hadronen gehören das Neutron und das Proton, die sich aus Quarks der ersten Generation zusammensetzen. Die elektromagnetische und die schwache Wechselwirkung können zu der elektroschwachen Wechselwirkung zusammengefassst werden[11].

Das Standardmodell wurde in den letzten Jahrzehnten entwickelt und durch zahlreiche

Wechselwirkung	koppelt an	Austauschteilchen	Masse (GeV/c^2)	J^P
stark	Farbe	8 Gluonen (g)	0	1^-
elektromagn.	elektr. Ladung	Photon (γ)	0	1^-
schwach	schwache Ladung	$W^\pm, Z^0$	$\approx 10^2$	1

Tabelle 2.4: Übersicht der bekannten elementaren Wechselwirkungen neben der Gravitation. Die Wechselwirkungen werden durch den Austausch von Vektorbosonen übertragen. [7]

Experimente bestätigt[12]. Obwohl mittels des SMs gute Vorhersagen vieler quantitativer Eigenschaften der fundamentalen Teilchen und Wechselwirkungen getroffen wurden, bleiben immer noch viele Fragen ungeklärt. Zum Beispiel beinhaltet das SM keine Theorie der Gravitation. Desweiteren gibt es eine große Anzahl von mindestens 19 freien Parametern, wie die Massen von Fermionen und Bosonen, die unterschiedlichen Kopp-

[11] In dieser vereinheitlichten Theorie werden die entprechenden Ladungen über den Weinberg-Winkel verknüpft.

[12] Eine Ausnahme bilden die Neutrinomassen. Im SM wird angenommen, dass die Neutrinos masselos sind. Durch die Beobachtung von Neutrino-Oszillationen wurde diese Annahme widerlegt.

lungskonstanten sowie die Koeffizienten der CKM-Matrix. Diese Parameter wurden experimentell mit hoher Präzision bestimmt. Für manche der vielen offenen Fragen existieren bereits Erweiterungen des Standardmodells, die erst durch Experimente überprüft werden müssen. Die aktuell bedeutendste Fragestellung ist jedoch, welcher Mechanismus den Fermionen und Bosonen Masse verleiht.

Die Suche nach dem Higgsboson

Eine Lösung des Masseproblems des Standardmodells ist der Higgs-Mechanismus, der im Jahr 1964 von dem britischen Physiker Peter Higgs ausgearbeitet wurde (siehe [8]). Dieser erklärt die Massen von Fermionen sowie von bestimmten Vektorbosonen durch spontane elektroschwache Symmetriebrechung. Es wird ein Dublett von komplexen skalaren Feldern eingeführt, von denen nach der Symmetriebrechung nur noch das neutrale skalare Higgsfeld übrigbleibt. Das Higgsboson ist innerhalb des Standardmodells das einzige bisher noch nicht beobachtete Teilchen.
Durch direkte Suche am Elektron-Positron-Beschleuniger LEP konnte eine untere Grenze von 114,4 GeV für dessen Masse gefunden werden. In der Annahme der vollständigen Korrektheit des Standardmodells kann durch einen globalen Fit aller vorhandenen elektroschwachen Daten eine obere Massengrenze bestimmt werden. Kombiniert man diese mit den Daten des LEP ergibt sich mit einer statistischen Sicherheit von 95 % die obere Massengrenze zu 191 GeV. Die Einschränkungen der Higgsmasse durch die zugrunde liegende Theorie sind schwächer und nach oben durch Unitaritätsargumente, nach unten durch die geforderte Stabilität des elektroschwachen Vakuums begründet.
Der erwartete NLO[13]-Produktions-Wirkungsquerschnitt des SM-Higgsbosons am LHC

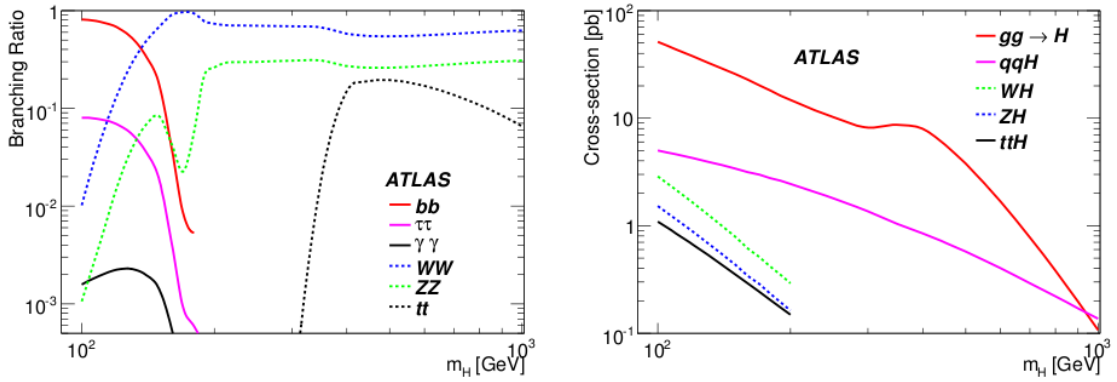


Abbildung 2.8: Links sieht man das Verzweigungsverhältnis der relevanten Zerfälle des SM-Higgsbosons als Funktion der Masse. Rechts ist der Wirkungsquerschnitt der fünf Produktionskanäle des SM-Higgsbosons als Funktion der Masse für eine Schwerpunktsenergie von 14 TeV am LHC zu erkennen. [9]

wird in Abb. 2.8 (rechts) abgebildet. Der dominierende Prozess ist die *Gluon-Gluon*-Fusion ($gg \rightarrow H$), die für niedrige Massen einen großen Untergrund hat. Die Vektorbo-

[13] Abkürzung für *Next to Leading Order*.

sonen-Fusion (VBF) ist der zweithäufigste Produktionsprozess und hat den Vorteil einer klaren Signatur ($qq \to Hqq$). Assoziierte Produktionsprozesse wie $q\bar{q}' \to HW$, $q\bar{q} \to HZ$, und $gg, q\bar{q} \to t\bar{t}H$ haben viel kleinere Wirkungsquerschnitte. Von Vorteil ist, dass die Produktionsprozesse, die neben dem Higgs ein W, Z oder t-Quark beziehungsweise im Fall der VBF zwei Jets mit großem p_T bei hohem η enthalten, mit hoher Effizienz getriggert werden können.
In Abb. 2.8 (links) wird das Verzweigungsverhältnis der Zerfälle des SM-Higgsbosons dargestellt. Für die Entdeckung des Higgsbosons sind die Zerfallskanäle wie folgt relevant:

- Obwohl der Kanal $H \to b\bar{b}$ für $m_H < 2m_W$ dominant ist, wird er wegen des enormen QCD-Untergrunds bei der Suche nach dem Higgs wahrscheinlich nur von untergeordneter Bedeutung sein.
- Für kleine Higgsmassen sind die $\gamma\gamma$-Endzustände trotz ihrer geringen Wahrscheinlichkeit sehr interessant. Sie entstehen durch Zerfälle des Higgs über b-Quark-, t-Quark- und *W-Loops*. Mit Hilfe einer guten γ/Jet-Unterscheidung und Energieauflösung der γ-Quanten verbessert sich das Verhältnis von Signal zu Untergrund.
- Endzustände von Higgsbosonen, die durch VBF entstanden sind, können durch die zwei einhergehenden Jets gefunden werden.
- Falls die Higgsmasse ausreicht, um WW- oder ZZ-Endzustände zu erzeugen, wären diese die wichtigsten Entdeckungskanäle. Der Kanal $H \to ZZ^* \to 4l$ wird auch als *Golden Mode* bezeichnet, da sich die vier Leptonen eindeutig triggern lassen und eine genaue Rekonstruktion der Higgsmasse erlauben.

Das ATLAS-Experiment sucht das Higgsboson in allen geeigneten Kanälen. Die erwartete Signifikanz der Entdeckung des SM-Higgsbosons bei Kombination der verschiedenen Endzustände in Abhängigkeit von der Higgsmasse und der integrierten Luminosität wird in Abb. 2.9 gezeigt. Unter der Annahme einer integrierten Luminosität des ATLAS-Experiments von einigen fb^{-1} in den ersten Jahren ist abzulesen, dass es das SM-Higgsboson in dieser Zeit entdecken kann.

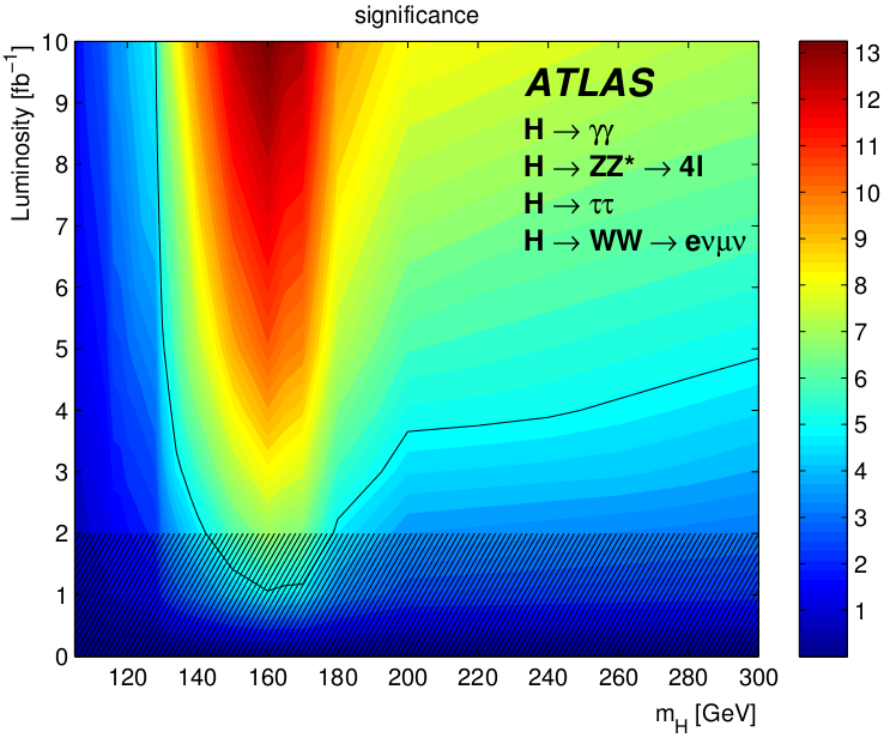


Abbildung 2.9: Signifikanzlinien für verschiedene SM-Higgsmassen und integrierte Luminositäten. Die schwarze Linie beschreibt die 5σ-Entdeckung. Die schraffierte Fläche unterhalb $2\,fb^{-1}$ bedeutet, dass die Berechnungen in diesem Bereich nicht sehr genau sind, jedoch als konservativ eingeschätzt werden. [9]

Kapitel 3

Das Trigger- und Datennahmesystem des ATLAS-Detektors

3.1 Überblick

Im LHC kollidieren alle 25 ns zwei Protonenbündel, in jedem dieser sogenannten Ereignisse gibt es ungefähr 20 inelastische Proton-Proton-Wechselwirkungen. Das Speichern und anschließende Verarbeiten aller zugehörigen Informationen ist weder technisch möglich noch sinnvoll, da die meisten Ereignisse physikalisch uninteressant sind. Um die seltenen interessanten Physikprozesse trotz der kleinen Wirkungsquerschnitte mit hoher Effizienz herauszufiltern, wurde ein dreistufiges System entworfen, das ATLAS Trigger- und Datennahmesystem (TDAQ[1]).

Die erste Stufe (Level-1) besteht aus speziell angefertigter Elektronik, die zweite und

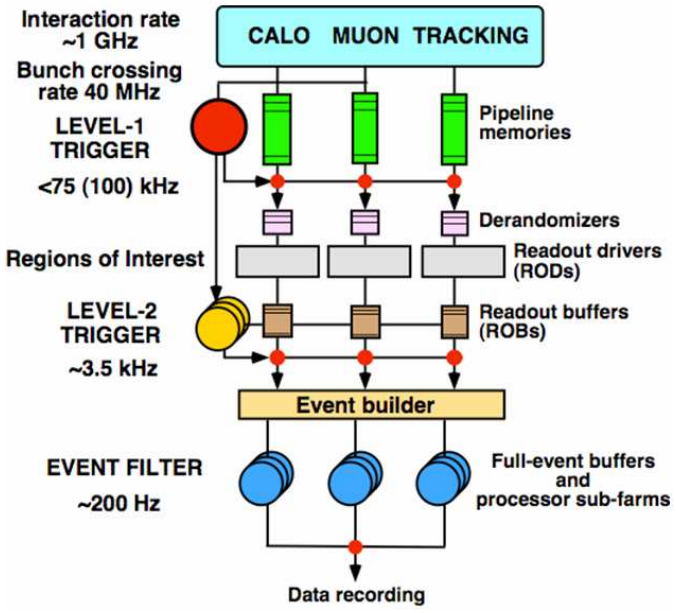


Abbildung 3.1: Das dreistufige Triggersystem des ATLAS-Detektors [5]

dritte Stufe (Level-2 und *Event Filter* (EF)) basieren auf kommerziellen Computern und Netzwerktechnologien und werden als *High Level Trigger* (HLT) bezeichnet. Ein

[1] Abkürzung für *Trigger and Data Acquisition*.

Diagramm der drei Stufen findet sich in Abb. 3.1.
Der Level-1-Trigger reduziert die Ereignisrate von 40 MHz auf eine Rate von 75 kHz. Die Detektordaten jedes Ereignisses werden für maximal 2.5 µs in der entsprechenden Subsystemelektronik gepuffert. Dies ist die Zeit, die der Level-1-Trigger für eine Entscheidung hat. In diese gehen Informationen niedriger Granularität des Myon- und Kalorimetersystems ein.
Der Level-2-Trigger hat für die Analyse eines Ereignisses bis zu 40 ms Zeit und reduziert die akzeptierten Ereignisse auf eine Rate von 3.5 kHz. Die Selektion der Ereignisse basiert auf sogenannten *Regions-Of-Interest* (ROIs), dies sind Regionen, in denen der Level-1-Trigger mögliche interessante Objekte identifiziert hat.
Der EF reduziert die Ereignisse nach einer Berechnungszeit von etwa einer Sekunde auf eine Rate von 200 Hz und die verbleibenden, ungefähr 1.5 MB großen Ereignisse werden gespeichert.

3.2 Der Level-1-Trigger

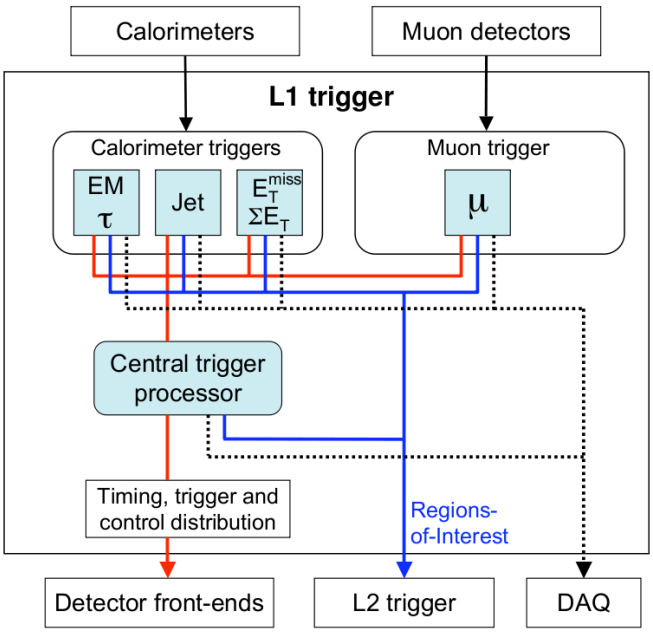


Abbildung 3.2: Blockdiagramm des Level-1-Triggers [5]

Der Level-1-Trigger (siehe Abb. 3.2) selektiert Ereignisse auf Basis von Informationen des Myon- und Kalorimetersystems.
Der Level-1 Kalorimeter Trigger (L1Calo) sucht nach Elektronen, Photonen, Jets, zu Hadronen zerfallenen τ-Leptonen und nach Ereignissen mit großer transversaler Energie beziehungsweise fehlender transversaler Energie. Zusätzlich sucht er nach hohen transversalen Energiesummen nur von Jets. Bei der Auswahl von Elektronen, Photonen und

τ-Leptonen können Isolationskriterien angewendet werden, das heißt, dass in einem vorgegebenen Raumwinkel um die getriggerten Teilchen herum keine zweite signifikante Energiedeposition erlaubt ist. In jedem *Bunch Crossing* zählt L1Calo die Multiplizitäten von 4 bis 16 programmierbaren E_T-Schwellen pro Objekttyp.
Der Level-1 Myon Trigger (L1Muon) findet Myonen mit großem p_T auf Basis von Informationen der RPCs und TGCs des Myonsystems. Er sucht nach Mustern von Raumpunkten, die konsistent mit von dem Kollisionspunkt ausgehenden Myonen sind. Der Myon-Trigger zählt die Multiplizitäten von 6 programmierbaren p_T-Schwellen.
Der *Central Trigger Processor* (CTP) kombiniert die Ergebnisse des Myon- und Kalorimeter-Triggers für verschiedene Objekttypen und entscheidet, ob ein Ereignis von dem Level-1-Trigger akzeptiert wird oder nicht. Der CTP greift dabei auf ein *trigger menu* mit bis zu 256 programmierbaren Einträgen zurück, die zum Großteil logische Verknüpfungen der Ergebnisse von L1Calo und L1Muon sind. Um die verfügbare Bandbreite von dem Level-1- zu dem Level-2-Trigger optimal zu nutzen, können die Einträge des *trigger menu* mit einem Skalierungsfaktor versehen werden. Der CTP schickt seine Entscheidung mittels des *Timing, Trigger and Control* Systems (TTC) an die Front-end Systeme des Detektors. Wird ein Ereignis akzeptiert, werden die im Level-1-Trigger nicht verwendeten, geometrischen Informationen der Objekte als ROIs an den Level-2-Trigger weitergereicht.

3.3 Die „High Level Trigger“

Beide *High Level Trigger* verwenden die volle Granularität[2] des Kalorimeter- und Myonsystems sowie Daten des inneren Spurdetektors, durch die mittels Spurrekonstruktion die Teilchenidentifikation effizienter wird. Beispielsweise wird die Selektion der Ereignisse durch die Unterscheidung von Elektronen und Photonen stark verbessert.
Im Folgenden wird das Datennahmesystem (DAQ[3]) anhand des Pfads der Daten von dem Detektor über die HLTs zum Massenspeicher erklärt (siehe Abb. 3.3). Wenn der Level-1-Trigger ein Ereignis akzeptiert, werden die zugehörigen Daten von subsystemspezifischen Auslesetreibern (RODs[4]) über 1574 Ausleseanschlüsse (ROLs[5]) zu den HLTs weitergereicht. Die Daten werden nun in 1574 Auslesepuffern (ROBs[6]) des Auslesesystems (ROS[7]) zwischengespeichert, von welchem sie bei Bedarf abgerufen werden können.
Für jedes von dem Level-1-Trigger akzeptierte Ereignis werden die gefundenen ROIs von dem sogenannten *ROI-Builder* zu einem Datenobjekt zusammengefasst, das an einen *L2 supervisor* (L2SV) übergeben wird. Der L2SV verwaltet die verschiedenen Daten innerhalb des Level-2-Triggers und weist sie zur Analyse verschiedenen Berechnungseinheiten (L2PUs[8]) zu. Die L2PUs forden die ausführlichen Detektordaten der ROIs von dem Aus-

[2] Für den Level-2-Trigger gilt dies allerdings nur in den ROIs.
[3] Abkürzung für *Data Acquisition*.
[4] Abkürzung für *Readout Drivers*.
[5] Abkürzung für *Readout Links*.
[6] Abkürzung für *Readout Buffers*.
[7] Abkürzung für *Readout System*.
[8] Abkürzung für *Level-2 Processing Units*.

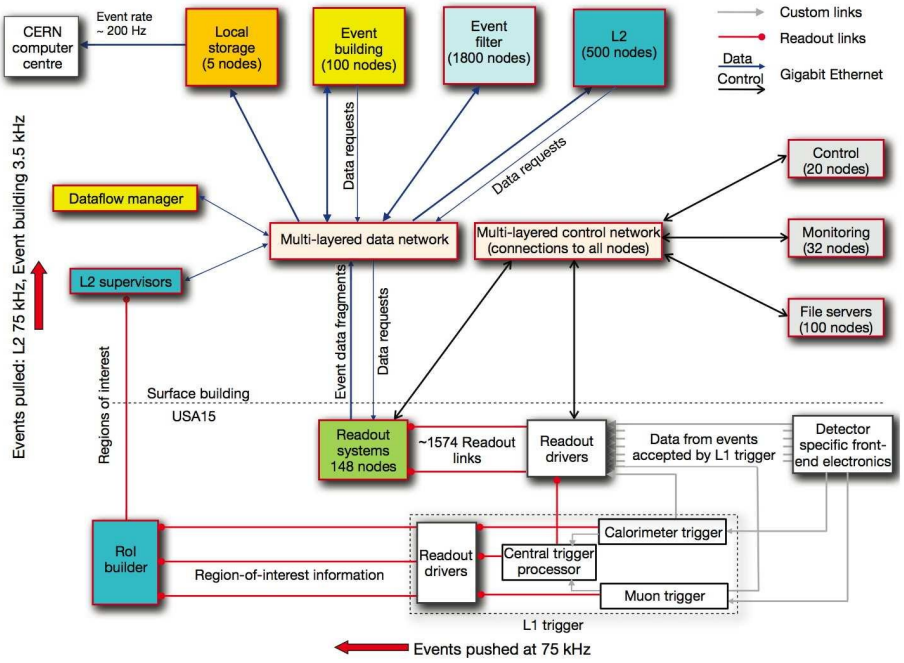


Abbildung 3.3: Blockdiagramm der *High Level Trigger* und des Datennahmesystems von ATLAS [5]

lesesystem an und geben das Ergebnis der Analyse an den L2SV zurück.
Der L2SV reicht das Ergebnis an den *DataFlow Manager* (DFM) weiter. Wurde das Ereignis akzeptiert, weist der DFM es einem *event builder*-Knoten (SFI[9]) zu, der alle Ereignisdaten von dem Auslesesystem anfordert, diese in einem Datenobjekt (*event*) zusammenführt und es dann dem *Event Filter* zur Analyse übergibt. Im Falle einer Selektion klassifiziert der EF das Ereignis, ordnet es einem *event stream* zu und sendet es an einen Ausgabeknoten (SFO[10]). Dieser speichert das Ereignis seiner Klassifizierung entsprechend lokal und übermittelt es der zentralen Datenspeichereinrichtung des CERN.
Abgesehen von der Kontrolle der Daten im Triggersystem bietet das Datennahmesystem Möglichkeiten zur Konfiguration, Kontrolle und Beobachtung des ATLAS-Experiments. Die Hardware-bezogene Überwachung durch das Detektorkontrollsystem (DCS) ist unabhängig von dem Datennahmesystem und besitzt eine eigene Infrastruktur.

[9] Abkürzung für *Subfarm Input*.
[10] Abkürzung für *Subfarm Output*.

3.4 Der Level-1 Kalorimeter Trigger

Überblick

Der Level-1 Kalorimeter Trigger ist ein digitales System mit fester Latenzzeit, das in speziell angefertigter Elektronik umgesetzt wurde. Er befindet sich außerhalb des Detektors in der Elektronik-Kaverne *USA15*. Die analogen Eingangssignale stammen von ungefähr 7200 sogenannten *Trigger-Towern*; dies sind Zellen reduzierter Granularität der Größe 0.1×0.1 in $\Delta\eta \times \Delta\phi$, die auf dem Detektor durch analoge Summen der Signale von elektromagnetischen und hadronischen Kalorimeterzellen gebildet werden. Die Bearbeitungszeit der L1Calo-Elektronik beträgt weniger als eine µs. Wenn zudem die durch Kabellängen bedingte Verzögerung und die Rechenzeit des CTPs berücksichtigt werden, vergrößert sich die Gesamtbearbeitungszeit auf 2.1 µs. Dies liegt innerhalb von 2.5 µs, was der maximalen Speicherdauer der Ereignisdaten in den Front-end Detektorsystemen entspricht.

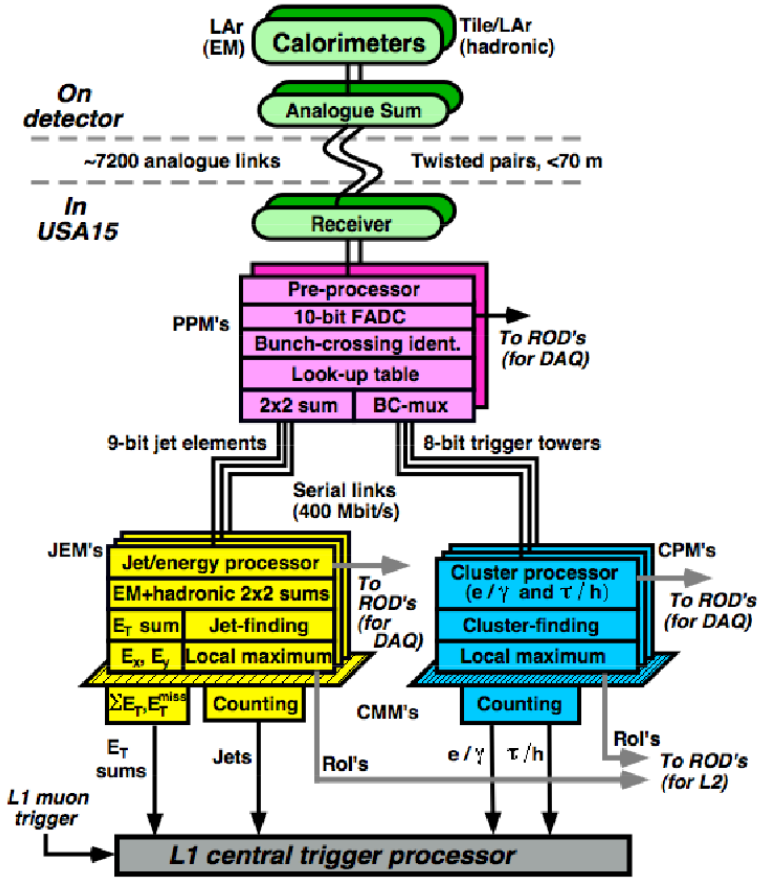


Abbildung 3.4: Blockdiagramm des Level-1 Kalorimeter Triggers. Die grauen Pfade repräsentieren Auslesedaten, die schwarzen Pfade stellen den Echtzeit-Weg der Daten innerhalb des Triggersystems dar. [10]

Das L1Calo-System besteht aus drei Subsystemen (siehe Abb. 3.4):

- Der *PreProcessor* (PPr) digitalisiert die analogen *Trigger-Tower*-Signale der Kalorimeter und ordnet sie einem *Bunch Crossing* zu. Da sich die Signale typerscherweise über 5 *Bunch Crossings* erstrecken, wird bei der Zuordnung digitale Filtertechnik eingesetzt. Die Subtraktion des Signalsockels, Rauschunterdrückung, finale E_T-Kalibration und das Abschalten problematischer *Trigger-Tower* wird mit Hilfe von sogenannten *Look-up*-Tabellen (LUT) in einem Schritt durchgeführt. Die Daten werden danach parallel zu den zwei anderen Subsystemen weitergeleitet.

- Der *Cluster*-Prozessor (CP) sucht nach Elektronen, Photonen und τ-Leptonen, die sowohl die programmierbaren E_T-Schwellen überschreiten als auch die Isolationskriterien erfüllen.

- Der Jet-Energiesummen-Prozessor (JEP) identifiziert Jets, die die programmierbaren E_T-Schwellen überschreiten. Aufgrund der räumlichen Ausdehnung von Jets beruht er auf Kalorimeterdaten gröberer Granularität. Desweiteren berechnet der JEP durch globale Summenbildung die totale und fehlende transversale Energie sowie die totale transversale Energie nur von Jets.

Der CP und der JEP senden die Multiplizitäten der verschiedenen getriggerten Objekte und Energien zur Weiterverarbeitung an den CTP. Die Daten des L1Calo-Systems werden über RODs ausgelesen und umfassen sowohl Ergebnisse der Ereignisprozessierung als auch Zwischenergebnisse, die zur Kalibration, Überwachung und Verifikation des Triggers verwendet werden können. Zeitgleich zur Auslese durch das DAQ-System werden die gefundenen ROIs dem *ROI-Builder* übergeben.
Die L1Calo-Module sind mit dem Detektorkontrollsystem verbunden, um deren Spannungen und Temperaturen zu überwachen.

Das analoge Front-end

Die *Trigger-Tower* haben die Größe 0.1×0.1 in $\Delta\eta \times \Delta\phi$, im FCAL und in den Endkappen sind sie etwas größer (siehe Abb. 3.5). Die Zellen des elektromagnetischen und des hadronischen Kalorimeters werden in voller Tiefe verwendet. Die *Trigger-Tower*-Signale entsprechen in den EM-Kalorimetern der transversaler Energie und in den HAD-Kalorimetern der Gesamtenergie. Die Anzahl der Kalorimeterzellen, die einen *Trigger-Tower* formen, variiert je nach Region[11].
Die analogen *Trigger-Tower*-Signale werden durch 616 abgeschirmte 16-fach *twisted-pair*-Kabel vom Detektor zum L1Calo-System übertragen. Um die Kabel insgesamt kurz zu halten, wurde die Abschirmung der USA15- von der ATLAS-Kaverne durch spezielle Kabellöcher durchbrochen.
Im L1Calo-System werden die Signale in *Receiver*-Modulen vorverarbeitet [11] :

[11] Im EM-Zentralbereich sind es 60 Zellen, in den Endkappen weniger. Im TileCal setzen sich die meisten *Trigger-Tower* aus 5 Photomultiplier-Signalen zusammen.

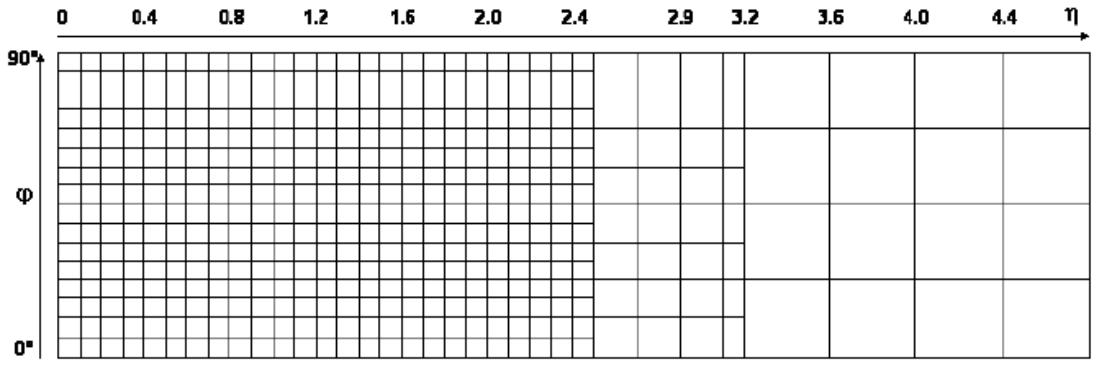


Abbildung 3.5: Die Granularität der *Trigger-Tower* für $\eta > 0$ und einen Quadrant in ϕ [10]

- **Signalverarbeitung**. Die Abschwächung der Signale in den Kabeln wird ausgeglichen. Die Signale werden feinkalibriert, sodass $E_T = 250\,\text{GeV}$ der Spannung $2.5\,\text{V}$ entspricht.
- **Signalordnung**. Die Signale werden den Anforderungen des Triggersystems entsprechend umgeordnet.
- **Überwachung**. Die *Receiver*-Module stellen eine begrenzte Auswahl an programmierbaren Ausgängen bereit, die die einzige Möglichkeit darstellen, auf analoge Kalorimetersignale zuzugreifen, wenn der ATLAS-Detektor nicht zugänglich ist.

Die so vorverarbeiteten Signale werden nun dem L1Calo *PreProcessor* übergeben.

Der „PreProcessor“

Der *PreProcessor* (PPr) digitalisiert die *Trigger-Tower*-Signale, ordnet sie einem *Bunch Crossing* zu und führt die finale Kalibration durch, damit sie anschließend in den algorithmischen Prozessoren weiterverwendet werden können. Der PPr besteht aus 124 *PreProcessor*-Modulen (PPMs), die in acht *Crates* untergebracht sind. Jeweils vier *Crates* verarbeiten die EM- beziehungsweise HAD-Signale. Um die LHC-Taktung zu verteilen, besitzt jedes *Crate* ein *Timing Control* Modul. Ein PPM verarbeitet 64 *Trigger-Tower*-Signale, die über vier analoge Kabel empfangen werden. In Abb. 3.6 findet sich ein Photo eines PPMs.

Analoger Signaleingang

Am Eingang des PPMs werden die differentiellen Signale in unipolare umgewandelt. Dabei werden diese für die Digitalisierung entsprechend verstärkt und mit einer Vorspannung versehen. Mit Hilfe eines programmierbaren DACs[12] wird die Sockelspannung ein-

[12] Abkürzung für *Digital to Analog Converter*.

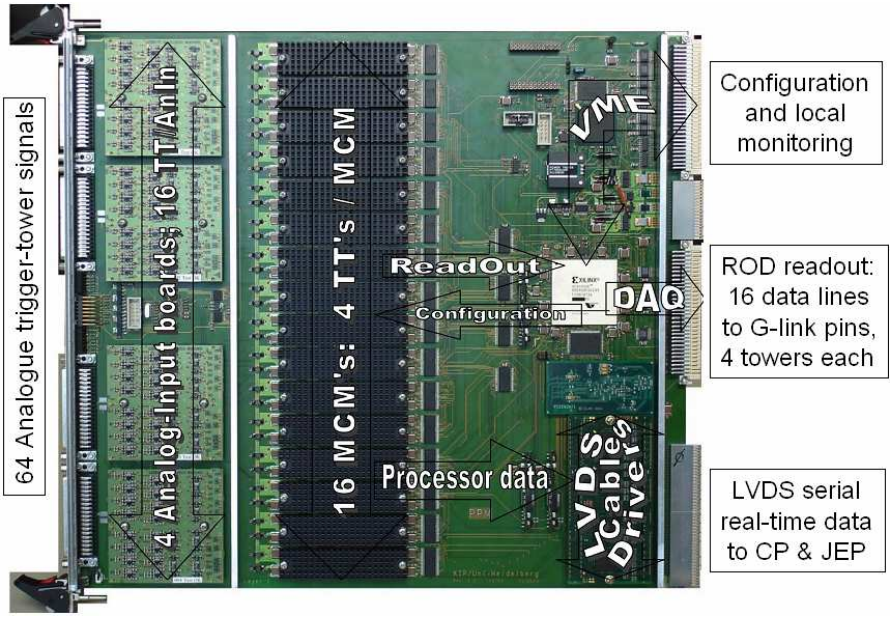


Abbildung 3.6: Photo eines PPMs mit Kennzeichnung der Hauptkomponenten [10]

gestellt. Die so veränderten Signale saturieren, falls sie einer Energie größer als 250 GeV entsprechen.

Das „PreProcessor Multi-Chip“ Modul

Die eigentliche Signalverarbeitung innerhalb des PPMs findet in den 16 *Multi-Chip* Modulen (MCMs) statt, die jeweils 4 *Trigger-Tower* verarbeiten. Ein Photo eines MCMs mit Hervorhebung der wichtigsten Komponenten findet sich in Abb. 3.7.
Die analogen Eingangssignale werden mit einer Frequenz von 40 MHz (*Bunch Crossing*-Frequenz) durch FADCs[13] mit einer Genauigkeit von 10 Bit digitalisiert[14]. Die digitalisierten Werte werden dann an den PPr-ASIC[15] weitergereicht[16]. Um die Signale trotz der Flugzeitdifferenzen und der durch die unterschiedlichen Kabellängen bedingten Verzögerungen auswerten zu können, werden sie durch den Einsatz zweier Komponenten zeitlich synchronisiert. Die erste Komponente ist ein am CERN entwickelter ASIC namens *PHOS4* und verschiebt die Abtastphase der FADCs in 1 ns Schritten bis zu 25 ns. Da dies der Zeit zwischen zwei *Bunch Crossings* (BC) entspricht, kann das Maximum der Signale immer mit der LHC-Taktung synchronisiert werden. Die zweite Komponente ermöglicht Zeitkorrekturen in 25 ns Schritten mit Hilfe eines Schieberegisters programmierbarer Länge und ordnet die Signale dem richtigen BC zu. Nach der Synchronisation werden die 10-Bit Werte im PPr-ASIC als Adresse einer *Look-up*-Tabelle (LUT) ver-

[13] Abkürzung für *Flash Analog to Digital Converter*.
[14] Streng genommen sogar mit 12 Bit Genauigkeit, die zwei niedrigsten Bits werden jedoch verworfen.
[15] Abkürzung für *Application Specific Integrated Circuit*.
[16] Dieser wurde im Heidelberger ASIC-Labor entworfen und in 0.6 µm-Technologie hergestellt.

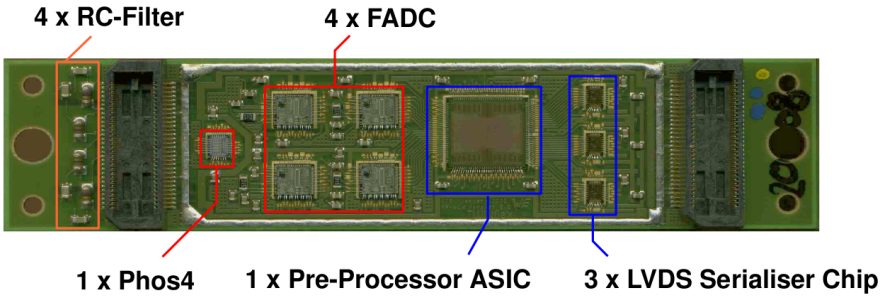


Abbildung 3.7: Photo eines *PreProcessor Multi-Chip* Moduls ohne Abdeckung [11]

wendet. Hierdurch werden die folgenden Operationen in einem Schritt durchgeführt: Subtraktion eines Signalsockels, Rauschunterdrückung, finale E_T-Kalibration und das Abschalten problematischer *Trigger-Tower*. Der Ausgang der LUTs ist ein 8-Bit Wert, der E_T in 1 GeV Einheiten entspricht. Signale der Größenordnung 1 GeV werden auf 0 gesetzt, da sie wahrscheinlich Ausleserauschen darstellen. Saturierte Signale werden auf den maximalen Wert 255 gesetzt. Die PPr-ASICs haben für jeden *Trigger-Tower* Register, die die Anzahl von Signalen über einer programmierbaren Energieschwelle zählen. Dies ermöglicht das Erstellen von Histogrammen mit Signalraten, die vollkommen unabhängig von der sonstigen Triggerlogik sind und zur Diagnose bei Fehlfunktionen herangezogen werden können.
Abschließend werden die LUT-Werte der vier *Trigger-Tower* des MCMs zu einem *jet element* addiert. Es entspricht einer Größe von 0.2×0.2 in $\Delta\eta \times \Delta\phi$ und besitzt einen E_T-Wert der Länge von 9 Bits. In seltenen Fällen tritt bei der Summation ein Überlauf auf, das *jet element* wird dann auf den Maximalwert gesetzt. Die *jet elements* sind die Eingangsdaten des Jet-Energiesummen-Prozessors, die LUT-Werte der vier *Trigger-Tower* sind die Eingangsdaten des *Cluster*-Prozessors.
Nach Durchlaufen des PPr-ASICs müssen die Daten im MCM zur Weiterübermittlung serialisiert werden, da die parallele Übertragung von 64 Kanälen eine zu große Anzahl von Kabeln und Steckern benötigen würde. Dies erledigen drei *National Semiconductor LVDS* Serialisierer mit einer Rate von 400 Mbit/s.

„Bunch Crossing“ Identifikation (BCID)

Die richtige Zuordnung von Kalorimetersignalen zu *Bunch Crossings* ist entscheidend. Nur wenn diese korrekt ist, können die Teilchenkollisionen ausgewertet werden. In Abb. 3.8 werden die unterschiedlichen Formen der *Trigger-Tower*-Pulse für das *Tile*- und das Flüssig-Argon-Kalorimeter (LAr) dargestellt. Während die LAr-Pulse bipolar sind und einen langen, negativen Unterschwung haben, sind die Tile-Pulse unipolar. Trotz der unterschiedlichen Formen der Pulse und Länge von mehreren BCs muss die BCID-Logik sicher den genauen Zeitpunkt der Signalspitze finden. Der PPr-ASIC setzt hierfür drei unabhängige Methoden ein:

- Für normale, nicht saturierte Pulse wird ein *finite impulse-response* (FIR) Filter

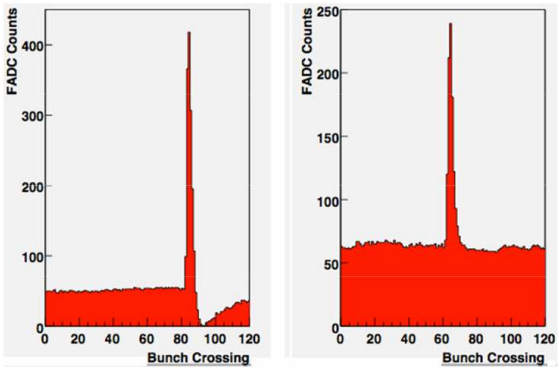


Abbildung 3.8: Mit 40 MHz von dem PPM digitalisierte Pulse des Flüssig-Argon-Kalorimeters (links) und des *Tile*-Kalorimeters (rechts) [10]

verwendet. Die Funktionsweise wird in Abb. 3.9 verdeutlicht. Durch Multiplikation von fünf hintereinander abgetasteten Werten mit vorgegebenen Koeffizienten und anschließender Summation wird das Signal-über-Rausch-Verhältnis verbessert. Die Koeffizienten sind für die Pulsformen des jeweiligen Kalorimeters optimiert. Das Maximum wird durch einfache Vergleiche der Ergebnisse des FIR-Filters mit den zwei direkt benachbarten FIR-Filter-Ergebnissen bestimmt. Simulationen haben gezeigt, dass diese Methode für Energien von wenigen GeV bis zu Energien nahe dem Saturationslevel (255 GeV) gut funktioniert [12].

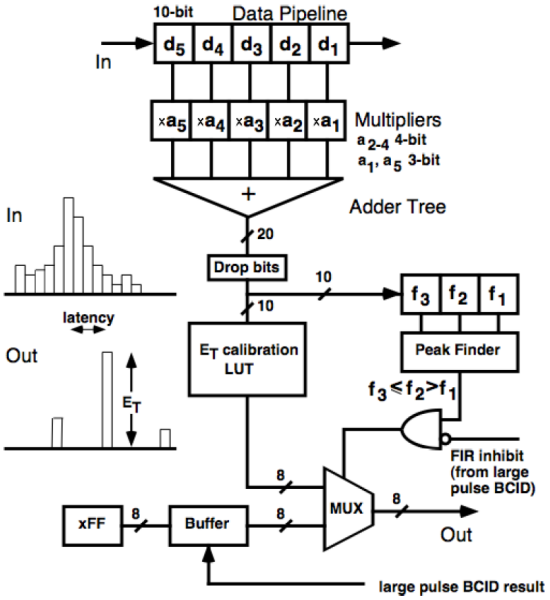


Abbildung 3.9: Der Einsatz von FIR-Filtern auf Kalorimeterpulsen zur Identifikation der zugehörigen BCs [10]

- Da saturierte Signale keine klar definierte Spitze haben, wird bei diesen zur BC-Zuordnung die steigende Flanke genutzt. Es wird eine tiefe und eine hohe Schwelle definiert und anschließend die Zeit gemessen, die ein Signal benötigt, um nach Überschreiten der tiefen Schwelle die hohe Schwelle zu überschreiten. Aus dieser Zeit kann auf den Ort der Signalspitze geschlossen werden, wenn das Signal nicht saturiert wäre. Diese Methode liefert ab Energien von 200 GeV gute Ergebnisse.

- Die dritte Methode detektiert Signale, sobald sie einen bestimmten Schwellenwert erreicht haben, und addiert dann auf diesen Zeitpunkt eine vorgegebene Pulsform-abhängige Verzögerung. Die so ermittelten Positionen der Signalspitze sind über weite Energiebereiche gute Abschätzungen. Diese Methode wird hauptsächlich zur Überprüfung der anderen zwei Methoden eingesetzt.

Der „Cluster"- und der Jet-Energiesummen-Prozessor

Der CP besteht aus 56 *Cluster*-Prozessor Modulen (CPMs), die über vier *Crates* verteilt sind. Jedes *Crate* verarbeitet die Kalorimeterdaten aus einem Quadrant in ϕ (siehe Abb. 3.10). Der JEP besteht aus 32 Jet/Energie Modulen (JEMs), die über 2 *Crates* verteilt sind. Diese *Crates* verarbeiten jeweils die Kalorimeterdaten aus zwei Quadranten in ϕ. Jedes CPM oder JEM ist einer dünnen Scheibe in η und 90° in ϕ zugeordnet. Die CPMs und JEMs senden ihre Ergebnisse an zwei auf den *Crates* sitzende *Common Merger* Module (CMMs). Diese führen die Ergebnisse der verschiedenen Module des *Crates* zusammen und leiten sie an CMMs weiter, die abschließend das Ergebnis des Gesamtsystems berechnen und an den CTP senden.

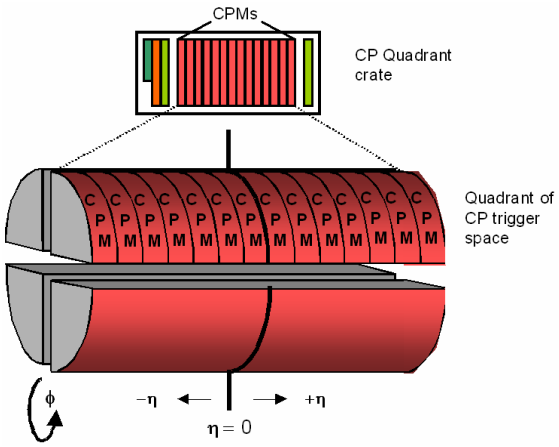


Abbildung 3.10: Ein CPM-Crate mit 14 CPMs deckt einen Quadrant in ϕ ab [10]

Der CP berücksichtigt Kalorimeterdaten aus dem Bereich $\eta < |2.5|$. Um eine hohe Genauigkeit der E_T^{miss}- und Gesamt-E_T-Trigger zu erreichen, berücksichtigt der JEP Kalorimeterdaten aus dem gesamten η-Bereich.

Die e/γ- und τ/Hadronen-Algorithmen

Die Aufgabe eines CPMs ist das Auffinden und Auswerten von Teilchenschauern im zugeordneten Bereich. Hierfür werden zwei Algorithmen eingesetzt: Einer für Elektronen und Photonen sowie einer für τ-Leptonen und Hadronen. Beide Algorithmen werden auf alle überlappenden 4×4 großen *Trigger-Tower* Fenster angewendet und aufgrund ihrer Ähnlichkeit zusammen ausgeführt. Der e/γ-Algorithmus sucht nach schmalen Schauern mit hohem E_T im EM-Kalorimeter. Zur Unterdrückung des großen hadronischen Untergrunds kann gefordert werden, dass die Schauer nicht in das HAD-Kalorimeter eindringen und räumlich isoliert auftreten. Der τ/Hadronen-Algorithmus identifiziert τ-Leptonen dadurch, dass diese nach Zerfall in Hadronen zu kollimierten Schauern im EM- und HAD- Kalorimeter führen.

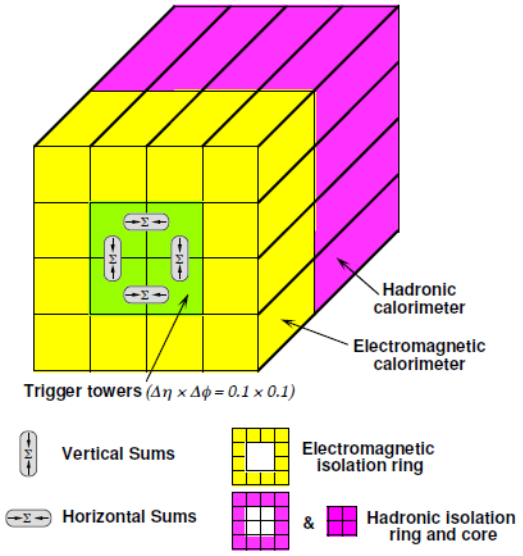


Abbildung 3.11: Die Eingangselemente des e/γ und τ/Hadronen Algorithmus [13]

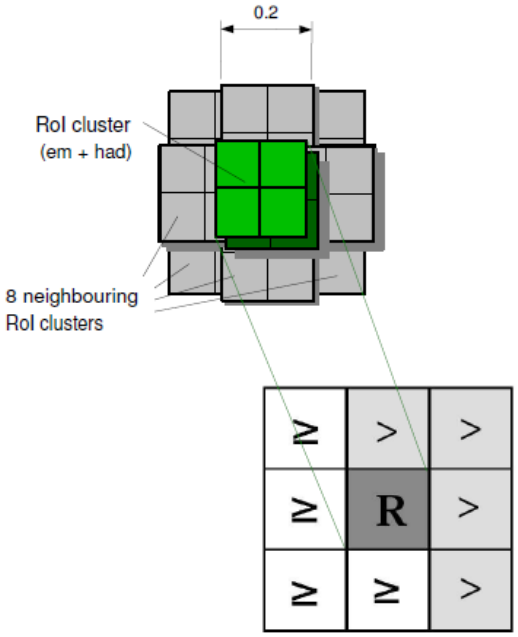


Abbildung 3.12: Der lokale E_T-Maximum-Test wird auf *ROI Cluster* der Größe 0.2×0.2 in $\Delta\eta \times \Delta\phi$ angewendet. Zu sehen sind ebenfalls die benachbarten *Cluster*. [14]

Der e/γ-Algorithmus addiert im 2×2 großen Zentrum des betrachteten 4×4 großen Fensters jeweils die E_T-Werte benachbarter EM *Trigger-Tower* (siehe Abb. 3.11). Es wird gefordert, dass mindestens eine der vier Summen eine *Cluster*-Schwelle überschreitet. Durch die Summenbildung werden schmale Schauer auch dann gefunden, wenn sie sich auf der Grenze zweier *Trigger-Tower* befinden. Der τ/Hadronen-Algorithmus arbeitet ähnlich, addiert jedoch vor dem Vergleich mit einer *Cluster*-Schwelle zu jeder der vier Summen noch die E_T-Gesamtsumme der hadronischen 2×2 Zentrums-*Trigger-Tower*.

Der e/γ-Algorithmus kann zusätzlich überprüfen, ob die E_T-Summe der hadronischen Zentrums-*Trigger-Tower* unter einer Veto-Schwelle liegt. Dies entspricht einem Schauer, der im EM-Kalorimeter gestoppt wurde.
Um sicherzustellen, dass ein Schauer lokal isoliert ist, kann gefordert werden, dass die E_T-Summe der 12 EM *Trigger-Tower*, die das Zentrum umgeben, eine EM Isolations-Schwelle nicht überschreiten. Analog können auch HAD Isolations-Schwellen festgelegt werden.
Im CP können 16 Kombinationen von Cluster- und Isolations-Schwellen programmiert werden. Acht dieser Kombinationen sind für den e/γ-Algorithmus reserviert, die restlichen können dem e/γ- oder τ/Hadronen-Algorithmus zugeordnet werden.
Es kann vorkommen, dass die Bedingungen der Algorithmen zeitgleich in überlappenden Fenstern erfüllt werden, wie zum Beispiel durch ein hochenergetisches Photon oder Elektron, das seine Energie in nur einem *Trigger-Tower* deponiert. Um dennoch eine eindeutige 2×2 große *Cluster ROI* definieren zu können, wird ein lokaler Maximum-Test angewendet. In diesem werden die E_T-Summen[17] der entsprechenden Zentrums-*Trigger-Tower* wie in Abb. 3.12 dargestellt mit den E_T-Summen ihrer acht benachbarten 2×2 großen *Trigger-Tower*-Zentren verglichen. Da digitale Werte gleich sein können, sind die Vergleiche in eine Richtung von dem Typ „größer gleich“ und in die andere von dem Typ „echt größer“.

Die Jet-Algorithmen

In den JEMs werden Jet-Algorithmen auf den zugeordneten Bereich angewendet. Da Jets eine große räumliche Ausdehnung haben, sind die Basiselemente der Jet-Algorithmen die *jet elements*. Die Jet-Algorithmen arbeiten ähnlich wie die e/γ- und τ/Hadronen-Algorithmen, fassen jedoch EM und HAD *Trigger-Tower* zusammen. Die Fenster, über deren Elemente die E_T-Summen gebildet werden, haben eine Größe von 2×2, 3×3 sowie 4×4 *jet elements*. Ein kleineres Fenster kann Jets genauer auflösen und besser auseinanderhalten, ein größeres Fenster beinhaltet hingegen einen größeren Teil der Jet-Energie und hat deswegen eine höhere Effizienz. Eine Jet *ROI* hat die Größe von 2×2 *jet elements* und muss ein lokales E_T-Maximum sein. Der eingesetzte Maximum-Test ähnelt dem der e/γ- und τ/Hadronen-Algorithmen. Es können acht Kombinationen von Schwellen vorgegeben werden, die jeweils aus einer Fenstergröße und einer E_T-Schwelle bestehen. Die Multiplizität der Jets wird bis zum Limit von 7 berechnet und dann an den CTP gesendet.
Die JEMs führen zudem erste Vorberechnungen von E_T^{miss} und E_T^{total} durch. Die endgültige Bestimmung und der Vergleich mit Energie-Schwellen wird in den CMMs durchgeführt.

[17] In beiden Algorithmen werden hier sowohl HAD wie EM *Trigger-Tower* berücksichtigt. Der e/γ Algorithmus arbeitet nicht wesentlich effizienter, wenn in diesem nur EM *Trigger-Tower* berücksichtigt werden.

Kapitel 4

Überwachung des ATLAS-Experiments

In diesem Kapitel wird ein Überblick über die *Online*-Überwachungs-Infrastruktur und die Systeme zur Fernüberwachung des ATLAS-Experiments gegeben.

4.1 Die Online-Überwachung

Das *Online Monitoring Framework* ist eine Umgebung, die das Verteilen, Analysieren und Überwachen der operativen Daten ermöglicht. Operative Daten umfassen unter anderem die Qualität und Integrität der aufgenommenen Physikdaten, die Konsistenz der Triggerinformationen, den Zustand der Detektor-Subsysteme sowie den Status von Hardware- und Softwarekomponenten.
Das *Online Monitoring Framework* basiert auf dem sogenannten *Information Service* (IS). Dieser ist eine zentrale Einrichtung zum Austausch von Daten. Die Hauptkomponenten des IS sind die Anwendungs-Schnittstellen und die *Information Server*, in denen die Daten liegen, die ausgetauscht werden. Die *Information Server* setzen auf Datenbanksysteme auf, deren Struktur vor den Nutzern des IS verborgen wird, da diese ausschließlich Kenntnis der Anwendungs-Schnittstellen benötigen. In Abb. 4.1 wird die

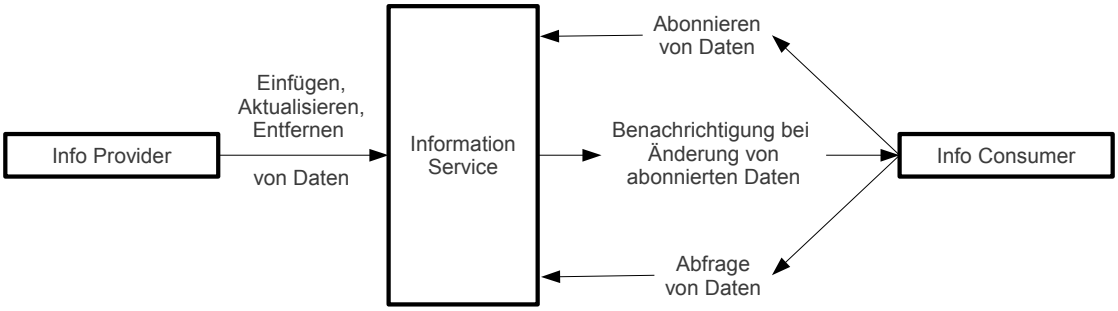


Abbildung 4.1: Die Funktionsweise des *Information Service*.

Funktionsweise des IS dargestellt: *Information Provider* aktualisieren die Daten im IS, fügen neue hinzu oder entfernen Daten. Die *Information Consumer* fragen die IS-Daten ab. Sie können Daten abonnieren und werden dann immer benachrichtigt, wenn sich diese ändern. Ein Beispiel für einen *Information Provider* ist der CTP, der die aktuellen Level-1-Triggerraten in den IS schreibt. Ein Beispiel für einen *Information Consumer* ist eine Anwendung, die die Raten graphisch darstellt.
Daten im IS können einfache Variablen oder komplexe Klassen sein. Der *Online Histogramming Service* (OHS) basiert auf dem IS und erweitert diesen um Schnittstellen

zum Austausch von Histogrammen. Unterstützte Histogrammformate sind das ROOT[1]-Format und ein einfaches Roh-Format, das die Histogramminhalte in Vektorform enthält.

4.2 Systeme zur Fernüberwachung

Die Infrastruktur zur Überwachung des ATLAS-Detektors befindet sich aufgrund von Sicherheitsbestimmungen innerhalb eines streng abgeschirmten Netzwerks (*Point 1*). Es werden daher drei spezielle Systeme zur Fernüberwachung angeboten:

- Das *NX Monitoring*
- Das *Web Monitoring Interface*
- Das *Web-IS Interface*

Diese werden im folgenden Abschnitt verglichen und deren Zielgruppen sowie Vor- und Nachteile herausgestellt. Damit die Datennahme nicht beeinflusst oder gestört wird, können alle Fernüberwachungssysteme nur lesend auf die IS- oder OHS-Daten zugreifen.

Das „NX Monitoring“

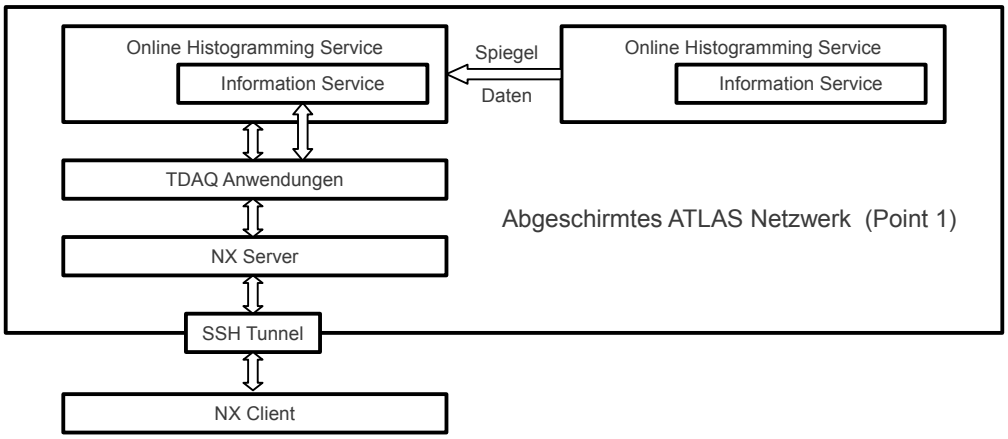


Abbildung 4.2: Blockdiagramm der Funktionsweise des *NX Monitoring*. Die Pfeile beschreiben den Datenfluss und dessen Richtung.

Das *NX Monitoring* basiert auf der Übertragung des Bildschirminhalts eines Rechners, der sich innerhalb *Point 1* befindet, auf einen Rechner außerhalb. Der Anwender kann dann so an diesem arbeiten, als ob er sich innerhalb *Point 1* befindet. Ein Blockdiagramm der Funktionsweise befindet sich in Abb. 4.2. Die Übertragung des Bildschirminhalts erledigt die *Remote-Desktop*-Software NX[2]. Die Verbindung läuft über einen

[1] Eine am CERN entwickelte, objektorientierte Software zur Analyse von Daten.

[2] NX wird von der italienischen Firma *NoMachine* entwickelt und steht in Teilen unter einer *Open-Source*-Lizenz. Die schnelle Übertragung des Bildschirminhalts wird durch eine Datenkompression des Netzwerkverkehrs erreicht. Nähere Information finden sich unter [15].

verschlüsselten *SSH*-Tunnel. Da nur authorisierte Personen den *SSH*-Tunnel erzeugen können, ist unbefugter Zugriff nicht möglich.
Während der NX-Sitzung können die TDAQ-Anwendungen wie in *Point 1* gestartet und genutzt werden. Um sicherzustellen, dass die TDAQ-Software nur zur Überwachung und nicht zur Steuerung eingesetzt werden kann, wurde folgende Infrastruktur gewählt: Der ATLAS IS Server wird gespiegelt und die auf den NX-Servern laufende TDAQ-Software hat ausschließlich Zugriff auf die gespiegelten Daten.
Der Vorteil des *NX Monitoring* für den Anwender ist, dass die zahlreichen von *Point 1* gewohnten TDAQ-Anwendungen zur Verfügung stehen. Der Nachteil ist die hohe Systemlast innerhalb *Point 1*. Zwei dafür zweckbestimmte Rechner ermöglichen nur eine begrenzte Anzahl von NX-Sitzungen. Zudem besitzen nicht alle ATLAS-Physiker die benötigten erweiterten Zugangsrechte, um eine NX-Sitzung zu starten.
Die Zielgruppe des *NX Monitoring* sind *Remote Shifter*. Aufgabe dieser ist es, die Personen im ATLAS Kontrollraum bei Problemen zu unterstützen. Die *Remote Shifter* sind meist Experten für bestimmte Subsysteme und benötigen während ihrer Schicht Zugriff auf alle TDAQ-Diagnosewerkzeuge.

Das „Web Monitoring Interface“

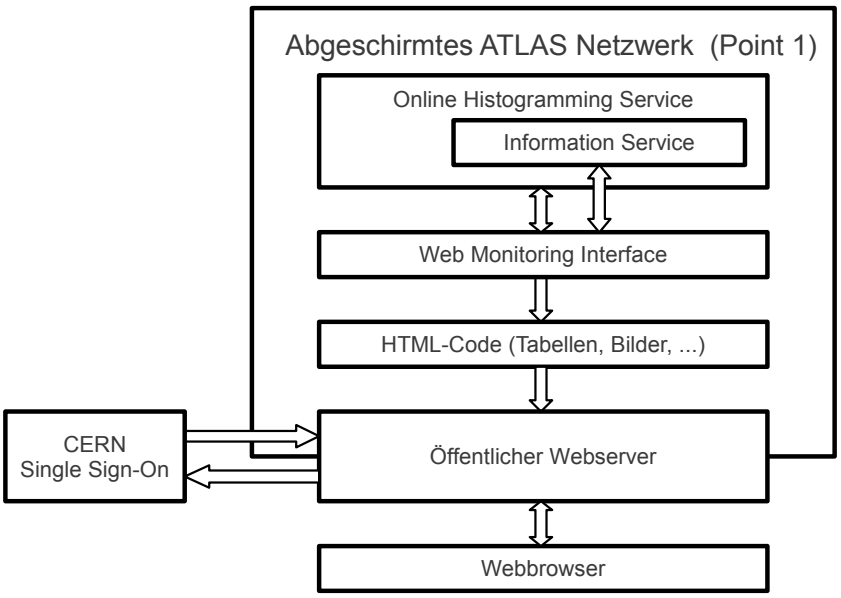


Abbildung 4.3: Blockdiagramm des *Web Monitoring Interface*. Die Pfeile beschreiben den Datenfluss und dessen Richtung.

Das *Web Monitoring Interface* (WMI) generiert aus den Daten im IS oder OHS Webseiten. In Abb. 4.3 wird die Funktionsweise dargestellt. Das WMI basiert auf der Programmiersprache C++ und besteht aus einem festen Kern (Server) und einer beliebigen Anzahl von *Plugins*. Der WMI-Server läuft innerhalb von *Point 1* und startet in konfi-

gurierbaren Intervallen die *Plugins*, die aus IS- und OHS-Daten HTML-Code erzeugen. Anschließend kopiert der WMI-Server diesen auf einen öffentlichen Webserver. In einem Webbrowser können die erzeugten Webseiten nach einer Authentifizierung mittels *CERN Single Sign-On* betrachtet werden. Nähere Informationen zur *CERN Single Sign-On* Authentifizierung finden sich in Abschnitt 6.1.
Der Vorteil der WMI-Architektur ist, dass die Richtung des Datenflusses immer von *Point 1* nach außen zeigt. Dadurch sind weder unerlaubte Zugriffe noch eine Beeinflussung der ATLAS-Datennahme möglich. Die Systemlast innerhalb *Point 1* ist unabhängig von der Anzahl der Webseitenabrufe. Der Nachteil des WMI ist, dass nur ein kleiner Teil der IS- und OHS-Informationen verfügbar gemacht werden kann. Das Erzeugen von HTML-Code für alle vorhandenen Informationen ist aufgrund deren Menge technisch schwer realisierbar. Informationen, die während der Generierung der Webseiten durch kein *Plugin* berücksichtigt werden, sind später nicht verfügbar. Das heißt, man muss bei der Erstellung der *Plugins* genau wissen, welche Informationen später gebraucht werden, dies ist jedoch kaum möglich.
Die Zielgruppe des WMI sind interressierte ATLAS-Physiker und Personen, die nur einen Überblick über den aktuellen Status des ATLAS-Experiments brauchen, da in den wenigsten Fällen notwendige Detailinformationen zur Fehleranalyse vorhanden sind.

Das „Web-IS Interface“

Das *Web-IS Interface* bietet über einen Webserver Zugang zu allen im IS und OHS enthaltenen Daten. In Abb. 4.4 findet sich ein Blockdiagramm, das die Funktionsweise beschreibt. Die Abfrage von IS- und OHS-Daten wird in einer URL[3] codiert, deren Aufruf das *Web-IS Interface* auf einem öffentlichen Webserver am CERN startet. Nach der Authentifizierung mittels *CERN Single Sign-On* werden die angefragten Informationen durch das *Web-IS Interface* von dem IS oder OHS angefordert und zurückgegeben. Es sind zwei Arten von Antworten möglich: Eine IS-Abfrage liefert die gewünschten Informationen in einer XML[4]-Darstellung. Eine OHS-Abfrage liefert das Histogramm in einem Format, das bei der Anfrage spezifiziert wurde. Das *Default*-Format ist PNG. Die unterstützten Formate entsprechen denen von ROOT, da das Histogramm im OHS in der Regel im ROOT-Format vorliegt und das *Web-IS Interface* dieses mit ROOT-Bibliotheken in andere Formate umwandeln kann. Aus demselbem Grund können in der Anfrage ROOT Histogramm-Parameter vorgegeben werden. So können beispielsweise logarithmische Achsen eingestellt werden. Werden die Histogramme im binären ROOT-Format angefordert, können diese unter ROOT genauer analysiert werden.
Der Vorteil des *Web-IS Interface* im Gegensatz zum WMI ist die Verfügbarkeit aller IS- und OHS-Informationen. Die verursachte Systemlast innerhalb *Point 1* ist geringer als im *NX Monitoring*. Durch die Übertragung der IS-Daten in XML-Darstellung und Verbindungen über gewöhnliche Webserver, gibt es nur zwei Anforderungen an einen möglichen *Client*: Das HTTP(S)-Protokoll muss unterstützt und ein XML-Parser vor-

[3] Abkürzung für *Uniform Resource Locator*. Beschreibt die Adresse einer Ressource in Netzwerken.
[4] Abkürzung für *Extensible Markup Language*. Diese stellt hierarchisch strukturierte Daten in Textform dar.

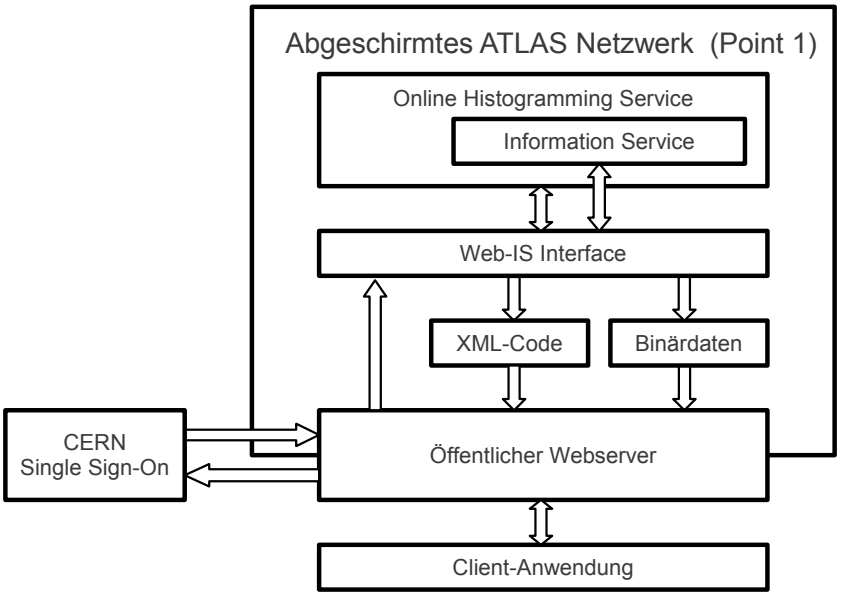


Abbildung 4.4: Blockdiagramm des *Web-IS Interface*. Die Pfeile beschreiben den Datenfluss und dessen Richtung.

handen sein. Ein Webbrowser zum Beispiel erfüllt diese Voraussetzungen und kann die IS-Daten in Verbindung mit einer XSL-Stilvorlage[5] formatiert wiedergeben. Der Nachteil des *Web-IS Interface* ist, dass jede Anfrage Rechenlast innerhalb *Point 1* verursacht. Um sicherzustellen, dass es zu keiner Störung der ATLAS-Datennahme kommen kann, wurden Belastungstests durchgeführt. Diese zeigten, dass der IS Server durch eine große Zahl von Anfragen nicht beeinträchtigt wird[6]. Der Grund hierfür liegt im Aufbau des *Web-IS Interface*. Es besteht aus einer festen Anzahl von identischen, parallel laufenden Python-Skripten, die nur lesend auf den IS oder OHS zugreifen und Anfragen jeweils sequentiell bearbeiten. Die Anzahl der Skripte stellt einen natürlichen Flaschenhals dar und ermöglicht es, die Systemlast zu kontrollieren. Hierbei wird in Kauf genommen, dass sich Anfragen stauen können, denn der reibungslose Betrieb der Datennahme hat Priorität.
Die Zielgruppe des *Web-IS Interface* ähnelt der des WMI. Im Gegensatz zu diesem ist es zusätzlich für Experten geeignet, die zwar Zugriff auf alle Informationen des IS oder OHS benötigen, jedoch nicht auf TDAQ-Anwendungen angewiesen sind. Die Dokumentation des *Web-IS Interface* findet sich unter [17].

[5] Abkürzung für *Extensible Stylesheet Language*. Eine in XML notierte Transformationssprache zur Definition von Layouts für XML-Dokumente.
[6] Nähere Informationen zu den Belastungstests finden sich unter [16].

Kapitel 5

Entwicklung eines Systems zur Fernüberwachung

Ziel dieser Arbeit ist die Erstellung einer Anwendung zur Fernüberwachung des Level-1 Kalorimeter Triggers. Voraussetzung hierfür ist der lesende Zugriff von außen auf den IS innerhalb *Point 1*. Um diesen zu ermöglichen, wurde als Basis der Anwendung ein eigenes System zur Fernüberwachung entwickelt, da das *Web-IS Interface* zu Beginn dieser Arbeit noch nicht existierte[1]. Das *NX Monitoring* ist aufgrund der limitierten Anzahl an Sitzungen nur eingeschränkt verfügbar und das *Web Monitoring Interface* bietet nur Zugriff auf eine kleine Teilmenge der IS-Daten.
Als sich herausstellte, dass parallel zum eigenen System am CERN das *Web-IS Interface* entstand, wurde dieses als Basis gewählt, da es zusätzlich auf OHS-Daten zugreifen kann. Der Wechsel stellte keine Probleme dar, da sich die zugrunde liegenden Prinzipien sowie die eingesetzten XML-Darstellungen des *Web-IS Interface* und des eigenen Systems ähneln. In der weiteren Entwicklung wurde zusammengearbeitet und von der Anwendung zur Fernüberwachung benötigte Änderungen in das *Web-IS Interface* integriert.
Im Folgenden werden die Anforderungen an das eigene System formuliert und eine zugehörige Schnittstelle beschrieben. Diese nimmt IS-Anfragen von einem *Client* entgegen und liefert die angefragten Informationen zurück. Anschließend wird ein Prototyp beschrieben, welcher die definierte Schnittstelle auf *Server*-Seite umsetzt, sowie eine Anwendung, die die Schnittstelle auf *Client*-Seite implementiert.

5.1 Beschreibung des Systems

Die wichtigsten Anforderungen an das System können wie folgt zusammengefasst werden:

- Das System soll die Abfrage aller im IS vorhandenen Daten erlauben.
- Damit die Datennahme des ATLAS-Experiments nicht beeinträchtigt werden kann, muss das System Mechanismen zur Zugangskontrolle unterstützen.
- Das System muss sicherstellen, dass keine Schreibzugriffe auf die internen ATLAS-Systeme erfolgen können.
- Um den Aufwand bei Erstellung und Wartung gering zu halten, soll das System auf verbreiteten und bewährten Technologien aufsetzen.

[1] Der Starttermin war im März 2010.

- Das System soll dem *Client* die IS-Daten in einer plattform- und implementationsunabhängigen Form liefern.

Die Anforderungen werden durch den Einsatz folgender Technologien erfüllt: Innerhalb *Point 1* läuft ein Webserver, der IS-Anfragen einer *Client*-Anwendung entgegennimmt, die in einem *Query String* codiert sind. Ein *Query String* ist eine Zeichenfolge, die an eine URL angehängt wird. Im vorgeschlagenen System zeigt die URL auf ein Programm, welches den *Query String* decodiert und die gewünschten Informationen von dem IS anfordert. Diese werden an den Webserver zurückgegeben. Der Webserver sendet die Daten dann als Antwort der Anfrage an den *Client*. Eine Darstellung dieses Prozesses findet sich in Abb. 5.1. Direkte Anfragen von außen an einen Webserver innerhalb *Point 1* stellen ein Sicherheitsrisiko dar und werden ohne vorherige Authentifizierung nicht zugelassen. Zugangskontrollen können zum Beispiel mit dem in Abschnitt 6.1 vorgestellten *CERN Single Sign-On* Verfahren umgesetzt werden.

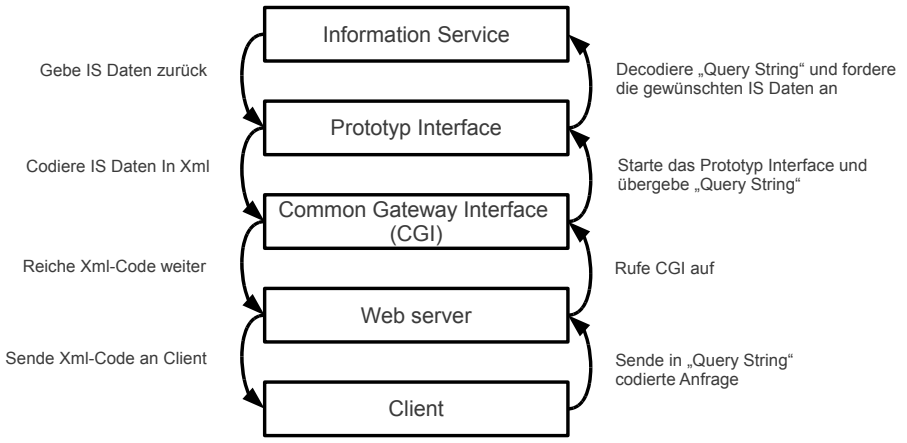


Abbildung 5.1: Schematische Darstellung der Beabeitung einer IS-Anfrage

Der Webserver startet das Programm, das die Daten von dem IS anfordert und zurückgibt, mittels CGI[2]. Zur Übertragung der IS-Daten wird deren Darstellung in XML vorgeschlagen. Ein großer Vorteil von XML ist, dass die Daten in einer selbsterklärenden Struktur angeordnet werden können und dass effiziente XML-Parser auf nahezu allen Plattformen verfügbar sind.

5.2 Der Prototyp

Im Folgenden wird nach einem Überblick über die Hierarchie der Daten im IS ein Prototyp des Programms vorgestellt, das mittels CGI von dem Webserver gestartet wird,

[2] Abkürzung für *Common Gateway Interface*. Dieses ist ein weit verbreiteter Schnittstellenstandard zum Datenaustausch zwischen einem Webserver und einem Programm, das die Anfragen bearbeitet. Die Spezifikation findet man unter [18]. Mittlerweile findet *FastCGI* mehr Verbreitung, das Geschwindigkeitsnachteile von CGI ausgleicht und als dessen Nachfolger gilt.

um die IS-Anfragen zu bearbeiten.

Hierarchie der Daten im „Information Service“

Die Daten innerhalb des IS sind, wie in Abb. 5.2 verdeutlicht, hierarchisch angeordnet. Die Elemente der obersten Ebene heißen Partitionen und können zur Laufzeit des IS gestartet oder gestoppt werden. Eine Partition beherbergt einen oder mehrere Server, welche wiederum sogenannte Objekte beinhalten. Die Objekte enthalten einer flexible Anzahl von Attributen. Dies sind Name-Wert-Paare der zur Überwachung relevanten operativen Daten.

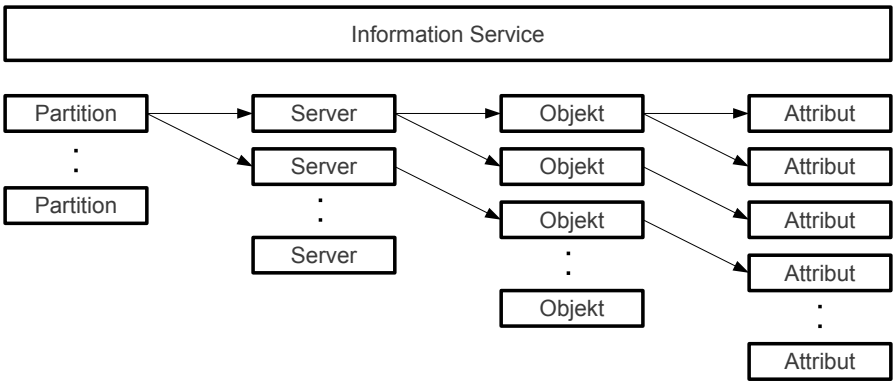


Abbildung 5.2: Die Hierarchie der Daten im *Information Service*

Einige Beispiele für IS-Attribute sind:

- Das Attribut L1A_COUNTER enthält die Anzahl der von dem Level-1-Trigger akzeptierten Ereignisse im aktuellen *Run*. Es befindet sich im Objekt ISCTPCORE des Servers L1CT.
- Das Attribut READOUTCONFIG enthält den Bezeichner der Auslese-Konfiguration des Level-1 Kalorimeter Triggers. Es befindet sich im Objekt GENERALPARS des Servers L1CALOSTATUS.
- Das Attribut ACTIVETIME enthält die bisherige Dauer des aktuellen *Run* in Sekunden. Es befindet sich im Objekt RUNINFO des Servers RUNPARAMS.
- Das Attribut BEAM_ENERGY enthält die aktuelle Strahlenergie in GeV. Es befindet sich im Objekt RUNPARAMS des Servers RUNPARAMS.

Alle genannten Attribute befinden sich in der Partition *ATLAS*.

Codierung der IS-Anfragen

Zur Codierung der IS-Anfragen in einem *Query String* bietet sich eine Struktur an, die die Hierarchie des IS widerspiegelt. In Tabelle 5.1 findet sich die für den Prototyp

gewählte Codierung. Durch das Senden eines leeren *Query String* können die Namen aller im IS verfügbaren Partitionen abgerufen werden. Um die Namen der Server einer Partition abzufragen, wird im *Query String* der Partitionsname angegeben. Durch die Angabe des zugehörigen Partitions- und Servernamen können die verfügbaren Objekte eines bestimmten Servers abgefragt werden. Eine Besonderheit ist die Abfrage der Attribute eines Objekts; in der Praxis kann das gleichzeitige Auslesen von Attributen verschiedener Objekte eines Servers hilfreich sein, daher kann statt eines konkreten Objektnamens auch ein regulärer Ausdruck angegeben werden. Die Antwort auf Anfragen dieses Typs enthält alle Objektnamen des angegebenen Servers, die den regulären Ausdruck erfüllen sowie alle zugehörigen Attribute. Die Codierung der Anfragen in einem *Query String* führt die *Client*-Anwendung durch.

IS-Anfrage	**Codierung in *Query String***
Partitionen	**?**
Server einer Partition	**?p=**Partition
Objekte eines Servers	**?p=**Partition**&s=**Server
Attribute eines Objekts	**?p=**Partition**&s=**Server**&o=**Objekt
Bestimmtes Attribut	**?p=**Partition**&s=**Server**&o=**Objekt**&a=**Attribut

Tabelle 5.1: Codierung einer IS-Anfrage in einem *Query String*

Decodierung der IS-Anfragen

Der Prototyp wird auf dem Server mittels CGI gestartet. Er wurde in C++ umgesetzt, da so eine hohe Ausführungsgeschwindigkeit erreicht wird und der IS eine C++ API[3] bereitstellt. Die Dokumentation dieser findet sich in [19]. Die Übergabe des *Query String* an den Prototyp mittels CGI geschieht durch das Setzen der Umgebungsvariablen QUERY_STRING. Diese wird innerhalb des Prototyps, wie in Listing 5.1 dargestellt, verarbeitet.

Listing 5.1: Decodierung des *Query String* im Prototyp

```
1 CGImap query(getenv("QUERY_STRING"));
2
3 std::string partition_name, server_name,
4       object_name, attribute_name;
5
6 partition_name = query["p"];
7 server_name    = query["s"];
8 object_name    = query["o"];
9 attribute_name = query["a"];
```

[3] Abkürzung für *Application Programming Interface*. API bezeichnet eine Schnittstelle, die ein Programm einem anderen Programm zur Anbindung zur Verfügung stellt.

Nachdem die Umgebungsvariable QUERY_STRING ausgelesen und dem Konstruktor eines Objekts der Klasse CGIMap[4] übergeben worden ist, werden die Anfrageparameter decodiert. Der Prototyp fordert die gewünschten Daten von dem IS an und gibt sie in einer XML-Darstellung zurück, die dem Typ der IS-Anfrage entspricht. Die kommentierte Basisdatei des Prototyps findet sich in Anhang A.1. Eine Diskussion des Auslesens der Daten aus dem IS und der Klassen, die den XML-Code erzeugen, findet hier nicht statt, da die Schnittstellen-Definition im Vordergrund stehen soll.

Gewählte XML-Darstellung der IS-Daten

Im Folgenden wird die gewählte XML-Darstellung der IS-Daten erläutert. Eine Abfrage der IS-Partitionen liefert eine Liste der Namen der vorhandenen Partitionen:

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes"?>
<is>
   <partitions>
      <part name = "PartitionA"/>
      ...
   </partitions>
</is>
```

Eine Abfrage der IS-Server einer bestimmten IS-Partition liefert neben deren Namen zusätzlich deren Startzeitpunkt, *Host*[5], Besitzer und PID[6]:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<is>
   <server>
      <serv name="ServerA" started="time of start"
            host="server host" owner="server owner" pid="server pid" />
      ...
   </server>
</is>
```

Eine Abfrage der IS-Objekte eines bestimmten IS-Servers liefert neben deren Namen zusätzlich deren Typ, Zeitpunkt der letzten Modifikation und Beschreibung:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<is>
   <objects partition="PartitionA" server="ServerA" >
      <object name="ObjectA" type="object type"
            modified="last modified time" descr="object description" >
      </object>
      ...
   </objects>
</is>
```

Eine Abfrage der IS-Attribute eines bestimmten IS-Objekts liefert neben deren Namen zusätzlich deren Wert, Typ und Beschreibung. Wenn in der Anfrage statt eines kon-

[4] Die Klasse CGIMap wurde [20] entnommen.
[5] Name des Rechners, auf dem die IS-Server Daten liegen.
[6] Abkürzung für *process identifier*. Bezeichnet eine eindeutige Proszessidentifikationsnummer.

kreten Objekts ein regulärer Ausdruck angegeben wird, kann die Antwort die Attribute mehrerer Objekte enthalten:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<is>
   <objects partition="PartitionA" server="ServerA" >
      <object name="ObjectA" type="object type"
            modified="last modified time" descr="object description" >
         <attr name="AttributeA" desc="attribute description"
               type="attribute time" val="attribute value" />
         ...
      </object>
      ...
   </objects>
</is>
```

Installation am CERN

Zu Testzwecken wurde der Prototyp auf einem öffentlichen Webserver am CERN installiert. Der den Tests zugrunde liegende IS befand sich außerhalb *Point 1*. Den Programmbibliotheken, die die Kommunikation mit dem IS bereitstellen, wird durch das Setzen der Umgebungsvariablen TDAQ_IPC_INIT_REF der Ort des IS mitgeteilt. Da dies vor dem Starten des Prototyps geschehen muss, wird die Variable in der Datei .HTACCESS gesetzt:

```
SetEnv TDAQ_IPC_INIT_REF file:/somePath/ipc_root.ref
```

Wenn sich diese Datei im gleichen Verzeichnis wie der Prototyp befindet, führt der Webserver bei einer Anfrage die in der Datei enthaltenen Anweisungen aus, bevor der Prototyp mittels CGI gestartet wird.

5.3 Der „Remote IS Monitor“

Der *IS Monitor* ist Teil der ATLAS TDAQ-Software und wird zum Anzeigen und Durchsuchen der im IS vorhandenen Daten genutzt. Um zu demonstrieren, dass das in diesem Kapitel vorgestellte System zur Fernüberwachung geeignet ist, wurde ein *Remote IS Monitor* erstellt, der nahezu die gleiche Funktionalität wie der *IS Monitor* bietet, jedoch auf dem Prototyp aufsetzt. Der *Remote IS Monitor* ist daher auch außerhalb *Point 1* lauffähig. Er wurde in Java entwickelt und kann mittels *Web Start*-Technologie aus einem Webbrowser heraus gestartet werden. Die Vorteile von Java und Java *Web Start* werden in Abschnitt 6.1 diskutiert.

Erzeugen des „Query String“

In Listing 5.2 befindet sich die Methode, die innerhalb des *Remote IS Monitor* die Abfrage von IS-Daten in einem *Query String* codiert. Diese wird exemplarisch erläutert, da eine Codierung in jedem *Client* durchgeführt werden muss. Die Parameter der Funktion beschreiben die geforderten IS-Daten. Der Anfragetyp des erzeugten *Query String* ergibt

Listing 5.2: Erzeugen des *Query String* im *Client*

```
1  public String createURL(String partitionName,
2              String serverName, String objectName)
3  {
4      try {
5          partitionName = java.net.URLEncoder.encode(partitionName, "UTF-8");
6          serverName = java.net.URLEncoder.encode(serverName, "UTF-8");
7          objectName = java.net.URLEncoder.encode(objectName, "UTF-8");
8      } catch (UnsupportedEncodingException ex) {
9          Logger.getLogger(IsData.class.getName())
10                 .log(Level.SEVERE, null, ex);
11     }
12     String urlBase = mSettings.get("UrlBase");
13     String urlPath = "";
14     if (!partitionName.isEmpty()){
15         urlPath = "p=" + partitionName;
16         if (!serverName.isEmpty()){
17             urlPath = urlPath + "&s=" + serverName;
18             if (!objectName.isEmpty()){
19                 urlPath = urlPath + "&o=" + objectName;
20             }
21         }
22     }
23     return (urlBase + "?" + urlPath);
24 }
```

sich aus den aufrufenden Parametern. Wenn kein Partitionsname angegeben wird, wird ein *Query String* erzeugt, der einer Abfrage der vorhandenen IS-Partitionen entspricht (Zeile 13). Wird nur ein Partitionsname angegeben, entspricht der *Query String* einer Abfrage aller zugehörigen IS-Server (Zeile 15). Die Angabe weiterer Parameter führt analog zu einem *Query String*, der einer Abfrage der IS-Objekte (Zeile 17) beziehungsweise IS-Attribute (Zeile 19) entspricht. Sehr wichtig ist das Maskieren von möglichen Sonderzeichen innerhalb der IS-Partitions-, Server- und Objektnamen (Zeile 5-7), anderenfalls werden diese von dem Webserver fälschlicherweise als Steuerzeichen interpretiert und der Prototyp wird nicht gestartet.

Der XML-Parser

Eine wichtige Komponente der *Client*-Anwendung ist der XML-Parser. Dieser extrahiert aus der XML-Antwort des Protoyps die IS-Daten. Grundsätzlich lassen sich XML-Parser in zwei Kategorien einteilen, die jeweils verschiedene Vor- und Nachteile haben. Zum einen gibt es die SAX-Parser[7], die ein XML-Dokument linear von Anfang bis Ende durchlaufen und ereignisbasiert sind. Ereignisse sind zum Beispiel der Start oder das Ende eines Elements. Wenn ein Ereignis auftritt, wird dies der Anwendung mitgeteilt, die den SAX-Parser einsetzt. Die Anwendung ist dann selbst für die Weiterverarbei-

[7] Abkürzung für *Simple API for XML*.

tung der Informationen verantwortlich. Im Gegensatz hierzu stehen die DOM-Parser[8]. Diese bilden das komplette XML-Dokument in einem sogenannten Dokumentenbaum im Speicher ab, von dem die enthaltenen Informationen durch bereitgestellte Methoden abgerufen werden können. Die Stärke der SAX-Parser ist ihre hohe Geschwindigkeit und der problemlose Einsatz mit großen XML-Dokumenten aufgrund ihres sequentiellen Charakters. Die Stärke der DOM-Parser sind komfortable Schnittstellen zur Manipulation der Dokumentenstruktur und gezielten Abfrage einzelner Elemente. Ein ausführlicher Vergleich beider Ansätze findet sich in [21].
Im *Remote IS Monitor* wird ein SAX-Parser eingesetzt, da dieser nicht speicherintensiv ist und Geschwindigkeitsvorteile bietet. Diese ergeben sich insbesondere daraus, dass der XML-Code wegen dessen Übertragung über das Internet als Datenstrom vorliegt und parallel zur Übermittlung verarbeitet werden kann. In Anhang A.2 befindet sich der kommentierte Quellcode des eingesetzten SAX-Parsers.

Einsatz in der Praxis

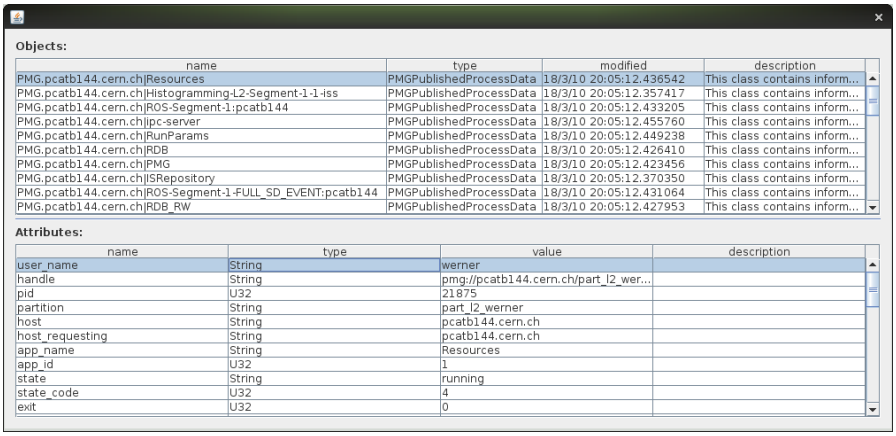


Abbildung 5.3: Die IS-Objekte Ansicht des *Remote IS Monitor*.

Nach dem Start fordert der *Remote IS Monitor* die aktiven IS-Partitionen an und stellt sie in einer Auswahlbox dar. Wenn eine IS-Partition ausgewählt wird, werden die zugehörigen IS-Server heruntergeladen und in einer Tabelle angezeigt. Ein Doppelklick auf einen IS-Server öffnet ein neues Fenster, in dessen oberer Hälfte die zugehörigen IS-Objekte dargestellt werden. Wenn ein IS-Objekt angeklickt wird, werden dessen IS-Attribute in der unteren Hälfte angezeigt (siehe Abb. 5.3). Alle Tabellen der Anwendung können sortiert werden.
Der Prototyp und der *Remote IS Monitor* als Beispiel-*Client* funktionieren in der Praxis wie gewünscht und demonstrieren, dass das vorgestellte System zur Fernüberwachung geeignet ist.

[8] Abkürzung für *Document Object Model*.

Kapitel 6

Fernüberwachung des Level-1 Kalorimeter Triggers

Dieses Kapitel beschreibt den Aufbau und die Funktionsweise der in dieser Arbeit entwickelten Anwendung zur Fernüberwachung des Level-1 Kalorimeter Triggers, im Folgenden *L1CaloDisplay* genannt. Ursprünglich wurde die Anwendung für L1Calo-Experten entwickelt, die sich nicht am CERN aufhalten und umfassende Informationen zum aktuellen Status des *L1Calo*-Systems benötigen. Mittlerweile wird sie auch von anderen ATLAS-Physikern genutzt, die sich einen Überblick über das Experiment verschaffen wollen. Das *L1CaloDisplay* ist zurzeit die einzige Anwendung mit graphischer Benutzeroberfläche, die es über das Internet ermöglicht, alle im IS vorliegenden Daten und Histogramme darzustellen und zu durchsuchen und die für alle ATLAS-Physiker zur Verfügung steht.

6.1 Eingesetzte Techniken

In diesem Abschnitt werden die bei der Entwicklung des *L1CaloDisplay* eingesetzten Technologien beschrieben sowie Besonderheiten und Herausforderungen bei der Umsetzung aufgezeigt. Eine Auflistung der bei der Entwicklung eingesetzten Software und der benutzten Bibliotheken findet sich in Anhang B.1.

Java

Das *L1CaloDisplay* wurde in Java entwickelt und kann mittels *Web Start* Technologie aus dem Webbrowser heraus gestartet werden. Die Motivation zur Verwendung von Java liegt in der Plattformunabhängigkeit der mit Java erstellten Anwendungen. Diese wird durch das Kompilieren des Java-Quellcodes in sogenannten *Bytecode* erreicht. Der *Bytecode* wird in einer Laufzeitumgebung namens *Java Virtual Machine* (JVM) ausgeführt. Die JVM ist selbst nicht plattformunabhängig, wird jedoch für die verbreiteten Betriebssysteme zur Verfügung gestellt. Sie muss auf dem Rechner, auf dem die Java-Anwendung ausgeführt werden soll, installiert sein.
Eine Erweiterung von Java stellt die *Web Start* Technologie dar. Diese ermöglicht es, eine Java-Anwendung aus einem Webbrowser heraus zu starten. Hierfür muss der Entwickler auf dem Server eine spezielle XML-Datei im Format *Java Network Launching Protocol* (JNLP) bereitstellen. Die JNLP-Datei enthält Beschreibungen der einzelnen Programmdateien, den Namen der Hauptklasse sowie Parameter für die aufrufende JVM. Ein richtig konfigurierter Webbrowser übergibt die JNLP-Datei nach Anklicken der lokalen JVM, die die benötigten Programmdateien von dem Server herunterlädt und ausführt. Die *Web*

Start Technologie sorgt dafür, dass die Programmdateien nur heruntergeladen werden, falls sie nicht schon im *Cache* vorhanden sind. Um sicherzustellen, dass der Anwender immer mit der aktuellsten Version der Anwendung arbeitet, werden stets die Zeitstempel der im Internet vorliegenden Programmdateien abgefragt und mit den Zeitstempeln der im *Cache* vorliegenden Programmdateien verglichen. Nähere Informationen zu Java *Web Start* finden sich unter [22].

Anfragen an das „Web-IS Interface“ und dessen Antwort

Um von dem *Web-IS Interface* IS- oder OHS-Daten abzufragen, wird eine URL aufgerufen. Der erste Teil der URL ist die Netzwerkadresse des *Web-IS Interface*, der zweite Teil beschreibt die angeforderten Daten. Das Prinzip, die angeforderten Daten in einer URL zu codieren, liegt auch dem in dieser Arbeit entwickelten Fernüberwachungssystem zugrunde. Zudem ähneln sich die XML-Darstellungen der IS-Daten und die Codierungen der Anfragen. In diesem Abschnitt werden daher lediglich Beispiele für konkrete *Web-IS Interface* Anfragen gegeben:

Wenn man den aktuellen *Prescale Key* des Level-1-Triggers des ATLAS-Detektors abrufen möchte, werden folgenden Informationen benötigt:

- Die URL des *Web-IS Interface*: **https://atlasop.cern.ch/info/current/**
- Der Name und Typ des IS-Objekts, in dessen Attributen die gewünschte Information vorliegt: **Physics.L1PsKey** und **TrigConfL1PsKey**
- Der Name des IS-Servers, in dem sich das IS-Objekt befindet: **RunParams**
- Der Name der IS-Partition, in der sich der IS-Server befindet: **ATLAS**

Die Informationen werden in folgender URL codiert:

`https://atlasop.cern.ch/info/current/ATLAS/is/RunParams/RunParams.Physics.L1PsKey?type=TrigConfL1PsKey`

Das *Web-IS Interface* gibt den aktuellen *Level-1 Prescale Key* zurück, wie in Listing 6.1 dargestellt (Zeile 5-7).

Listing 6.1: Antwort des *Web-IS Interface* auf eine Abfrage des *Level-1 Prescale Key*

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <objects release="current" partition="ATLAS" server="RunParams" tag="">
3     <obj name="RunParams.Physics.L1PsKey" type="TrigConfL1PsKey"
4           time="2/4/10 09:57:43.620000">
5        <attr name="L1PrescaleKey" descr="L1 prescale key"
6              type="u32">
7           <v>1272</v>
8        </attr>
9        <attr name="L1PrescaleComment" descr="L1 prescale comment"
10             type="string">
```

```
11            <v>Physics_lowRate_LArCosmics_noBPTX_RD1_1Hz</v>
12          </attr>
13      </obj>
14  </objects>
```

Wenn man sich ein Histogramm mit den aktuellen *Trigger-Tower*-Raten des EM-Kalorimeters anschauen möchte, werden folgende Informationen benötigt:

- Der Name des Histogramms: **l1calo_rates./2D_ELM**
- Der Name des IS-Servers, in dem sich das Histogramm befindet: **L1CaloHistos**
- Der Name der IS-Partition, in der sich der IS-Server befindet: **ATLAS**

Das Histogramm enthält die Raten aller *Trigger-Tower*, von Interesse sind jedoch nur Raten aus dem Bereich $-1 < \eta < 1$ und $0 < \phi < 3.14$. Die Histogramm-Informationen und die Darstellungsparameter werden in folgender URL codiert:

```
https://atlasop.cern.ch/info/current/ATLAS/oh/L1CaloHistos.l1calo_rates./
2D_ELM?rangex=-1,1&rangey=0,3.14
```

Das *Web-IS Interface* gibt ein Bild des Histogramms im PNG-Format zurück, das nur den gewünschten Bereich darstellt. Es existieren weitere Darstellungsparameter und ein Parameter, der das gewünschte Format des Histogramms vorgibt.
Die entwickelten Methoden zur Codierung einer IS- oder OHS-Anfrage in einem *Query String* und zum Parsen der XML-Antworten im *L1CaloDisplay* ähneln den in Kapitel 5 vorgestellten und werden daher nicht näher beschrieben.

Digitale Signierung des „L1CaloDisplay“

Die durch die *Web Start* Technologie bereitgestellte JVM läuft in der Regel mit eingeschränkten Rechten. Beispielsweise werden Netzwerkverbindungen nur zum Server der JNLP-Datei zugelassen und der Zugang zum Dateisystem des Anwenders stark eingeschränkt. Dies ist Teil des *Web Start* Sicherheitskonzepts und soll möglichen Schaden durch böshafte Software begrenzen.
Anwendungen, die vollen Zugriff auf das Dateisystem benötigen und eigenständig Netzwerkverbindungen aufbauen, müssen in einer JVM mit erweiterten Rechten gestartet werden. Die *Web Start* Technologie erlaubt dies, falls die Programmdateien auf dem Server mit einem digitalen Zertifikat signiert worden sind. Wenn die Anwendung per *Web Start* gestartet wird, überprüft eine Sicherheitsroutine, ob sich eine der Programmdateien seit der Signierung verändert hat. Ist dies der Fall wird das Starten der Anwendung abgebrochen. Anderenfalls erfragt ein Dialog, ob der Anwender dem Zertifikat vertraut. Nach einer Bestätigung wird die Anwendung mit erweiterten Rechten ausgeführt.
Die Programmdateien des *L1CaloDisplay* wurden signiert, da es Zugriff auf das *Web-IS Interface* und das lokale Dateisystem benötigt. Für die Signierung wurde ein Programm verwendet, das der Java-Installation beiliegt.

„Multithreading“ im „L1CaloDisplay“

Um die Plattformunabhängigkeit, der mit Java erstellten Anwendungen zu gewährleisten, muss Java eine eigene Graphikbibliothek zur Verfügung stellen, da sich diese von Plattform zu Plattform unterscheiden. Graphische Oberflächen unter Java werden in der Regel unter der Benutzung der Graphikbibiliothek *Swing* erstellt, welche selbst in Java geschrieben ist und einen umfangreichen Satz von graphischen Benutzerschnittstellen beinhaltet.
Bei der Entwicklung einer Anwendung mit graphischer Oberfläche ist es wichtig, alle zeitintensiven Programmbestandteile in separate Prozesse auszulagern. Dieses Verfahren bezeichnet man als *Multithreading*. Wird es nicht beachtet, blockiert der Programmprozess während zeitintensiver Operationen die Oberfläche und der Anwender empfindet die Anwendung als langsam. Das Auslagern geschieht mit Hilfe sogenannter *Threads*. Dies sind leichtgewichtige Prozesse, die durch einen übergeordneten Prozess gestartet werden. Da *Swing* nicht *Thread*-sicher ist, müssen bei der Entwicklung spezielle Strategien beachtet werden, um zu verhindern, dass zwei *Threads* zur selben Zeit auf dieselbe Komponente der Oberfläche zugreifen. Ansonsten kann es zu plötzlichen Programmabstürzen kommen, deren Ursache später nur schwer zu finden ist.

Listing 6.2: Die Klasse *SwingWorker* zum Auslagern zeitintensiver Operationen

```
1  //Definiere und instanziiere SwingWorker
2  SwingWorker<Object, Void> worker = new SwingWorker<Object, Void>()
3  {
4      //Hintergrund-Thread
5      @Override
6      protected Object doInBackground()
7      {
8          //Fuehre zeitintensive Operationen aus
9          ..
10         //Gebe das Ergebnis zurueck
11         return Object;
12     }
13     //Event-Dispatch-Thread
14     @Override
15     protected void done()
16     {
17         //Hole das Ergebnis
18         Object object = get();
19         //Stelle Ergebnis in der Oberflaeche dar
20         ..
21 };
22 //Fuehre SwingWorker aus
23 worker.execute();
```

Der Entwickler einer *Multithreading*-Anwendung muss dafür sorgen, dass die Methoden der *Swing*-Komponenten nie direkt aus verschiedenen *Threads* aufgerufen werden, sondern stets dem *Event Dispatch Thread* zur Ausführung übergeben werden. Dieser *Thread* arbeitet die ihm übergebenen Methoden sequentiell ab und stellt so sicher, dass zu keinem Zeitpunkt mehr als eine Methode auf eine *Swing*-Komponente zugreift.

In Java existiert zur Auslagerung von zeitintensiven Operationen eine Hilfsklasse namens *SwingWorker*, die das genannte Prinzip beachtet. Die Klasse bietet zwei zu überladende Methoden an (siehe Listing 6.2): Die eine Methode (Zeile 6) führt die zeitintensiven Operationen im Hintergrund aus, die andere (Zeile 15) wird im Anschluss dem *Event Dispatch Thread* zur Ausführung übergeben und darf auf die *Swing*-Komponenten der Oberfläche zugreifen. Nähere Infomationen zur *Multithreading*-Problematik unter *Swing* und zur *SwingWorker*-Klasse finden sich in [23].
Im *L1CaloDisplay* wird die *SwingWorker*-Klasse konsequent für alle rechen- oder zeitintensiven Programmbestandteile eingesetzt. Zu diesen gehören beispielsweise das Sortieren von Daten, das Warten auf die Antwort von IS- und OHS-Anfragen sowie die Aufbereitung von Überwachungsdaten vor dem Darstellen. Die Oberfläche wird zu keinem Zeitpunkt blockiert und reagiert stets auf von dem Anwender ausgelöste Ereignisse.

Dauerhaftes Speichern von Einstellungen

Im *L1CaloDisplay* können eine Reihe von Einstellungen festgelegt werden. Hierunter fallen Netzwerk- und Benutzereinstellungen sowie allgemeine Programmeinstellungen. Damit Einstellungen über die Laufzeit einer Anwendung hinaus bestehen bleiben, bietet Java *Web Start* eine spezielle API namens *PersistenceService* an. Mit dieser werden die Einstellungen in sogenannten *Muffins* lokal auf dem *Client*-Rechner gespeichert. *Muffins* sind das Java-Analogon zu den aus Webbrowsern bekannten *Cookies*. Die *Muffins* werden immer der URL zugeordnet, von welcher die erzeugende Anwendung heruntergeladen wurde. Die JVM stellt sicher, dass eine Anwendung immer nur die *Muffins* auslesen kann, die seiner URL zugeordnet sind. Durch diese Maßnahme kann eine per *Web Start* gestartete Anwendung den Anwender im Allgemeinen nicht ausspionieren[1].
Das *L1CaloDisplay* setzt *Muffins* sowohl zum Speichern von Einstellungen als auch zum Merken von Verzeichnissen in Datei-Dialogen ein.

Authentifizierung mittels „CERN Single Sign-On“

Der Zugriff des *L1CaloDisplay* auf das *Web-IS Interface* ist erst nach einer Authorisierung möglich. Für diese wird der von dem CERN bereitgestellte *Single Sign-On* Mechanismus genutzt. Dieses setzt auf das sogenannte *Shibboleth*-Verfahren auf, welches eine verteilte Authentifizierung und Authorisierung für Webanwendungen und Webseiten ermöglicht. Webserver am CERN, die Inhalte mit *Shibboleth* schützen, können zur Prüfung der Authorisation des Anwenders den zentralen CERN Authentifizierungsserver nutzen. Nähere Information zu *Shibboleth* finden sich unter [24].

Ablauf der Authentifizierung

Die zeitliche Abfolge der Authentifizierungsschritte wird in Abb. 6.1 dargestellt:

[1] Wenn die Anwendung mit uneingeschränkten Rechten gestartet wird, hat sie Zugriff auf das lokale Dateisystem und das Auslesen sensibler Daten kann nicht verhindert werden.

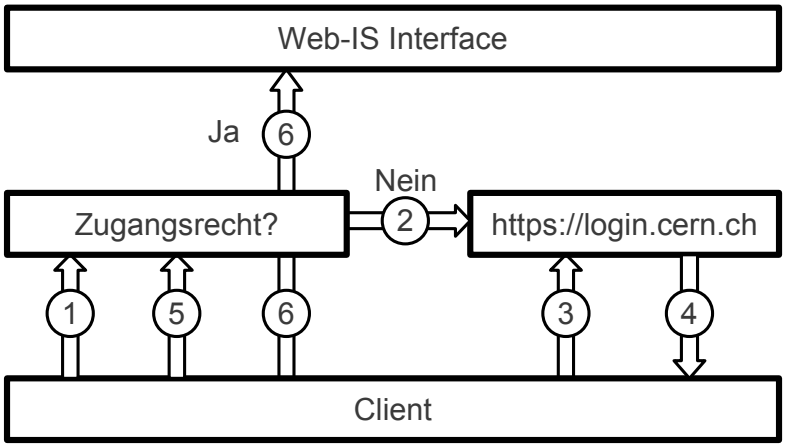


Abbildung 6.1: Ablauf der *CERN Single Sign-On* Authentifizierung

1. Das *L1CaloDisplay* sendet eine Anfrage an das *Web-IS Interface*. Der Webserver des *Web-IS Interface* überprüft die Zugangsberechtigung des Anwenders.
2. Bei der ersten Anfrage der Sitzung hat der Anwender noch keine Zugangsberechtigung und wird auf den zentralen *CERN Login Server* umgeleitet.
3. Das *L1CaloDisplay* sendet an diesen nun die von dem Anwender angegebenen Zugangsdaten, bestehend aus Benutzername und Passwort.
4. Der *CERN Login Server* überprüft die Gültigkeit der Zugangsdaten und generiert ein XHTML-Dokument, das zurück an das *L1CaloDisplay* gesendet wird. Das Dokument enthält Attribute, die die Zugangsrechte des Anwenders beschreiben.
5. Das *L1CaloDisplay* parst die in den Attributen enthaltenen Daten, erstellt aus diesen ein HTTP-Formular und sendet es an den Webserver des *Web-IS Interface*. Der Webserver prüft, ob die im Formular beschriebenen Rechte des Anwenders ausreichen und lässt, je nach Ausgang der Prüfung, von nun an Verbindungen zu. Aufgrund des niedrigen Implementationsaufwands wird im *L1CaloDisplay* zum Parsen des XHTML-Dokuments ein DOM-Parser verwendet. Um sicherzustellen, dass die Attributdaten nicht manipuliert wurden und von dem *CERN Login Server* stammen, sind die wesentlichen Informationen verschlüsselt und digital signiert.
6. Das *L1CaloDisplay* greift nach erfolgreicher Authorisierung auf das *Web-IS Interface* zu.

Während der Authentifizierung wird eine einmalige, nicht vorhersehbare Identifikations-Nummer der Sitzung generiert, die in einem HTTP-*Cookie* gespeichert wird. Dieses *Cookie* wird mit jeder Anfrage mitgesendet und ermöglicht dem *Web-IS Interface* Webserver, die Anfrage einer *L1CaloDisplay*-Instanz zuzuordnen. Diese hat dadurch ohne erneute Authentifizierung solange Zugriff auf das *Web-IS Interface*, bis die maximal erlaubte Sitzungsdauer erreicht ist.

Die Klasse „CernSsoConnection“

Die im Rahmen dieser Arbeit entwickelte Klasse *CernSsoConnection* bietet einen transparenten Zugang zu Webanwendungen und Webseiten, die eine Authentifizierung mittels *CERN Single Sign-On* erfordern. Sie wurde für das *L1CaloDisplay* entwickelt, kann aber ebenso in jeder anderen Java-Anwendung verwendet werden.
Die wichtigsten Eigenschaften der Klasse sind:

- Einfache Benutzung
- *Proxy Server*-Unterstützung
- *Multithreading*-Unterstützung
- Automatisches Erneuern der *CERN Single Sign-On* Sitzung bei Ablauf

Im Folgenden wird die Benutzung der *CernSsoConnection*-Klasse anhand ihrer öffentlichen Methoden erläutert:

```
public static void initialize()
```

Diese Methode initialisiert die Klasse und muss vor allen anderen Methoden ausgeführt werden. Da die Standard-Java-Bibliotheken zum Erstellen von HTTP-Verbindungen keine automatische *Cookie*-Verwaltung bieten, werden stattdessen die *httpclient*-Bibliotheken von *Apache* verwendet[2]. Diese enthalten zudem einen *Multithreading*-fähigen Verbindungsmanager, der in dieser Methode instanziiert wird.

```
public static void setProxy(String proxyHost, String proxyPort,
                             String proxyUser, String proxyPass)
```

Diese Methode setzt einen *Proxy Server*, über den alle Verbindungen laufen sollen. Die Angabe eines *Proxy*-Benutzernamens und eines *Proxy*-Passworts kann entfallen. Die Methode ist optional und wird nicht ausgeführt, wenn die Verbindung direkt ist.

```
public static void setUserAndPassforSSO(String user, String pass)
```

Diese Methode setzt den Benutzernamen und das Passwort, das in der *CERN Single Sign-On* Authentifizierung benutzt wird, und ist optional. Wenn sie nicht ausgeführt wird, können nur Verbindungen zu nicht geschützten Webanwendungen oder Webseiten aufgebaut werden.

```
public static InputStream getInputStream(String url)
```

Diese Methode stellt eine Verbindung zu einer geschützten oder ungeschützten URL her und gibt den zugehörigen Datenstrom zurück. Der Datenstrom enthält die Daten, die der Webserver an die anfragende Anwendung sendet. Wenn auf eine URL zugegriffen wird,

[2] Nähere Informationen zu diesen finden sich unter [25].

die nicht geschützt ist, entfällt die Authentifizierung. Eine Anfrage an eine URL, die mittels *CERN Single Sign-On* geschützt ist, löst das Ausführen einer privaten Methode aus, die die Authentifizierung durchführt. Die automatische *Cookie*-Verwaltung stellt sicher, dass in jeder Anfrage die Identifikations-Nummer der Sitzung übertragen wird, welche von dem Webserver benötigt wird, um die Authorisierung bis zum Ablauf der maximal erlaubten Sitzungdauer aufrecht zu erhalten. Wenn diese überschritten ist, führt die Methode bei der nächsten Anfrage einer durch *CERN Single Sign-On* geschützten URL automatisch die erneute Authentifizierung durch. Die Methode ist *Thread*-sicher und kann parallel aus verschiedenen *Threads* aufgerufen werden.

Sperren der Authentifizierungs-Methode

Der *Multithreading*-fähige Verbindungsmanager ermöglicht parallele Verbindungen, die von verschiedenen *Threads* aufgebaut werden. Dennoch genügt dies allein nicht, um die *CernSsoConnection*-Klasse *Multithreading*-fähig zu machen. Es muss garantiert werden, dass zur Vermeidung folgender Problematik immer nur ein *Thread* die Authentifizierungs-Methode durchführt: Ein Thread stellt eine Verbindung zu einer URL her, die in einem mit *CERN Single Sign-On* geschützten Bereich liegt, und wird auf den zentralen *Login Server* umgeleitet. Bevor die Authentifizierung abgeschlossen ist, stellen weitere *Threads* Verbindungen zum geschützten Bereich her, werden ebenfalls umgeleitet und rufen parallel die Authentifizierungs-Methode auf. Der *Login Server* kann die verschiedenen Authentifizierungsversuche der *Threads* nicht auseinanderhalten und sie misslingen. Selbst wenn die parallele Authentifizierung der *Threads* möglich wäre, sollte diese zur Optimierung der Geschwindigkeit und Minimierung des Datenverkehrs nur ein *Thread* durchführen.
Daher wurde ein Mechanismus implementiert, der die Authentifizierungs-Methode sperren kann. Der erste *Thread*, der auf einen mit *CERN Single Sign-On* geschützten Bereich zugreift, sperrt die Authentifizierungs-Methode und führt sie aus; alle anderen *Threads* warten solange. Der Mechanismus beruht auf der privaten *Boolean*-Variable

```
private static Boolean mAuthenticationIsLocked
```

Deren Wert beschreibt den aktuellen Status der Sperre: *true* entspricht dem gesperrten und *false* dem nicht gesperrten Status. Die Sperr-Methoden, über die der Status der Sperre festgelegt und abgefragt wird, sind:

```
private static synchronized Boolean requestLock()
```

Diese Methode fordert eine Sperre an. Wenn diese schon gesetzt ist, wird *false* zurückgegeben. Ansonsten wird die Sperre eingerichtet und der Wert *true* zurückgegeben.

```
private static synchronized void releaseLock()
```

Diese Methode hebt die Sperre auf.

Listing 6.3: Sperren und Entsperren der Authentifizierungs-Methode

```
1  private static synchronized Boolean requestLock(){
2      if (mAuthenticationIsLocked){
3          return false;
4      } else {
5          mAuthenticationIsLocked = true;
6          return true;
7      }
8  }
9
10 private static synchronized void releaseLock(){
11     mAuthenticationIsLocked = false;
12 }
13
14 private static synchronized Boolean isLocked(){
15     return mAuthenticationIsLocked;
16 }
```

`private static synchronized Boolean isLocked()`

Diese Methode gibt den Status der Sperre zurück.

Alle Sperr-Methoden enthalten in ihrer Deklaration das Schlüsselwort **synchronized**. Dies teilt der JVM mit, dass die Methoden *Thread*-sicher ausgeführt werden müssen: Während der Ausführung einer Sperr-Methode in einem *Thread*, kann kein zweiter *Thread* eine Sperr-Methode ausführen.[3].
Listing 6.3 beinhaltet den vollständigen Quellcode der Sperr-Methoden. Wenn auf das **synchronized**-Schlüsselwort verzichtet wird, kann folgender Fall eintreten: Ein *Thread* fordert durch den Aufruf der Methode requestLock() (Zeile 1) die Sperre der Authentifizierungs-Methode an. Da sie noch nicht gesetzt ist (Zeile 2), soll sie eingerichtet werden (Zeile 5). Unmittelbar vor dem Sperren wird die Ausführung des *Threads* unterbrochen und ein zweiter *Thread* ruft die Methode requestLock() auf. Da der erste *Thread* die Sperre noch nicht eingerichtet hat, will dieser ebenfalls eine Sperre einrichten. Unabhängig davon wie die *Threads* nun wechseln, wird beiden mitgeteilt, die Authentifizierungs-Methode gesperrt zu haben (Zeile 6). Die *Threads* führen daher parallel die Authentifizierung durch und sie misslingt.
Der Einsatz der Sperr-Methoden in der getInputStream-Methode der *CernSsoConnection*-Klasse wird in Listing 6.4 demonstriert. Wenn ein *Thread* auf eine URL zugreift, die in einem mit *CERN Single Sign-On* geschützten Bereich liegt, wird die Anfrage, für den Fall dass noch keine Authentifizierung durchgeführt oder die maximale Sitzungsdauer erreicht wurde, auf den zentralen *Login Server* umgeleitet.

[3] Die JVM führt eine Liste aller Objekte, die aus einer **synchronized**-Methode heraus verwendet werden, und merkt sich für jedes der Objekte, von welchen **synchronized**-Methoden es verwendet wird. Führt ein *Thread* eine **synchronized**-Methode aus, stellt die JVM sicher, dass zum gleichen Zeitpunkt kein zweiter *Thread* eine **synchronized**-Methode ausführt, die ein gleiches Objekt verwendet.

Die getInputStream-Methode überprüft ständig, ob eine Anfrage umgeleitet wurde (Zeile 2), und veranlasst gegebenenfalls die Authentifizierung. Der erste *Thread*, der die Sperre der Authentifizierungs-Methode anfordert (Zeile 3), führt die Authentifizierung durch (Zeile 5). Für diesen *Thread* liefert die Funktion requestLock() den Wert *true* zurück.
Alle weiteren *Threads*, deren Anfragen umgeleitet worden sind, fordern ebenfalls die Sperre an, werden jedoch abgewiesen, da die Sperre schon eingerichtet wurde. In diesem Fall liefert die Funktion requestLock() den Wert *false* zurück. Diese *Threads* warten nun solange (Zeile 11-12) in einer Schleife, bis der erste Thread die *Authentifizierung* abgeschlossen hat und die Sperre aufhebt (Zeile 8). Die getInputStream-Methode führt die ursprünglichen Anfragen der *Threads* automatisch noch einmal aus und liefert die angeforderten Daten zurück.
Das Schlüsselwort **finally** in Zeile 7 bewirkt, dass die Sperre selbst dann aufgehoben wird, wenn in der Authentifizierungs-Methode ein unvorhergesehener Fehler auftritt. Nun kann ein anderer *Thread* die Sperre anfordern und die Authentifizierungs-Methode ein weiteres Mal aufrufen.

Listing 6.4: Authentifizierung im Falle einer Umleitung

```
1  ..
2  if (redirectedUrl.startsWith("https://login.cern.ch/adfs/ls")){
3      if (requestLock()){
4          try {
5              statusCode = authenticateSso(redirectedUrl, mUser, mPass);
6          } ..
7          finally {
8              releaseLock();
9          }
10     } else {
11         while (isLocked()){
12             Thread.sleep(100);
13         }
14 ..
15 }
```

Verschlüsseln der Verbindung

Das *L1CaloDisplay* setzt in Verbindungen das *HTTP over SSL*[4] Protokoll (HTTPS) ein. Mit SSL wird eine abhörsichere Verbindung zwischen zwei Parteien hergestellt, die im Folgenden *Server* und *Client* genannt werden. Die sendende Partei verschlüsselt die Daten und die empfangende Partei entschlüsselt sie. SSL-Verbindungen beruhen auf einer symmetrischen Verschlüsselung, das heißt, die Schlüssel zum Ver- und Entschlüsseln der Daten sind identisch. Der Vorteil dieser Verschlüsselung ist der niedrige Rechenaufwand auf *Server*- und *Client*-Seite. Eine Schwachstelle der symmetrischen Verschlüsselung ist der Austausch des Schlüssels. Nur wenn sichergestellt wird, dass außer dem *Client* und dem *Server* keine dritte Partei den Schlüssel besitzt, ist die Verbindung sicher.

[4] Abkürzung für *Secure Sockets Layer*. Mittlerweile ist SSL durch dessen Nachfolger TLS (*Transport Layer Security*) abgelöst worden, der Begriff SSL hat sich dennoch gehalten.

Für die Übertragung des symmetrischen Schlüssels setzt SSL die rechenintensive asymmetrische Verschlüsselung ein. Diese beruht auf einem Paar nicht identischer Schlüssel, die invers zueinander sind. Daten, die mit einem der beiden Schlüssel verschlüsselt werden, können nur mit dem anderen entschlüsselt werden. Der *Server* veröffentlicht einen der beiden Schlüssel (öffentlicher Schlüssel) und behält den anderen für sich (privater Schlüssel). Verschlüsselt der *Client* Daten mit dem öffentlichen Schlüssel, kann sie nur der *Server* entschlüsseln, da nur er den benötigten privaten Schlüssel hat. Daten, die der *Client* mit dem öffentlichen Schlüssel des *Server* entschlüsselt, können aus dem gleichen Grund nur von dem *Server* stammen. Der *Client* generiert einen symmetrischen Schlüssel, verschlüsselt diesen mit dem asymmetrischen Schlüssel des *Server* und sendet ihn an diesen. Nur der *Server* kann diesen symmetrischen Schlüssel lesen und stellt mit ihm eine gesicherte Verbindung zum *Client* her.
Es ist wichtig, dass der *Client* den richtigen öffentlichen Schlüssel des *Server* kennt, ansonsten kann eine dritte Partei vorgeben, der *Server* zu sein. Dieses Problem wird mit Hilfe digital signierter Zertifikate gelöst, die den öffentlichen Schlüssel und den Namen des Zertifikateigentümers enthalten. Die Signatur kann als fälschungssicherer Ausweis betrachtet werden, der von einer Einrichtung ausgestellt wird, der man vertraut. Diese bürgt dafür, dass das Zertifikat wirklich dem genannten Eigentümer gehört.
Die vertrauenswürdigen Zertifikate speichert Java in einer sogennanten *Keystore*-Datei. SSL-Verbindungen zu einem *Server*, dessen Zertifikat nicht im *Keystore* enthalten ist, werden nicht erlaubt. Im Rahmen dieser Arbeit wurde eine *Keystore*-Datei mit den benötigten öffentlichen CERN-Zertifikaten erstellt, die das *L1CaloDisplay* zur Authentifizierung mittels *CERN Single Sign-On* und zur Kommunikation mit dem *Web-IS Interface* benötigt. Diese *Keystore*-Datei wird auf der Website, von der das *L1CaloDisplay* gestartet wird, zum Herunterladen angeboten. Die Signatur der Zertifikate kann durch Hilfsprogramme überprüft werden, die der Java-Installation beiliegen. Eine Anleitung zum Erstellen der *Keystore*-Datei befindet sich in Anhang B.2.

Aktualisierung der Daten

Eine Grundvoraussetzung für eine effektive Überwachung ist die regelmäßige Aktualisierung der dargestellten Informationen. Im *L1CaloDisplay* befinden sich die überwachten Daten thematisch sortiert in verschiedenen Klassen. Diese laden unter Einsatz spezieller Methoden zur Kommunikation mit dem *Web-IS Interface* die IS- oder OHS-Daten herunter und stellen sie in der Oberfläche dar. Jede der Klassen besitzt eine öffentliche `Update()`-Methode, die zur Aktualisierung aufgerufen werden kann.
In der Basisklasse des *L1CaloDisplay* existiert ein Aktualisierungsmanager, der diese Methoden in regelmäßigen Abständen ausführt. Er basiert auf einer Java-Klasse namens *ScheduledExecutorService*, die in regelmäßigen Abständen *Threads* startet und ausführt. Die Benutzung der Klasse wird in Listing 6.5 deutlich. In Zeile 1 wird eine Instanz erzeugt, die bis zu 10 *Threads* gleichzeitig ausführt. Aus den Aktualisierungsmethoden werden *Threads* erzeugt (Zeile 4-8), die dem *ScheduledExecutorService* übergeben werden (Zeile 3). Dieser startet die *Threads* in regelmäßigen Intervallen, die bei der Übergabe festgelegt werden (Zeile 9).

Listing 6.5: Beispiel für die Nutzung des *ScheduledExecutorService*

```
1  scheduler = Executors.newScheduledThreadPool(10);
2  ..
3  scheduler.scheduleAtFixedRate(
4     new Runnable() {
5        @Override
6        public void run() {
7           jPanelStatus.UpdateRootControllerStatus()
8     }
9  }, 0, updateInterval, TimeUnit.SECONDS);
10 ..
```

6.2 Beschreibung des „L1CaloDisplay“

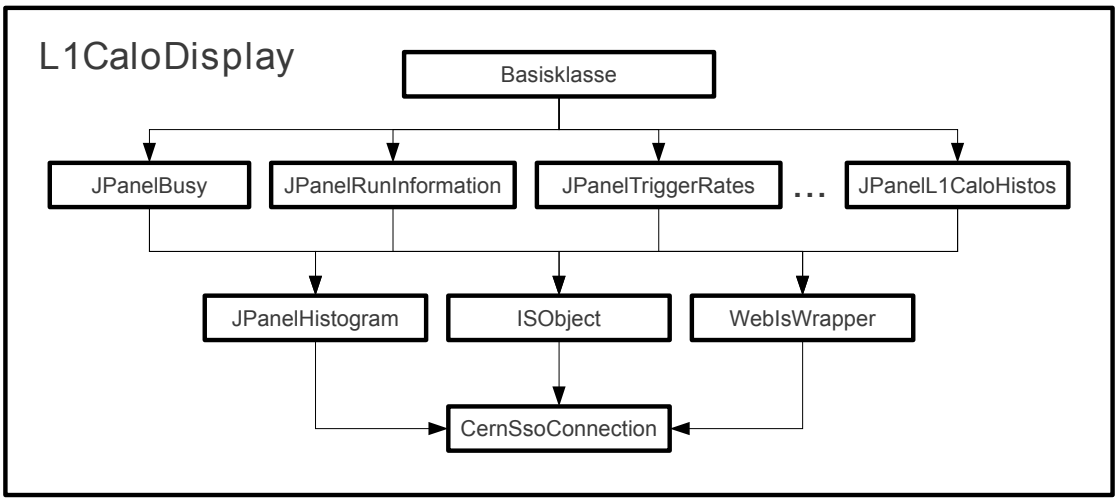


Abbildung 6.2: Vereinfachte Darstellung der wichtigsten Klassen des *L1CaloDisplay*. Die Pfeile symbolisieren, welche Klassen eine Klasse verwendet.

Eine vereinfachte Darstellung der wichtigsten Klassen des *L1CaloDisplay* findet sich in Abb. 6.2. Die Basisklasse stellt das Grundgerüst der Anwendung dar. Sie ist von der Java-Klasse *JFrame* abgeleitet und stellt der Anwendung eine graphische Oberfläche zur Verfügung. Die Klassen, die die zu überwachenden Informationen graphisch darstellen, sind von der Java-Klasse *JPanel* abgeleitet und werden im Folgenden als Überwachungsfelder bezeichnet. Ein *JPanel* kann innerhalb eines *JFrame* platziert werden und wird dadurch sichtbar. Die Überwachungsfelder stellen innerhalb der Anwendung eigenständige Module dar und sind in der Instanz der Basisklasse durch verschiedene Registerreiter zugänglich. Zum Beispiel gibt es Überwachungsfelder, die den *Busy*-Status des Detektors oder die aktuellen Triggerraten darstellen. Das Erweitern der Oberfläche um neue zu überwachende Informationen ist durch den modularen Aufbau sehr einfach und erfordert keine Veränderungen anderer Programmteile. Damit die

Überwachungsfelder Zugriff auf alle Programmeinstellungen haben, wird ihnen während des Programmstarts eine Referenz auf ein *Thread*-sicheres Objekt übergeben, das die Einstellungen beinhaltet. Die Überwachungsfelder stellen eine öffentliche Aktualisierungsmethode bereit, die von dem Aktualisierungsmanager der Basisklassen-Instanz in gewünschten Zeitintervallen ausgeführt wird.
Die Überwachungsfelder nutzen im Wesentlichen Objekte der drei folgenden Klassen, um die IS- oder OHS-Informationen zu verarbeiten und darzustellen:

- Die von *JPanel* abgeleitete Klasse **JPanelHistogram** ist ein Histogramm-Container. Wenn einem Objekt dieser Klasse ein Histogramm zugewiesen wird, wird das Histogramm in der Oberfläche dargestellt. Ein Rechtsklick auf das Histogramm öffnet ein *Pop-up*-Menü, durch das Methoden gestartet werden können, die das Histogramm einmalig oder in regelmäßigen Abständen aktualisieren sowie in einem gewünschten Format auf der lokalen Festplatte speichern. Unterstützte Formate sind unter anderem XML, PDF, PNG und das ROOT-Format. Ein Doppelklick auf ein Histogramm stellt dieses in einem separaten Fenster dar, das eine spezielle Oberfläche zum Wählen von Histogramm-Darstellungsparametern besitzt. Da die Klasse für alle dargestellten Histogramme genutzt wird, stehen für alle die genannten Methoden zur Verfügung.

- Die Klasse **ISObject** repräsentiert ein IS-Objekt und bietet komfortable Funktionen zur Abfrage von IS-Attributwerten und IS-Objekteigenschaften. Dem Konstruktor der Klasse wird die zugehörige *Web-IS Interface* URL des IS-Objekts übergeben. Die *ISObject*-Klasse besitzt eine öffentliche Methode, deren Aufruf die Aktualisierung der Attribute auslöst. Zum Parsen der XML-Antwort des *Web-IS Interface* wird eine XML-Parser-Klasse eingesetzt, die für das *Web-IS Interface* optimiert ist.

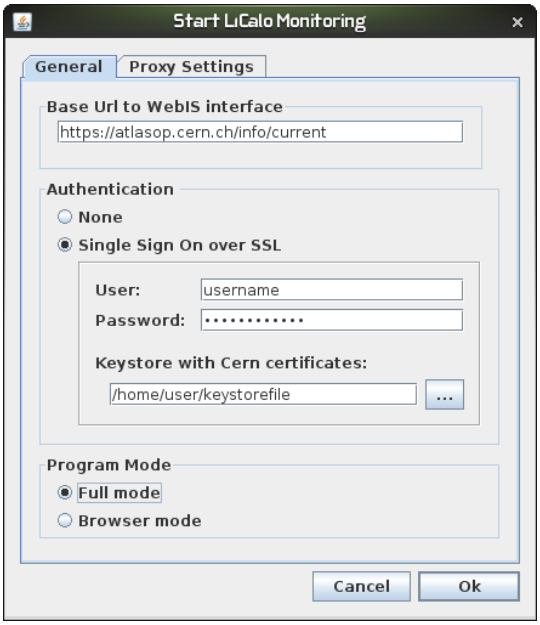
- Die Klasse **WebIsWrapper** bietet Methoden zur Abfrage der verfügbaren Informationen des IS. Zum Beispiel besitzt sie Methoden zur Abfrage einer Liste der Namen aller IS-Objekte oder Histogramme eines IS-Servers, der vorhandenen IS-Partitionen oder der IS-Server einer bestimmten IS-Partition. Zum Parsen der XML-Antwort wird die gleiche XML-Parser-Klasse eingesetzt wie in der *ISObject*-Klasse.

Alle drei genannten Klassen greifen über die Klasse *CernSsoConnection* auf das *Web-IS Interface* zu. Zur Codierung der IS- oder OHS-Anfragen in einer URL stehen spezielle Methoden zur Verfügung.

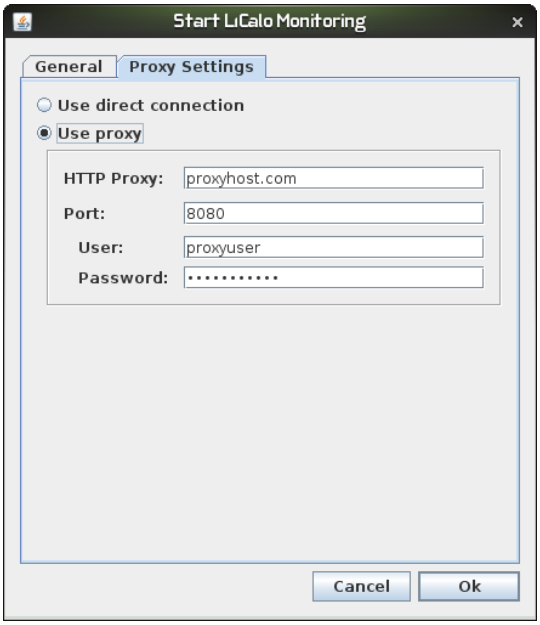
Der Start-Dialog

Wenn das *L1CaloDisplay* gestartet wird, öffnet sich der in Abb. 6.3 gezeigte Dialog. In diesem werden allgemeine Programmeinstellungen und *Proxy*-Einstellungen festgelegt:

- Allgemeine Programmeinstellungen:
  - *Base URL to Web-IS Interface*: Die URL des *Web-IS Interface*.



(a) *Allgemeine Einstellungen*



(b) *Proxy-Einstellungen*

Abbildung 6.3: Der Start-Dialog des *L1CaloDisplay*

  - *Authentication*: Hier wird angegeben, ob das *Web-IS Interface* eine Authentifizierung benötigt[5]. Falls die Option *Cern Single Sign-On* gewählt wird, muss der benötigte Benutzername mit zugehörigem Passwort angegeben werden. Zusätzlich wird der Pfad zur *Keystore*-Datei benötigt, die die zum Aufbau der SSL-Verbindung benötigten CERN-Zertifikate enthält.
  - *Program Mode*: Im *Full mode* werden alle überwachten Daten regelmäßig aktualisiert. Hat ein Anwender eine langsame Internet-Anbindung und möchte lediglich den IS oder OHS durchsuchen, sollte zur Minimierung des Netzwerkverkehrs der *Browser mode* gewählt werden. In diesem Modus werden keine automatischen Aktualisierungen durchgeführt.

- *Proxy*-Einstellungen:
  - Hier wird gewählt, ob direkt oder über einen *Proxy Server* auf das *Web-IS Interface* zugegriffen werden soll. Wenn ein *Proxy Server* keine Authentifizierung verlangt, können die entsprechenden Textfelder leer gelassen werden.

Ein Klick auf den *Ok*-Knopf startet die Überwachung, ein Klick auf den *Cancel*-Knopf bricht den Startvorgang ab. Damit die Einstellungen nicht bei jedem Start erneut eingegeben werden müssen, werden sie in Java-*Muffins* gespeichert. Das Passwort bildet eine Ausnahme und wird aus Sicherheitsgründen nicht gespeichert.

[5] Zu Testzwecken wurde im Laufe dieser Arbeit oft ein *Web-IS Interface* benutzt, das auf einen Entwicklungs-IS zugreift und daher keine Authentifizierung benötigt.

Die Statusleiste

Abbildung 6.4: Die Statusleiste zeigt den ATLAS *Root Controller* Zustand (links) und den Verbindungsstatus zum *Web-IS Interface* (rechts)

Nach dem Start der Anwendung ist die Statusleiste permanent sichtbar und zeigt den aktuellen ATLAS *Root Controller* Zustand und den Verbindungsstatus zum *Web-IS Interface* (siehe Abb. 6.4).
Der ATLAS *Root Controller* Zustand ist der Zustand der *Run Control* Komponente, die das DAQ-System startet, abschaltet und in Betrieb hält. Es existieren 17 Zustände, denen in TDAQ-Anwendungen jeweils eine Farbe zugeordnet wird. Im *L1CaloDisplay* wird dieses Farbschema bei der Darstellung des *Root Controller* Zustands übernommen. Im Regelbetrieb befindet sich der *Root Controller* im Zustand *Running*.
Der angezeigte Verbindungsstatus zum *Web-IS Interface* entspricht dem HTTP-Statuscode, den jeder Webserver bei einer HTTP-Anfrage zurückgibt. Der Statuscode enthält Informationen darüber, ob die Anfrage erfolgreich bearbeitet wurde. Beispiele für Statuscodes sind *OK*, *Bad Request*, *Not Found*, *Internal Server Error* oder *Forbidden*[6]. Wenn der HTTP-Statuscode den Wert *Ok* hat, wird er mit grüner Schriftfarbe dargestellt, anonsten mit roter. Wenn zum Beispiel versucht wird, ein nicht existierendes IS-Objekt abzufragen, liefert der Webserver den Statuscode *Not Found* zurück.
Wenn die *Single Sign-On* Authentifizierung misslingt, ist dies aus den empfangenen HTTP-Statuscodes nicht erkennbar, da das *Single Sign-On* Verfahren zwar auf HTTP-Verbindungen basiert, jedoch keine HTTP-Authentifizierung ist. Um den Anwender dennoch über eine misslungene Authentifizierung zu informieren, gibt die Authentifizierungs-Methode der *CernSsoConnection*-Klasse einen Pseudo-HTTP-Statuscode zurück: Wenn ein Fehler auftritt, ist dessen Wert *Forbidden*.

Das „General“-Überwachungsfeld

In Abb. 6.5 wird das *General*-Überwachungsfeld des *L1CaloDisplay* gezeigt. Dieses ist zu Beginn der Anwendung aktiviert und gibt einen Überblick über den aktuellen Status des Detektors. Es gliedert sich in die drei Bereiche *Run Information*, *Trigger Configuration* und *Trigger Rates*.

- Der Bereich *Run Information* enthält Informationen über den aktuellen *Run* des ATLAS-Experiments. Hierzu gehören unter anderem dessen Typ, Start- und Endzeit sowie Dauer in Sekunden. Ist ein *Run* aktiv, wird dessen Dauer als 0 angezeigt. Eine wichtige Kennung ist die *Run Number*, die jeden *Run* eindeutig identifiziert. Die Strahlenergie während des *Run* wird in GeV angegeben.
- Im Bereich *Trigger Configuration* findet man die aktuelle Konfiguration des Triggers. Diese ist in eindeutigen Schlüsseln codiert, welche mit einem zugehörigen

[6] Eine Liste aller HTTP-Statuscodes findet sich in [26].

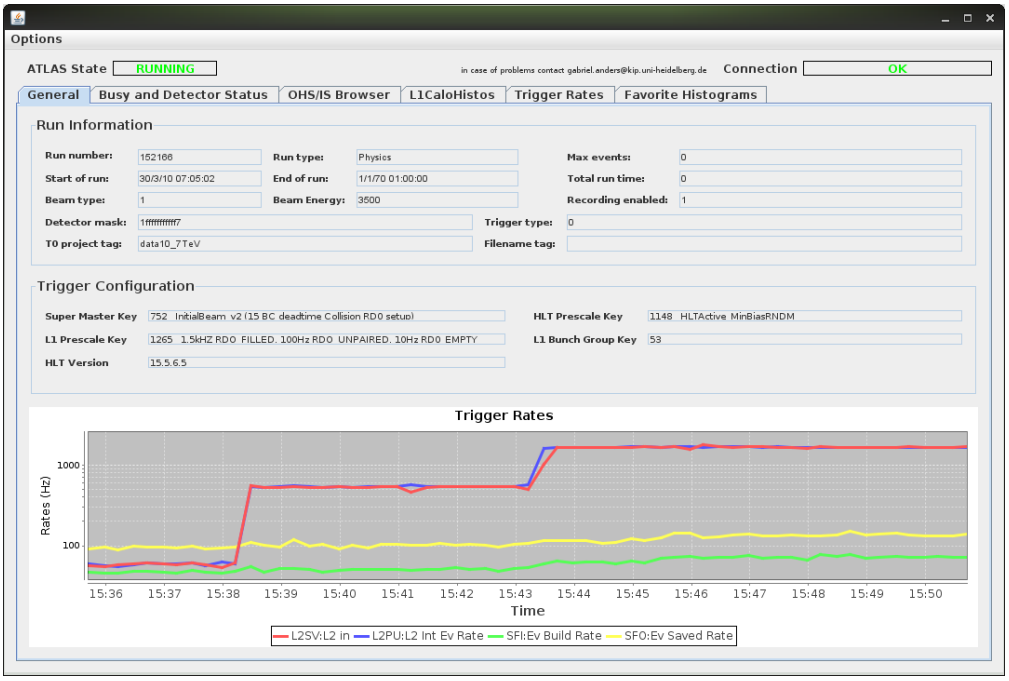


Abbildung 6.5: Das *General*-Überwachungsfeld gibt einen Überblick über den aktuellen Status des ATLAS-Detektors

Kommentar dargestellt werden. Der *L1 Prescale Key* und der *HLT Prescale Key* beschreiben die *Prescale*-Konfiguration des Level-1 und des *High Level Trigger*. Mit Hilfe des *Prescale Key* können Triggerraten nach unten skaliert werden, indem ein Teil der ursprünglich von dem Trigger akzeptierten Ereignisse nicht weitergereicht wird.

- Die *Trigger Rates* Graphik zeigt die Triggerraten der letzten 15 Minuten an verschiedenen Stellen der Triggerkette:
  - *L2SV:L2 in* entspricht der Rate der Ereignisse, die der *L2 supervisor* von dem ROI-Builder empfängt. Diese kann auch als die Rate des Level-1-Triggers bezeichnet werden.
  - *L2PU:L2 Int Ev Rate* entspricht der Rate der Ereignisse, die in den *L2 processing units* verarbeitet werden. Wenn diese Rate nicht mit der *L2SV:L2 in* Rate übereinstimmt, deutet dies auf einen Fehler hin.
  - *SFI:Ev Build Rate* entspricht der Rate der Ereignisse, die von dem *event builder*-Knoten verarbeitet werden. Diese kann auch als die Rate des Level-2-Triggers bezeichnet werden.
  - *SFO:Ev Saved Rate* entspricht der Rate der Ereignisse, die der *Event Filter* zur späteren Physik-Analyse speichert. Diese kann auch als die Rate des *Event Filter* bezeichnet werden.

Eine detaillierte Erklärung der Triggerkette findet sich in Kapitel 3.

Das „Busy and Detector Status“-Überwachungsfeld

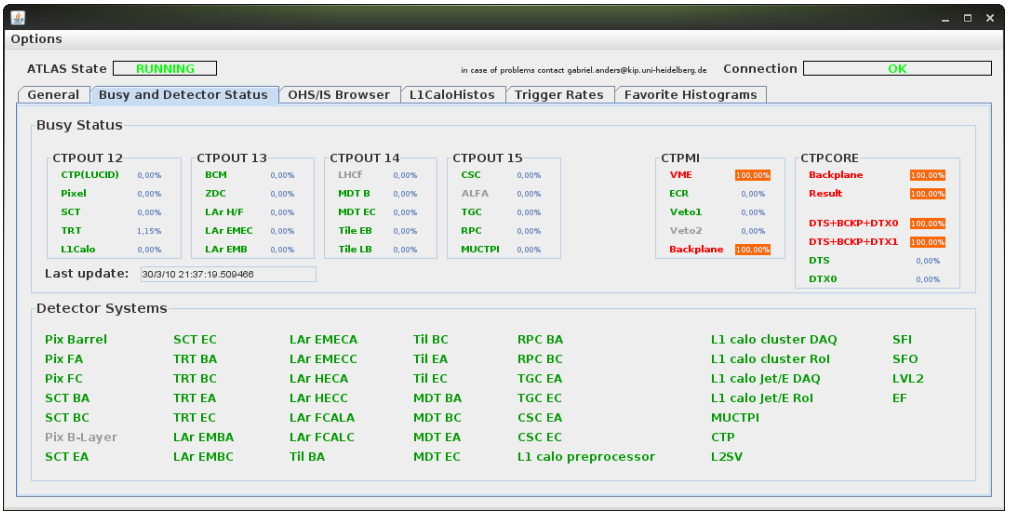


Abbildung 6.6: Das *Busy and Detector Status*-Überwachungsfeld zeigt die Überlastungs-Häufigkeit ausgewählter Detektorkomponenten und den Status der Detektorsysteme

In Abb. 6.6 ist das *Busy and Detector Status*-Überwachungsfeld des *L1CaloDisplay* zu sehen. Dieses zeigt die Überlastungs-Häufigkeit ausgewählter Detektorkomponenten und gibt Auskunft darüber, welche Detektorsysteme aktiv oder inaktiv sind.

- Im *Busy Status* Bereich wird dargestellt, wie häufig verschiedene Detektorkomponenten überlastet sind:
  - Die *CTPOUT*-Felder beschreiben, wie häufig die Auslesetreiber der Sub-Detektorsysteme überlastet sind. Wenn der CTP ein Ereignis akzeptiert, benachrichtigt er die Elektronik der Sub-Detektorsysteme, die die Ereignisdaten auslesen und an den Level-2-Trigger übergeben. Während der Datennahme können die Auslesetreiber überlastet werden. Mögliche Gründe sind zum Beispiel, dass die Triggerraten so hoch sind, dass die Ereignisse von dem Auslesetreiber nicht schnell genug ausgelesen oder die Daten von dem Level-2-Trigger nicht schnell genug in Empfang genommen werden können. Die Überwachung der Überlastungen ist wichtig, da der CTP nur Ereignisse akzeptiert, wenn kein Auslesetreiber überlastet ist. Falls die Raten zu hoch sind, können sie mit einem *Prescale*-Faktor versehen werden. In einem der Felder wird die Überlastungs-Häufigkeit des L1Calo-Systems gezeigt.
  - Das *CTPMI*-Feld beschreibt, wie häufig das CTP *machine interface* überlastet ist, das die LHC-Taktung empfängt.

  - Das *CTPCORE*-Feld beschreibt, wie häufig der CTP-Kern überlastet ist, der die Level-1-Entscheidung trifft und zugehörige Informationen an den Level-2-Trigger und das DAQ-System sendet.

  Während der Datennahme deaktivierte Systeme werden grau dargestellt, aktivierte Systeme grün. Die Häufigkeit der Überlastungen wird zur besseren Übersicht in Balken dargestellt; überschreitet sie 90 %, wird das System rot gefärbt.

- Im *Detector Status* Bereich werden die aktiven Detektorsysteme dargestellt, die in einem IS-Attribut namens *Detektor Mask* codiert sind. Der Wert des Attributs wird als *Bit*-Maske interpretiert, in der jedes *Bit* den Status eines Detektorsystems beschreibt. Ist ein *Bit* gesetzt, ist das zugehörige Detektorsystem aktiviert, ansonsten deaktiviert. Die Zuordnung der *Bits* zu Detektorsystemen verändert sich nicht. Da die Detektormaske im IS in Dezimaldarstellung vorliegt, wird sie von dem *L1CaloDisplay* zur Interpretation in Binärdarstellung umgewandelt. Ist ein Detektorsystem aktiviert, wird es grün dargestellt, ansonsten grau. Die Detektormaske enthält unter anderem den Status verschiedener L1Calo-Subsysteme.
  Im *General*-Überwachungsfeld wird die Detektormaske zusätzlich in Hexadezimaldarstellung gezeigt.

Das „OHS/IS Browser"-Überwachungsfeld

Im *OHS/IS Browser*-Überwachungsfeld befinden sich *Browser*, die alle Informationen des IS oder OHS zugänglich machen. In einer Auswahlbox wird aus einer Liste die IS-Partition ausgewählt, deren IS und OHS Informationen man im *IS* oder *OHS Browser* anschauen möchte. Die Liste der vorhandenen IS-Partitionen kann durch das Drücken des *Load Partitions*-Knopfs aktualisiert werden.

Der „OHS Browser"

In Abb. 6.7 ist der *OHS Browser* zu sehen. Zu Beginn zeigt dieser eine Liste aller verfügbaren *Histogram Provider*. Dies sind IS-Server, die Histogramme enthalten. Wenn ein *Histogram Provider* angeklickt wird, werden die Namen der zugehörigen Histogramme geladen. Das *L1CaloDisplay* sortiert die Histogramme zur Übersichtlichkeit in eine Ordnerstruktur ein, die in den Histogrammnamen codiert ist. Dieses Prinzip wird am Histogrammnamen

`/SHIFT/L1Calo/Overview/Errors/l1calo_2d_GlobalOverview`

erläutert: `l1calo_2d_GlobalOverview` ist der verkürzte Name des Histograms. Dieser wird dem Ordner `Errors` zugeordnet, der dem Ordner `Overview` zugeordnet ist. Dieser wiederum ist dem Ordner `L1Calo` zugeordnet, der dem Ordner `SHIFT` zugeordnet ist.
Wenn im *OHS Browser* ein Histogrammname angeklickt wird, wird das zugehörige Histogramm geladen und angezeigt. Um die Darstellungs-Parameter des Histogramms zu verändern, stellt die Oberfläche spezielle Steuerelemente bereit.

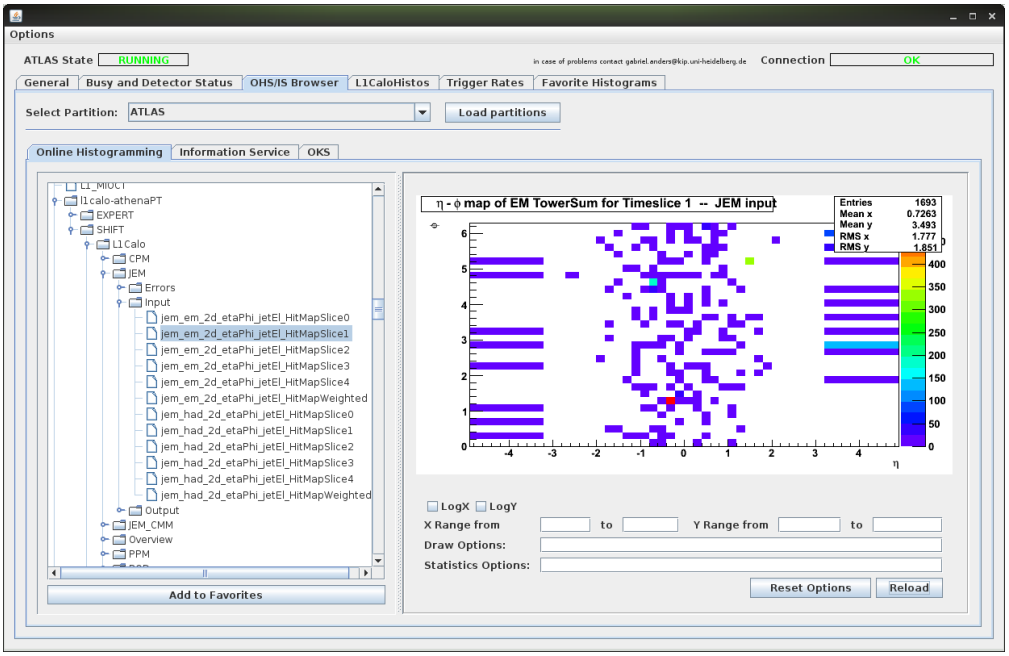


Abbildung 6.7: Der *OHS Browser* des *L1CaloDisplay*. In der linken Hälfte wird ein Histogramm eines *Histogram Provider* ausgewählt, das daraufhin in der rechten Hälfte dargestellt wird.

Histogramme können einer Favoriten-Liste hinzugefügt werden, die im *Favorite Histograms*-Überwachungsfeld verwaltet wird. Das dargestellte Histogramm wird der Favoriten-Liste durch das Drücken des *Add to Favorites*-Knopfs hinzugefügt.

Der „IS Browser“

In Abb. 6.8 ist der *IS Browser* zu sehen. Zu Beginn zeigt dieser eine Liste aller verfügbaren IS-Server. Wenn ein IS-Server angeklickt wird, wird eine Liste der zugehörigen IS-Objekte geladen und in einer Tabelle dargestellt. Die Liste enthält den Namen der IS-Objekte, deren Typ sowie den Zeitpunkt der letzten Modifikation. Ein Klick auf ein IS-Objekt lädt eine Liste der zugehörigen IS-Attribute. Die Liste der IS-Attribute wird ebenfalls in einer Tabelle dargestellt. Sie enthält die Namen und Werte der IS-Attribute sowie deren Typ und hinterlegte Beschreibung.
Alle Spalten der Tabellen des *IS Browser* werden durch einen Klick auf deren Überschrift sortiert. Diese Funktion ist hilfreich, wenn die Tabellen groß sind und bestimmte IS-Objekte oder Attribute gesucht werden.

Das „L1Calo Histograms“-Überwachungsfeld

Es wurde eine Auswahl von Histogrammen getroffen, die zeigen, ob das L1Calo-System korrekt arbeitet. Die Histogramme werden verschiedenen Bereichen zugeordnet (siehe

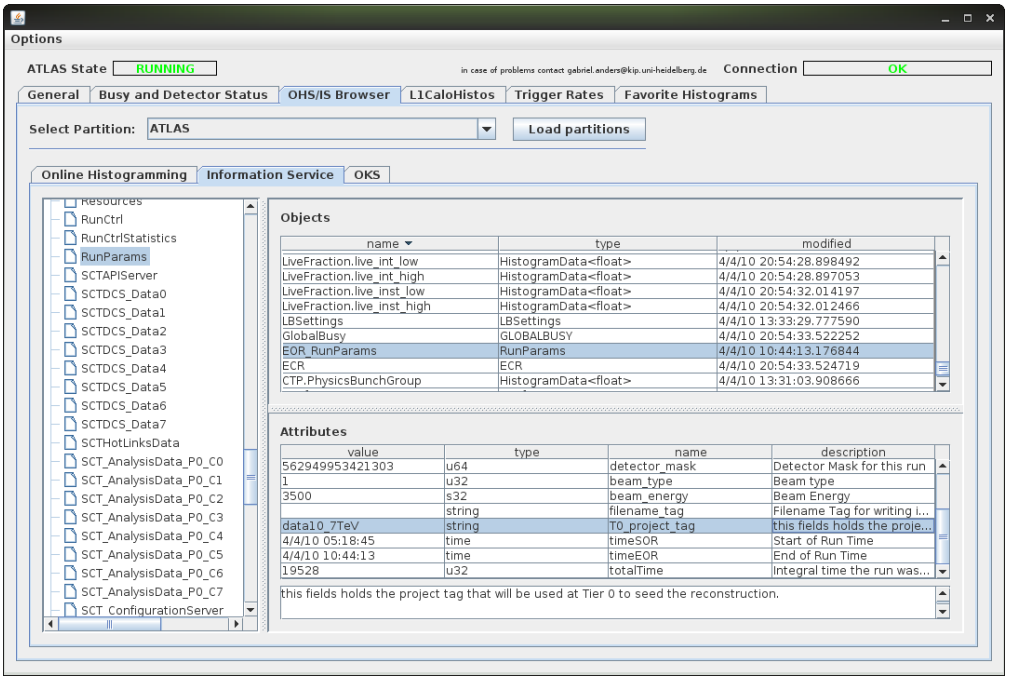


Abbildung 6.8: Der *IS Browser* des *L1CaloDisplay*. In der linken Hälfte wird ein IS-Server ausgewählt, dessen IS-Objekte daraufhin in der oberen rechten Tabelle dargestellt werden. Wenn ein IS-Objekt ausgewählt wird, werden die zugehörigen IS-Attribute in der unteren rechten Tabelle dargestellt.

Abb. 6.9):

- Fehler-Zusammenfassungen:
  Diese Histogramme stellen übersichtlich Elektronikdefekte oder Datenübertragungsprobleme dar, die während des Betriebs des L1Calo-Systems aufgetreten sind. Beispielsweise werden immer die *Trigger-Tower*-Energien der akzeptierten Ereignisse sowohl im PPr als auch im CPM ausgelesen und verglichen; sind diese nicht identisch, wird ein Fehler angezeigt. Ein analoger Vergleich wird zwischen den *jet element*-Energien im PPr und im JEM durchgeführt.

- Schwellen-Multiplizitäten:
  In diesen Histogrammen ist zu erkennen, wie häufig eine Schwelle überschritten wurde. Beispiele hierfür sind Multiplizitäten der E_T^{miss}- oder Gesamt-E_T-Schwellen, welche auf den CMMs bestimmt werden, oder Multiplizitäten der E_T-Schwellen für Jets, welche auf den JEMs bestimmt werden. Abweichungen der Multiplizitäten vom Normalverhalten sind ein Hinweis auf Fehler.

- η-ϕ-Trefferkarten:
  Diese Histogramme sind zweidimensionale Karten des kompletten Raumwinkels und enthalten die Anzahl von Treffern pro $\Delta\eta \times \Delta\phi$, die vorgegebene Schwellen

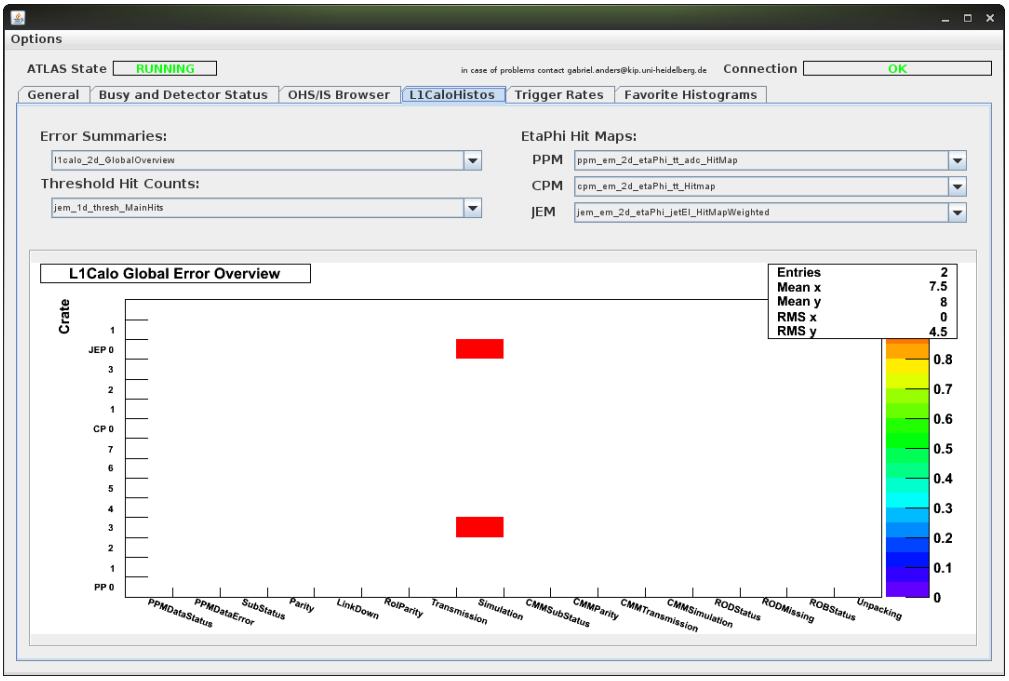


Abbildung 6.9: Im *L1Calo Histograms*-Überwachungsfeld können ausgesuchte L1Calo-Histogramme betrachtet werden.

überschritten haben. Wenn das L1Calo-System korrekt arbeitet, sollten die Treffer gleichmäßig in ϕ und symmetrisch zu $\eta = 0$ in η verteilt sein.

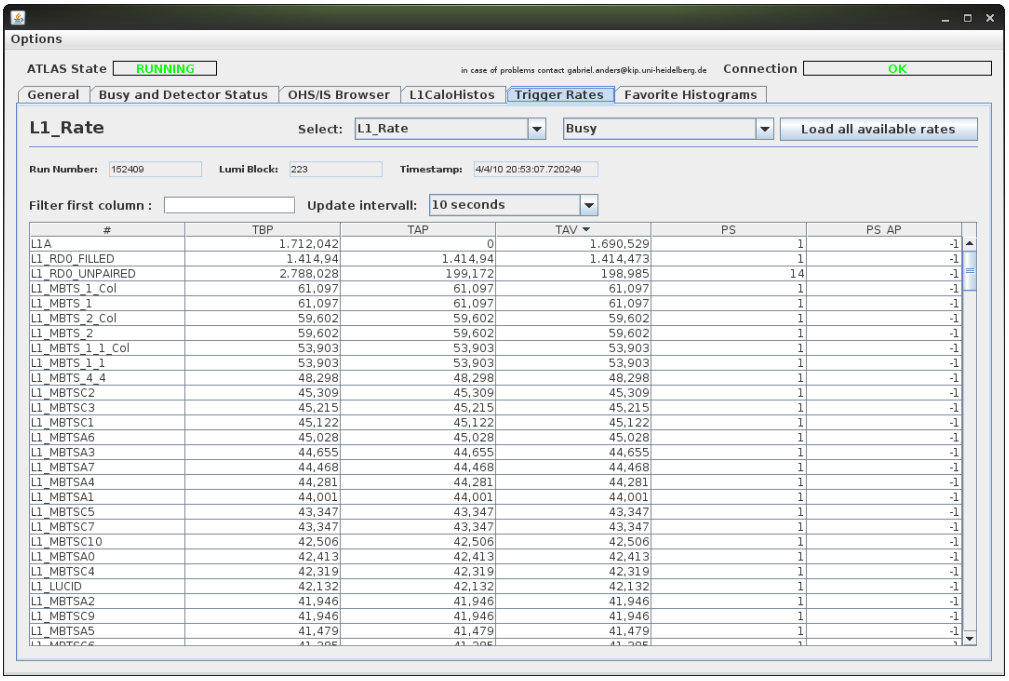
- Alle PPM-Trefferkarten basieren auf *Trigger-Tower*-Treffern im EM- oder HAD-Kalorimeter. Die Schwellen sind LUT-Energien oder ADC-Werte, die bei der Digitalisierung der analogen Kalorimetersignale erzeugt werden. In den PPM-Trefferkarten sind defekte *Trigger-Tower* oftmals aufgrund der zu hohen oder niedrigen Anzahl von Treffern gut zu erkennen.
- Die CPM-Trefferkarten basieren auf *Trigger-Tower*- oder *Cluster ROI*-Treffern.
- Die JEM-Trefferkarten basieren auf *jet element*- oder Jet *ROI*-Treffern.

Eine detaillierte Erklärung der genannten Begriffe findet sich in Kapitel 3.4.

Das „Trigger Rates“-Überwachungsfeld

Im *Trigger Rates*-Überwachungsfeld werden Triggerraten in Tabellenform dargestellt (siehe Abb. 6.10). Zu Beginn stehen die Raten des Level-1-Triggers, des Level-2-Triggers und des *Event Filter* zur Auswahl. Die Spalten der Tabellen können sortiert[7] oder durch die Eingabe eines regulären Ausdrucks gefiltert werden. In einer Auswahlbox kann eingestellt werden, in welchen Intervallen die dargestellten Raten aktualisiert werden sollen,

[7] Die Namen der Raten werden lexikographisch sortiert, die Raten nach ihrem Wert.



#	TBP	TAP	TAV	PS	PS_AP
L1A	1.712,042	0	1.690,529	1	-1
L1_RD0_FILLED	1.414,94	1.414,94	1.414,473	1	-1
L1_RD0_UNPAIRED	2.788,028	199,172	198,985	14	-1
L1_MBTS_1_Col	61,097	61,097	61,097	1	-1
L1_MBTS_1	61,097	61,097	61,097	1	-1
L1_MBTS_2_Col	59,602	59,602	59,602	1	-1
L1_MBTS_2	59,602	59,602	59,602	1	-1
L1_MBTS_1_1_Col	53,903	53,903	53,903	1	-1
L1_MBTS_1_1	53,903	53,903	53,903	1	-1
L1_MBTS_4_4	48,298	48,298	48,298	1	-1
L1_MBTSC2	45,309	45,309	45,309	1	-1
L1_MBTSC3	45,215	45,215	45,215	1	-1
L1_MBTSC1	45,122	45,122	45,122	1	-1
L1_MBTSA6	45,028	45,028	45,028	1	-1
L1_MBTSA3	44,655	44,655	44,655	1	-1
L1_MBTSA7	44,468	44,468	44,468	1	-1
L1_MBTSA4	44,281	44,281	44,281	1	-1
L1_MBTSA1	44,001	44,001	44,001	1	-1
L1_MBTSC5	43,347	43,347	43,347	1	-1
L1_MBTSC7	43,347	43,347	43,347	1	-1
L1_MBTSC10	42,506	42,506	42,506	1	-1
L1_MBTSA0	42,413	42,413	42,413	1	-1
L1_MBTSC4	42,319	42,319	42,319	1	-1
L1_LUCID	42,132	42,132	42,132	1	-1
L1_MBTSA2	41,946	41,946	41,946	1	-1
L1_MBTSC9	41,946	41,946	41,946	1	-1
L1_MBTSA5	41,479	41,479	41,479	1	-1

Abbildung 6.10: Im *Trigger Rates*-Überwachungsfeld können die aktuellen Raten des Level-1-Triggers, des Level-2-Triggers und des *Event Filter* betrachtet werden

die gewählte Sortierung und der Filter bleiben dabei erhalten.
Für die Überwachung des L1Calo-Systems sind die Level-1-Raten eine hilfreiche Information, da die Level-1-Trigger-Entscheidung des CTP zum Teil auf L1Calo-Ergebnissen basiert. Die dargestellten Level-1-Raten zeigen Triggerraten für alle Einträge des *trigger menu*. Für jeden Eintrag werden drei Triggerraten und ein *Prescale*-Faktor gezeigt. Die erste Rate (TBP) ist die Triggerrate vor dem Anwenden des *Prescale*-Faktors, die zweite (TAP) ist die Triggerrate nach Anwenden des *Prescale*-Faktors. In der dritten Rate (TAV) wird berücksichtigt, dass nach einem akzeptierten Ereignis innerhalb einiger *Bunch Crossings* kein zweites Ereignis akzeptiert wird.
Die Triggerraten liegen in einer Tabellenstruktur in IS-Objekten des Typs *TimePoint_IS* vor, der die folgenden Attribute besitzt:

- *RunNumber* ist die Nummer des aktuellen *Run*.
- *TimeStamp* ist der Zeitpunkt der letzten Aktualisierung der Raten.
- *LumiBlock* ist die Nummer des aktuellen Luminositäts-Blocks.
- *XLabels* ist eine Liste, die die Zeilenüberschriften der Tabelle beinhaltet.
- *YLabels* ist eine Liste, die die Spaltenüberschriften der Tabelle beinhaltet.
- *Data* enthält eine Liste der aktuellen Raten.

Um aus den Attributen eine Tabelle der Raten zu erzeugen, wird zuerst eine leere Tabelle mit den angegebenen Zeilen- und Spaltenüberschriften erzeugt (*XLabels* und *YLabels*). Diese wird danach von links nach rechts und Zeile für Zeile mit den Werten gefüllt, die sich in der Liste des IS-Attributs *Data* befinden.
Das *Trigger Rates*-Überwachungsfeld kann alle Raten und Informationen anzeigen, die in IS-Objekten des Typs *TimePoint_IS* enthalten sind. Das Drücken des *Load all available rates*-Knopfs lädt eine Liste aller im IS vorhandenen Objekte dieses Typs in eine Auswahlbox.

Das „Favorite Histograms“-Überwachungsfeld

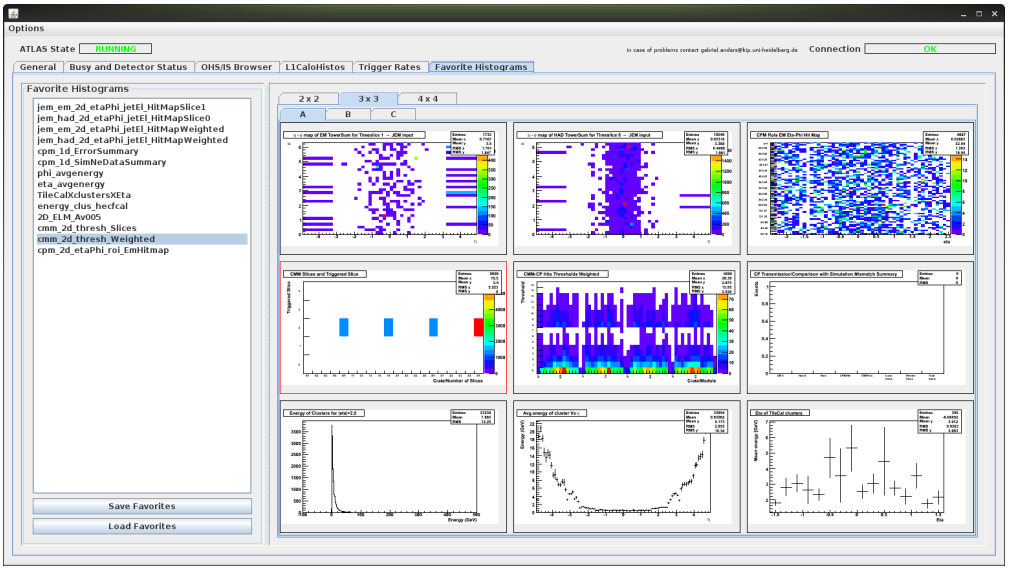


Abbildung 6.11: Im *Favorite Histograms*-Überwachungsfeld können Histogramme der Favoriten-Liste betrachtet werden. Der ausgewählte Histogramm-Container ist rot umrandet.

Im *Favorite Histograms*-Überwachungsfeld werden die Histogramme der Favoriten-Liste aufgelistet und betrachtet. Histogramme werden der Liste im *OHS Browser* hinzugefügt. Markierte Histogramme können von der Liste durch das Drücken der *DEL*-Taste entfernt werden. Die Histogramme können in verschiedenen Registerreitern betrachtet werden, die Histogramm-Container enthalten, die in einer 2x2-, 3x3- oder 4x4-Struktur angeordnet sind. Ein Doppelklick auf ein Histogramm der Favoriten-Liste stellt dieses in einem Histogramm-Container dar, der zuvor durch Anklicken ausgewählt wurde.
Das *L1CaloDisplay* kann Favoriten-Listen in einer Datei speichern oder von einer Datei laden. Sie gehen dadurch nach Beendigung des *L1CaloDisplay* nicht verloren und der Anwender muss häufig angeschaute Histogramme nicht bei jedem Start des *L1CaloDisplay* erneut im *OHS Browser* suchen. Die Favoriten-Listen können zudem an andere Anwender weiteregeben werden.

Kapitel 7

Zusammenfassung

Zur Fernüberwachung des Level-1 Kalorimeter Triggers wurde in der vorliegenden Arbeit die Java-Anwendung *L1CaloDisplay* entwickelt, mit der über das Internet der Status des L1Calo-Systems beobachtet und Fehleranalysen durchgeführt werden könnnen.
Die für die *Online*-Überwachung des ATLAS-Experiments relevanten Daten befinden sich innerhalb des streng abgeschirmten ATLAS-Netzerks im sogenannten *Information Service*. Da die existierenden Fernüberwachungssysteme entweder durch ihre Funktionalität oder die Anzahl von erlaubten Sitzungen limitiert waren, wurde ein neues, zweikomponentiges System konzipiert. Die erste Komponente ist eine Anwendung, die innerhalb des ATLAS-Netzwerks läuft und Anfragen von außen entgegennimmt, diese bearbeitet und die angeforderten Daten zurückgibt. Die zweite Komponente ist eine überall lauffähige graphische Benutzeroberfläche, mit deren Hilfe Überwachungsdaten angefordert werden. Es wurde ein Prototyp erstellt, der durch das Aufrufen einer URL gestartet, in einem *Query String* codierte Anfragen bearbeitet und die angeforderten Daten in XML-Darstellung zurückgibt. Dessen Funktionsfähigkeit wurde mit einer *Client*-Anwendung demonstriert. Als bekannt wurde, dass zeitgleich am CERN ein ähnliches Fernüberwachungssystem namens *Web-IS Interface* entstand, wurde dieses als Basis des *L1CaloDisplay* gewählt und mit dessen Entwickler zusammengearbeitet.
Das *L1CaloDisplay* ist plattformunabhängig und kann aus einem *Webbrowser* heraus gestartet werden. Alle zeit- oder rechenintensiven Programmbestandteile wurden in separate *Threads* ausgelagert. Zur Kommunikation mit dem *Web-IS Interface* besitzt das *L1CaloDisplay* einen auf das *Web-IS Interface* optimierten XML-Parser und Klassen, die die vollständige Funktionalität des *Web-IS Interface* zur Verfügung stellen. Zur Authentifizierung am *Web-IS Interface* wurde eine *Multithreading*-fähige Java-Klasse entwickelt, die über eine mit SSL verschlüsselte Verbindung einen transparenten Zugriff auf mit *CERN Single Sign-On* geschützte Webanwendungen oder Webseiten erlaubt und leicht in anderen Java-Anwendungen verwendet werden kann.
Es wurde eine Auswahl von Informationen getroffen, die im *L1CaloDisplay* graphisch aufbereitet dargestellt werden und zeigen, ob das L1Calo-System korrekt arbeitet. Jeder Anwender kann das *L1CaloDisplay* durch das Erstellen eigener Favoriten-Listen von Histogrammen an die eigenen Anforderungen anpassen.
Das *L1CaloDisplay* ist nicht nur für L1Calo-Experten geeignet, da es zurzeit die einzige Anwendung mit graphischer Benutzeroberfläche ist, mit der über das Internet alle im *Information Service* vorliegenden Daten und Histogramme dargestellt und durchsucht werden können und die für alle ATLAS-Physiker zur Verfügung steht. Aufgrund der hohen Modularität kann das *L1CaloDisplay* zukünftig auch zur Fernüberwachung anderer Detektorsysteme eingesetzt werden.

Anhang A

Fernüberwachungssystem

A.1 Basisdatei des Prototyps

Listing A.1: Basisdatei des Prototyps

```
1  // Dies ist die Basisdatei des Prototyps. In dieser werden
2  // der Query String decodiert und je nach Typ der IS-Anfrage
3  // die IS-Daten in entsprechender XML-Darstellung zurueckgegeben.
4  // Autor: Gabriel Anders
5
6  #include "CGImap.h"
7  #include "xml_utils.h"
8  #include <iostream>
9
10 using namespace std;
11
12 int main(int argc, char** argv)
13 {
14    //Diese Zeile teilt dem Client den Content-Type
15    //der Antwort mit, in diesem Fall XML.
16    cout <<  "Content-Type: text/xml" << endl;
17    cout << endl;
18    if (getenv("QUERY_STRING") != NULL)
19       {
20       //Parse Query String
21       CGImap query(getenv("QUERY_STRING"));
22
23       //Initialisiere Kommunikation ueber IS API
24       IPCCore::init( argc, argv );
25
26       std::string partition_name, server_name,
27             object_name, attribute_name;
28
29       partition_name = query["p"];
30       server_name = query["s"];
31       object_name = query["o"];
32       attribute_name = query["a"];
33
34       //Erzeuge ein Objekt der Klasse IS_to_Xml.
35       //Diese Klasse holt die angefragten Daten aus dem IS und erzeugt
36       //den Xml-Code, welcher mit verschiedenen "Print-Optionen"
37       //genauer spezifiziert werden kann
38       IS_to_Xml Data1;
39       Data1.set_attr_print_options(true, true, true, true);
40       Data1.set_obj_print_options(true, true, true, true, false);
```

```
41        //Erzeuge je nach Typ der angefragten IS-Daten
42        //verschieden codierte XML-Antworten
43        if (partition_name.empty())
44        {
45           //Erzeuge XML-Code der vorhandenen IS-Partitionen
46           Data1.print_partitions();
47        }
48        else
49        {
50           if (server_name.empty())
51           {
52              //Erzeuge XML-Code der vorhandenen IS-Server
53              Data1.print_server(partition_name);
54           }
55           else
56           {
57              if (object_name.empty())
58              {
59                 object_name = ".*";
60                 Data1.add_objects(
61                    partition_name, server_name,
62                    object_name, attribute_name);
63                 //Erzeuge XML-Code der vorhandenen IS-Objekte
64                 //exklusive zugehoerigen IS-Attributen
65                 Data1.print_objects("", false, false);
66              }
67              else
68              {
69                 Data1.add_objects(
70                    partition_name, server_name,
71                    object_name, attribute_name);
72                 //Erzeuge XML-Code der angefragten IS-Objekte
73                 //inklusive zugehoerigen IS-Attributen
74                 Data1.print_objects("", true, false);
75              }
76           }
77        }
78     }
79   return 0;
80 }
```

A.2 SAX-Parser des „Remote IS Monitor“

Listing A.2: Sax-Parser des *Remote IS Monitor*

```
1 /**
2  * Eine Klasse, welche die XML-Antwort des Prototyps
3  * parst und zur Weiterverarbeitung in Vektoren speichert.
4  * @author Gabriel Anders
5  */
6 public class XmlToVectorHandler extends org.xml.sax.helpers.DefaultHandler
7 {
```

```
8   private java.util.Vector m_xmlData = null;
9
10  /**
11   * Konstruktor
12   * @param xmlData Ein Vektor in welchem die geparsten
13   *   IS-Daten gespeichert werden
14   */
15  public XmlToVectorHandler(java.util.Vector xmlData){
16    m_xmlData = xmlData;
17  }
18
19    /**
20     * Diese Methode wird aufgerufen, wenn das Ereignis
21     * "Start Element" eintritt. Also immer wenn der Parser
22     * auf ein neues XML-Tag im XML-Code trifft.
23     * @param namespaceURI URI des Namensraumes
24     * @param localName Elementname ohne evt. Praefix
25     * @param qName Elementname mit evt. Praefix
26     * @param attrs Attribute des Elementes
27     */
28    @Override
29  public void startElement( String namespaceURI, String localName,
30                            String qName, org.xml.sax.Attributes attrs )
31  throws org.xml.sax.SAXException
32  {
33    java.util.Vector row = new java.util.Vector();
34    String eName = ( "".equals( localName ) ) ? qName : localName;
35    //Parse die IS-Partitionen
36    if (eName.equals("part")){
37        if( attrs != null )
38        {
39          for( int i=0; i<attrs.getLength(); i++ )
40          {
41            String aName = attrs.getLocalName( i ); //Attributname
42            if( "".equals( aName ) ) aName = attrs.getQName( i );
43            String aVal = attrs.getValue( i );
44            if (aName.equals("name")) row.add(aVal);
45          }
46        }
47        m_xmlData.add(row);
48    }
49    //Parse die IS-Server
50    if (eName.equals("serv")){
51        if( attrs != null )
52        {
53          for( int i=0; i<attrs.getLength(); i++ )
54          {
55            String aName = attrs.getLocalName( i ); //Attributname
56            if( "".equals( aName ) ) aName = attrs.getQName( i );
57            String aVal = attrs.getValue( i );
58            if (aName.equals("name")) row.add(aVal);
59            if (aName.equals("started")) row.add(aVal);
60            if (aName.equals("host")) row.add(aVal);
61            if (aName.equals("owner")) row.add(aVal);
```

```
62              if (aName.equals("pid")) row.add(aVal);
63            }
64          }
65          m_xmlData.add(row);
66      }
67      //Parse die IS-Objekte
68      if (eName.equals("object")){
69          if( attrs != null )
70          {
71            for( int i=0; i<attrs.getLength(); i++ )
72            {
73              String aName = attrs.getLocalName( i ); //Attributname
74              if( "".equals( aName ) )  aName = attrs.getQName( i );
75              String aVal = attrs.getValue( i );
76              if (aName.equals("name")) row.add(aVal);
77              if (aName.equals("type")) row.add(aVal);
78              if (aName.equals("modified")) row.add(aVal);
79              if (aName.equals("descr")) row.add(aVal);
80            }
81          }
82          m_xmlData.add(row);
83      }
84      //Parse die IS-Attribute
85      if (eName.equals("attr")){
86          if( attrs != null )
87          {
88            for( int i=0; i<attrs.getLength(); i++ )
89            {
90              String aName = attrs.getLocalName( i ); //Attributname
91              if( "".equals( aName ) )  aName = attrs.getQName( i );
92              String aVal = attrs.getValue( i );
93              if (aName.equals("val")) row.add(aVal);
94              if (aName.equals("type")) row.add(aVal);
95              if (aName.equals("name")) row.add(aVal);
96              if (aName.equals("descr")) row.add(aVal);
97            }
98          }
99          m_xmlData.add(row);z
100     }
101   }
102 }
```

Anhang B

„L1CaloDisplay“

B.1 Eingesetzte Software bei der Entwicklung des „L1CaloDisplay“

Eine Auflistung der eingesetzten Software bei der Entwicklung des *L1CaloDisplay* und der benutzten Bibliotheken findet sich in Tabelle B.1. Als Entwicklungsumgebung wurde *Netbeans IDE* in Verbindung mit der Java-Plattform *OpenJDK 1.6* gewählt.
Für die automatische *Cookie*-Verwaltung werden Bibliotheken der *Apache Software Foundation* verwendet [25]. Zur graphischen Darstellung der Triggerraten werden *JFreeChart*-Bibliotheken [27] eingesetzt.
Die eingesetzte Software und alle benutzten Bibliotheken stehen unter einer *Open Source* Lizenz.

Eingesetzte Software/Bibliotheken	**Version**
NetBeans IDE	6.5 (Build 091101)
OpenJDK	1.6.0_0
Apache Commons Bibliotheken	
commons-codec	1.4
commons-logging	1.1.1
commons-httpclient3	3.1
JFreeChart Bibliotheken	
jfreechart	1.0.13
jcommon	1.0.16

Tabelle B.1: Software und Bibliotheken bei der Entwicklung des *L1CaloDisplay*

B.2 Erstellen des Zertifikate-„Keystore“

Das *L1CaloDisplay* benötigt eine *Keystore*-Datei mit gültigen Zertifikaten, um SSL-Verbindungen mit dem *Web-IS Interface* und dem *CERN Login Server* aufbauen zu können. Für die Erstellung der *Keystore*-Datei werden zunächst die Zertifikate des Webservers, auf dem sich das *Web-IS Interface* befindet, und des *CERN Login Server* beschafft. Hierzu wird jeweils eine beliebige Webseite, die sich auf den Servern befindet,

in einem Webbrowser geöffnet, der Funktionen zum Speichern des Zertifikats der aktuell betrachteten Webseite bereitstellt.
Aus den abgespeicherten Zertifikaten wird mit Hile des Kommandozeilen-Programms keytool, das der Java-Installation beiliegt, die *Keystore*-Datei erstellt. Zunächst wird das Zertifikat des Webservers hinzugefügt, auf welchem sich das *Web-IS Interface* befindet (`https://atlasop.cern.ch`):

```
keytool -importcert -file atlasop.cern.ch -alias atlasop.cern.ch
    -keystore jssecacerts
```

Anschließend wird das Zertifikat des *CERN Login Server* hinzugefügt (`https://login.cern.ch`):

```
keytool -importcert -file login.cern.ch -alias login.cern.ch
    -keystore jssecacerts
```

Die Bedeutung der Parameter ist:

- importcert weist das keytool an, ein Zertifikat in einen *Keystore* zu importieren.
- file ist der Name der Datei, die das Zertifikat enthält.
- alias ist eine frei wählbare Bezeichnung, die das Zertifikat im *Keystore* eindeutig identifiziert.
- keystore ist der Name der *Keystore*-Datei, in welche das Zertifikat importiert werden soll. Wenn die *Keystore*-Datei noch nicht existiert, wird sie von dem keytool erzeugt, ansonsten wird sie um das angegebene Zertifikat erweitert.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Abachi, S. et al.: *Observation of the Top Quark.* Phys. Rev. Lett., 74(14):2632–2637, Apr 1995.

[2] Team, AC: *The four main LHC experiments.* Jun 1999.

[3] Bruening, Oliver Simon et al.: *LHC Design Report.* CERN, Geneva, 2004.

[4] Evans, Lyndon R und Philip Bryant: *LHC Machine.* J. Instrum., 3:S08001. 164 p, 2008.

[5] Aad, G et al.: *The ATLAS Experiment at the CERN Large Hadron Collider.* J. Instrum., 3:S08003, 2008.

[6] *The Expected Performance of the ATLAS Inner Detector.* Technischer Bericht ATL-PHYS-PUB-2009-002. ATL-COM-PHYS-2008-105, CERN, Geneva, Aug 2008.

[7] Povh, Bogdan, Klaus Rith, Christoph Scholz und Frank Zetsche: *Teilchen und Kerne.* Springer Berlin Heidelberg, 8 Auflage, 2009.

[8] Higgs, Peter Ware: *Broken symmetries and the masses of gauge bosons.* Phys. Rev. Lett., 13:508–509, 1964.

[9] Aad, G et al.: *Expected performance of the ATLAS experiment: detector, trigger and physics.* Technischer Bericht arXiv:0901.0512. CERN-OPEN-2008-020, Geneva, 2009.

[10] Achenbach, R et al.: *The ATLAS Level-1 Calorimeter Trigger.* Technischer Bericht ATL-DAQ-PUB-2008-001. ATL-COM-DAQ-2008-002, CERN, Geneva, Jan 2008.

[11] Weber, Pavel: *ATLAS Calorimetry: Trigger, Simulation and Jet Calibration.* Doktorarbeit, Kirchhoff Institute for Physics, University of Heidelberg, 2008.

[12] Pfeiffer, Ullrich und Wolfgang Hötzel: *Bunch-Crossing Identification for saturated calorimeter signals.* Technischer Bericht ATL-DAQ-99-009, CERN, Geneva, May 1999.

[13] Eisenhandler, E F: *ATLAS Level-1 Calorimeter Trigger Algorithms.* Technischer Bericht ATL-DAQ-2004-011. CERN-ATL-DAQ-2004-011, CERN, Geneva, Sep 2004.

[14] Staley, Richard, Paul Bright-Thomas, Alan Watson und Norman Gee: *ATLAS Level-1 Calorimeter Trigger Cluster Processor Module (Version: 2.03).* Technischer Bericht, University of Birmingham.

[15] NoMachine: *NX Remote Desktop Software*. http://www.nomachine.com/. am 24. März 2010.

[16] Hauser, Reiner: *Web IS Measurements @ Point 1*. http://indico.cern.ch/conferenceDisplay.py?confId=70802. Präsentation in der Monitoring Working Group am 25. März 2009.

[17] Hauser, Reiner: *Web Access to Information Service*. http://rhauser.web.cern.ch/rhauser/doc/interface.html. am 26. März 2010.

[18] World Wide Web Consortium: *CGI Spezifikation*. http://www.w3.org/CGI/. am 23. März 2010.

[19] Kolos, Serguei: *Information Service ATLAS TDAQ - User's Guide*, November 2005.

[20] Eckel, Bruce: *Thinking in C++, Volume I: Introduction to Standard C++, Second Edition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[21] McLaughlin, Brett und Justin Edelson: *Java & XML*. O'Reilly Media, Inc., 2006.

[22] Oracle Corporation: *Java Web Start Technology*. http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp. am 23. März 2010.

[23] Ullenboom, Christian: *Java ist auch eine Insel*. Galileo Computing, 8., aktualisierte Auflage, 2009.

[24] Internet2 Arbeitsgemeinschaft: *Shibboleth Website*. http://shibboleth.internet2.edu/. am 12.04.2010.

[25] Apache Software Foundation: *Apache Commons Website*. http://commons.apache.org/. am 01.04.2010.

[26] World Wide Web Consortium: *HTTP Spezifikation*. http://www.w3.org/Protocols/rfc2616/rfc2616.html. am 07.04.2010.

[27] Object Refinery Limited: *JFreeChart Webseite*. http://www.jfree.org/jfreechart/. am 11.04.2010.

Danksagung

Ich möchte allen Personen danken, die zur Entstehung dieser Diplomarbeit beigetragen haben. Besonderer Dank geht an:

- Prof. Dr. Hans-Christian Schultz-Coulon dafür, mir diese Diplomarbeit angeboten zu haben. Zusätzlich möchte ich mich für die hervorragenden Forschungsbedingungen, den Enthusiasmus bei der Umsetzung des *L1CaloDisplay* sowie die ausgezeichnete Betreuung bedanken.

- Prof. Dr. Udo Kebschull für sehr interessante technische Gespräche und die Übernahme der Zweitkorrektur dieser Arbeit.

- Dr. John Taylor Childers und Dr. Martin Wessels für viele hilfreiche Diskussionen über die Auswahl der Daten, die für die Fernüberwachung des Level-1 Kalorimeter Triggers benötigt werden.

- Dr. Reiner Hauser für wertvolle Informationen zur *CERN Single Sign-On* Authentifikation und die Beantwortung aller Fragen bezüglich des *Web-IS Interface*.

- Dr. Rainer Stamen und Michael Henke für das präzise Korrekturlesen dieser Arbeit und zahlreiche Verbesserungsvorschläge.

Ansonsten bedanke ich mich auch bei allen nicht genannten Mitgliedern der ATLAS-Gruppe am Kirchhoff-Institut für Physik in Heidelberg. Es hat riesige Freude gemacht Teil dieser Arbeitsgruppe zu sein.
Ganz besonders möchte ich meinen Eltern danken, die immer für mich da waren und mir ein sorgenfreies Studium ermöglicht haben.
Mein letzter Dank gilt meiner Frau Juliane Anders, die mich immer unterstützt hat und auf die ich mich jederzeit verlassen konnte. Vielen Dank für deine Freundschaft.

Erklärung:

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 20.04.2010

.....
(Unterschrift)