Eine Software zur Echtzeitanalyse von experimentellen Daten im Flexible Image Transport System (FITS)

Diplomarbeit
vorgelegt von
Stefan Schwarzburg

Eberhard Karls Universität Tübingen
Fakultät für Mathematik und Physik
Institut für Astronomie und Astrophysik
Abteilung Astronomie

Juni 2005

Inhaltsverzeichnis

Die wichtigsten Abkürzungen

ASCII American Standard Code for Information Interchange
ARF Ancilliary Response File
CCD Charge Coupled Device
CERN Conseil Européen pour la Recherche Nucléaire
CFITSIO C[1] FITS Input-Output
CPU Central Processing Unit
DNS Dynamic Name Server
DUO Dark Universe Observatory
ESA European Space Agency
ESO European Southern Observatory
ESTEC European Space Research and Technology Centre
FITS Flexible Image Transport System
fv fits view
GNU Gnu is Not Unix
GTI Good Time Interval
GUI Graphical User Interface
HDU Header and Data Unit
HEASARC High Energy Astrophysics Science Archive Research Center
HFWG HEASARC Fits Working Group
IAAT Institut für Astronomie und Astrophysik der Universität Tübingen
IAU International Astronomical Union
IAUFWG IAU Fits Working Group
IDL Interaktive Data Language
IEEE Institute of Electrical and Electronics Engineers
IPC Inter Process Communication
IRAF Image Reduction and Analysis Facility
ISDC Integral Science Data Center
MIP Minimal Ionizing Particle
MJD Modified Julian Date
MPE Max Planck Institut für Extraterrestrische Physik in Garching
NASA National Aeronautic and Space Administration
NOAO National Optical Astronomy Observatories
NOST Nasa/Science Office of Standards and Technology
NRAO National Radio Astronomy Observatory
OFWG OGIP Fits Working Group
OGIP Office of Guest Investigator Programs
PHA Pulse Hight Analyser
PI Pulse Invariant
PIL Parameter Interface Library

[1] Das C steht für die Programmiersprache C. Die entsprechende Bibliothek für Fortran heisst FITSIO.

RMF Redistribution Matrix File
SAO Smithsonian Astrophysical Observatory
UHURU ist keine Abkürzung, sondern heißt ‘Freiheit’ auf Swahili
UT Universal Time
WCS World Coordinate System
XEUS X-Ray Evolving Universe Spectrometer
XMM X-Ray Multi Mirror
XPA X Public Access

Kapitel 1

Einleitung: Zur Erforschung des Universums im 21. Jahrhundert

1.1 Politische Vorstellungen und ihre mediale Inszenierung: ‘A Renewed Spirit of Discovery’?

Abbildung 1.1: Banner der US-Regierung zu den neuen Raumfahrtplänen

Am 14. Januar 2004 hielt George W. Bush eine Rede vor der amerikanischen Weltraumbehörde NASA, die unter anderem auch von Astrophysikern in aller Welt mit Interesse erwartet worden war: für elf Milliarden Dollar allein in den kommenden fünf Jahren [1][1,2] wollen die USA ab 2008 vom Mond aus das Sonnensystem erkunden, um dort möglicherweise Alternativen für die ‘menschliche Gegenwart’ aufzuspüren. Roboter sollen auf dem Mond bemannte Aktionen vorbereiten, die ab 2015 folgen sollen [2][3]. Schon im Vorfeld der Rede war angekündigt worden, dass die Pläne neuen Schwung in das US-Raumfahrtprogramm bringen sollen, welches unter einer Serie von Rückschlägen leidet. Nach dem Absturz der Raumfähre ‘Columbia’ hatte die Untersuchungskommission neue Ziele für die US-Raumfahrt gefordert, der es seit 30 Jahren an einer klaren Vision fehle. Zugleich, so die ersten kritischen Anmerkungen, wolle der

[1] vgl. FR vom 16. 01. 2004, näheres zum Gesamtbudget des Projektes vgl. www.whitehouse.gov

[2] Alle folgenden Zitate entsprechen dieser Zitierform. Die Abkürzungen und die dazu gehörigen Artikel und Bücher sind im Literaturverzeichnis zu finden

[3] Bushs Rede ‘Let us continue the journey‘ vom 14. Januar 2004 im NASA-Hauptquartier findet sich im Ressort Reden der ZEIT in englischer Sprache abgedruckt. Die Rede ist auch über die Webseite des Weißen Hauses einzusehen unter: www.whitehouse.gov/news/release/2004/01/20040114-3.html

Präsident seinen Wahlkampf mit einem zukunftsträchtigen Thema besetzen [3].
Der Pressespiegel hielt dem mächtigsten Mann der Welt prompt das vor Augen, was die Öffentlichkeit von seinen Plänen hält: die Reaktionen reichten von vereinzelter Begeisterung für die neue Vision bis hin zu enttäuschter Kritik. In Zeiten massiver Finanznöte ist wohl nur schwer Verständnis für die horrenden Kosten des Projektes aufzubringen. Gegen das milliardenschwere Raumprogramm hat sich eine Koalition aus oppositionellen Demokraten aufgebaut, die mehr Geld für Bildung und soziale Programme fordern, sowie konservativen Abgeordneten, denen die eskalierenden Staatsdefizite wachsende Sorge bereiten [1].
Unabhängig davon haben aber auch Wissenschaftler Gründe zur Skepsis, die vielleicht weniger offensichtlich auf der Hand liegen. Statt Menschen sollten die USA lieber vermehrt Roboter wie die Mars-Sonde '*Spirit*' ins Weltall schicken, sagte Physik-Nobelpreisträger Douglas Osheroff in einer ersten Reaktion: 'Ich glaube, wir sind noch 30 Jahre davon entfernt, den Mars betreten zu können'. Und was aus wissenschaftlicher Perspektive noch viel interessanter sein könnte ist, dass er ergänzt: 'Und ich weiß nicht, ob es einen Grund gibt, es zu tun' [3].
Bushs Pläne haben außerdem eine Parallele zu dem, was bisher in der Astrophysik wissenschaftlich aufgebaut wurde: während Gelder in die neuen Ambitionen der Regierung fließen, werden sie woanders abgezogen, von Projekten wie dem *Hubble-Space-Teleskop* zum Beispiel. Dieses wird in Zukunft nicht mehr gewartet werden, alle Versuche, diese Entscheidung noch in letzter Minute abzuwenden, liefen ins Leere. Europäische Wissenschaftler fürchten außerdem um die versprochene Beteiligung der Amerikaner an der Internationalen Raumstation ISS. Die Versorgungsflüge der Space Shuttle werden zwar, so wird versichert, nicht ausbleiben, immerhin hat der Präsident erklärt, die USA würden ihre Verpflichtungen einhalten und die Raumstation bis 2010 fertig stellen [1], aber insgesamt verläuft das nun eingeschränkte Engagement der Amerikaner wohl so, dass der Eindruck entsteht, man werde sich langsam aber sicher aus diesem Projekt zurückziehen.
Wegen dieser möglichen Streichungen, die nebenbei gesagt auch weltraumgestützte Klimastudien betreffen, erklärte John Bahcall[4], es sei unklar, 'ob die Wissenschaft von den neuen Vorschlägen profitiert' [1].
So umstritten Bushs Rede auch sein mag, was sie sicher ganz deutlich gezeigt hat ist, dass Wissenschaft nicht unabhängig vom politischen Weltgeschehen und Regierungsinteressen abläuft. So urteilt Gero von Randow über die neuen Raumfahrtspläne des US-Präsidenten zu Mond und Mars, dass die Gründe für diese eher politisch als wissenschaftlich seien [4]. Es sieht so aus, als würden auch im 21. Jahrhundert die Raumfahrtpläne von Seiten der Politik gekonnt medial inszeniert, um eine enorme Publikumswirkung zu entfalten, aber das ist schließlich nichts neues: Bushs Rede erinnert deutlich – und wohl auch mit Absicht – an eine Rede seines Vaters von 1989: der hatte im Prinzip damals schon denselben langfristigen Plan vorzuschlagen: 'zurück zum Mond, zurück in die Zukunft, und diesmal, um zu bleiben' gefolgt von 'einer Reise ins Morgen, eine Reise zu einem anderen Planeten, eine bemannte Mission zum Mars' (zitiert nach [5]). Das war am 20. Jahrestag der ersten Mondlandung eines Menschen. Seine Pläne starben damals jedoch schnell, als der Kongress das Preisschild sah [5][5]. Schon vor der Rede

[4]John Bahcall ist Professor in Princton, wird hier aber wegen seiner Position als Nasa-Berater zitiert
[5]vgl. Christoph Drösser und Tobias Beck in ihrem Artikel 'Rot ist die Zukunft' in der ZEIT vom 15. 01. 2004

von George Bush war klar, dass er mit seinem Programm in die Fußstapfen seines Vaters tritt, was die Ziele für das 21. Jahrhundert angeht. Bush, so kommentiert Dietmar Ostermann aus Washington für die Frankfurter Rundschau, schwebe aber mehr vor: 'eine dauerhafte Kolonie auf dem Erdtrabanten, die Stars and Stripes am Mast. Spektakulärer geht es kaum - ein Projekt für die Geschichtsbücher. Bei der gebeutelten Nasa dürften die Sektkorken knallen.' Das gilt sicher nicht für die gesamte Nasa und erst recht nicht für Astrophysiker in aller Welt. Denn, so auch Ostermann 'über den wissenschaftlichen Nutzen einer möglichen bemannten Mondstation mag man ebenso streiten wie darüber, ob es klug wäre, ein derart teures Prestigeprojekt anzugehen, wenn gleichzeitig der in Finanznöten steckenden Internationalen Raumstation im Wortsinn die Luft entweicht und die alternde Shuttle-Flotte nach der Columbia-Katastrophe noch immer nicht wieder startklar ist'. Auch er kommt zu dem Ergebnis: 'die Wissenschaft war bei der bemannten Raumfahrt selten der entscheidende Impuls'. Schon beim Wettrennen mit der Sowjetunion ging es vor allem um nationales Prestige. Eine neue Mondmission, wenn sie überhaupt zustande komme, würde wohl nicht nur die Sonderstellung der einzigen Supermacht symbolisieren, sondern auch, dass diese ihre tiefen Wunden des 11. September endgültig überwunden habe [6]. Insgesamt scheint die Entwicklung, die die amerikanische Weltraumforschung momentan nimmt, deutlich auf eine stärkere Verknüpfung mit militärischer Sicherheitspolitik hinauszulaufen und auch die Europäische Weltraumorganisation ESA schwankt, was ihre Kursbestimmung angeht. Als aber der EU-Beauftragte für Außen- und Sicherheitspolitik Javier Solana 2003 auflistete, was einer künftigen EU-Eingreiftruppe von 60000 Mann zur technischen Ausrüstung noch fehle, da rangierten 'weltraumgestützte Sicherheitskomponenten' ganz oben, vom Navigationssystem bis zum Aufklärungssatelliten [7]. Die ESA arbeitet schon lange mit dualen Techniken, die sowohl zivil als auch militärisch nutzbar sind. In ihrem Testlabor im niederländischen Noordwijk etwa ist der französische *Helios I*-Satellit geprüft worden, der bei seinen Erdbeobachtungen auch Truppenbewegungen erspähen kann. Ähnliches gilt für optische Sensoren an Bord des Umweltsatelliten Envisat. Auch in der ESA wird es weiterhin um Fragen der Finanzierung dieses neuen Kurses gehen, da sie in einer Finanzkrise steckt: schon 2003 war die Lage die, dass die weltweiten Überkapazitäten bei den Trägerraketen die Startpreise mehr als halbiert haben. Chinesen, Inder, Russen, und Amerikaner bieten nun Dumpingpreise – und die europäische Raumfahrt wird Opfer ihres einstigen Erfolgs: sie verbucht zwar nach 30 Erfolgsjahren die Hälfte des Trägerraketenmarkts für sich – bei jedem Start macht sie aber Verluste [7]. 'Eine echte Krise', diagnostizierte der italienische Generaldirektor der ESA Antonio Rodota schon 2003 [7].

Ingesamt scheint der Fokus der internationalen Raumfahrt also konzeptionell auf eine verstärkte Verknüpfung von zivilen und militärischen Techniken gerichtet, während man sich inhaltlich – zumindest was NASA und US-Regierung angeht – wieder der bemannten Raumfahrt zuwendet. Auch längerfristig hat das wahrscheinlich finanzielle Umwälzungen in der Forschung zur Folge.

Nr. 4, vgl. auch FR vom 10.01.2004

1.2 Wissenschaftliche Interessen und ihre Rahmenbedingungen

Das bedauerliche an der gesamten momentanen Entwicklung der Raumfahrt ist meiner Meinung nach, dass der falsche Eindruck entstehen könnte, es gäbe im Weltall nichts anderes interessantes mehr zu erforschen, als die Bodenbeschaffenheit auf dem Mars und den möglichen Einsatz weltraumgestützter Sicherheitskomponenten. Dabei öffnete sich in den letzten Jahren der astrophysikalischen Forschung ein neues Fenster: sie hat sich aufgemacht, das Phänomen der Röntgenstrahlung und die damit verbundenen Geheimnisse näher ins Auge zu fassen. Eigentlich gibt es in der Physik kaum ein Thema, das so viel gespannte Aufmerksamkeit auch der breiten Öffentlichkeit auf sich zieht wie das, was sie versucht zu erforschen: Schwarze Löcher, Supernovae und Dunkle Materie zum Beispiel.

1.2.1 Allgemeines zur Röntgenastronomie

Die Vorgänge, mit denen die Röntgenastronomie zu tun hat, spielen sich nicht im sichtbaren Wellenlängenbereich ab, sondern im Bereich der Strahlung, die Wilhelm Röntgen 1895 bei der Untersuchung von Gasentladungsröhren entdeckt hat. Röntgenstrahlen wurden erst 1912 als elektromagnetische Wellen erkannt, ihre Wellenlänge liegt zwischen etwa 10^1nm - 10^{-3}nm oder – in einer anderen gebräuchlichen Einheit – zwischen etwa $0,1$ - 1000keV. Die Röntgenstrahlen grenzen damit einerseits an das Vakuum-Ultraviolett und überlappen bei kurzen Wellenlängen z.T. mit den Gammastrahlen [8, S. 400]. Insgesamt gibt es mehrere Arten zur Eerzeugung von Röntgenstrahlung, z.B. Schwarzkörperstrahlung, Synchrotonstrahlung, Zyklotronstrahlung und Bremsstrahlung. Auch bei Fluoreszenz und Kernprozessen, wie dem inversen β-Zerfall (K-Schaleneinfang), werden Röntgenstrahlen freigestzt. Ausserdem werden bei der inversen Comptonstreuung vorhandene Photonen zu höheren Energien gestreut [9]. Als Quellen für Röntgenstrahlung kommen sowohl galaktische Quellen – wie beispielsweise Sterne und Röntgendoppelsterne als auch extragalaktische Quellen, wie Aktive Galactic Nuclei (AGN, Kernregionen aktiver Galaxien) und Galaxienhaufen – in Frage [10].
Die erste extrasolare Röntgenquelle (Sco X-1) wurde erst 1962 von Giacconi et al. entdeckt. Geht man von idealisierter thermischer Strahlung – Schwarzkörperstrahlung– aus, dann liegt das Maximum des so entstandenen Spektrums bei $h\nu_{max} = 2,82kT$. Röntgenstrahlung wird also von extrem heißen Objekten in das Weltall emittiert und ist vom Erdboden aus nicht messbar, da sie in der Erdatmosphäre absorbiert wird. Für ihre Erforschung muss deshalb hoch in die Atmosphäre gegangen werden, entweder mit Hilfe von Raketen oder mit Ballonexperimenten. 1970-1973 wurde der erste Satellit (*UHURU*) eingesetzt, um Messungen dieser Art durchzuführen.

1.2.2 Einige Satellitenmissionen

Inzwischen kann die Röntgenastronomie auf Satelliten zurückgreifen, deren energieauflösende pixelbasierte Detektoren technisch sehr hoch entwickelt sind. Zurzeit stehen mehrere Satelliten dieser Art zur Verfügung, unter anderem *Chandra*, ein amerikanischer NASA Satellit, und zwei europäische Satelliten, *XMM-Newton* und *INTEGRAL*. Sie alle beobachten mit einem sehr hohen Niveau an Orts- und Energieauflösung das Hochenergiegeschehen in unserem Universum.

Chandra, nach dem berühmten Astrophysiker Subrahmanyan Chandrasekhar benannt, wurde im Juli 1999 mit einem NASA Space Shuttle gestartet. *XMM-Newton* gelangte im Dezember 1999 mit einer Ariane 5 Rakete in die Umlaufbahn und ist auf eine Lebensdauer von 10 Jahren angelegt, soll also eine Laufzeit bis mindestens 2009 haben. Im Gegensatz zu *Chandra* können alle seine Instrumente gleichzeitig betrieben werden, er hat sehr große Spiegelflächen mit einer hohen Sensitivität ($\sim 4400cm^2$ @ $1.5keV$), eine hohe Ortsauflösung ($6''$ FWHM) und eine sehr gute Energieauflösung($E/\Delta E \sim 20 - 500$ je nach Instrument). Durch den 48h-Orbit kann lange und ununterbrochen beobachtet werden, simultane Beobachtungen im optischen und im UV Bereich sind über ein 30cm Teleskop möglich. Sein wissenschaftliches Hauptziel ist die spektroskopische Untersuchung von ausgedehnten Röntgenquellen. Mit seinen Instrumenten können spektroskopische, abbildende und zeitliche Analysen durchgeführt werden, z.B. an AGN, an Quasaren und an zeitlich schnell variierenden Quellen wie Röntgendoppelsternsystemen [11].
Geplant sind weitere Projekte, wie die *XEUS*-Mission der ESA, das X-Ray Evolving Universe Spectrometer. *XEUS* soll Erkenntnisse zur Strukturbildung und Entwicklung unseres Universums liefern und kann mit speziellen und neuartigen Detektoren und Spiegeln außer den hellen Röntgenquellen auch leuchtschwächere und weiter entfernte Objekte abbilden. Die Mission soll z.B. klären, wie sich die Galaxien und Galaxienhaufen gebildet haben, wann supermassive Schwarze Löcher in den Zentren der Galaxien entstanden und in welchem Zusammenhang die Entwicklung von Sternen und Schwarzen Löchern steht [12].
Das für die nächsten Jahre geplante *DUO*-Projekt (Dark Universe Observatory) hätte sich mit einer weiteren noch ungelösten Frage der Röntgenastronomie beschäftigt: Wie beeinflusst Dunkle Energie Galaxienhaufen? *DUO* wäre eine Kleinsatellitenmission (small explorer SMEX) der Nasa gewesen, nichts desto trotz sind Satellitenprojekte Millionenprojekte. *DUO* wäre eine Zusammenarbeit verschiedenster Institute gewesen, unter anderem dem Institut für Astronomie und Astrophysik der Universität Tübingen (IAAT) und dem Max Planck Institut für Extraterrestrische Physik in Garching (MPE) [13]. Das Projekt war abhängig von der Finanzierung durch die NASA, im Winter 2004 wurde dann klar, dass es nicht finanziert wird. Es wäre eines der ersten Projekte gewesen, das durch die Ideen der hier vorgelegten Arbeit hätte unterstützt werden können. Sie stellt nämlich eine Software für solche Satellitenprojekte vor, die sowohl in der Testphase vor dem Start, als auch in der Auswertung nach dem Start in der Forschung eingesetzt werden kann.

1.2.3 Detektoren im Testlabor und im All

Bevor ein Satellit ins All befördert wird, durchläuft er nach jahrelanger Planungsphase – XEUS, der sich immer noch in Planung befindet, soll erst zwischen 2015 und 2019 starten – verschiedene Tests, um abschätzen zu können, wie seine Detektoren, z.B. Charge Coupled Devices, kurz CCDs, im All arbeiten werden. Im Labor wird dazu ein baugleicher Detektor aufgebaut und künstlich erzeugter Strahlung ausgesetzt. Dabei sind bestimmte Fragen von Interesse: wie reagieren zum Beispiel die Teile des CCDs auf Photonen? Welchen Einfluss hat die Temperatur und wieviele Photonen können nachgewiesen werden? Wenn ein Forscher hierfür beispielsweise die Temperatur erhöht, dann ist es wünschenswert, dass er den entsprechenden Effekt, der sich

daraus ergibt, in Echtzeit am Monitor ablesen kann, ohne dass wahrnehmbare Verzögerungen entstehen. Später im All fängt der Detektor dann natürliche Röntgenstrahlung auf. Nachdem die Detektoren ihre Daten aufgenommen haben, werden diese auf der Erde von Forschergruppen hinsichtlich bestimmter Fragestellungen analysiert. Die Daten werden reduziert, Rauschen wird entfernt, es finden u.a. Offsetkorrektur und Bad Pixel Filterung statt.
Programme zur Echtzeitanalyse und Darstellung von Hochenergiedetektordaten für die Labortests vor dem Start gibt es viele verschiedene. Der Grund für diese Vielzahl ist hauptsächlich der, dass einmal für jedes Satellitenprojekt gesonderte spezifische Software entwickelt wird. Daneben existiert außerdem wieder unterschiedliche Software für die spätere Datenbearbeitung. Im Folgenden soll zunächst gezeigt werden, wie die bisher im Labor benutzten Programme aufgebaut sind, um deutlich machen zu können, was das hier vorliegende Programmpaket ausmacht und was es von anderen Programmen unterscheidet.

1.2.4 Zum Aufbau der bisher im Labor benutzten Programme

Ein Beispiel für ein bereits existierendes Echtzeitanalyseprogramm ist das in IDL (Interactive Data Language, eine kommerzielle Programmiersprache) geschriebene ‘ON’, das hauptsächlich von Gisela Hartner (MPE) entwickelt wurde und das am IAAT und in einer Partneranlage in Neuried häufig eingesetzt wird, um die Daten der pn-CCDs auszuwerten.
Eine weitere Software dieser Art, vom MPE in Garching entwickelt, wurde während der Entstehung meiner Arbeit dem Tübinger Institut für Astronomie zur Verfügung gestellt, um die Daten der ursprünglich für *DUO* gedachten Detektoren zu testen. Die Arbeit mit diesen Programmen erwies sich als in mancher Hinsicht problematisch, was unter anderem der Anlass für die Überlegungen zu einem neuem Programmtyp war. Woraus diese Probleme resultieren und welche Vorteile diese Programme trotzdem bieten, werde ich hier kurz beschreiben:
Echtzeitprogramme dieser Art können spezielle, an den aktuell benutzten Detektor angepasste Datenformate verarbeiten und dazu auf die zur Entwicklung und für die Tests benutzte Hardware Rücksicht nehmen. Das ist besonders dann wichtig, wenn die Kapazität der zur Verfügung stehenden Computer voll ausgenutzt werden muss, um Echtzeitdarstellung zu erreichen.
Ein Merkmal vieler dieser Programme ist die Nutzung einer speziellen Sprache oder Entwicklungsumgebung, die auf die Bedürfnisse der Arbeit im Labor mit Meßinstrumenten zugeschnitten ist. Ein Beispiel für eine solche Entwicklungsumgebung ist LabView und die darunter liegende graphische Programmiersprache "G"[6]. Die Art dieser Programmierung, d.h. die graphische Darstellung des Datenflusses ist der Art, Hardwaredesign zu erstellen, besonders ähnlich. Die durch LabView bereits vorgefertigten Bausteine zur Darstellung von x-y-Plots oder Bildern inklusive Zoom- und Druck-Funktionen erleichtern das Erstellen von Laborsoftware erheblich: Mit wenigen Mausklicks liegt eine Graphik wie die blaue rechts im Fenster (Bild 1.2.4) zur Darstellung einer CCD Intensity Map vor. Zwei große daraus folgende Vorteile sind, dass diese Programme sehr schnell geschrieben werden können und komfortabel für den Benutzer sind. Die schnelle Programmierbarkeit ergibt sich daraus, dass Sprachen speziell für

[6]Programmiersprachen wie IDL und LabView können mehrere Tausend Dollar kosten, mehr unter www.rsinc.com/idl/ oder www.ni.com/labview/

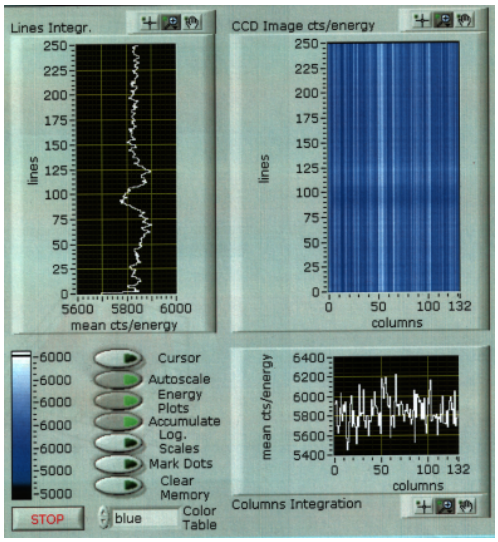

Abbildung 1.2: Ausschnitt aus einem für DUO geschriebenen Programm.

diesen Zweck entwickelt und angeschafft werden. Außerdem haben die Programmierer genaue Kenntnis, was die benötigten Informationen und die gewünschte Darstellungsart angeht. Die Komfortabilität für den Benutzer ergibt sich dadurch, dass das Programm meist bereits in eine graphische Benutzeroberfläche integriert ist. Dadurch, und weil das ganze Programm in einem Guss programmiert wurde, ist es außerdem besonders kompakt: Jedes seiner Teile kann jederzeit auf alle internen Daten zugreifen. Ist eine Information erst einmal aus den Daten extrahiert, steht sie sofort der weiteren Verarbeitung und Darstellung zur Verfügung und zwar jedem einzelnen Element des Programms. Der Speicherbedarf bleibt somit, also dadurch, dass jede Information nur ein einziges mal gespeichert werden muss, gering.
Daraus ergibt sich direkt ein großer Nachteil: aufgrund der Kompaktheit und Abgeschlossenheit der Programme ist es kaum möglich, sie einfach anderen Erfordernissen anzupassen. Sie sind nicht korrigierbar, neue Bausteine, später Arbeitsmodule genannt, können nicht eingefügt und die Programme deshalb auch nicht für Arbeiten mit einem anderen Detektor wiederverwendet werden. Hinzu kommt, dass diese Programme so individuell auf die Daten eines Detektors angepasst sind, dass sie nur in der Phase im Labor verwendet werden können, nicht mehr aber später, in der Auswertungsphase, die ja eigentlich denselben Detektor mit denselben Fragestellungen im Blick hat. Dies kann als äußerst unökonomisch bezeichnet werden. Würden sich die Programmierer der Testsoftware an die Standards halten, die sich unter den Programmierern der Auswertungssoftware bereits durchgesetzt haben, dann könnte mehrfach eingespart werden: ein allgemeineres Programm könnte für mehrere verschiedene Detektoren verwendet werden und für diese sowohl in der Testphase vor dem Start im Labor als auch in der Auswertungsphase danach zur Verfügung stehen. Auch wichtig ist hier, dass dieses Programm dann besser getestet wurde (siehe auch Abschnitt 1.2.5).
Ein weiterer großer Nachteil der bisherigen Programme ist der hohe Grad an Kommerzialisie-

rung, dem solche Programme unterliegen, die auf einer kommerziellen Sprache wie IDL oder LabView basieren. Um mit diesen Sprachen programmieren zu können, müssen die Forschergruppen tausende von Euro pro Arbeitsplatz bezahlen. Geht ein Konzern bankrott, oder lohnt es sich nicht mehr, die Sprache weiter zu vertreiben, dann geht das komplette Programm verloren, da es als Ganzes vom Markt genommen wird. Auch die darauf basierenden Programme werden damit untauglich. IDL basierte Programme wären dann immerhin mit einiger Mühe übersetzbar in eine freie Programmiersprache[7], da sie aus einem menschenlesbaren Code aufgebaut sind. Die Benutzer von LabView hingegen hätten das Problem, dass ihre Programme vollständig aus Zeichnungen bestehen, die nicht in eine andere Programmiersprache übertragbar sind.

1.2.5 Über den Wert von Standards und das Konzept eines allgemeinen Pipelinedrivers

Bei einer Optimierung für einen speziellen Datentyp oder eine spezielle Hardwareumgebung werden, was die Portabilität angeht, immer grosse Abstriche gemacht. Deshalb muss tatsächlich für jeden neuen Detektor ein neues, komplettes Programm geschrieben werden. Dies bringt natürlich weitere erhebliche Nachteile mit sich.
Zunächst ist die Entwicklung eines neuen Echtzeitprogrammes mit personellem und zeitlichem Aufwand verbunden. Besonders wichtig ist aber auch, dass dadurch die Gefahr einer höheren Fehleranfälligkeit in Kauf genommen wird. Es muss davon ausgegangen werden, dass jedes neue Programm auch neue Fehler enthält, die erst in einer ausgiebigen Testphase zum Vorschein kommen, was wiederum mit Kosten und Zeit verbunden ist, oder – was noch schlimmer sein könnte – erst während der Verwendung der Software entdeckt werden.
Wäre es stattdessen möglich, bei einem neuen Detektor auf ein bereits vorhandenes Programm zurückgegreifen, und dieses dann noch zudem sowohl für die Tests der CCDs als auch für die Auswertung der Daten zu verwenden, dann könnte auf diese Weise erheblicher zeitlicher und personeller Aufwand eingespart werden. Der Einsatz von solchen bereits existierenden erprobten Programmen gelingt dann am besten, wenn die Daten einem international in der wissenschaftlichen Gemeinschaft gültigen Standard entsprechen. Damit sind standardisierte Datenformate gemeint. Ich werde im Verlauf dieser Arbeit auf das Datenformat FITS als möglichen geeigneten Standard Bezug nehmen. Nur wenn mit Standards gearbeitet wird, können Forschergruppen Daten und Ergebnisse austauschen und die Programme in ihre Arbeit integrieren.
Die Nützlichkeit standardisierter Datenformate wird in der Astronomie schon lange wahrgenommen, und es gibt seit einiger Zeit Bemühungen, in der wissenschaftlichen Auswertung die Vielzahl der weltweit gemachten Beobachtungsdaten für die gesamte Wissenschaft nutzbar und vor allem auch vergleichbar zu gestalten. So haben sich in der Astronomie mittlerweile einige Standards herauskristallisiert. Diese werden bereits von einem breiten Spektrum von

[7] GDL - The GNU Data Language würde sich hier anbieten. Dieser unter der GNU General Public Licence (GPL) veröffentlichte Compilier kann IDL Code bis IDL 6.0 interpretieren. Er unterstützt aber unter anderem keine GUIs [14]

Programmen bedient.
Was für die Auswertung gilt, gilt aber noch nicht für die Testphasen im Labor, wo sich wie gesagt, noch kein Standard durchgesetzt hat. Das bedeutet, dass ein solcher Standard, und zwar am besten derselbe, der in der wissenschaftlichen Auswertung bereits verbreitet und erprobt ist, adaptiert werden muss für die Testphasen.
Die folgende Auflistung soll deutlich machen, wie viele Überschneidungen es tatsächlich zwischen den Analysen im Testlabor und später bei der Auswertung gibt, die ein neues 'ökonomischeres' Programm verkürzen könnte.
Die grundlegende Analyse, die beiden Arbeitsbereichen gemeinsam ist, setzt sich aus folgenden Arbeitsschritten zusammen:

- Spezielle Auswertungen für CCDs, z.B. MIPS (Minimal Ionizing Particles) und Splitevents finden.
- Anwenden einer Offsetmap [8]
- Eine Bad Pixel Filterung durchführen [9]
- Erstellen eines Intensitätsbildes
- Erstellen eines Spektrums
- Erstellen einer Lichtkurve
- ...

Ausserdem haben beide Arbeitsbereiche gemeinsam, dass sie die ausgewerteten Daten auch darstellen müssen. Dabei ist zu beachten, dass alles, was im Labor an Auswertung und Darstellung geschieht, in Echtzeit ablaufen muss. Diesem Anspruch muss die Software in der wissenschaftlichen Auswertung nicht gerecht werden.
Aus all diesen Überlegungen ergibt sich folgendes Bild eines neuartigen Programmtyps, der all diesen Anforderungen gerecht werden muss und dabei insofern ökonomischer verfährt, als dass er die sich überschneidenden Arbeitsbereiche in einem gemeinsamen Ablauf konzentriert.

Die gerade aufgelisteten Arbeitsschritte werden von einer Reihe von einzelnen Programmen, die ich im folgenden Arbeitsmodule nennen werde, durchgeführt. Entscheidend ist nun, dass je nach Forschungsziel, je nach Detektor, je nach Datenlage, in die bearbeitende Software Arbeitsmodule integrierbar sein sollten. Damit ein flexibles, multifunktionales Gesamtprogramm entsteht, müssen Einzelteile problemlos austauschbar sein. Anstelle eines kompakten, abgeschlossenen Programms, sollen nun verschiedene, austauschbare Arbeitsmodule über andere Programme, sogenannte Steuermodule oder Pipelinedriver, koordiniert werden. Dabei müssen mehrere Probleme gelöst werden.
Erstens muss sichergestellt werden, dass jedes Arbeitsmodul seine Aufgabe erfüllt, dass also kein Schritt im Verarbeitungsprozess ausgelassen wird oder Daten auf dem Weg der Verarbeitung verloren gehen. Das mag trivial klingen, passiert aber sehr leicht und ist ein grosses Problem.

[8] Offsetmaps sind Tabellen, die die Differenz von tatsächlichem Energiewert zu dem im Mittel vom CCD-Pixel gelieferten Wert darstellen.

[9] Bad Pixel sind Pixel, deren Werten grundsätzlich nicht getraut werden darf, weil diese Pixel defekt sind.

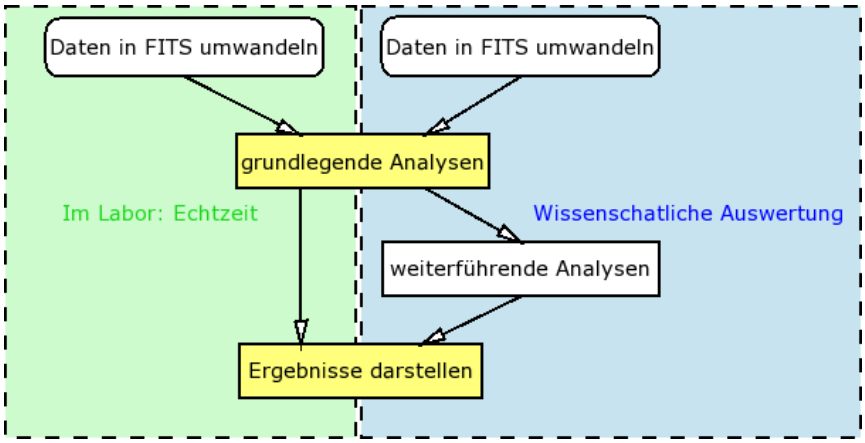

Abbildung 1.3: Überschneidungen zwischen Testlabor und Auswertung

Zweitens dürfen die Programme sich nicht gegenseitig behindern, indem sie beispielsweise versuchen gleichzeitig auf dieselben Daten zuzugreifen. Auch das können die Arbeitsmodule, die womöglich aus einem ganz anderen Kontext kommen, nicht selbst regeln. Das von mir benutzte Programm ds9 wurde z.B. ursprünglich für ganz andere Projekte programmiert. Deshalb muss ein Steuermodul dafür sorgen, dass die Arbeitsmodule sich nicht gegenseitig blockieren.
Ein drittes großes Problem stellt die CPU-Zeit dar. Die Arbeiten der einzelnen Programme müssen nicht nur möglichst schnell vor sich gehen, es muss auch sichergestellt werden, dass keine zusätzliche CPU-Zeit durch wartende Programme verbraucht wird. Es sollte im Idealfall eine Load kleiner als 1 herrschen, das heißt, zu jedem Zeitpunkt sollte ein oder gar kein Programm die CPU komplett ausnutzen und es sollte nie mehr als ein Programm diese zur gleichen Zeit benötigen, dann ergibt sich im zeitlichen Mittel als Anzahl der gleichzeitig CPU-Zeit verlangenden Programme der Wert 1 oder kleiner (daher Load < 1). Das bedeutet,dass unbedingt beachtet werden muss, dass kein Polling durchgeführt wird. Mit Polling wird das Verhalten eines Programmes umschrieben, in einer Schleife immer wieder eine Abfrage durchzuführen, bis das Ergebnis der Abfrage endlich positiv ist. Dies ist zwar eine einfache, aber keine ökonomische Art, Veränderungen in Dateien oder anderen Resourcen zu beobachten. Alternativ sollte immer ein Weg gefunden werden, der Programme schlafen lässt während sie warten. ‘Schlafen’ ist der offizielle Ausdruck, der verwendet wird, wenn ein Programm nichts tut, im Sinne von ‘keine einzige Operation durchführt’. Dies sollte grundsätzlich die bevorzugte Art des Wartens sein. Der Pipelinedriver kann das alleine nicht leisten, aber er trägt durch ein gelungenes Design mit zu diesem Ziel bei.
Ein letzter aber wichtiger Punkt ist der, dass die Steuermodule keinen Zugriff und Einfluss auf die von den Arbeitsmodulen gerade bearbeiteten Daten haben dürfen. Das ist deshalb wichtig, weil Steuermodule gewissermaßen funktionale Operatoren sind, die allein dazu existieren, den Arbeitsmodulen Impulse zu geben, damit sie wissen, wann sie arbeiten sollen. Die koordinierende Struktur muss unbedingt unabhängig von den einzelnen Arbeitsabläufen und Dateninhalten exis-

tieren, weil diese ja jederzeit austauschbar bleiben sollen. Eine Abhängigkeit oder Vermischung dieser beiden Strukturen kann deshalb auf keinen Fall wünschenswert sein. Bei der Entwicklung muss also immer im Blick behalten werden, dass die Steuermodule der Pipeline keinerlei Informationen über die Daten in der Pipeline, die Arbeitsmoduldaten sind, erhalten. Dazu gehört auch, dass alle für den jeweils nächsten Arbeitsschritt notwendigen Informationen in der Datenausgabe des zuvor aufgerufenen Arbeitsmoduls enthalten sein müssen, oder dass jedes Programm alle nötigen Informationen selbst extrahiert, weil diese Informationen nicht aus den Steuerelementen kommen können. Diese Auflage führt auf jeden Fall leider dazu, dass Rechenzeit und benötigter Speicherplatz im Vergleich zu kompakten All-In-One Programmen höher sind. Das muss in Kauf genommen werden, um die beiden Ebenen, die Datenebene und die Koordinationsebene nicht zu verwischen, denn sonst würden die Vorteile des neuen Laborprogrammtyps, Flexibilität und Mehrfachverwendbarkeit, aufgegeben werden. Nur die Verwendbarkeit für jede Art von Datenpipeline macht aber diesen Ansatz effizient und praktikabel.
Der grösste Vorteil eines solch allgemeinen Pipelinedrivers, der keine Informationen über die verarbeiteten Daten enthält, sei aber nochmals abschließend zusammengefasst: Jede Art von Pipeline kann mit ihm betrieben werden, und somit kann jederzeit die Struktur, die Zusammensetzung der Arbeitsmodule verändert werden, ohne dass dies einen Unterschied für die Steuermodule macht. Dies steht im direkten Gegensatz zu den kompakten Echtzeitprogrammen, bei denen Änderungen in der Verarbeitung der Daten und jede andere Art von neuen Anforderungen meist große Probleme mit sich bringen.

1.2.6 Was diese Arbeit zeigen soll

Mein Programm soll die Idee dieses neuen Programmtyps vorstellen und prüfen.
Es ist deshalb als eine Machbarkeitsstudie konzipiert, die von folgenden Fragen ausgehen wird und mögliche Schwachpunkte aufzeigen soll:
Zunächst bestehen allgemeine Fragen die generelle Machbarkeit des Projekts an sich betreffend:

- Ist es möglich eine Software zu entwickeln, die beide Arbeitsbereiche einer Satellitenmisson (Testphase und Auswertungsphase) insofern miteinander verbindet, als dass sie Testsoftware und Auswertungssoftware in einem ist?
- Kann eine Software so flexibel programmiert werden, dass sie für mehrere Detektortypen verwendet werden kann?
- Ist eine Echtzeitdarstellung im Rahmen der derzeit verfügbaren Rechnerleistung überhaupt möglich?

Zweitens bestehen spezifischere Fragen den Aufbau eines solchen Softwarepaketes, speziell den Aufbau der Arbeitsmodule und Steuermodule, betreffend:

- Aus welchen Einzelbausteinen besteht das Softwarepaket?
- Welche Arbeitsmodule werden benötigt für welche Aufgaben? Worauf sollte beim Erstellen neuer Arbeitsmodule geachtet werden?
- Wie sind sie sinnvoll anzuordnen?
- Wie können die Daten zwischen den Arbeitsmodulen transportiert werden?
- Wie müssen sie untereinander verbunden werden?

- Ist ein grosses Steuerzentrum sinnvoll, oder werden besser viele kleine einzelne Steuermodule eingesetzt?
- Wie sieht das Netzwerk aus, das sich daraus ergibt?
- Ist es möglich dieses neue Programm tatsächlich so aufzubauen, dass Steuermodule und Arbeitsmodule zwei unabhängig voneinander arbeitende Strukturen darstellen?
- Können auf diese Weise wirklich Arbeitsmodule ergänzt und ausgetauscht werden?

Ausserdem muss der nötige Arbeitsaufwand geklärt werden für ein solches Softwarepaket:

- Gibt es z.B. bereits einen Datentyp, der sich als Standard besonders anbietet? Was macht ihn so geeignet?
- Inwiefern sind existierende Programme wie ds9 tatsächlich verwendbar?

Es wird am Ende dieser Arbeit wichtig sein, dass während des Programmierens ein Weg gegangen wurde, der möglichst viele Schwierigkeiten aufgezeigt hat. Dabei muss nicht unbedingt alles ausführt werden, was angedacht wird, da es hauptsächlich darum geht, den prinzipiellen Weg zu zeigen. Ich werde im folgenden zunächst ausführlicher auf das Datenformat FITS und seine Besonderheiten eingehen und darstellen, inwiefern dieses Datenformat tatsächlich besonders gut geeignet für meinen angestrebten Programmtyp ist. Danach werde ich meine Programmstruktur vorstellen und auf ihre Arbeitsmodule, Steuermodule und ihre graphische Benutzeroberfläche eingehen.

Kapitel 2

Das Datenformat

Ein wichtiges Kriterium beim Erstellen dieser Diplomarbeit war die Verwertbarkeit des Programmpaketes für eine möglichst grosse Anzahl von Projekten. Darum ist die Wahl des verwendeten Formates zur Speicherung der Daten wichtig. Es muss darauf geachtet werden, dass das Format weit verbreitet ist, von vielen astronomischen Software Paketen unterstützt wird, dass es flexibel an die jeweils zu bearbeitenden Daten anpassbar ist und nicht zuletzt, dass es Archivierbarkeit garantiert, d.h. es muss garantiert sein, dass auch in mehreren Jahren noch Software existiert, die dieses Format unterstützt. In der Astronomie bietet sich das FITS (Flexible Image Transport System) Format an, das all diesen Anforderungen gerecht wird [15] [16] .

2.1 Das FITS-Format

Die Entwicklung des FITS Formates begann 1979. In diesem Jahr wurden FITS Dateien zum ersten mal benutzt.
Ein Jahr später wurde die Konvention für ‘random groups’ eingeführt. Die erste Veröffentlichung zum ursprünglichen Format wurde 1981 geschrieben [15], zu diesem Zeitpunkt bestand eine FITS Datei noch aus einer einzelnen HDU (Header and Data Unit). Im selben Jahr wurde auch die Definition der ‘random groups’ veröffentlicht [17].
Die International Astronomical Union (IAU) erkannte das Format 1982 formal an [18].
Die heute wichtige Eigenschaft, dass eine FITS Datei mehrere Erweiterungen haben kann, wurde 1988 definiert [19]. Im selben Jahr wurde auch die IAUFWG (IAU FITS Working Group) gegründet [20] und die Möglichkeit ASCII Tabellen in FITS einzubinden wurde festgelegt [21].
1990 wurde dann der Standard für IEEE floating-point Daten festgelegt [22].
Etwa fünf Jahre später wurden zwei weitere wichtige Neuerungen in den FITS Standard aufgenommen, die Möglichkeit für weitere Bild-Arrays in IMAGE Erweiterungen und die Möglichkeit, Tabellen auch im Binärformat zu speichern [23] und [24].
Die neueren Veröffentlichungen von 2002 fassten die bisherigen Änderungen in einem Paper zusammen und schrieben bereits existierende Konventionen für ‘world coordinates’ und ‘celestial coordinates’ fest [25] [26]. Die auch heute gültige Veröffentlichung ist der NOST (Nasa/Science Office of Standards and Technology) FITS Standard [16].

2.1.1 Zur Flexibilität des FITS Formats

Die bereits im Namen genannte Flexibilität wird bei FITS Dateien durch die Unterteilung der Daten in einen 'Header-' und einen 'Data'-Teil erreicht. Diese ergeben zusammen eine so genannte HDU. Eine einzelne FITS Datei kann dabei beliebig viele HDUs enthalten, mit der Einschränkung, dass die erste HDU ein Bild (d.h. ein möglicherweise leeres Array, das so genannte Primary Array) ist.
Der 'Header' besteht aus einer Reihe von obligatorischen und optionalen 'card images' oder 'keyword records'. Solch ein 'card image' besteht aus 80 Zeichen, die wie folgt zusammengesetzt sind [16]:
1-8 Name des Keywords
9-10 '= ' (falls das Keyword einen Wert hat ('value indicator'))
11-80 Wert des Keywords, gefolgt von einem Kommentar. Der Kommentar wird durch '/' eingeleitet.
Somit sieht ein typischer 'card image' so aus: `KEYNAME = value / comment string`
Einige Keywords müssen vorhanden sein, damit die Datei dem FITS Standard entspricht, dazu zählen unter anderem 'SIMPLE', 'BITPIX', 'NAXIS', 'NAXISn', 'XTENSION', 'EXTEND' und 'END'.
Eine weitere Gruppe von Keywords hat festgelegte Bedeutungen, und darf nicht anders verwendet werden, dazu zählen zum Beispiel 'DATE', 'ORIGIN' und 'TELESCOP', aber auch noch viele andere (siehe z.B.: [27] und [32]).
Keywords die nicht in die oben genannten Gruppen fallen, können jederzeit der FITS Datei hinzugefügt werden, um zusätzliche Informationen zu speichern.
Die Keywords legen fest, ob Erweiterungen in der Datei vorhanden sind. Mögliche Erweiterungen sind:

- ein Bild (d.h. ein n-dimensionales Array) oder
- eine ASCII-Tabelle oder
- eine Binary-Tabelle

Obwohl ein Bild beliebig viele Dimensionen haben kann (in Integer Grenzen), sind 1D-Spektren, 2D-Bilder und 3D-Datenwürfel die am häufigsten anzutreffenden Bildformate. Die beiden Tabellenarten unterscheiden sich hauptsächlich in der Speicherung der Daten, aber es gibt bei Binary-Arrays noch zusätzlich die Möglichkeit, sogenannte 'variable length columns' zu nutzen, bei denen die Länge der Daten in den einzelnen Spalten nicht festgelegt werden muss. Möglich wird dies durch eine Konstruktion, die erlaubt, dass nach den eigentlichen Daten der Tabelle noch zusätzliche Daten geschrieben werden können, die nicht dem FITS Standard entsprechen müssen. Bei 'variable length arrays' wird dieser Platz genutzt, um die einzelnen Einträge der Tabelle aufzunehmen. In der Tabelle selbst steht dann nur eine Beschreibung derjenigen Daten, die im 'heap', dem Speicherplatz nach den regulären Tabellendaten, enthaltenen sind. Diese Struktur ist dann: [table_header] [record_storage_area] [heap_area]
Die Anzahl und Grösse normaler Tabellen befindet sich auch im Header, ebenso wie die Grösse von Bildern. Dazu kommen noch Beschreibungen der Spaltennamen, Spalteneinheiten (z.B. Interger, Real) und der physikalischen Einheiten.
Alle im Rahmen dieser Diplomarbeit entwickelten Arbeitsmodule schreiben einen Eintrag in den

Header aller von ihnen erstellten oder geänderten FITS-Dateien. Die Einträge richten sich dabei nach Empfehlungen der OGIP (Office of Guest Investigator Programs) [27][28][29].
Wird die Datei von den Arbeitsmodulen erstellt, so schreiben sie auf jeden Fall folgende Keywords in den Header:

- ORIGIN = *Benutzereingabe*
- CREATOR = *program* v Versions Nummer
- DATE = *yyyy-mm-ddTHH:MM:SS*/date of file creation
- TELESCOP = *Benutzereingabe* /name of telescope
- INSTRUME = *Benutzereingabe* /name of instrument
- OBSERVER = *Benutzereingabe* /observer who acquired the data
- CONFIGUR = *Benutzereingabe* /software configuration used to process the data

Existiert die Datei bereits und wird nur von diesem Arbeitsmodul weiter bearbeitet, dann werden nicht alle diese Keywords in den Header geschrieben.
Die Funktion, die die Daten vom Detektor holt, schreibt außerdem noch die Keywords:

- DATE-OBS = *yyyy-mm-ddTHH:MM:SS* /date of the beginning of observation
- DATE-END = *yyyy-mm-ddTHH:MM:SS* /date of the end of observation

Die jeweiligen Einträge, die mit Benutzereingabe gekennzeichnet sind, werden über Parameter der Arbeitsmodule eingegeben, meist ist als Standardparameter eine Umgebungsvariable gesetzt. Mehr dazu unter Abschnitt 3.1 und 3.2.

2.1.2 Zur Archivierbarkeit des FITS Formats

> ‘An archival format must be utterly portable and self-describing, on the assumption that, apart from the transcription device, neither the software nor the hardware that wrote the data will be available when the data are read.’
> (Steering Committee for the Study on the Long-Term Retention of Selected Scientific and Technical Records of the Federal Government,[US] National Research Council[1], [US]zitiert nach [16, S. 1]).

Das FITS Format ist grundsätzlich dazu entworfen worden, um für Transport und Archivierung von astrophysikalischen Daten gut geeignet zu sein. Neben dem selbstbeschreibenden Aufbau der FITS Dateien ist dabei noch ein zweiter Aspekt von Bedeutung. Dieser betrifft die Definition von Neuerungen im Standard. Die wichtige Regel diesbezüglich wird als die ‘Once FITS, always FITS’ Regel bezeichnet. Danach darf eine neue Definition keine bisher dem Standard entsprechende Datei ungültig machen. Dies garantiert die Archivierbarkeit der gesammelten Daten zusätzlich zur Unabhängigkeit des FITS von Speichermedien und dem selbstbeschreibenden Aufbau.

[1] Steering Committee for the Study on the Long-Term Retention of Selected Scientific and Technical Records of the Federal Government, [US] National Research Council: Preserving Scientific Data on our Physical Universe. National Academy Press, 1995. Seite 60

Die Archivierbarkeit ist auch mit ein Grund, warum bei der Programmierung meines Softwarepaketes auf einige proprietäre Programmiersprachen oder Entwicklungsumgebungen verzichtet wurde, da einige von ihnen nur über einen eigenen, d.h. nur von diesen Sprachen lesbaren, Quellcode verfügen. Dies widerspricht aber stark den Bemühungen dieser Arbeit, die Daten von heute auch in Zukunft nutzen zu können.

2.1.3 Zur Verbreitung des FITS Formats

Das FITS-Format wird heute von einer Vielzahl von in der Astronomie verwendeten Programmen unterstützt. Eine Liste der Programme mit denen sich FITS-Dateien oder Bilder anzeigen lassen ist auf der Internetseite des FITS Support Office der NASA zu finden [30], hier nur einige Auszüge daraus:

- SAOImage ds9 – ein Programm zum Darstellen von (FITS) Bildern, entwickelt am SAO (Smithsonian Astrophysical Observatory)
- fv – ermöglicht das Erstellen, Bearbeiten und Anschauen von beliebigen FITS Dateien
- FITSview – ein FITS Bildbetrachter vom NRAO (National Radio Astronomy Observatory)
- SkyCat – ein Program der ESO
- Aladin – ein interaktiver Himmelsatlas
- Avis FITS Viewer and Fits4Win Viewer – zwei Betrachter für Windows
- xv – Bildbetrachter, der auch FITS Bilder anzeigen kann
- Liberator – ein plug-in für den Adobe Photoshop

Dazu kommen noch Programme, die FITS und andere Graphikformate ineinander umwandeln können, auch hier ein paar Beispiele, die das FITS Support Office aufführt [30]:

- ImageMagick – ein Formatierungsprogramm, das 68 Formate unterstützt, eines davon ist das FITS Format
- pbmplus – ein anderes Konvertierungsprogramm
- gimp – das GNU Image Manipulation Package
- FITS2jpeg – von Bill Cotton (NRAO)entwickelt, benötigt CFITSIO und libjpeg
- fts2gif – von Michal Szymanski (Warsaw University Observatory) entwickelt

Die Anzahl der Programme die mit FITS arbeiten können, ist natürlich wesentlich grösser.
Ausser diesen kompletten Programmen gibt es noch eine Vielzahl an Programmierbibliotheken in verschiedenen Sprachen. Auch hier eine kleine Auswahl aus der Zusammenstellung des FITS Support Office [30]:

- CFITSIO in den Sprachen: C/Fortran, zusätzlich aufrufbar aus den folgenden Sprachen:
  - C++: CCfits
  - C-#: FitsLib (.Net)
  - Perl: CFITSIO.pm bzw. Astro::FITS::CFITSIO
  - Tcl: fitsTcl
  - Python: pCFITSIO
  - MatLab: MFITSIO
- WCS FITS library in C (Fortran callable)
- fitsy/funtools in C

- qfits in den Sprachen C und Python
- C++ FITS in C++
- MRDFITS/MWRFITS in IDL
- FX library in IDL
- READFITS/WRITEFITS in IDL
- FITS_library in IDL
- IUEDAC in IDL
- nom.tam.fits in Java
- eap.fits in Java
- jfits in Java
- PDL FITS in Perl
- PyFITS in Python

Von diesen Bibliotheken können nicht alle mit beliebigen FITS-Dateien umgehen, beispielsweise unterstützt PyFITS keine Gruppen und die FX Library keine ASCII-Tabellen.
Die CFITSIO Bibliothek hat unter diesen eine Sonderrolle, nicht nur, weil sie alle FITS-Dateien unterstützt, sie verfügt auch noch über eine ganze Reihe von zusätzlichen Funktionen. Das FITS Support Office bezeichnet sie als die fortgeschrittenste Bibliothek, um mit FITS-Dateien zu arbeiten [30] . Da ich mich ab jetzt immer wieder auf die CFITSIO Bibliothek beziehen werde, möchte ich im Folgenden diese Bibliothek vorstellen. Sie wird dann nicht mehr im Kapitel über die Arbeitsmodule behandelt, wo ich die anderen beiden von mir verwendeten Programmierbibliotheken beschreibe.

2.1.4 Die verwendete Bibliothek zum Bearbeiten von FITS Dateien: CFITSIO

Die wichtigste Bibliothek, die in diesem Softwarepaket verwendet wurde, ist sicher die CFITSIO Bibliothek. Entwickelt wurde sie ursprünglich vom HEASARC (High Energy Astrophysics Science Archive Research Center) am NASA Goddard Space Flight Center. Die heutige Version enthält aber viele wichtige Beiträge von anderer Seite, darunter vom ISDC (Integral Science Data Center), von XMM/ESA/ESTEC und von vielen einzelnen Personen [31].
In diesem Abschnitt werde ich einige Gründe aufzeigen, die für die Verwendung dieser Bibliothek sprechen und die die CFITSIO vor den meisten anderen verfügbaren Bibliotheken auszeichnen. Wie in Abschnitt 2.1 gezeigt, gibt es sehr viele Programmierbibliotheken, die sich für das Lesen und Schreiben von FITS-Dateien eignen. Sie verfügen aber nicht über einige sehr nützliche Eigenschaften von CFITSIO. Alle meine Arbeitsmodule arbeiten mit CFITSIO und profitieren somit von deren Vorteilen.
Nicht alle davon sind für den Benutzer bei der Anwendung sichtbar, einige haben aber entscheidenden Einfluss beim Aufsetzen einer neuen Pipeline. Im Gegensatz zu manchen anderen FITS-Bibliotheken unterstützt die CFITSIO alle durch die NOST definierten FITS-Formate. Auf diese, eigentlich für eine FITS-Bibliothek fast vorauszusetzende Eigenschaft, werde ich aber nicht eingehen. Stattdessen werde ich nur darüber hinausgehende Möglichkeiten beschreiben.

2.1.4.1 Von CFITSIO lesbare oder schreibbare Datenformate

Die CFITSIO-Bibliothek basiert seit Version 2.0 auf einem Treiberkonzept, welches vom ISDC (von Jurek Borkowski, Bruce O'Neel und Don Jennings) entwickelt und implementiert wurde.
Nach diesem Konzept werden die zu öffnenden Dateien (wenn nötig) intern in eine neue FITS-Datei umgewandelt und diese wird dann von den FITS lesenden und schreibenden Routinen geöffnet. Auf diese Art lassen sich verschiedene Dateiformate lesen und schreiben. Diese Formate hängen eng mit der 'extended file name syntax' zusammen. Die wichtigsten Dateiformate sind hier zusammengestellt und die dazugehörige Dateinamensyntax angegeben:

IRAF-Bilder – (Image Reduction and Analysis Facility, entwickelt an den National Optical Astronomy Observatories (NOAO)) können direkt gelesen, d.h. in ein internes FITS-Format umgewandelt werden. Die IRAF-Datei muss die Endung *.imh* haben.

raw binary data arrays – können ebenfalls direkt gelesen werden. Hier ist die Endung *.dat*. Dieser müssen weitere Informationen über den Datentyp (i für integer, ...) und eine Dimensionsangabe folgen, beispielsweise **.dat[d128,64]* für ein Bild im 'double'Format.

Komprimierte Dateien – im GNU zip oder UNIX COMPRESS Format können gelesen werden. Sollen Dateien komprimiert gespeichert werden, dann muss dies am Ende des Dateinamens in folgender Form angegeben werden: Datei.fit[compress] benutzt den Rice Algorithmus, Datei.fit[compress GZIP], Datei.fit[compress Rice] und Datei.fit[compress PLIO] benutzen jeweils den angegebenen Algorithmus.

ftp – kann benutzt werden, um Dateien von entfernten Rechnern zu holen. Die Syntax ist dann: *ftp://weitere.adresse.der/Datei.fits*.

http – kann ebenso als Quelle dienen: *http://weitere.adresse.der/Datei.fits*.

root – ist ein besonderes Protokoll des CERN, bei dem auch das Schreiben der Dateien erlaubt ist: *root://weitere.adresse.der/Datei.fits*.

Shared Memory – ist für diese Arbeit besonders wichtig, da dies die schnellste Form der IPC für größere Datenmengen ist. Die Syntax lautet hier: *shmem://hx*, wobei x zwischen 0 und 16 liegen muss. Damit lassen sich die Dateien im Shared Memory mit einem Namen beschreiben, was das Arbeiten sehr erleichtert. So wirken diese Daten für den Benutzer wie eine Datei, was sie im Sinne des UNIX Kernels, und damit für alle relevanten Methoden des Öffnens, Lesens und Speicherns, nicht sind. Der dazugehörende Treiber wurde von Jurek Borkowski vom ISDC geschrieben.

Diese Dateiformate lassen sich dank der 'extended file name syntax' direkt öffnen, aber dies ist nicht das einzige, was mit diesen Dateinamen-Erweiterungen gemacht werden kann.
Sie werden auch dazu benutzt, um schon beim Öffnen Berechnungen oder ähnliches auf den Daten vorzunehmen. Eine FITS-Datei kann viele Erweiterungen (HDUs) enthalten, von diesen kann eine direkt selektiert werden, wenn ihr Name oder ihre Nummer angegeben wird: *Datei.fits[EVENTS]*. Von Bildern können auch nur Ausschnitte geöffnet werden. Die Syntax dazu ist eine Angabe der Form 'start:end:step' für jede Achse, wobei 'step' eine Schrittlänge angibt. So liest ein 'step' von 2 nur jedes zweite Pixel. Ein * kann genutzt werden, um eine komplette Achse auszuwählen: *Datei.fits[*:2, 512:256:2]*. Hier wird das Bild an der zweiten

Achse gespiegelt, da die ‘start’-Angabe größer ist als die ‘end’-Angabe.
Bei Tabellen können einige Angaben gemacht werden, die nicht nur selektieren, sondern auch Berechnungen auf den Daten durchführen oder neue Einträge in die Tabelle machen:

Spalten selektieren – wird durch Angabe der Spaltennamen, getrennt duch Kommas, durchgeführt.
Spalten umbennen – kann durch ‘Neuer Name == Alter Name’ geschehen.
Neue Spalten einfügen – kann durch Angabe des neuen Namens geschehen, gefolgt vom Datentyp in Klammern und dem Wert, der in der Spalte stehen soll.
Berechnungen – können vorgenommen werden, indem der Name der Spalte, gefolgt von der Rechnung, angegeben wird: *[col PI=PHA * 1.1 + 0.2]*.
Zeilen Selektion – kann durch eine logische Angabe gemacht werden, so dass nur Einträge übernommen werden, bei denen der logische Ausdruck wahr ist: *[PHA > 30]* übernimmt nur Zeilen, in denen der Eintrag in der PHA Spalte größer als 30 ist. Die logischen und mathematischen Ausdrücke, die erlaubt sind, sind in Tabelle 2.1 zusammengefasst.

Diese ‘extended file name syntax’ ist beim Aufsetzen einer neuen Pipeline wichtig, da hiermit einige der Selektionen durchgeführt werden können, die die Grenzwerte für Energie oder Ähnliches betreffen.

2.1.4.2 Die Iterator-Funktion

Die Iterator-Funktion (`fits_iterate_data`) eröffnet dem Programmierer die Möglichkeit, das Öffnen, Lesen und Speichern von Tabellen oder Bildern allein der CFITSIO-Bibliothek zu überlassen. Der Programmierer schreibt dann nur noch die Funktion, die auf den Zeilen einer Tabelle oder den Pixeln eines Bildes arbeitet. Diese Funktion wird der Iterator-Funktion als Zeiger übergeben, welche diese sogenannte Arbeitsfunktion dann beliebig oft ausführt, und ihr dabei Teile der Tabelle oder des Bildes zur Bearbeitung zur Verfügung stellt. Die Vorteile dieses Ansatzes sind:

- Die Operation auf den Daten und das Lesen der Daten werden komplett getrennt. Dies macht den Aufbau der Programme modularer und einfacher zu ändern.
- Der Programmierer muss sich über wesentlich weniger Dateizugriffe und Speicherplatzbelegungen kümmern, was vor allem zu einem geringeren zeitlichen Aufwand und einer niedrigeren Anfälligkeit für Programmierfehler führt.
- Da die Programmierer der CFITSIO viel Zeit, Wissen und Können in die Optimierung des CFITSIO-Codes gesteckt haben, sorgt dieser Ansatz für eine besonders schnelle Bearbeitung der Daten. Die Iterator-Funktion errechnet zunächst die optimale Anzahl der Schritte und deren Datengröße, um dann mit entsprechend optimierten Datenpaketen die Arbeitsfunktion aufzurufen.
- Beim Entwickeln größerer Projekte können die Arbeitsfunktionen aufgrund ihres einheitlichen Aufrufes und ihrer einheitlichen Struktur leicht die Entwicklungszeit verkürzen, da sie unabhängig vom tatsächlichen FITS-Format sind.

Tabelle 2.1: Mögliche mathematische und logische Operationen in der ‘extended file name syntax’

Name	Symbole	Name	Symole
equal	.eq. .EQ. $==$	not equal	.ne. .NE. $!=$
less than	.lt. .LT. $<$	less than/equal	.le. .LE. $<==<$
greater than	.gt. .GT. $>$	greater than/equal	.ge. .GE. $>==>$
or	.or. .OR. $\|\|$	and	.and. .AND. &&
negation	.not. .NOT. !	approx. equal$1e-7$	$\sim$
addition	$+$	subtraction	$-$
multiplication	$*$	division	$/$
negation	$-$	exponentiation	^
absolute value	$abs(x)$	cosine	$cos(x)$
sine	$sin(x)$	tangent	$tan(x)$
arc cosine	$arccos(x)$	arc sine	$arcsin(x)$
arc tangent	$arctan(x)$	arc tangent	$arctan2(x,y)$
hyperbolic cos	$cosh(x)$	hyperbolic sin	$sinh(x)$
hyperbolic tan	$tanh(x)$	round to nearest int	$round(x)$
round down to int	$floor(x)$	round up to int	$ceil(x)$
exponential	$exp(x)$	square root	$sqrt(x)$
natural log	$log(x)$	common log	$log10(x)$
modulus	i % j	random # [0.0,1.0]	$random()$
minimum	$min(x,y)$	maximum	$max(x,y)$
cumulative sum	$accum(x)$	sequential difference	$seqdiff(x)$
if-then-else	$b?x:y$		
#pi	3.1415...	#e	2.7182...
#deg	$\#pi/180$	#row	current row number
#null	undefined value	#snull	undefined string

Vor dem Aufruf der Iterator-Funktion muss eine Treiber-Funktion die FITS-Dateien öffnen und die Zeiger der Fitsfile-Strukturen auf die jeweils benötigten HDUs richten. Die *iteratorCol*-Struktur, die von der Iterator-Funktion benötigt wird, muss vor dem Aufruf ausgefüllt werden. Dazu werden folgende Daten benötigt:

fptr - dies ist der in CFITSIO übliche Pointer auf die Fitsfile-Struktur
colnum - entspricht der Nummer der Spalte in der FITS-Datei
colname - kann anstelle der Nummer angegeben werden und enthält den Namen der Spalte (beides wird im Fall von Bildern ignoriert)
datatype - legt den Datentyp fest, der der Arbeitsfunktion geliefert werden soll, ist dies nicht der

Datentyp, in dem die Daten in der FITS-Datei gespeichert sind, dann nimmt die CFITSIO-Bibliothek eine Konvertierung vor

iotype - gibt an, ob die Daten gelesen, geschrieben oder gelesen und geschrieben werden sollen

Die Spalten, die so in die Struktur eingesetzt werden, müssen nicht aus der gleichen Datei stammen, sondern können beliebig zusammengesetzt sein. Obwohl dies eine sehr gute Eigenschaft ist, ist sie in den Arbeitsmodulen meiner Software nicht richtig nutzbar. Dies liegt daran, dass der Offset, über den das erste zu lesende Element gesetzt werden kann, bei den Arbeitsmodulen im Infile und Outfile, also den zu lesenden und zu schreibenden Dateien, unterschiedlich sein kann. Für die Zwecke meiner Arbeit wäre es dienlich gewesen, wenn pro Spalte ein eigener Offset hätte angegeben werden können. Sind die Daten ausgefüllt, kann die Iterator-Funktion aufgerufen werden:

```
int fits_iterate_data(int narrays, iteratorCol *data,
                      long offset, long nPerLoop,
                      int (*workFn)(), void *userPointer,
                      int *status);
```

Über den Parameter ‘Offset’ kann angegeben werden, ab welcher Zeile die Daten gelesen werden sollen, und über den ‘userPointer’ können der Arbeitsfunktion noch zusätzliche Informationen zur Verfügung gestellt werden. Beides wird in den Arbeitsmodulen dieser Diplomarbeit getan: der Offset wird benutzt, um es den Programmen zu ermöglichen, nur ungelesene Daten oder alle Daten zu lesen, bzw. Daten anzuhängen oder zu überschreiben. Dies sind für die Pipeline wichtige Eigenschaften. Über den ‘userPointer’ werden den Arbeitsmodulen Strukturen übergeben, in denen alle wichtigen Informationen enthalten sind.
Ein wichtiger Parameter ist noch ‘nPerLoop’. Hier kann die Iterator-Funktion gezwungen werden, eine bestimmte Anzahl an Zeilen oder Pixeln pro Aufruf der Arbeitsfunktion zu liefern, oder auch alle Zeilen auf einmal. Wird dieser Parameter auf Null gesetzt, dann wird es der Iterator-Funktion überlassen, den optimalen Wert zu bestimmen.
Die Syntax der Arbeitsfunktionen ist immer gleich:

```
int user_work_fctn(long totaln, long offset, long firstn,
                   long nvalues, int narrays, iteratorCol *data,
                   void *userPointer)
```

Über diese Parameter werden der Arbeitsfunktion einige möglicherweise wichtige Informationen über den momentanen Aufruf, die gesamten Daten und der oben erwähnte ‘userPointer’ übergeben.
Über den Parameter ‘firstn’ kann die Arbeitsfunktion testen, ob dies der erste ihrer Aufrufe ist. Dies ist dann der Fall, wenn `firstn == offset + 1` ist (nicht wie im Manual [31] angegeben `firstn == 1`).
Ebenso kann getestet werden, ob der Aufruf der letzte ist, dies ist dann der Fall, wenn `firstrow - 1 + nrows -offset == totalrows` ist (auch hier enthält das Manual nicht die richtigen Angaben).

Eine weitere Eigenschaft der Iterator-Funktion ist die, dass das Datenarray, welches der Arbeitsfunktion geliefert wird, mit dem Index 1 beginnend die Daten enthält und bei Index 0 den Wert enthält, der benutzt wird, um undefinierte Eintragungen anzuzeigen.

2.1.4.3 Die Region-Dateien

Eine wichtige Filtermethode, die aber eine externe Datei benötigt, ist über ein sogennantes 'Regionfile'gegeben.
Diese Datei besteht aus ASCII-Text und beschreibt eine geometrische Figur (circle, ellipse, box, etc.). Diese Datei wird typischerweise über ein externes Programm wie fv oder ds9 erstellt (siehe Abschnitt 3.1.1.1). Diese Region, die Koordinaten eines 2D Bildes (oder WCS) darstellt, kann nun genutzt werden, um Zeilen einer Tabelle zu selektieren. Die entsprechende Syntax dazu ist: 'regfilter("region.reg", XPOS, YPOS)', wobei XPOS und YPOS die Spalten der Tabelle angeben, die die x und y Kooradinaten enthalten. Wenn die Koordinaten nicht übereinstimmen, können die entsprechenden Werte angepasst werden, wie in diesem Beispiel: 'regfilter("rosat.reg", X/32.+.5, Y/32.+.5)'.
Da ds9 in meinem Softwarepaket zur Darstellung der Bilder eingeplant war, hätte so interaktiv während des Auslesens eine Region ausgewählt werden können, über die die Daten bei der weiteren Bearbeitung selektiert worden wären. Genaueres zu diesem Thema im Ausblick am Ende dieser Arbeit.

2.2 Vorgaben und Definitionen durch die OFWG und HFWG

Das OGIP (Office of Guest Investigator Programs) des Goddard Space Flight Centers hat eine 'FITS Working Group' (OFWG) eingerichtet, um sinnvolle Konventionen, die in der Hochenergie Astrophysik verwendet werden, zu dokumentieren und darauf zu achten, dass diese weder sich untereinander noch dem offiziellen FITS Standard widersprechen.
Da einige dieser Konventionen beim Aufbau der Arbeitsmodule eine Rolle gespielt haben, werden ich hier auf einige der OGIP Memos eingehen.

2.2.1 Vorgaben der OFWG für die Erstellung von Event Lists

In ihrem Memo zu Eventlists legt die OGIP fest, was eine Event Liste ist und wie diese gewöhnlich im FITS-Format gespeichert werden sollte [32].
Eine Event Liste ist bei den meisten Programmen die Grundlage für eine weitere Auswertung und Datenreduktion, wie dies auch in meinem Softwarepaket der Fall ist. Im Anschluss an die Erstellung einer Event Liste werden die Daten meist gefiltert, zusammengefasst und nach einer Reihe von Kriterien in ihrer Qualität bewertet und gegebenenfalls aussortiert. Nach diesen Schritten erfolgen dann im Normalfall weitere grundlegende Analyseschritte, die aus dieser Liste dann ein Spektrum, eine Lichtkurve oder ein Bild machen. Ich werde später noch näher auf diese Datenformate eingehen.

Eine Event Liste wird als Liste, d.h. im FITS Format als ASCII oder Binary Tabelle gespeichert, und sollte als Erweiterungsnamen ‘EVENTS’ haben. Jedem Event bzw. Ereignis wird dabei genau eine Zeile zugeordnet, egal ob es sich in dieser Rohform um tatsächliche Photonen oder um andere unerwünschte Teilchen (bei CCDs z.B. um ein Teil einer MIP Spur) oder um Hintergrundrauschen handelt.
Die Event Liste besteht in einem ersten Schritt aus Rohdaten, d.h. unbearbeiteten Daten, wie sie beispielsweise von der Ausleseelektronik eines Detektors geliefert weden. In diesem Fall schlägt die OFWG vor, die x und y Position des Ereignisses auf dem Detektor (bei abbildenden Detektoren) durch Tabellenspalten mit Namen ‘RAWX’ und ‘RAWY’ zu speichern. Die jeweiligen Einheiten sind natürliche vom jeweiligen Detektor abhängig. Werden diese Positionen linearisiert, so werden die Namen ‘DETX’ und ‘DETY’ vorgeschlagen. Weitere Positionsinformationen, die aber für das Labor keine Rolle spielen, sind ‘X’, ‘Y’, ‘RA’, ‘DEC’, ‘L’ und ‘B’. Nicht von der OFWG vorgeschlagen, aber bei CCDs oft benutzt, werden auch EVTX und EVTY, meist zusammen mit PATTERN und PATTID, welche sich auf die bei CCDs vorkommenden Splitevents beziehen. Bei Splitevents hinterlassen einzelne Photonen Elektronenwolken in mehreren Pixeln, und werden daher beim Auslesen ersteinmal als zwei oder mehr Ereignisse eingestuft. Wird dieses Splitevent dann als solches erkannt, wird den einzelnen Splitpartnern eine PATTERN ID zugewiesen, je nach Position, die das Teilevent im Split Muster eingenommen hat.
Für die spektrale Information wird von der OFWG eine Reihe von Vorschlägen gemacht [28]:

- die PHA (Pulse Height Analyzer bin number) ist am besten im Integer Format und ohne Einheit zu speichern, ‘chan’ ist aber möglich
- die PI (Pulse Invariant bin number) ist wie eine PHA zu behandeln
- die ENERGY wird gespeichert als Real und vorzugsweise mit den Einheiten ‘keV’ oder ‘MeV’
- die WAVELENGTH wird auch als Real gespeichert und vorzugsweise in ‘m’ oder ‘angstrom’

ENERGY und WAVELENGTH sind dabei Analyseergebnisse, während PHA und PI in den Rohdaten verwendet werden. Die von verschiedenen Missionen verwendeteten Einheiten ‘CHAN’, ‘bin’ und ‘Angstroms’ entsprechen nicht den OFWG-Vorgaben.
Für die zeitliche Einordnung gibt es nur einen Vorschlag: die TIME wird mit real oder double Genauigkeit und beliebigen zeitlichen Einheiten, vorzugsweise Sekunden ‘s’, gespeichert. Die Qualität eines Events sollte in der Spalte ‘STATUS’ eingetragen werden, wobei ein Status von 0 gute Qualität bedeuten sollte [33]. Die Status-Spalte hat keine Einheiten.

2.2.2 Vorgaben der OFWG für die Erstellung von Lichtkurven

Auch für Lichtkurven gibt es von der OFWG Empfehlungen für zu benutzende Spalten und Keywords [34]. Diese sogenannten ‘Rate Files’, mit Erweiterungsnamen ‘RATE’ oder ‘EVENTS’, können in mehreren verschiedenen Arten vorliegen:

- als Events
  im Prinzip eine Eventliste mit Zeitinformation
- gleichmäßig verteilt
  eine Tabelle mit Intensitätsmessungen, die immer den gleichen zeitlichen Abstand voneinander haben
- ungleichmäßig verteilt
  eine Tabelle mit Intensitätsmessungen in verschiedenen zeitlichen Abständen
- als 'Packet'
  ein Histogramm Array mit Zeit oder PHA Information zu einem bestimmten Referenzzeitpunkt. Auch hier können die Einträge wieder gleichmäßig verteilt sein oder nicht.

Bei allen Tabellen, die nicht gleichmäßige zeitliche Abstände haben, ist die Spalte 'TIME' nötig. Entweder als absolute Zeit, oder relativ zu einer im Header festgelegten Zeit. Haben die Einträge gleiche zeitliche Abstände, so müssen diese zusammen mit einer Startzeit im Header angegeben sein. Die zeitdefinierenden Keywords im Header der Lichtkurve können entweder aus Paaren bestehen, oder aus einzelnen Einträgen. Werden Paare benutzt, so gibt der erste Wert einen ganzzahligen Wert und der zweite einen Fließkommawert an:

- MJDREFI = *integer* / integer portion in MJD for reference time
- MJDREFF = *float* /fractional portion of MJD for reference time
- TSTARTI = *integer* / integer portion of start time
- TSTARTF = *float* / fractional portion of start time
- TSTOPI = *integer* / integer portion of stop time
- TSTOPF = *float* / fractional portion of stop time
- TIMEZERI = *integer* / time zero integer used to calculate the n-time event or the n-time bin
- TIMEZERF = *float* / fractional portion of TIMEZERI

Die entsprechenden einzelnen Keywords sind MJDREF, TSTRART, TSTOP, TIMEZERO. Unabhängig vom Format müssen aber folgende Keywords angegeben werden:

- TIMESYS, welches das Zeitsystem festlegt und üblicherweise MDJ, JD oder TJD ist. Ist das Zeitsystem von einem Zeitpunkt in UT (Universal Time) ab gemessen, dann wird dieser Zeitpunkt angegeben.
- TIMEUNIT legt die Einheiten für die Start-, Stop- und Nullpunkte fest und ist etwa 's' für Sekunden oder 'd' für Tage. Desweiteren ist CLOCKCOR entweder mit YES, NO oder UNKNOWN anzugeben, um anzuzeigen, ob Korrekturen der Systemzeit des Detektors bezüglich UT durchgeführt wurden.

Die Spalten der Lichtkurven sind je nach Aufzeichnungsart unterschiedlich.
Werden die Daten als Events oder zumindest nicht in gleichmäßigem zeitlichen Abstand festgehalten, dann muss die Spalte TIME vorhanden sein. Bei zusammengefassten Events (Bins)

ist dies der mittlere Wert des Bins. Entweder kann hier die komplette Zeit oder das Residuum bezüglich des in TIMEZERO angegebenen Wertes angegeben werden.
Das Keyword TIMEREF gibt an, auf welchen Bezugsrahmen sich die angegebenen Zeiten beziehen (LOCAL, SOLARSYSTEM, HELIOCENTRIC oder GEOCENTRIC). Dies zeigt an, ob zum Beispiel baryzentrische Korrekturen durchgeführt wurden.
Ausserdem wird durch das Keyword TASSIGN angezeigt, wo die Zuordnung von Events und Zeit stattgefunden hat (SATELLITE, Tuebingen, ...), gegebenenfalls mit den Ortsangaben, falls ein Ort auf der Erde angegeben wurde (GEOLAT, GEOLONG, ALTITUDE).
Wenn möglich, sollte auch die Genauigkeit der Zeitangaben angegeben werden durch die Keywords TIERRELA (relativer Fehler als dimensionslose Rate) und TIERABSO (absoluter Fehler in Sekunden).
Wenn die Lichtkurve nicht aus einem Eintrag pro Event besteht, dann muss noch eine Spalte die Intensität enthalten. Der Spaltenname muss dann entweder COUNTS sein (mit der Einheit 'count' und im Format Integer), oder RATE. Wird der Name COUNTS gewählt, dann enthält ein Entrag in dieser Spalte die Anzahl der Photonen im entsprechenden Zeitbin, wird dagegen RATE gewählt, dann enthält ein Eintrag folgendes: $Rate = (counts - backcounts)/inttime$ (in s^{-1}) als Fließkommazahlen. Dabei ist 'inttime' die totzeitkorrigierte Integrationszeit. Counts (der Quelle) und backcounts (Hintergrund-Photonen) sollten dabei natürlich vergleichbar sein und deshalb müssen für die Hintergrundphotonen gegebenenfalls Korrekturen stattfinden, um z.B. unterschiedlich größe Sammelflächen für Quell- und Hintergrundphotonen auszugleichen. Hintergrundsubtraktionen finden allerdings bei Lichtkurven selten statt.

2.2.3 Vorgaben der OFWG für die Erstellung von Spektren

Die Erstellung von Spektren wird natürlich auch von der OFWG behandelt [28]. In einem Spektrum, dass zur wissenschaftlichen Analyse genutzt werden soll, sollten mehrere Informationstypen enthalten sein:

- eine Daten Erweiterung
- eine 'Good Time Interval' Erweiterung (GTI) und
- eine'(Selektor) History' Erweiterung

Die Arbeitsmodule meines Softwarepakets ähneln wie bereits erwähnt in vielerlei Hinsicht eher einer Machbarkeitsstudie, und sind keine kompletten Programme zur wissenschaftlichen Auswertung, wie etwa XSPEC. Aus diesem Grund ist auch das als Arbeitsmodul geschriebene Programm zur Erstellung eines Spektrums nur mit grundlegenden Eigenschaften ausgestattet und erstellt und beachtet die GTI und Selektor History Erweiterungen nicht. Trotzdem versucht es, ein einfaches aber OGIP konformes Spektrum zu erstellen. Daher werde ich an dieser Stelle auch nur auf die tatsächliche Datenerweiterung eingehen, so wie sie auch im Labor bei einer Echtzeitanwendung benötigt wird. Ebenso werde ich auch nicht auf alle Keywords im Detail eingehen, wenn diese nicht von meiner Software genutzt werden.
Die Daten Erweiterung eines Spektrums besteht aus einer Binärtabelle, wobei die Anzahl der

Zeilen der Anzahl der Energiekanäle des Detektors entspricht. Der Name der Erweiterung ist SPECTRUM. Die für ein OGIP konformes Spektrum benötigten Keywords (abgesehen von den üblichen wie TELESCOP, INSTRUME, ORIGIN, ...) werden in Tabelle 2.2 genannt.

Tabelle 2.2: Die nach dem OGIP Memo zu Spektren notwendigen Keywords in einer Spektrum Daten Erweiterung [28]

Keyword	Beschreibung
EXPOSURE	korrigierte Integrationszeit in Sekunden
BACKFILE	Name der Hintergrunddatei
CORRFILE	Name der Korrektionsdatei
CORRSCAL	Korrektionsskalierungsfaktor
RESPFILE	Name der Redistribution Matix Datei (RMF)
ANCFILE	Name der Ancillary Response Datei (ARF)
HDUCLASS	'= OGIP' wenn dies dem Standard entspricht
HDUCLAS1	'= SPECTRUM'
HDUVERS	OGIP Memo Nummer, der die Datei entspricht
POISSERR	'= T' wenn keine STAT_ERR Spalte vorhanden ist
CHANTYPE	PHA wenn nicht korrigiert, sonst z.B. PI
DETCHANS	Gesamtzahl der Detektorkanäle
TLMIN*n*	Falls die CHANNEL Spalte nicht mit 1 beginnt
TLMAX*n*	Falls die CHANNEL Spalte nicht mit 1 beginnt
CHANMIN	kleinster Detektor Channel
CHANMAX	größter Detektor Channel
QUALITY	'= 0' wenn alle Events als gut angesehen werden
GROUPING	'= 0' wenn kein Grouping stattgefunden hat
OBJECT	beobachtetes Objekt (optional)

Neben diesen notwendigen Keywords gibt es noch eine Reihe optionaler Keywords: XFLTXXX (bezieht sich auf den XSPEC Filter), RA-OBJ, DEC-OBJ, EQUINOX, RADECSYS. Diese werden nicht von FPSpectrum geschrieben.
Desweiteren gibt es die optionalen Keywords DATE-OBS, TIME-OBS, DATE-END, TIME-END, CREATOR, HDUCLAS2 (BKG für ein Hintergrundspektrum, NET für ein Hintergund subtrahiertes Spektrum, TOTAL für beides zusammen) und HDUCLAS3 (COUNT für PHA in counts oder RATE für PHA in $counts/s$). Diese Keywords werden von FPSpektrum geschrieben.
Für die Spalten der Datentabelle gibt es folgende Vorschläge von Seiten der OFWG:

CHANNEL – ein 2 oder 4 byte Integer, der die Kanalnummer des Detektors angibt und keine Einheiten besitzt.
RATE – falls die Anzahl der Photonen in $counts/s$ angegeben werden soll. Das Format ist eine

4-byte Fließkommazahl.

COUNTS – falls sie in $counts$ angegeben wird, dann ist das Format ein 2 oder 4 byte Integer.

STAT_ERR – eine 4-byte Fließkommazahl, die den statistischen Fehler der COUNTS oder RATE Daten angibt. Alternativ geht auch das POISSERR = T Keyword.

SYS_ERR – der fraktionale, systematische Fehler. Alternativ geht auch das SYS_ERR = 0 Keyword, wenn alle Einträge den Wert 0 hätten.

QUALITY – gibt an, ob die Daten gut ($= 0$) oder schlecht ($\neq 0$) sind. Zusätzlich zählt: 1: bad(software), 2: dubious(software) und 5: bad(user). Dies hat ein 2-byte Integer Format und besitzt keine Einheiten. Sind alle QUALITY Werte Null, so kann alternativ auch das QUALITY = 0 Keyword gesetzt werden.

GROUPING – legt fest, ob der Kanal am Begin (+1) oder Teil eines schon begonnen Bins ist (-1). Ein Wert von 0 sagt, dass keine Gruppierung stattfindet. Format ist ein 2-byte Integer ohne Einheiten. Alternativ zu einer Spalte mit Nullwerten kann auch GROUPING = 0 im Header angegeben werden.

AREASCAL – eine 4-byte Fließkommazahl, die einen Skalierungsfaktor aufgrund der effektiven Detektorfläche darstellt.

BACKSCAL – ebenfalls eine 4-byte Fließkommazahl, die einen Skalierungsfaktor, diesmal für den Hintergrund, der über BACKFILE eingelesen und vom Spektrum subtrahiert wird, angibt. Diese Spalte und AREASCAL wird von FPSpectrum nicht geschrieben oder angewandt. Allerdings wird die Alternative, nämlich AREASCAL = 1.0 und BACKSCAL = 1.0 im Header angegeben.

2.2.4 Zusätzliche Vorgaben durch die HFWG

Außer der OGIP bzw. OFWG macht auch noch die HFWG (HEASARC[2] FITS Working Group) Vorschläge, was spezielle Einträge in den Header angeht. Wichtig in Bezug auf mein Softwarepaket ist erstens die Recommendation R7 [29], die das Keyword CREATOR beschreibt. Damit soll das Programm benannt werden, das benutzt wurde, um dieses FITS-File herzustellen. Das Format für diesen CREATOR Eintrag sollte folgenden Aufbau haben:
CREATOR = ‘Programmname’v ‘Versionnummer’ (mit dem Kommentar: programm that created this FITS File)
Das Keyword AUTHOR sollte dazu nicht benutzt werden, da es reserviert ist für eine ursprünglich andere Bedeutung, nämlich die, eine Veröffentlichung zu zitieren.
Recommendation R9 [33] der HFWG ist ebenfalls von Bedeutung: sie legt fest, dass immer dann, wenn die Qualität eines Events beschrieben werden soll, der Wert 0 für ‘gut’ steht.
Die dritte wichtige Recommendation der HFWG ist R10 [35]. Sie besagt, dass in verarbeiteten Dateien, wie z.B. Bildern, Lichtkurven und Spektren, die Grenzen für Energiekanäle des Detektors angegeben werden können. In diesem Fall sollten die benutzten Keywords CHANMIN und CHANMAX heißen. Über CHANTYPE kann als Wert PHA oder PI das Kanalsystem angegeben werden, auf den sich CHANMIN und CHANMAX beziehen. Sollen nicht die Grenzen

[2]High Energy Astrophysics Science Archive Research Center

der Kanäle, sondern Energiegrenzen angegeben werden, dann stehen dafür die drei Keywords E_MIN, E_MAX und EUNIT zur Verfügung. In diesem Memo wird auch festgehalten, dass folgende Keywords nicht mehr benutzt werden sollten: MINCHAN, MAXCHAN, XS-MINCH, MINPI, MINPHA, XS-MAXCH, MAXPI, MAXPHA, XS-CHAN, PICHANS, PHACHANS, PIMIN, PIMAX, PHAMIN und PHAMAX.
Recommendation R6 [36] muss ebennfalls beachtet werden. Sie legt fest, wie Maximaldaten und Minimaldaten beschrieben werden. Tatsächlich vorkommende Grenzdaten werden durch TDMINn und TDMAXn (n ist dabei die Spaltennummer) kenntlich gemacht. Durch TLMINn und TLMAXn werden Datengrenzen festgelegt, unabhängig davon, ob die als Endpunkte genannten Daten auch wirklich vorkommen.

Kapitel 3

Die Programme: Aufbau und Ablauf

Eine wichtige Zielsetzung beim Entwurf dieses Programmpaketes war die Möglichkeit externe Programme einzubinden, wie z.B. ds9 oder eines der FTOOL Programme, auf deren Programmierung kein Einfluss genommen werden kann und soll. Um dieses Ziel zu erreichen, habe ich das Softwarepaket in zwei grosse Modulbereiche aufgeteilt.
Die einzelnen Modulbereiche und ihr Zusammenspiel sind dabei so konzipiert, dass jeder Einzelbaustein leicht austauschbar ist. Dies ist besonders dann wichtig, wenn das gesamte Programm für einen neuen Detektor eingesetzt werden soll oder neue Anforderungen an die Analyse einen Umbau der Pipeline notwendig machen.
Die auffälligste Unterteilung des Softwarepaketes liegt in der Unterscheidung zwischen mehreren Arbeits- und Steuermodulen und der graphischen Benutzeroberfläche (GUI). In diesem Kapitel soll im Detail auf die Unterteilung der Software eingegangen werden.

3.1 Die Arbeitsmodule: Aufbau und Ablauf

Die Arbeitsmodule, die im Laufe dieser Arbeit geschrieben wurden, sind (im Unterschied zu den externen Programmen) fast alle nach dem gleichen Prinzip aufgebaut[1]. Sie führen nur kleine, klar definierte Aufgaben durch. Die Daten, die von diesen Programmen gelesen oder geschrieben werden, sind außerdem immer im FITS-Format gehalten. Diese beiden Eigenschaften ermöglichen den leichten Austausch von Arbeitsmodulen bei sich ändernden Anforderungen.
Die Aufgaben der Arbeitsmodule sind:

- Umwandeln der Detektordaten in das FITS-Format: diese Aufgabe ist durch

[1] Ich habe für die Entwicklung der Arbeitsmodule die Programmiersprache C benutzt. Dies bringt in erster Linie Geschwindigkeitsvorteile. Obwohl C++ momentan sicher als die modernere Sprache gilt, da zur Zeit eher objektorientiert programmiert wird, war sie für mich nicht die bessere Alternative. Erstens muss man sagen, dass durch den erzwungenen Objektcharakter ein oft unnötiger und verlangsamender Overhead ersteht und zweitens bieten sich die Vorteile des objektorientierten Programmierens, nämlich die Verbindung von Daten und Funktionen, mit jeder grundlegenden Sprache, ob sie nun zur Objektorientierung zwingt, oder nicht.

`FPEventFeeder` realisiert, wobei der Detektor von diesem Programm noch durch eine Event-Liste simuliert wird.
- Datenreduktion: durch `FPOmap` wird einer von vielen möglichen Reduktionsschritten – das Anwenden einer Offsetmap – durchgeführt.
- Datenextraktion: die Arbeitsmodule `FPLightcurve`, `FPSpectrum` und `FPIntensity` erstellen aus Eventlisten Lichtkurven, Spektren und Intensitätsbilder.
- Darstellung der extrahierten Daten: diese Aufgabe wird von `FPPlot` ausgeführt.
- Pipelinespezifische Aufgaben werden von `FPCopyControl` durchgeführt.

Auf jedes dieser Arbeitsmodule wird später einzeln eingegangen.
Im folgenden fasse ich nun zunächst die grundlegenden Design-Anforderungen an diese Arbeitsmodule zusammen:

1. Die Programme sollten eigenständig arbeiten, sofern sie nicht ausschließlich in einer Pipeline sinnvoll sind. Sie sollten sich also im Einzelbetrieb wie ein FTOOL benutzen lassen.

2. Die Arbeitsmodule sollten für Echtzeitanwendungen optimiert sein. Das heißt, sie sollten schnell arbeiten, Lese- und Schreibzugriffe auf die Festplatte vermeiden, soweit es sinnvoll ist.

3. Ein Servermodus sollte vorhanden sein. Dies ist sinnvoll, um die Startup-Zeit, die jedes Programm benötigt, nur einmal im Laufe der Pipeline zu verlieren.

4. Wie der Servermodus bereits impliziert, sollten die Programme eine Form der Interprozesskommunikation (IPC) unterstützen. Im Fall dieser Programme ist dies in Form von XPAAccessPoints geschehen, über die beispielsweise auch ds9 und fv verfügen.

5. Die verwendeten Formate sollten sich, soweit möglich und dem Rahmen dieser Arbeit angemessen, an die in der Astronomie üblichen Standards halten. Sie sollten sich also an den Vorgaben der OGIP oder anderer, weit verbreiteter Formate orientieren.

3.1.1 Die externen Programme

Das Einbinden externer Programme bietet mehrere Vorteile:
- Teile der gewünschten Pipeline müssen nicht selbst geschrieben werden, dies kann erhebliche Entwicklungszeit sparen.
- Die externen Programme werden von externen Teams gewartet und von einer noch größeren Gruppe von Wissenschaftlern getestet. Dies reduziert einerseits die Fehleranfälligkeit der Pipeline und natürlich auch wieder die aufgewendete Zeit, die nötig ist, um die Pipeline auf Fehler hin zu testen.
- Wenn zusätzliche Funktionen in die Pipeline eingebunden werden sollen, die bereits von einem externen Programm erfüllt werden, so kann dies ohne Änderung des Quellcodes

der Pipeline geschehen, was wieder Entwicklungs- und Testzeit spart.

Die Verwendung externer Programme bringt aber auch Nachteile mit sich, oder ist nicht ohne weiters möglich.
Der wichtigste Punkt, auf den geachtet werden muss, ist das Datenformat, das von diesen Programmen unterstützt wird. Da alle Programme der Pipeline auf FITS basieren sollten, wird dadurch bereits eine Auswahl getroffen.
Zusätzlich sollte auf die Rechenzeit geachtet werden, die die Programme benötigen, um ihre Aufgaben durchzuführen. Insbesondere sollte bekannt sein, ob die Programme neben den FITS-Dateien, in denen die Daten stehen, noch weiter Dateien lesen müssen, da dies die Geschwindigkeit stark reduziert. Besonders geeignet sind dagegen Programme, bei deren Entwicklung bereits an die Arbeit in einer Pipeline gedacht wurde. Diese Programme verfügen oft über einen Server-Modus, bei dem sie nur einmal gestartet werden, und danach auf einen externen Befehl hin arbeiten. Dies kann erhebliche Zeiteinsparungen bedeuten.
Ausserdem verfügen diese Programme meist über die Unterstützung einer Form der IPC, um eben diese Befehle zu erhalten und am besten auch eine Rückmeldung über den Verlauf dieser Arbeit an das sendende Programm (den Client) zu schicken.

3.1.1.1 SAOImage ds9

Ein externes Programm, das einen Großteil dieser Anforderungen erfüllt, ist ds9 [37]. Einige der Eigenschaften, die für die Verwendung von ds9 in einer Pipeline sprechen, werde ich nun beschreiben:
Die erste Eigenschaft ist die Geschwindigkeit beim Öffnen von (großen) Bildern.
Eine zweite ist, dass ds9 entwickelt wurde, um in einer Pipeline zu laufen, denn es lässt sich als Server betreiben, der über XPA Kommandos von externen Programmen, von Skripten oder über die Kommandozeile entgegen nehmen kann, und damit eine Form der IPC unterstützt, die auch für diese Software benutzt wurde. Genau genommen wurde die Interprozesskommunikation XPA für ds9 entwickelt, ist aber mittlerweile ein eigenständiges Projekt. Dementsprechend läßt sich ds9 auch über eine sehr große Anzahl an XPA Kommandos steuern. Diese sollen und können hier nicht vollständig aufgelistet werden. Drei wichtige möchte ich aber nennen:

plot ist der Befehl, mit dem sich ein xy-Plot Fenster öffnen lässt. Mit einer Reihe weiterer Kommandos kann dieser Plot dann erstellt, die Skalierung, Farben und Symbole können geändert werden. Das Problem (in Bezug auf die hier erstellte Software) ist aber, dass für diesen xy-Plot die Daten nicht in Form einer FITS-Datei vorliegen dürfen, sondern im ASCII-Format gespeichert sein müssen. Dies ist äußert bedauerlich, da ich gerne alle Plotprogramme aus einer Hand genommen hätte und nicht ein eigenes Plotprogramm, das FITS lesen kann, schreiben wollte.

shm öffnet eine Datei aus dem Shared Memory. Dies ist eine sehr wichtige Eigenschaft für das Softwarepaket, da es die Geschwindigkeit gegenüber dem Lesen und Schreiben auf die Festplatte erhöht. Leider ist die Art der Speicherung der FITS-Dateien nicht direkt

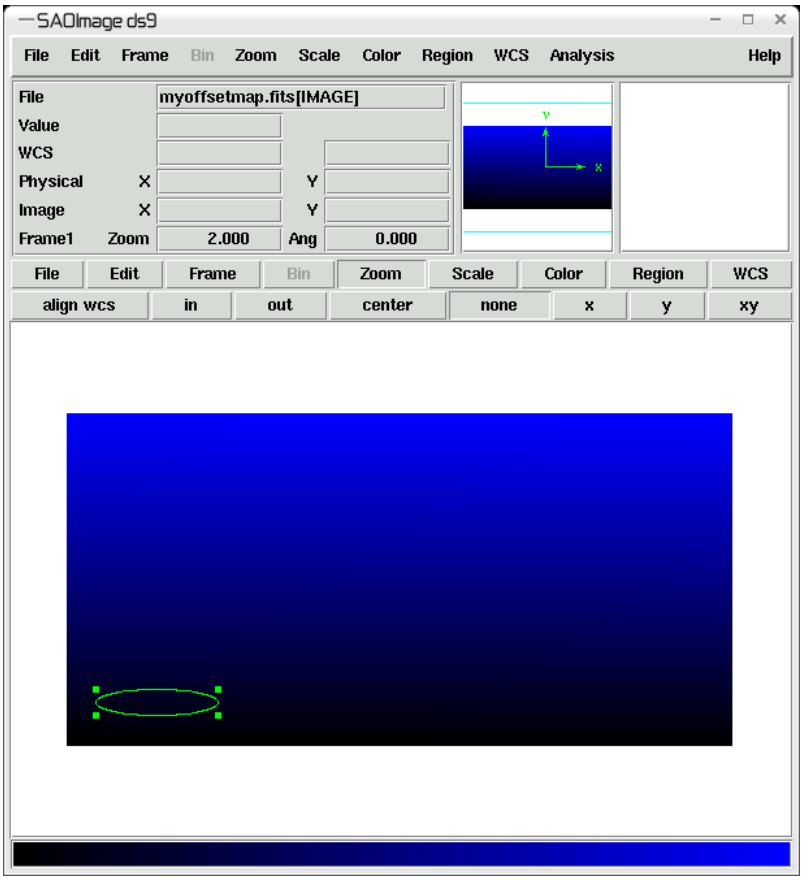

Abbildung 3.1: Das ds9 Fenster mit einem CCD Intensitätsbild

kompatibel mit der von CFITSIO implementierten Art. Um dies zu ändern, haben Jörn Wilms und ich ein Patch für CFITSIO und ds9 geschrieben, das es ds9 erlaubt (mit Hilfe einiger CFITSIO-Routinen) Dateien, die mit CFITSIO in den Shared Memory geschrieben wurden, anhand der Namensendung (siehe Abschnitt 2.1.4) zu öffnen. Dieses Patch ist noch nicht in einer offiziellen Version von ds9 oder CFITSIO veröffentlicht[2], aber die in diesem Softwarepaket verwendete Version von ds9 ist von mir bereits entsprechend verändert worden.

region ist ein weiteres XPA Kommando, das in diesem Softwarepaket sinnvoll eingesetzt werden kann. Mit diesem Kommando kann ein Region-File erstellt werden und es kann geladen, gespeichert, verändert werden und vieles mehr. Zusammen mit der in CFITSIO eingebauten ‘regfilter’-Methode (siehe: 2.1.4) lassen sich so interaktiv die in ds9 eingezeichneten Regionen in den Arbeitsmodulen als Filter für die Tabellendaten der

[2]Wir gehen aber davon aus, dass dies in absehbarer Zeit geschehen wird. Wir stehen in Kontakt mit den Programmierern von ds9, die auf diese Problematik von uns hingewiesen wurden.

Eventliste nutzen, um beispielsweise nur Photonen aus bestimmten Regionen des CCDs zu verarbeiten.

Die hier beschriebenen Funktionen lassen sich bisher nicht realisieren. Während meiner Arbeit gab es immer wieder Probleme bei der Einbindung von ds9, auf die ich kurz eingehen möchte.

3.1.1.2 Probleme mit ds9

Wie sich herausstellte, ist die Auswahl an Programmen, die xy-Plots in Echtzeit in einer FITS basierten Pipeline durchführen können gering.
Die Option des FTOOLS `fplot` konnte aufgrund der Geschwindigkeit nicht eingebaut werden. Die zweite, eigentlich ideale, Option, die Plotmöglichkeit von `ds9` zu nutzen, scheiterte am Datenformat. Ds9 akzeptiert für xy-Plots nur Daten im ASCII-Format. Auch bei der Darstellung von Bildern war meine erste Wahl ds9.
Die andere Option – `fv/POW` – schied aus, da fv weder über die 'extended file name syntax' von CFITSIO, noch über die Möglichkeit, Dateien aus dem Shared Memory zu lesen, verfügt.
Wie sich jedoch nach einiger Zeit der Suche zeigte, unterstützt ds9 zwar Shared Memory, aber weder über den CFITSIO Namen, noch über die ID der Datei in Shared Memory lassen sich Dateien die von CFITSIO dorthin geschrieben wurden von ds9 öffnen.
Nach regem email Austausch zwischen Jörn Wilms, Bill Joye (ds9), Bill Pence (CFITSIO)und mir stellte sich heraus, dass das Shared Memory Modell von CFITSIO nicht nur zusätzliche Shared Memory Strukturen anlegt, sondern, dass auch die Struktur, in der die Datei geschrieben wird, noch zusätzliche Daten enthält. Da weder die ds9- noch die CFITSIO-Entwickler spontan bereit waren, dies zu ändern, schrieben Jörn Wilms und ich einen Patch, der dies beseitigen sollte. Dieser Patch wurde auf seine Funktionstüchtigkeit von mir geprüft und ich konnte keine Fehler entdecken. Der einzige Nachteil, der sich für mich ergab, war die Tatsache, dass ds9 nun diese Datei sperrte. Da es in ds9 keine Funktion gibt, die die Sperre löst, konnte ich lange nicht testen, wie sich ds9 in der Pipeline verhalten würde, wenn zum Darstellen des Bildes nicht ein Bild mit anderer Adresse, also sozusagen ein 'Pausenbild', dazwischengeschaltet wird.
Mit der inzwischen neueren Version von ds9 änderte sich das. Zwar musste der Patch an den neuen Code von ds9 angepasst werden, aber nun sperrt ds9 das Bild nicht mehr, so dass es von anderen Programmen wieder erneuert werden kann. Aus diesem Grund konnte ich vor kurzem erstmals das Verhalten von ds9 in dieser Pipeline untersuchen – wobei sich wieder zwei neue Probleme auftaten:

- Erstens scheint das von uns veränderte ds9 bei erneutem Öffnen eines Bildes im Shared Memory Speicher neu zu reservieren, den es nicht mehr frei gibt (was 'memory leak' genannt wird). Dies führt zu einem stetigen Anstieg des Arbeitsspeicherbedarfs, der dann auch schnell komplett belegt ist, und die Pipeline zum Erliegen bringt.

- Das zweite Problem sind zwei Eigenschaften, die mir zuvor nicht auffallen konnten, da ich immer ein 'Zwischenbild' mit anderer Adresse öffnen musste: wenn ds9 erneut die

gleiche Shared Memory Adresse wie zuvor öffnet, bleiben zwar die Farbtafelwahl und der Zoomfaktor erhalten, die Position des Zooms aber springt wieder auf den Mittelpunkt des neuen Bildes zurück. Ausserdem werden eingetragene Regionen gelöscht, die ich für die Pipeline nutzen wollte, und von denen ich annahm, sie interaktiv über XPA Befehle (die es gibt, da habe ich mich nicht getäuscht!) schreiben und lesen zu können. Somit bleiben sie nicht lange genug erhalten, um sie zu speichern. Auch dieses Problem sollte sich durch einfache Änderungen in ds9 lösen lassen.

Aufgrund dieser Probleme ist eine Nutzung von ds9 in der Pipeline zumindest zum jetzigen Zeitpunkt noch nicht möglich und deshalb taucht ds9 in der Beispielpipeline nicht auf.
Die Probleme hinsichtlich der Integration von ds9 waren mir hier einen eigenen Abschnitt wert, da ich es persönlich besonders schade finde, kein darstellendes Programm am Ende des Softwarepaketes zur Verfügung zu haben: Leser, die vielleicht den Programmierschritten im einzelnen nicht ganz folgen können oder wollen, mögen zu recht eine gute Darstellung der Ergebnisse des Programms erwarten. Den Anspruch habe ich selbst aber auch, da ich den Sinn und Zweck dieser neuen Art von Software nie aus den Augen verlieren wollte: sie soll schließlich vor allem dazu entstehen, um im Labor sinnvoll verwendet werden zu können. Als sich zum ersten Mal Probleme mit ds9 herauskristallisierten, habe ich deshalb, wie schon erwähnt, versucht, Kontakt mit den Entwicklern aufzunehmen (was mich viel Zeit gekostet hat) und mich schließlich hingesetzt, um mit der Hilfe von Jörn Wilms selbst eine Lösung der Schwierigkeiten zu finden. Wenn aber immer wieder neue zusätzliche Probleme auftauchen, dann kann innerhalb einer zeitlich limitierten Arbeit, wie es diese Diplomarbeit ist, nicht alles gelöst werden (ein Trost ist immerhin, dass die Schwierigkeiten in diesem Fall nicht ‘hausgemacht’ sind). Ds9 ist ein international in der Astrophysik und auch von mir sehr geschätztes Programm. Trotzdem scheint es wohl Ecken im Programm zu geben, die noch nicht ganz ausgeleuchtet wurden (die Tatsache, dass Programme niemals fertig werden gilt also anscheinend auch für international weit verbreitete und hoch perfektionierte). Die Probleme mit CFITSIO ergaben sich wohl daraus, dass ich womöglich der erste war, der versucht hat, diese Bibliothek im Zusammenspiel mit ds9 einzusetzten. Wenn ich Bill Pence (CFITSIO) in unserer email Korespondenz richtig verstanden habe, dann war der Shared Memory Driver von CFITSIO genau dafür programmiert worden, wofür ich ihn einsetzten wollte, nur war ich eben der erste, der das auch tatsächlich versucht hat (zumindest mit ds9) und die Problematik war deshalb vorher noch niemandem aufgefallen. Es wäre daher verwunderlich, wenn die Entwickler von ds9 und CFITSIO nicht den Ehrgeiz hätten, Lösungen zu finden.

3.1.1.3 FTOOLS

Die FTOOLS schienen zu Beginn der Diplomarbeit gute Kandidaten für externe Programme, die in diesem Softwarepaket genutzt werden könnten. Da aber sicher war, dass einige Programme auf jeden Fall von mir selbst geschrieben werden sollten, um Prototypen für mögliche Arbeitsmodule, die in dieser Art von Pipeline besonders gut laufen können, zu erstellen, habe ich in der Beispiel-Pipeline keine FTOOLS – bis auf `fdelete` in den Steuermodulen – eingebaut.

Ein anderer Grund ist, dass die meisten FTOOLS nicht so programmiert sind, dass sie besonders schnell arbeiten.
Bei einem FTOOL wäre ich allerdings froh gewesen, wenn es nutzbar gewesen wäre. Gemeint ist fplot. Leider ist es aber deshalb nicht so gut für den Betrieb in einer Echtzeitpipeline geeignet, weil es zusätzliche Datei-Leseoperationen oder aber interaktive Benutzerkommandos über die Kommandozeile benötigt, um die gewünschten Darstellungen zu erhalten.
Ein weiteres FTOOL, das tatsächlich in der Pipeline benutzt wird, ist |fdelete|. Es hat zwar lediglich die Funktion, eine FITS-Datei zu löschen, ist aber trotzdem sinnvoll, weil es die 'extended file name syntax' der CFITSIO-Bibliothek unterstützt. So kann hier über den (CFITSIO) Namen einer Shared Memory Datei diese gelöscht werden. Eine eigene Löschmethode würde dagegen einen langen Programmcode erforden. fdelete wird somit zwar nicht in der Pipeline selber, dafür aber in der Pipelinesteuerung verwendet.
Prinzipiell ist das Softwarepaket aber im Hinblick auf die Integrierbarkeit von FTOOL-ähnlichen Programmen geschrieben, und jedes FTOOL kann prinzipiell jederzeit integriert werden. Dazu ist lediglich ein Eintrag in die Definitionsdatei der Pipeline notwendig, mehr dazu unter 3.2.1.

3.1.2 Allgemeines zum Aufbau der Arbeitsmodule

Alle Arbeitsmodule, die im Rahmen dieser Diplomarbeit erstellt wurden, lassen sich grundsätzlich auf zwei verschiedene Weisen starten (Ausnahme: FPEventFeeder).
Eine Möglichkeit ist der sogenannte *Single-Mode*. In diesem Modus wird das Programm einmal in der durch die übergebenen Parameter vorgegebenen Weise ausgeführt, d.h es liest die Daten aus einer FITS-Datei, bearbeitet diese und schreibt sie wieder in eine FITS-Datei oder stellt die Ergebnisse dar.
Bei der zweiten Möglichkeit werden die Arbeitsmodule im *Server-Mode* gestartet. In diesem Fall startet das Programm und trägt sich als XPA-Server ein [38]. Über eine Reihe von XPAAccess-Points kann der Arbeitsschritt dieses Programms dann von der Kommandozeile aus oder durch XPA-Clients z.B. in einem Skript beliebig oft ausgeführt werden, dazwischen können Parameter verändert und das Programm beendet werden. Dieser Modus eignet sich besonders, um in Echtzeitanwendungen verwendet zu werden: die Zeit, die das Programm normalerweise benötigt, um zu starten, d.h. sich in die Programmliste einzutragen, Platz für Variablen zu reservieren, aber vor allem natürlich die Zeit, die es benötigt, um die mitgegebenen Parameter auf Korrektheit zu testen und das Vorhandensein aller angegebenen Dateien usw. zu überprüfen, wird nur einmal gebraucht.
Ich werde zuerst zwei weitere verwendete Bibliotheken beschreiben, die neben der bereits vorgestellen (CFITSIO) in meiner Software verwendet wurden (PIL und XPA). Die Beschreibung der verwendeten Bibliotheken ist deshalb sinnvoll, weil diese merklich den Aufbau der Arbeitsmodule beeinflussen und somit grossen Einfluss auf die Geschwindigkeit und Benutzerfreundlichkeit haben. Die Auswahl dieser Bibliotheken war, ähnlich wie die Auswahl der verwendeten Sprachen und die der eingebundenen externen Programme, ein wichtiger Baustein beim Entwurf und beim Erstellen des gesamten Softwarepaketes, selbst wenn der Benutzer letztendlich wenig davon bemerken sollte. Danach werde ich auf den generellen Aufbau aller Arbeitsmodule eingehen, da die Arbeitsmodule über viele Eigenschaften verfügen, die allen gemeinsam sind.

Im Anschluss daran werde ich die einzelnen Programme im Detail betrachten, soweit dies noch nicht durch die vorherigen Beschreibungen abgedeckt ist.

3.1.3 Die verwendete Bibliothek zur Interprozesskommunikation: XPA

Der Name XPA stammt von *X Public Access*, entspricht aber nicht mehr wirklich dem, was diese Bibliothek heute leistet, da nicht nur X11 Programme miteinander kommunizieren können, wie es in den ersten Versionen der Fall war. Inzwischen stellt XPA eine vielschichte Oberfläche zur Kommunikation zwischen verschiedensten UNIX-Programmen dar. Vielschichtig bezieht sich hierbei auf die verschiedenen Arten, auf die XPA benutzt werden kann:

- XPA stellt eine Bibliothek für Server-Programme zur Verfügung, die es diesen ermöglicht, öffentlich zugängliche, benannte Accesspoints zu registrieren.
- XPA stellt eine Bibliothek für Client-Programme zu Verfügung, über welche diese auf Programmierebene mit registrierten XPA-Servern kommunizieren und Daten austauschen können.
- XPA stellt selbstständige Programme, `xpaget` und `xpaset`, zur Verfügung, die diese Bibliotheken benutzen und es dem Benutzer ermöglichen, über die Kommandozeile mit XPA-Servern zu kommunizieren.
- Mit `xpans` steht ein XPA-Nameserver zur Verfügung, über den die XPAAccessPoints registriert und für XPA Client-Programme verfügbar gemacht werden.

3.1.3.1 Die Kommunikationsmethoden von XPA

Die XPA-Bibliothek ist auf Sockets aufgebaut. Dabei werden drei Arten unterstützt: *inet*, *localhost* und *unix*. Welche dieser Methoden benutzt wird, lässt sich über die Umgebungsvariable XPA_METHOD steuern, die auf eben einen dieser Werte gesetzt werden kann. Standardmäßig ist *inet* eingestellt, was ermöglicht, über das Internet mit XPA-Servern auf entfernten Rechnern zu kommunizieren. Die in meinem Softwarepaket bevorzugte Einstellung ist allerdings *unix* (oder *local*), da dies auf einem einzelnen Host und unter Unix am schnellsten ist. Dies liegt daran, dass ‘local sockets’ auf dem Dateisystem beruhen, also über file handler erreichbar sind, und nicht auf die IP-Adressen von DNS-Name-Servern angewiesen sind ([38] vgl. auch [39]). Bei der Einstellung ‘XPA_METHOD = localhost’ wird nicht unbedingt ein DNS-Server benutzt, da dies von der Konfiguration abhängt. Diese Einstellung bietet außerdem den Vorteil, sowohl unter UNIX, als auch unter Windows möglich zu sein. Trotzdem empfehlen die Entwickler von XPA eine andere Einstellung zu benutzen (unix), wenn unter UNIX und nur auf einem Host gearbeitet wird. Dies hat sich auch in den Messungen bestätigt, die ich von den Programmen `stest.c` und `ctest.c` gemacht habe, die den XPA Distributionen beiliegen. Sie stellen nicht die reinen Kommunikationszeiten dar, da noch einige interne Abfragen und Ausgaben stattfinden, können aber als Vergleichswerte für die verschiedenen Socket Methoden benutzt werden:

Tabelle 3.1: Vergleich der XPA Kommunikationsmethoden

Zeiten	inet	localhost	unix (= local)
real [s]	5.48	5.55	1.72
user [s]	0.29	0.29	0.15
system [s]	2.12	2.17	0.29

Es wurden je 1000 Kommunikationen zwischen Server und Client durchgeführt und die entsprechenden Zeiten gemessen. Die angegebenen Werte sind jeweils der Durchschnittswert 10 solcher Messungen.
Normalerweise kommunizieren die XPA Programme direkt miteinander. Es ist aber noch eine weitere Möglichkeit in XPA eingebaut, die einen 'Message Bus' emuliert. Hierbei wird das Programm `xpamb` vom Client-Programm angesprochen, dieses leitet dann die Anfragen an die entsprechenden XPA-Server weiter, nimmt die zurückkommenden Daten in Empfang und übergibt diese dann wieder an das Client-Programm. Dieser Zwischenschritt ist eingebaut, da bei einer großen Anzahl an zu kontaktierenden Servern die Einzelkommunikation für den Client langsamer ist, als die Daten an ein Zwischenprogramm zu senden, das dann die weitere Verteilung übernimmt. In den Pipelines, wie sie in diesem Softwarepaket eingeplant sind, ist diese Art der Kommunikation nicht der bevorzugte Weg, weshalb ich diese Option auch nicht weiter ausführen werde.

3.1.3.2 Der XPA-Server

In Abbildung 3.2 ist der prinzipielle Aufbau eines XPA-Servers zu sehen.
Die Funktionen `SendCallBack` und `ReceiveCallBack` werden ausgeführt, wenn ein Client-Programm über einen XPAAccessPoint, oder genauer gesagt über ein XPACommand eines XPAAccessPoints (von denen es viele geben kann), entweder mit `xpaget` (SendCallBack) oder xpaset (ReceiveCallBack) zugreift[3]. Diese Callbacks haben eine festgelegte Parameterstruktur und können sowohl programminterne Daten als auch vom Client-Programm stammende Daten, beide durch Zeiger übergeben, verarbeiten.
Jeder XPA-Server legt XPAAccessPoints fest. Diese Punkte haben einen Namen, über den sie erreichbar sind. Ein solcher Name folgt der Struktur `class:name`, beide kleiner als 1024 Zeichen. Um einen XPAAccessPoint zu erreichen, muss dieser Name angeben werden, wobei mehr als ein XPAAccessPoint benachrichtigt wird, wenn mehrere auf das angegebene Template zutreffen. Die XPA Routinen akzeptieren bei der Namensangabe folgende Formen:

[3]Ich unterscheide hier nicht zwischen den ausführbaren Programmen `xpaget` und der Routine `xpaget(...)` der XPA-Bibliothek, da dies für den Server keine Rolle spielt.

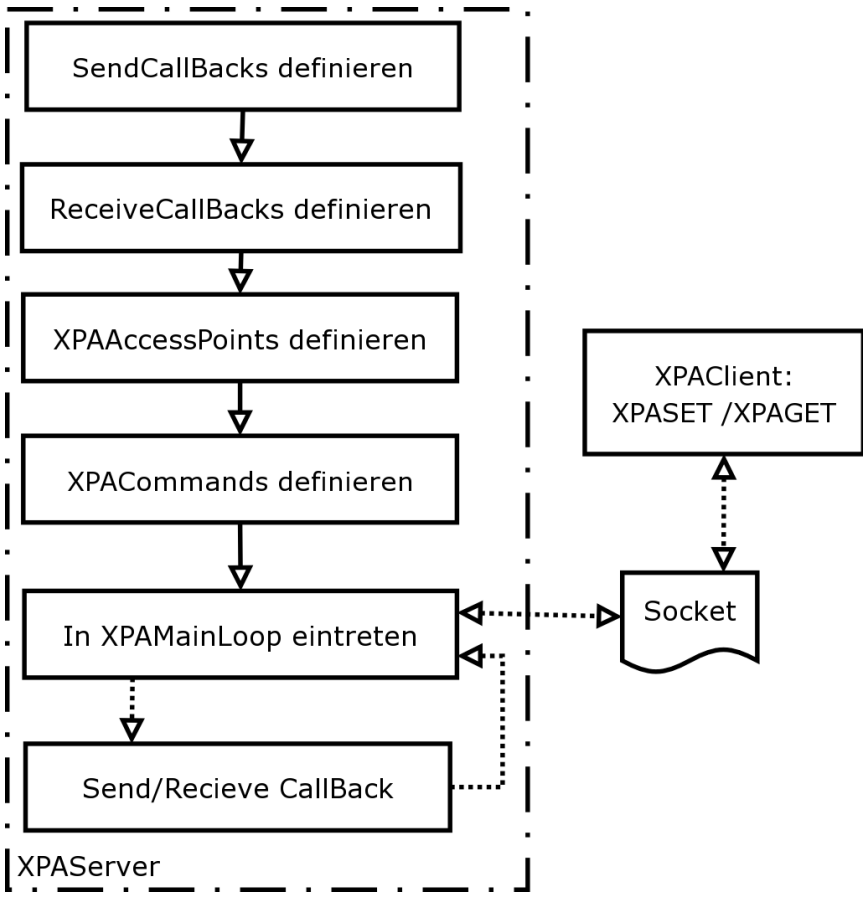

Abbildung 3.2: Prinzipieller Aufbau eines XPAServers, der die Standardabfrage des Sockets über die XPAMainLoop vornimmt.

Tabelle 3.2: XPAAccesspoint Templates

Template	Erklärung
`class:name`	Genaue Übereinstimmung von ‘`class`’ und ‘`name`’
`name`	Alle ‘`class`’, die mit ‘`name`’ übereinstimmen
`*:name`	Alle ‘`class`’, die mit ‘`name`’ übereinstimmen
`class:*`	Alle ‘`name`’, die mit ‘`class`’ übereinstimmen
`*:*`	Alle

Bei diesen Templates können außer *, was mindestens null Vorkommen bedeutet, auch die wild cards ‘?’ (= mindestens ein Vorkommen) und ‘[. . .]’ (= ein inklusive set) verwendet werden. Die Arbeitsmodule haben einen Parameter, über den sich der XPA-Server ‘`name`’ festlegen läßt, während der ‘`class`’ Teil des Servernamens immer dem Programmnamen entspricht. Dies ist

dann sinnvoll, wenn beispielsweise das xy-Plotprogramm FPPlot einmal ein Spektrum und einmal eine Lichtkurve darstellen soll. In diesem Fall kann das Programm zweimal, jeweils unter einem anderen XPA-Namen, gestartet und damit auch über diesen Namen kontaktiert werden.

3.1.3.3 Die XPAMainLoop

Alle Arbeitsmodule benutzen die Standard-XPAMainLoop, wie sie in der XPA Distribution vorhanden ist. Dies ist deshalb sinnvoll, weil diese Programme keine anderen Resourcen im Auge behalten müssen, als die über XPA definierten Sockets.
Die XPAMainLoop ruft (abgesehen von Fehlerabfragen) nur die `select()` Funktion auf, und zwar ohne Angabe einer maximalen Wartezeit. Dies bedeutet für das Programm, dass es schläft bis der Kernel ein Ereigniss in den angegebenen Sockets meldet. Wird ein Programm auf diese Art geweckt, dann werden alle externen Abfragen nacheinander abgearbeitet, indem die zu diesem Accesspoint gehörenden Funktionen aufgerufen werden. Der Rückgabewert dieser Funktionen wird dem Client, der diese Anfrage gestellt hat, zurückgegeben.
Eine Alternative zur XPAMainLoop wäre ein Aufruf von XPAPoll, bei dem angegeben werden kann, wie lange maximal auf eine Anfrage gewartet werden soll (in ms). Dies wäre sinnvoll bei Programmen, die noch weitere Resourcen beobachten müssen, die nicht über einen `select()` Aufruf abgefragt werden können. In den Arbeitsmodulen dieser Diplomarbeit war die XPAMainLoop aber die schnellere und damit bessere Wahl.

3.1.3.4 Besteht die Gefahr von XPA Race Conditions?

Race Conditions stellen bei miteinander kommunizierenden Programmen immer eine Gefahr dar. Von einer Race Condition Konstellationen spricht man, wenn das Ergebnis einer zusammengesetzten Operation vom zeitlichen Verhalten bestimmter Einzeloperationen abhängt. Solche Konstellationen sind im Allgemeinen schwer zu rekonstruieren. Im Fall der Kommunikation zwischen mehreren Programmen ist die kritische Operationen vor allem das Schicken von Anfragen mit anschließendem Warten auf eine Antwort. Geht eine solche Race Condition schlecht aus, dann entsteht ein sogenanntes Deadlock (auch Verklemmung), bei der alle beteiligten Prozesse zum Stillstand kommen.
Bei den Arbeitsmodulen dieser Diplomarbeit sind keine auf XPA zurückzuführenden Race Conditions möglich. Die bisher einzige bekannte Möglichkeit einer Race Condition durch XPA ist, wenn zwei XPA-Server, die auch gleichzeitig als XPA-Client fungieren, sich gegenseitig Anfragen schicken und auf die Antwort des anderen Servers warten. Dazu müssen beide Server die Abfrage über einen XPAPoll Aufruf mit Zeitlimit machen, und zwischen zwei Abfragen beide die Anfrage an den anderen Server senden.
Die Arbeitsmodule benutzen aber weder XPAPoll, noch fungieren sie als ein XPA-Client, der anderen Arbeitsmodulen Anfragen schickt. Es besteht von dieser Seite also kein Deadlock Risiko.

3.1.3.5 Die XPAAccessPoints der Arbeitsmodule

Für die Pipeline stellen die Accesspoints der Arbeitsmodule die wichtigste Möglichkeit dar, um Benutzern Kommandos zur Verfügung zu stellen, über welche das Verhalten der Pipeline beeinflusst werden kann. Die Standard-Accesspoints, über die alle Arbeitsmodule verfügen (siehe Abschnitt 3.2.2) sind PROCESS_DATA, EXIT und bei den meisten Programmen RESET und INFILE, mit welchem die einzulesende Datei geändert werden kann. Dies ist ein mächtiger Accesspoint, da alle Arbeitsmodule die 'extended file name syntax' der CFITSIO Bibliothek unterstützen. Somit können auf diesem Weg z.B. Energieschwellen verändert, neue ds9 Region Dateien angegeben werden und vieles mehr.

3.1.4 Die verwendete Bibliothek zum Einlesen der Parameterdateien: PIL

Wird ein FTOOL ohne Parameterangaben gestartet, dann wird automatisch nach einer Reihe von Parametern gefragt, und andere Parameter, die angeben werden könnten, werden auf einen Wert gesetzt. Wird dieses Programm erneut gestartet, so werden manche der Parameter automatisch auf den Wert gesetzt, der beim letzten Starten angegeben worden war, ohne dass der Benutzer gefragt wird. Dieses Verhalten wird gesteuert über sogenannte Parameterdateien. FTOOLS (aber beispielsweise auch die meisten der am Integral Science Data Center (ISDC) geschriebenen ausführbaren Programme) verfügen über solch ein Parameterfile. Alle Arbeitsmodule meines Softwarepaketes tun dies auch. Die Bibliothek, die zum Auslesen und zur Interaktion mit dem Benutzer genutzt wurde, ist die PIL (Parameter Interface Library) die am INTEGRAL Science Data Center geschrieben wurde [40]. Die PIL entstand, um ISDC Anwendungen den Umgang mit den IRAF Parameterdateien zu ermöglichen.
Die Parameterdateien enthalten reinen ASCII Text, wobei jede Zeile einem Parameter entspricht. Das Format jeder Zeile ist: 'Name,Typ,Modus,Standardwert,Min,Max,Prompt'. Die Bedeutung dieser Einträge ist folgende:

Name – der Parameter-Name, der frei gewählt werden kann

Typ – einer der folgenden Einträge

- b für einen boolschen Wert (Boolian)
- i für einen Ganzzahlwert (Integer)
- r für eine Fließkommazahl (Real)
- s für eine Zeichenkette (String)
- f für einen Dateinamen (File)

Modus – ist eine Kombination aus den Buchstaben a(auto), h(hide), l(lern), q(query) und bestimmt das Verhalten bei Abfrage dieses Parameters. 'ql' heißt beispielsweise, dass der Benutzer gefragt werden soll und dass der eingegebene Wert gelernt, d.h. in die Parameterdatei als neuer Standardwert geschrieben werden soll.

Standardwert – wird dem Benutzer als Wert angeboten, oder wenn nicht gefragt werden soll, sofort als Wert des Parameters akzeptiert.

Min – legt das Minimum für einen (Zahlen)Wert fest, wenn auch Max angegeben ist. Min kann auch eine durch '|' getrennte Liste von erlaubten Werten enthalten.

Max – ist der maximale Wert, der aber nur abgefragt wird, wenn auch ein minimaler Wert angegeben ist.

Prompt – wird dem Benutzer als Hilfestellung ausgegeben, wenn eine Abfrage dieses Parameters gemacht wird.

Die Parameterdateien werden unter dem Namen des Programms mit der Endung '.par' und an der durch die Umgebungsvariablen PFILES angegebenen Stelle gespeichert und gelesen. Die PIL sucht bei undefinierter Umgebungsvariable, anders als die FTOOLS, SAOrd und XPI, im momentanen Arbeitsverzeichnis nach der Parameterdatei.
Weitere Besonderheiten der PIL gegenüber den IRAF Parameterdateien sind:

- Die Zeilenlänge muss kleiner als 2000 Zeichen sein (80 oder 255 bei IRAF)
- Ein Eintrag von ${*parameterx*} wird durch die Umgebungsvariable *parameterx* ersetzt
- PIL unterstützt auch Vektoreinträge für Integer und Fließkommazahlen

Wird ein Programm mit Parametern aufgerufen, dann können diese einfach in derselben Reihenfolge wie im Parameterfile angegeben werden. Die sicher bessere Form ist aber die, den Parameternamen, gefolgt von einem '=' und dem Parameterwert, anzugeben, wobei vor und nach dem '=' Leerzeichen vorkommen dürfen.
Die Arbeitsmodule nutzen nicht alle Möglichkeiten aus, die ihnen durch die PIL zur Verfügung stehen, beispielsweise werden nach dem Start, bei dem die Datei einmal gelesen und bearbeitet wird, keine Einträge mehr aktualisiert und es wird auch nicht nach neueren Einträgen gesucht. Alle Besonderheiten, die aber auch beim einmaligen Durcharbeiten der Parameterdatei zum Tragen kommen, stehen den Arbeitsmodulen zur Verfügung.

Die dritte und wichtigste von mir benutze Bibliothek CFITSIO wurde bereits im Kapitel über das Datenformat behandelt (siehe Abschnitt 2.1.4).

3.1.5 Genaueres zum Aufbau der Arbeitsmodule

Mit den bisher beschriebenen Bibliotheken (XPA, PIL und CFITSIO) ergibt sich für den generellen Aufbau der Arbeitsmodule das folgende Bild:

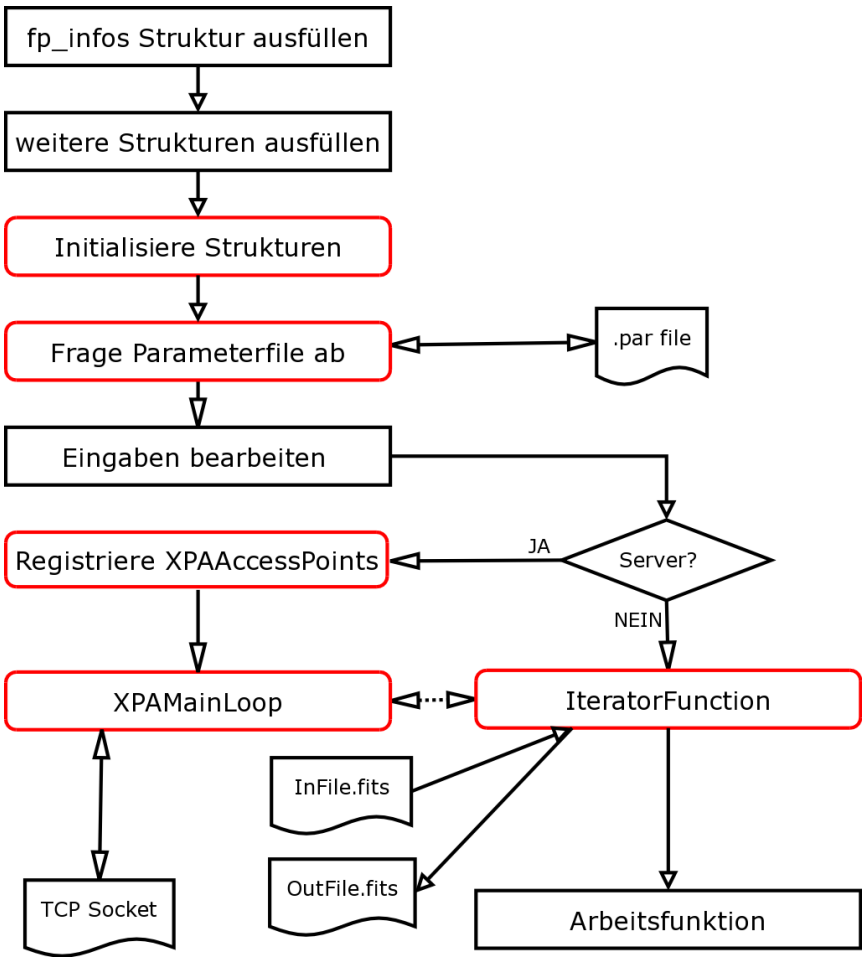

Abbildung 3.3: Genereller Aufbau der Arbeitsmodule

Das Bild zeigt den generellen Aufbau der Arbeitsmodule (Funktionen in gerundeten Rahmen sind komplette Funktionen, die entweder aus externen Bibliotheken kommen (XPAMainLoop) oder in meinem Softwarepaket geschrieben wurden. Diese Funktionen stellen eine Art von Template für Arbeitsmodule dar, das für neue Arbeitsmodule genutzt werden kann). Im Folgenden werde ich genauer auf die einzelnen Schritte eingehen, mit denen die Arbeitsmodule aufgebaut wurden.

3.1.5.1 Der Aufruf der Template-Funktion

Alle wichtigen Informationen der Arbeitsmodule sind in Strukturen enthalten, die den Objekten des objektorientierten Programmierens sehr ähnlich sind, da sie sowohl Daten enthalten, als auch Funktionen, die auf diesen Daten arbeiten.
Die Basis-Struktur (fast) aller Arbeitsmodule ist die `fp_infos`-Struktur. Sie enthält neben einigen Standardinformationen auch die arbeitsmodul-spezifischen Strukturen, die `xpa_infos`-Struktur (mit der die XPAAccessPoints von `fpTemplate` erstellt werden)

und auch die für die Iterator-Funktion der CFITSIO Bibliothek nötigen Informationen und Funktionen. Diese Funktionen (*workfunctions*) werden von der Iterator-Funktion aufgerufen. Der genaue Ablauf ist in Abbildung 3.4 und Abbildung 3.5 dargestellt. Der Programmcode

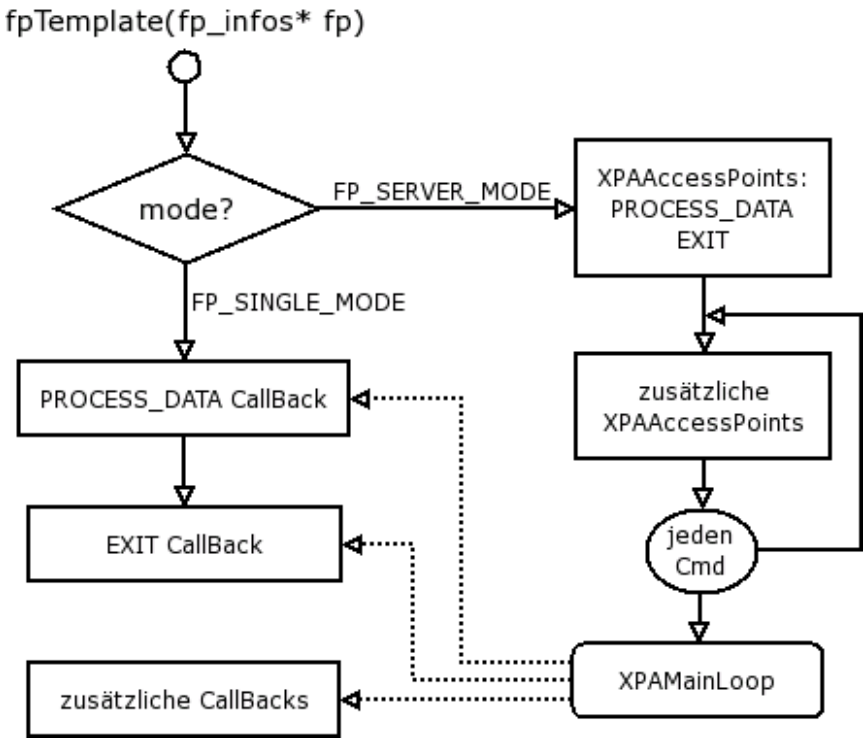

Abbildung 3.4: Der Ablauf des Aufrufes der Template-Funktion. Die `fp_infos` Struktur enthält alle Informationen, die für den Ablauf nötig sind.

der Arbeitsmodule endet mit dem Aufruf der Template-Funktion, da ab diesem Aufruf alle weiteren Funktionsaufrufe in der Template-Funktion selber enthalten sind oder über die in der `fp_struktur` eingetragenen Funktionspointer ausgeführt werden. Im Single-Modus führt fpTemplate nur den PROCESS_DATA- und danach den EXIT-Callback aus, ohne sich als XPA-Server zu registrieren.
Im Server-Modus werden zuerst die beiden Standard-Accesspoints registriert. Danach noch alle in der `xpa_infos` Struktur enthaltenen XPAAccessPoints. Diese Struktur wird wie alle anderen Information auch zu Beginn der Arbeitsmodule ausgefüllt.

3.1.5.2 Der PROCESS_DATA-Callback

Der für den Ablauf der Module wichtigste Callback ist PROCESS_DATA. Sein Ablauf wird in Abbildung 3.5 gezeigt. Die Struktur (Driver-Iterator-Driver) folgt der von Peter Wilson (einem

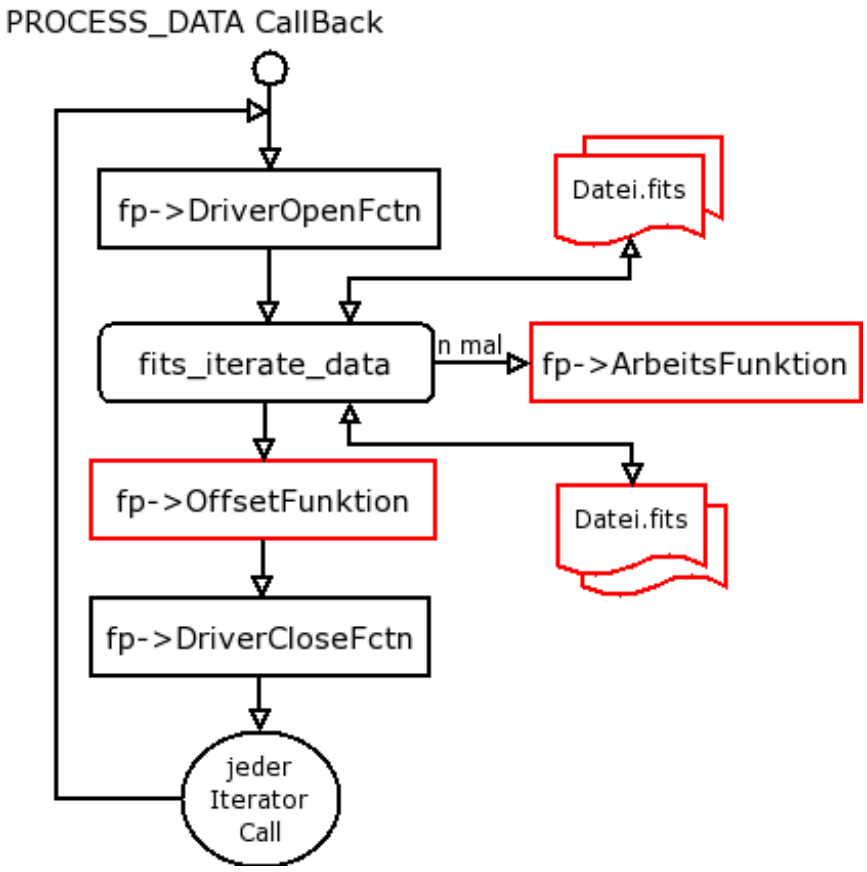

Abbildung 3.5: Ablauf des PROCESS_DATA-Callbacks

CFITSIO Mitentwickler) vorgeschlagenen Vorgehensweise. Diese forciert einen objektorientierteren und – was hier von Bedeutung ist – einen zudem auf Geschwindigkeit optimierten Ansatz zum Bearbeiten von FITS Dateien.
In meiner Software wird zuerst eine Treiber Funktion ausgeführt (`fp.DriverOpenFctn`), die die Dateien öffnet. Da diese Funktion, `fpDriverOpen` (Abbildung 3.6), einige Leseoperationen auf der FITS-Datei und Kopieroperationen ausführt, um die Informationen, die die Iterator-Funktion benötigt, zu erhalten, läuft sie nur beim ersten Öffnen der Datei ab. Sie ersetzt deshalb den Zeiger auf sich selbst in der `fp_infos` Struktur durch einen Zeiger auf die `fpDriverReopen` Funktion (Abbildung 3.7). Die Reopen-Funktion lässt diese zeitaufwendigen Schritte aus. Allerdings macht dies nur im Servermodus einen Unterschied, da ja im Single-Modus die Dateien nur ein einziges Mal geöffnet werden müssen. Im Servermodus aber müssen diese nach jedem Schreibvorgang geschlossen werden, damit die anderen Prozesse, die diese Datei lesen wollen, dies auch können.
Wenn die Datei, in die geschrieben werden soll, noch nicht existiert, dann wird die Dateierstellungsfunktion der `fp_struktur` (`fp.createFileFctn`) ausgeführt und im Anschluß

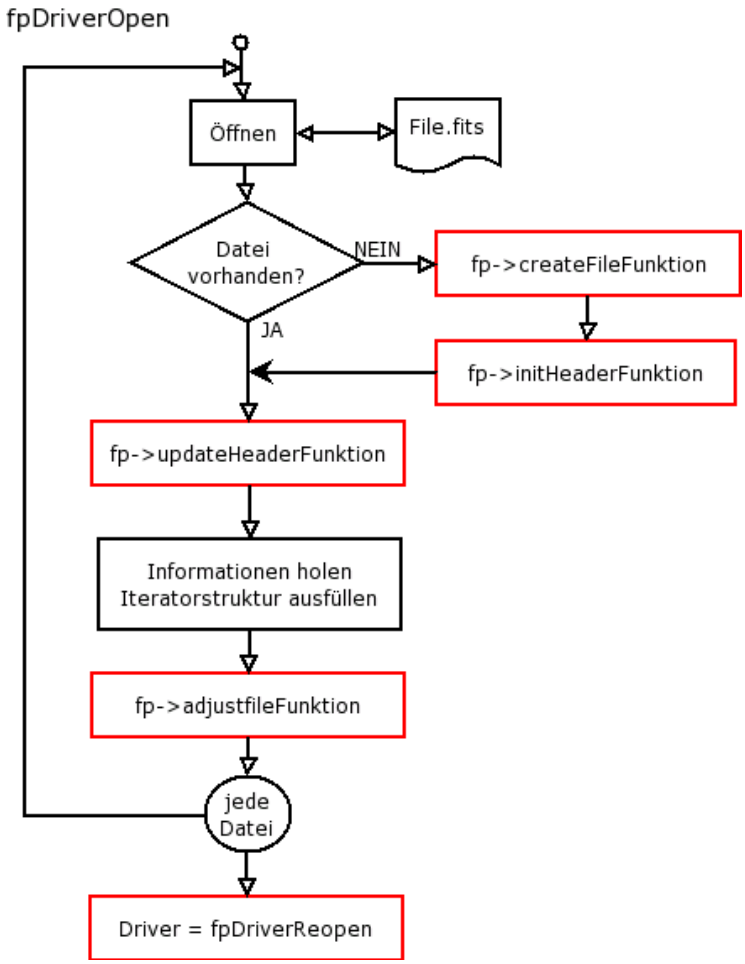

Abbildung 3.6: `fpDriverOpen`

daran der Header mit der Initialisierungsfunktion (`fp.initHeader`) erstellt. Dann wird der Header entweder der neu erstellten oder der bereits vorhandenen, geöffneten Datei aktualisiert, dies geschieht auch wieder mit der in `fp_infos` Struktur angegebenen Funktion (`fp.updateHeaderFctn`).
Im Anschluss daran kann noch eine Funktion ausgeführt werden (`fp.adjustFileFctn`), die auf diese Datei zugreift und Veränderungen vornimmt. In den Arbeitsmodulen ist dies dann nötig, wenn nach dem Einlesen und Bearbeiten der ersten FITS-Dateien in der ersten Arbeitsfunktion klar wird, dass die Grösse o.ä. in den weiteren FITS-Dateien für den nächsten Aufruf der Arbeitsfunktion angepasst werden muß.
Nachdem die Datei geöffnet ist, wird die Iterator-Funktion ausgeführt, und im Anschluss daran werden die Dateien wieder von einer Driver-Funktion, `fpDriverClose`, geschlossen.
Diese Treiberfunktion macht nichts, außer die Datei zu schliessen. Die Vorgehensweise der Iterator-Funktion wurde schon in 2.1.4 beschrieben. Sie wird im PROCESS_DATA-Callback so

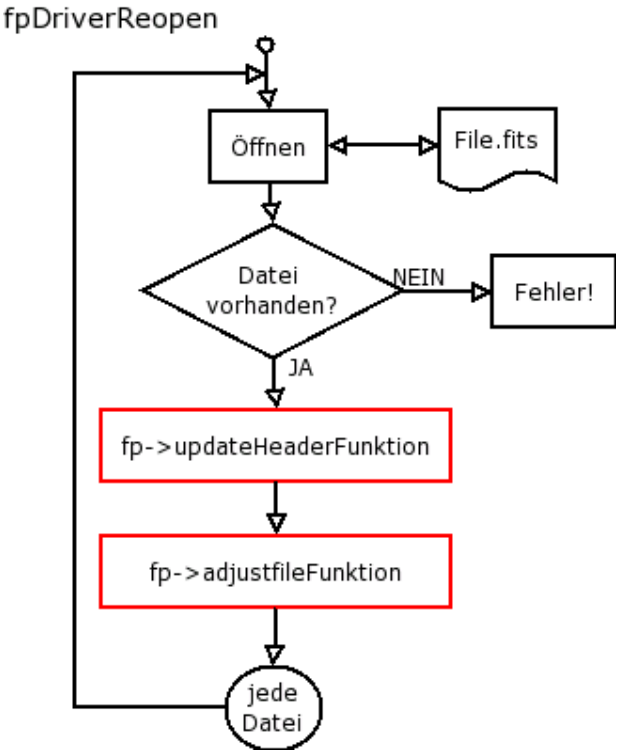

Abbildung 3.7: `fpDriverReopen`

oft aufgerufen, wie dies in der `fp_struktur` eingetragen ist.
Im Anschluss an die Iterator-Funktion wird eine modulspezifische Offsetfunktion (`fp.setOffsetFctn`) ausgeführt, wenn diese vorhanden ist. Diese Offsetfunktion wird von den Arbeitsmodulen benutzt, um dem Benutzer die Möglichkeit zu geben, nur die Bearbeitung ungelesener Daten oder das Überschreiben der bisherigen Daten, einzustellen. Ein paar einfache Offsetfunktionen, die von den Arbeitsmodulen benutzt werden, sind `fpSetOffsetAppend`, `fpSetOffsetAdd`, `fpSetOffsetOverwrite`, `fpSetOffsetUnread` und `fpSetOffsetComplete`. Diese Funktionen decken die häufigsten Fälle ab, die in einer Pipeline von Arbeitsmodulen gewünscht werden können.

3.1.5.3 Der EXIT-Callback

Ein weiterer Standardaccesspoint der Arbeitsmodule ist EXIT (siehe: 3.8). Dieser Accesspoint, oder vielmehr die EXIT-Callback Funktion, die auch sonst bei einer Beendigung des Programmes ausgeführt wird, enthält einige wichtige Operationen. Wenn sich das Programm im Servermodus befindet, also XPAAccessPoints definiert waren, werden diese über die XPAFree Funktion entfernt. Anschließend wird für jede Datei, falls definiert, eine CloseHeader-Funktion aufgerufen.

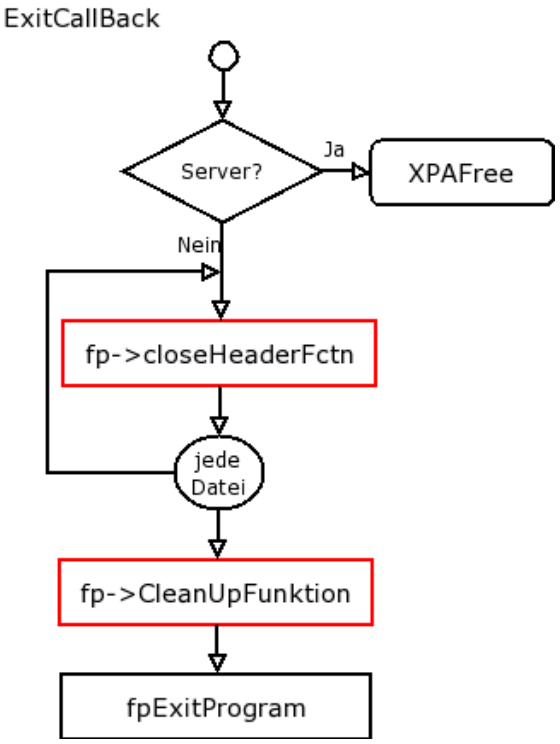

Abbildung 3.8: Ablauf des EXIT-Callbacks

Diese übernimmt bei den Arbeitsmodulen die wesentlichen Eintragungen in den Header. Diese Eintragungen sind – soweit möglich – an das Ende des Programms verlegt, um während der Pipeline keine Zeit zu verbrauchen.
Schließlich, nach dem Ausführen einer Aufräumfunktion (`fp.cleanUpFctn`), wird das Programm über die Exit Routine von meines Softwarepaketes beendet. Die Cleanup-Funktion ist dazu da, Systemressourcen ordnungsgemäss freizugegeben, die sonst mühsam von Hand entfernt werden müssten und das gesamte System verlangsamen könnten.
Die meisten Arbeitsmodule benötigen aber keine Systemresourcen, die durch eine solche Cleanup-Funktion freigegeben werden müssten. Die Ausnahme ist hier FPPlot, welches die geöffneten Plot-Devices schließt. Wird dies nicht getan, und als Device ist beispielsweise eine Postscript-Datei angegeben, dann wird diese nicht korrekt geschrieben und ist anschließend leer.

3.1.6 Das FPEventFeeder Programm

Das Programm FPEventFeeder ist der bisherige Ersatz für ein Programm, das die Daten von einem Detektor holt und in das FITS-Format umwandelt.
Es kann dazu verwendet werden, eine Pipeline zu testen. Dazu liest FPEventFeeder eine bereits existierende Eventliste ein und gibt die Events im richtigen zeitlichen Abstand an die

Pipeline weiter. Dieser richtige zeitliche Abstand wird aus der in der Eventliste eingetragenen Framenummer und einer Framezeit errechnet.
Das Programm ist das einzige der Pipeline, das das Readers-Writer-Locking (siehe: 3.2.6) selbst übernimmt und damit als (von mir so betitelter) fpipeserver fungiert. Um dieses Locking, also das Sperren und Entsperren der Dateien im Shared Memory – über von den Steuermodulen erstellte Semaphore – selbst durchzuführen, benötigt es eine Reihe von Parametern, die nicht vom Benutzer eingegeben werden können, und nur in einer Pipeline Sinn machen. Dadurch ist auch der Aufbau von FPEventFeeder komplett verschieden von dem der anderen Arbeitsmodule. Er stellt keinen XPA-Server dar, benutzt nicht die Iterator-Funktion der CFITSIO-Bibliothek und hat auch nicht die internen Strukturen der anderen Arbeitsmodule. Dies liegt daran, dass das Einlesen der bereits existierenden Eventliste so implementiert ist, wie dies wahrscheinlich auch beim Holen der Events von einem richtigen Detektor der Fall ist.
Dieses Programm, in der jetzigen Form und in der an einen Detektor angepassten Form, muss Pollings betreiben, also eine andauernde Abfrage in einer Schleife durchführen, um die Daten des Detektors rechtzeitig holen zu können. Dies wäre nur dann nicht nötig, wenn die Abfrage, ob neue Daten vom Detektor bereits vorhanden sind, über eine Struktur erfolgen würde, deren Veränderung vom Kernel über einen Select Aufruf beobachtet werden könnte. Dann wäre auch hier das Polling durch einen Schlaf (über den `select()`-Aufruf in einer der XPAMainLoop ähnlichen Schleife) möglich.
Die Parameter des FPEventFeeder sind in der folgenden Tabelle zu sehen:

Tabelle 3.3: Parameter von FPEventFeeder

Name	Format	Nachfrage	Default	Min/Max
infile	f	ql		
outfile	f	hl		
xcol	s	hl	RAWX	
ycol	s	hl	RAWY	
ecol	s	hl	PHA	
fcol	s	hl	FRAME	
frametime	r	hl	0.073	
geiger	b	h	n	
nsems	i	q	-1	
semid	i	q	-1	
wlock	i	q	-1	
nor	i	q	-1	
or	s	q	-1	

Die Parameter ‘nsems’ (Number of Semaphores), ‘semid’ (Semaphore Identification Number), ‘wlock’ (Write Lock Position), ‘nor’ (Number of other readers at this write lock) und ‘or’ (List of other Readers at this write lock) werden von den Steuermodulen meines Softwarepaketes ausgefüllt (siehe Abschnitt 3.2).
Der Parameter ‘Frametime’ gibt die Zeit in Sekunden an, die vom Start eines Frames bis

zum Start des nächsten Frames vergeht. Damit wird der in der FRAME-Spalte der Eventliste vorgefundene Wert multipliziert, um den Zeitpunkt dieses Events zu bestimmen.
Der Parameter ‘geiger’ wurde von Jörn Wilms zu Testzwecken eingebaut, um die Events über die Soundkarte des Computers hörbar zu machen (wie bei einem Geigerzähler, daher der Name des Parameters). Möglicherweise kann dies im Labor tatsächlich nützlich sein.

3.1.7 Das FPOmap Programm

Das Arbeitsmodul FPOmap ist dazu da, auf eine Eventliste eine Offsetmap anzuwenden.
Die Offsetmap besteht aus Energiewerten, die den Unterschied zwischen den tatsächlichen Energien der Photonen und den vom Detektor (im Mittel) gelieferten Energien darstellen.
Die meisten Detektoren (zumindest CCDs) liefern – abhängig von der Position des Pixels auf dem Detektor – einen spezifischen, vom realen Wert abweichenden Energiewert für diese Position. Diese Werte können für FPOmap entweder in Form eines Bildes (also eines 2D-Arrays mit den jeweiligen zu subtrahierenden Energiewerten), als Liste (mit je einer Spalte für x, y und dem Offsetwert), oder in Form einer vollständigen Liste (nur aus einer Energiewertspalte bestehend) eingelesen werden.
Bei der Angabe der x- und y-Koordinaten muss natürlich der Benutzer selbst aufpassen, dass diese mit den anderen in der Pipeline verwendeten Programmen übereinstimmen. Die Koordinaten des Bildes liegen, wie bei den FTOOLS auch, in der Form *Bild(y)(x)* vor.
Die Parameter, die an das FPOmap Programm übergeben werden können, und deren Eigenschaften sind in der folgenden Tabelle zusammengefasst:

Tabelle 3.4: Parameter von FPOmap

Name	Format	Nachfrage	Default	Min/Max
servermode	b	h	n	
infile	f	ql		
outfile	f	ql		
inmode	i	h	4	4 / 5
outmode	i	h	1	1 / 2
omapfile	f	ql		
omaptype	i	ql	1	0 1 2
omapxcol	s	hl	X	
omapycol	s	hl	Y	
omapvcol	s	hl	VALUE	
xdim	i	ql		
ydim	i	ql		
inxcol	s	hl	RAWX	
inycol	s	hl	RAWY	
inecol	s	hl	PHA	
intcol	s	hl	TIME	
outxcol	s	hl	RAWX	
outycol	s	hl	RAWY	
outecol	s	hl	PHA	
outtcol	s	hl	TIME	
xpaname	s	h	FPOmap	
instrume	s	h	${FP_INSTRUME}	
telescop	s	h	${FP_TELESCOP}	
creator	s	h	"FPOmap V.1.0.0"	
filter	s	h	${FP_FILTER}	
object	s	h	${FP_OBJECT}	
origin	s	h	IAAT	
observer	s	h	${FP_OBSERVER}	
configur	s	h	${FP_CONFIGUR}	

Diese Tabelle entspricht im wesentlichen dem Parameterfile. Neben den Standardparametern gibt es hier noch:

omapfile – Datei mit der Offsetmap. Diese Angabe kann natürlich die ‘extended file name syntax’ der CFITSIO Bibliothek enthalten.

omaptype – Typ der Offsetmap: 0:image, 1: Liste mit Spalten ‘x/y/Offset’, 2: Liste mit Spalte ‘Offset’.

omapxcol – Name der Spalte, die die x Koordinate enthält. Dieser Parameter wird ignoriert, wenn als Typ der Offsetmap 0 oder 2 angegeben ist.

omapycol – wie omapxcol

omapvcol – wie omapxcol, enthält den (Energie)Wert, in den Einheiten, die auch in der Eventliste benutzt werden.
xdim – Die Detektor Dimension in X Richtung, in Einheiten wie sie in der Eventliste für die Position angegeben sind.
ydim – wie xdim

3.1.8 Das FPCopyControl Programm

Dieses Programm ist, genau wie FPEventFeeder, nur in einer Pipeline sinnvoll einsetzbar.
Seine Aufgabe besteht darin, die Eventliste aus dem Infile in das Outfile zu schreiben. Dabei setzt das Programm eine Obergrenze der zu lesenden Events. Ist die Zahl der neuen Events größer als diese Grenze, dann werden in diesem Schritt nicht alle Events zur weiteren Verarbeitung dem Rest der Pipeline übergeben, stattdessen folgen die überzähligen Events erst in den nächsten Aufrufen. Der Grund für diese Funktion liegt in der Benutzerinteraktion während des Betriebs der Pipeline. Um die Pipeline in Echtzeit zu betreiben, können nicht alle Alternativen parallel gerechnet werden. Wird beispielsweise eine Splitkorrektur durchgeführt, und die Anzeige auf Single-Events beschränkt, dann können nicht gleichzeitig auch noch die Bilder für Double-Events, Triple-Events, Quadruple-Events und ihre Kombinationen gerechnet werden. Soll nun, während des Betriebs der Pipeline, die Verarbeitung auf Doubles umgestellt werden, dann müssen die bisher gesammelten Events mit den neuen Parametern neu verarbeitet werden. Würden alle Events auf einen Schlag eingelesen, käme es zu einem Stillstand der Pipeline. So aber kann die Pipeline diese Schritt für Schritt verarbeiten, bis wieder in Echtzeit dargestellt werden kann. Die Parameter sind hier aufgeführt:

Tabelle 3.5: Parameter von FPCopyControl

Name	Format	Nachfrage	Default	Min/Max
servermode	b	h	n	
infile	f	ql		
outfile	f	ql		
inmode	i	h	5	4/5
outmode	i	h	2	0/3
maxrows	i	h	50	
xpaname	s	h	"FPCopyControl"	

Der einzige interessante Parameter ist *maxrows*, der die maximale Anzahl der zu kopierenden Zeilen (d.h. Events) festlegt. Alle anderen Parameter sind die für die Arbeitsmodule üblichen. FPCopyControl folgt aufgrund der anderen Anforderungen nicht dem für die Arbeitsmodule üblichen Aufbau, sondern hat intern eine eigene Struktur. Dies folgt aus einem Nachteil der Iterator-Funktion der CFITSIO-Bibliothek. Diese Funktion ermöglicht es leider nicht, nur Teile der Datei zu lesen, wenn diese Teile am Anfang der Spalten stehen. Nur für die letzten Teile einer Spalte ist dies mit dem Offset Parameter möglich. FPCopyControl besitzt bisher nur eine

eingeschränkte Funktion, insofern es nur auf einem Teil der möglichen Datentypen einer FITS-Tabelle arbeiten kann. Dies zu ändern dürfte nur wenig Arbeit erfordern, musste aber trotzdem aus Zeitgründen auf einen späteren Zeitpunkt verschoben werden. Alle Datentypen, die von den anderen Arbeitsmodulen geschrieben werden, können natürlich gelesen werden.
Der RESET-Callback dieses Programms verfügt über eine Besonderheit: FPCopyControl schläft nach diesem Aufruf für 50 Sekunden oder bis ein Signal empfangen wird. Diese Eigenschaft kann genutzt werden, um die Pipeline für die Dauer von kurzen Benutzereingaben oder für die Dauer der Aufrufe der RESET-Callbacks der anderen Pipelinefunktionen anzuhalten. FPCopyControl sollte am Ende dieser Pause ein 'USR1'-Signal geschickt werden, in meiner interaktiven Beispielpipeline wird dieses Signal durch den 'set'-Aufruf '`killall -USR1 FPCopyControl`' gegeben.

3.1.9 Das FPPlot Programm

Das Arbeitsmodul zum Erstellen von xy-Plots bietet nur einen geringen Komfort und wurde geschrieben, weil keine externen Programme auffindbar waren, die allen Anforderungen gerecht wurden (siehe: 3.1.1).
Die Vorteile von FPPlot gegenüber externen Programmen sind: erstens, die Möglichkeit FITS Dateien zu lesen, zweitens, in einer Pipeline arbeiten zu können und drittens, über eine Form der Interprozesskommunikation zu verfügen. Nachteile sind dagegen die geringe Möglichkeit einer Benutzerinteraktion und die doch eher unflexible Optik. Die sicherlich beste Möglichkeit zur Verbesserung der Pipeline hinsichtlich der xy-Plots wäre wahrscheinlich, den ds9-Code zu verändern, so dass auch FITS-Dateien für xy-Plots gelesen werden können. Trotzdem sind in der folgenden Tabelle noch die Parameter, die von FPPlot akzeptiert werden, aufgelistet:

Tabelle 3.6: Parameter von FPPlot

Name	Format	Nachfrage	Default	Min/Max
servermode	b	h	n	
inmode	i	h	5	4/5
infile	f	ql		
xmin	i	h	0	
xmax	i	h	500	
ymin	i	h	0	
ymax	i	h	500	
xcol	s	hl	"FRAME"	
ycol	s	hl	"N"	
device	s	hl	"/XSERVE"	
symbol	i	h	1	-8/15
line	b	h	n	
scatter	b	h	n	
foreground	s	h	"white"	
background	s	h	"black"	
xpaname	s	h	"FPPlot"	

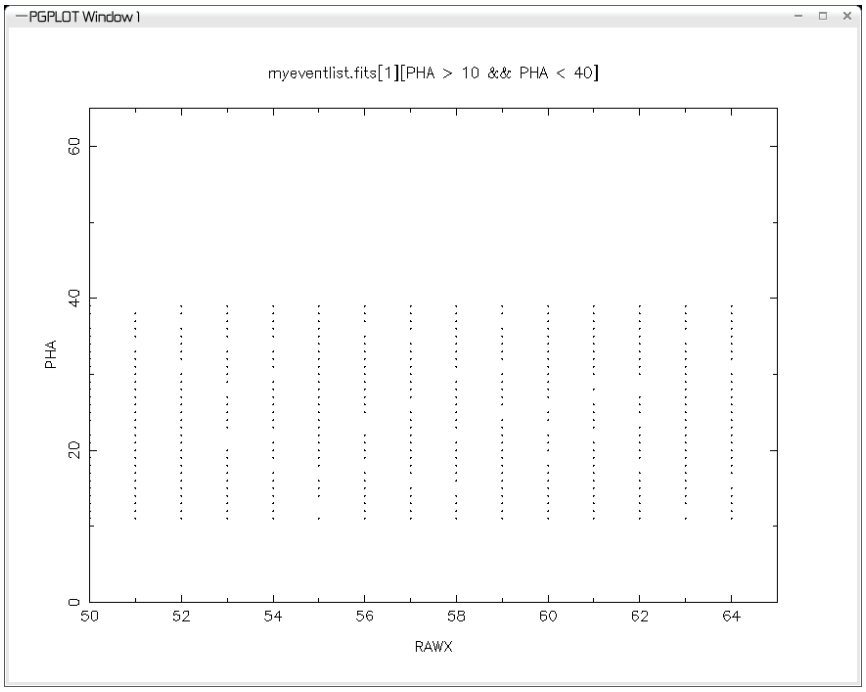

Abbildung 3.9: Das FPPlot Fenster ohne Scatter, device = /XSERVE, symbol = 1

Die diesbezüglich interessanten Parameter sind:

xcol – gibt die Spalte an, die die Abszisse des Plots enthält.
ycol – gibt die Spalte an, die die Ordinate des Plots enthält.
device – die plottende Bibliothek hinter FPPlot ist pgplot. Dieser Bibliothek können viele Ausgabemedien angegeben werden. Hier nur eine nützliche Auswahl:

- GIF file: /GIF, /VGIF
- PostScript page description language: /PS, /VPS, /CPS, /VCPS
- X Window dump file: /WD, /VWD.
- Workstations running X Window System: /XWINDOW, /XSERVE
- Tektronix terminals and emulators: /TEK4010, /GF, /RETRO, /GTERM, /XTERM, /ZSTEM, /V603, /KRM3. /TK4100
- Tektronix-format disk file: /TFILE
- GOC Sigma T5670 terminal: /GOC
- PGPLOT metafile: /PGMF
- TeX PK Font Output files: /TX
- LaTeX picture environment: /LATEX[4]

[4]Hier sei angemerkt, dass ich nicht alle dieser von pgplot angegebenen Devices in den Tests nutzen konnte.

symbol – legt das Symbol fest, dass zur Darstellung eines Punktes genutzt wird. Die möglichen Werte sind: -1, kleinstmöglicher Punkt (ein Pixel); 0 bis 31, verschiedene Symbole; -3 bis -8, reguläre Polygone mit entsprechend vielen Seiten; 33 bis 127, entsprechende ASCII-Zeichen.

line – erwartet eine Artwort auf die Frage, ob eine Linie anstelle von Punkten gezeichnet werden soll.

scatter – legt fest, ob anstelle der genauen Abszisse eine verschmierte Abszisse genutzt wird. Dies ist manchmal sinnvoll, wenn man auf einen Blick übereinander liegende Punkte, die man sonst nur als einen Punkt wahrnehmen würde, in ihrer Masse sehen möchte. Anstelle eines Netzes von kleinen Punkten (wie in Abbildung 3.9) entsteht so ein Band (wie es in Abbildung3.10 zu sehen ist), dass sich etwa zur Darstellung der Charge Transfer Efficiency eines CCDs eignet, da mit Hilfe dieser Darstellung sich das entstehende Bild schneller interpretieren läßt.

foreground – legt die Farbe des Vordergrundes fest, d.h. Schrift und Punkte. Wenn die Umgebungsvariable PGPLOT_FOREGROUND gesetzt ist, wird diese stattdessen benutzt.

background – legt die Hintergrundfarbe fest. Sonst wie *foreground*.

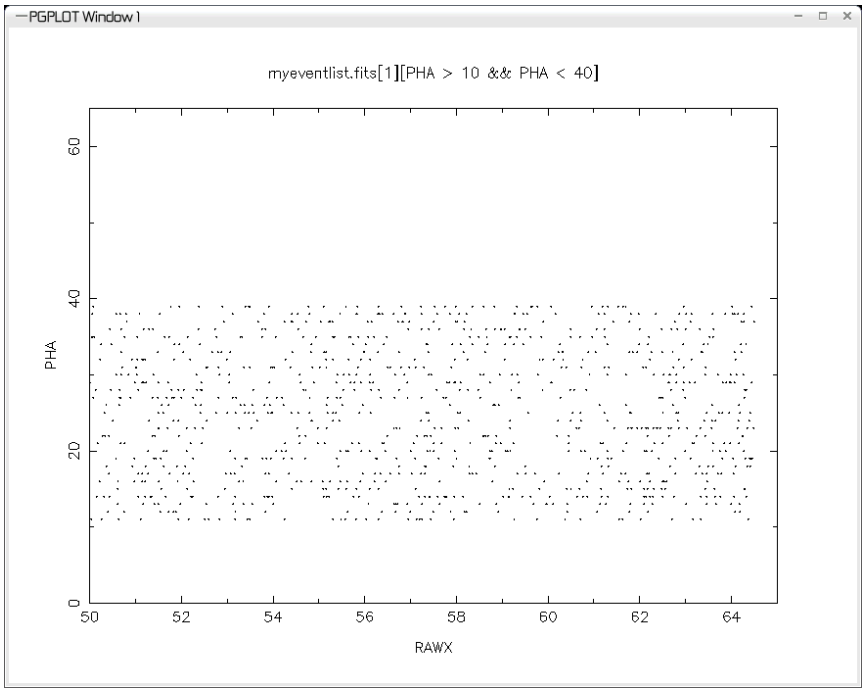

Abbildung 3.10: Das FPPlot Fenster mit Scatter, device = /XSERVE, symbol = 1

3.1.10 Das FPSpectrum Programm

Das Arbeitsmodul FPSpectrum erstellt aus einer Eventliste ein Spektrum.
Genauer gesagt werden die Events für jeden Detektorkanal gezählt und in einer Binarytable in der FITS-Datei gespeichert. Dies ist weitestgehend OGIP konform (siehe: 2.2.3), wobei die Unterschiede zu einem korrekten OGIP Spektrum hauptsächlich auf die Situation im Labor zurückgehen. So werden die Positionen des beobachteten Objekts und die RMF- und ARF-Dateien nicht angegeben (siehe 2.2.3 für die OGIP Vorgaben), bzw. es gibt anstelle des Dateinamens 'none' eingetragen.
Die Parameter von FPSpectrum sind:

Tabelle 3.7: Parameter von FPSpectrum

Name	Format	Nachfrage	Default	Min/Max
servermode	b	h	n	
inmode	i	h	4	4/5
outmode	i	h	2	0/3
infile	f	ql		
outfile	f	ql		
inecol	s	hl	"PHA"	
outncol	s	hl	"COUNTS"	
outecol	s	hl	"CHANNEL"	
emin	i	h	0	
emax	i	h	256	
xpaname	s	h	"FPSpectrum"	
instrume	s	h	${FP_INSTRUME}	
telescop	s	h	${FP_TELESCOP}	
creator	s	h	"FPSpectrum V.1.0.0"	
filter	s	h	${FP_FILTER}	
object	s	h	${FP_OBJECT}	
spectype	s	h	TOTAL	BKG|NET|TOTAL
chantype	s	h	PHA	PHA|PI
origin	s	h	IAAT	
observer	s	h	${FP_OBSERVER}	
configur	s	h	${FP_CONFIGUR}	

Davon interessant sind:

outncol – der Name der Spalte, in die die Anzahl der Events geschrieben werden soll. Dies sollte möglichst COUNTS sein.

outecol – der Name der Spalte, in die der Energiewert geschrieben werden soll. Dies sollte möglichst CHANNEL sein.

emin – ist der kleinste Kanal, oder der kleinste Energiewert. Normalerweise sollten Kanäle mit 1 anfangen.

emax – ist der größte erlaubte Kanal, oder der größte Energiewert.
spectype – FPSpectrum macht keine Hintergrundsubtraktion, deshalb ist hier entweder BKG oder TOTAL einzutragen.
chantype – ist PHA für Rohdetektorkanäle oder PI für korrigierte Kanäle.

Die üblichen Keywords wie INSTRUME, TELESCOP, ORIGIN, ... werden, wenn nicht vom Benutzer andere Angaben gemacht werden, über Umgebungsvariablen gesetzt, die beispielsweise von der GUI stammen können.
Die XPAAccessPoints, die von FPSpectrum registriert werden, sind PROCESS_DATA, EXIT, INFILE und RESET. Der RESET Befehl löscht das bisherige Spektrum und fängt neu an zu akkumulieren.

3.1.11 Das FPLightcurve Programm

Das Programm FPLightcurve erzeugt aus einer Eventliste eine Lichtkurve. Diese entspricht weitestgehend den OGIP-Anforderungen (siehe Abschnitt 2.2.2). Die Erweiterung der FITS-Datei, die die Lichtkurve enthält, ist eine Binarytable und hat den Namen EVENTS. Von den verschiedenen möglichen Arten, diese Lichtkurve zu speichern, unterstützt dieses Programm nur eine, nämlich die einer Eventliste mit gleichmäßig verteilten zeitlichen Abständen. Die von FPLightcurve akzeptierten Parameter sind:

Tabelle 3.8: Parameter von FPLightcurve

Name	Format	Nachfrage	Default	Min/Max
servermode	b	h	n	
inmode	i	h	4	4/5
outmode	i	h	2	0/3
infile	f	ql		
outfile	f	ql		
intcol	s	ql	"TIME"	
outncol	s	ql	"COUNTS"	
outtcol	s	ql	"TIME"	
timedel	i	q	1	
xpaname	s	h	"FPLightcurve"	
instrume	s	h	${FP_INSTRUME}	
telescop	s	h	${FP_TELESCOP}	
creator	s	h	"FPLightcurve V.1.0.0"	
filter	s	h	${FP_FILTER}	
object	s	h	${FP_OBJECT}	
origin	s	h	IAAT	
observer	s	h	${FP_OBSERVER}	
configur	s	h	${FP_CONFIGUR}	
tassign	s	h	TUEBINGEN	
tierrela	r	q		
tierabso	r	q		
chanmin	i	h	${FP_CHANMIN}	
chanmax	i	h	${FP_CHANMAX}	
tassign	s	h	s	
geolat	r	h	${FP_GEOLAT}	
geolong	r	h	${FP_GEOLONG}	
altitude	r	h	${FP_ALTITUDE}	
clockcor	s	h	${FP_CLOCKCOR}	YES|NO|UNKNOWN

Die in diesem Zusammenhang wichtigen Parameter sind:

outncol – sollte (wie standardmäßig eingestellt) COUNTS sein.
outecol – sollte TIME sein.
timedel – legt die Bingrösse fest, d.h. das Zeitintervall, in dem Events verschiedener Zeiten zusammengefasst werden. Die in der Lichtkurve angegebene Zeit in der Spalte TIME liegt immer in der Mitte des Bins, also von $TIME[ii] - TIMEDEL/2$ bis $TIME[ii] + TIMEDEL/2$.
tassign – beschreibt den Ort, an dem die Zeit zu einem Event zugewiesen wurde. Typische Werte hierfür sind ‘TUEBINGEN’ oder ‘SATELLITE’, der Standardwert wird über die

Umgebungsvariable festgelegt.

geolat – ist anzugeben, wenn für ‘tassign’ ein Ort auf der Erde angegeben wurde.

geolong – wie geolat

altitude – wie geolat

tierrela – legt den relativen Teil des Fehlers der Zeitangaben fest, der durch die Genauigkeit der Uhr bestimmt wird, und sollte als dimensionslose Rate angegeben werden.

tieabso – legt den absoluten Fehler der Zeitangabe fest. Dieser sollte alle anderen Unsicherheiten, z.B. durch Korrekturen, enthalten und in Sekunden angegeben werden.

chanmin – legt die untere Grenze der betrachteten Energiekanäle fest.

chanmax – legt die obere Grenze fest. Beides wird standardmäßig über Umgebungsvariablen gesetzt.

timeunit – legt die Einheit aller Zeitangaben in der Lichtkurve fest.

clockcor – gibt an, ob eine Korrektur gegenüber der Universal Time angebracht wurde oder nicht.

Beim Erstellen der Lichtkurve wird erwartet, dass die Zeit der Events nicht abnimmt, da sonst das Photon falsch eingeordnet wird. Dies geschieht, weil für einen neuen Event so lange die Zeit in Schritten von ‘timedel’ erhöht wird, bis die Eventzeit kleiner ist als der Zeiteintrag. Wenn also eine Eventzeit kleiner ist als der bisherige Zeiteintrag, dann wird er einfach zum aktuellen Eintrag addiert.
Zwar wäre es leicht, dies zu ändern und jedes Event korrekt in die Lichtkurve einzuordnen. Die Annahme, das die Events in ihrer zeitlichen Abfolge auch so bei diesem Programm ankommen, führt jedoch zu einem erheblichen Geschwindigkeitszuwachs. Denn bei einer Lichtkurve würde die Datenmenge sehr groß, wenn bei jedem neuen Eventschub die komplette Lichtkurve gelesen und geschrieben werden müsste oder sogar Einträge vor dem bisher ersten eingefügt werden müssten.
Diese Annahme scheint mir auch zu keinen besonderen Einschränkungen zu führen, weder in Laborbetrieb, noch in einer wissenschaftlichen Auswertung.
Manche der Keyword-Einträge, die von der OGIP in ihrem Memo zu Lichtkurven [34] gefordert werden, scheinen nicht konsistent mit anderen OGIP Memos zu sein, z.B. mit den Aussagen des Spektrum-Memos [28]. Deshalb schreibt FPLightcurve zusätzlich zu den im Lichtkurven-Memo [34] angegebenen Keywords noch folgende:

HDUCLAS1 = LIGHTCURVE dies geschieht in Anlehnung an die Spektrum Definition durch die OGIP [28], welche dieses Keyword durch den Wert SPECTRUM kennzeichnet.

HDUVERS ist auch aus der Spektrum Definition und entspricht dem für Lichtkurven definierten TIMVERS Keyword.

HDUCLAS2 = EQUALLY SPACED gibt ebenfalls, ähnlich wie beim Spektrum, weitere Information über die Art der Speicherung der Lichtkurve an.

3.1.12 Das FPIntensity Programm

Dieses Programm erstellt aus einer Eventliste ein Intensitätsbild. In diesem Bild wird die Anzahl der Events pro Pixel als Wert eingetragen.
Die Parameter, die dieses Programm akzeptiert, sind in folgender Tabelle aufgelistet:

Tabelle 3.9: Parameter von FPIntensity

Name	Format	Nachfrage	Default	Min/Max
servermode	b	h	n	
inmode	i	h	4	4/5
outmode	i	h	2	0/3
sourcefile	f	ql		
destfile	f	ql		
xcol	s	hl	"RAWX"	
ycol	s	hl	"RAWY"	
xdim	i	h		
ydim	i	h		
xpaname	s	h	"FPIntensity"	
instrume	s	h	${FP_INSTRUME}	
telescop	s	h	${FP_TELESCOP}	
creator	s	h	"FPIntensity V.1.0.0"	
filter	s	h	${FP_FILTER}	
object	s	h	${FP_OBJECT}	
origin	s	h	IAAT	
observer	s	h	${FP_OBSERVER}	
configur	s	h	${FP_CONFIGUR}	

In Ermangelung einer OGIP Definition schreibt dieses Programm einfach die Parameter HDUCLAS1 = INTENSITYMAP und HDUCLAS3 = COUNTS. Dies schien mir am besten zu den Keywords zu passen, die von der OGIP für Spektren definiert wurden, und die ich auch schon für die Lichtkurven zusätzlich zu den definierten geschrieben habe. Die Parameter erklären sich im Prinzip alle selbst oder wurden schon bei den bisherigen Arbeitsmodulen beschrieben.

3.2 Die Steuermodule: Aufbau und Ablauf

Die Steuermodule (Routinen, die von GUIs oder Kommandozeilenprogrammen aufgerufen werden können) haben die Aufgabe, die Arbeitsmodule zu koordinieren und von der Benutzeroberfläche Kommandos entgegenzunehmen und auszuführen.
Beim Design des Softwarepaketes habe ich darauf geachtet, diese Aufgaben getrennt von der Benutzeroberfläche zu halten, damit sie leichter an die Bedürfnisse der jeweiligen Anwender angepasst werden kann[5], ohne dabei die Funktionalität der Pipeline zu beeinträchtigen.
Die Aufgaben der Steuermodule sind im einzelnen:

- Einlesen der Pipelinedefinition
- Interpretieren der Pipelinedefinition und Erstellen einer internen Informationsstrukur über die Pipeline
- Starten der Arbeitsmodule, die über einen Servermodus verfügen
- Aufrufen der Arbeitsmodule, die nicht über einen Servermodus verfügen
- Organisation der Zugriffsrechte für die Dateien
- Benachrichtigung der Arbeitsmodule im Servermodus bei Ankunft neuer Daten
- Bearbeiten der Parameter für Aufrufe der Arbeitsmodule, die nicht über einen Servermodus verfügen und
- die Weitergabe von Benutzerkommandos an die Arbeitsmodule, wie z. B.
  - Änderungen von Grenzwerten
  - Änderungen in der Darstellung wie Vergrößerung einer Region
  - Abspeichern von Bildern
  - ...

Wichtig ist hierbei, dass die Steuermodule solche Benutzerkommandos von der Benutzeroberfläche bekommen, teilweise bearbeiten und danach an die Arbeitsmodule weiterleiten, während die Bereitstellung und Ausführung dieser Kommandos ausschließlich in den Arbeitsmodulen erfolgt .
Diese wichtige Trennung der Information über die Struktur der Pipeline von der Information über die Daten in der Pipeline kann aus meiner Sicht gut eingehalten werden. Es erfordert allerdings eine gewisse Zeit, bis man sich daran gewöhnt hat, dass nicht das Steuermodul und die GUI darüber ‘entscheiden’, ob z.B. eine Energieselektion stattfinden kann, sondern dass dies allein bei der aufgesetzten Pipeline und den in ihr enthaltenen Programmen liegt.
Einzig die Möglichkeit, solche Benutzerinteraktionen entgegenzunehmen und weiterzuleiten, ist fest in den Steuermodulen und der Benutzeroberfläche verankert.

[5] Die GUI ist in Perl/Tk geschrieben. Um sie z.B. an ein anderes Widget-System anzupassen, müssten keine Steuermodule geändert werden.

3.2.1 Die Pipelinedefinition

Der tatsächliche Aufbau der Pipeline, d.h. die Festlegung der Arbeitsmodule, die Festlegung der Reihenfolge, in der sie ausgeführt werden sowie die Festlegung weiterer Details sind nicht in meinem Softwarepaket oder in den Steuermodulen festgeschrieben, sondern können individuell zusammengestellt und verändert werden.
Das ist möglich durch die Platzierung der Definition der Pipeline in separaten Dateien. Diese Dateien können über die Graphische Benutzeroberfläche ('pipeline definition file') ausgewählt und von den Steuermodulen geladen werden.
Das Format, in dem diese Definitionen gspeichert werden, lässt sich über die Dateiendung festlegen – andere Arten der Dateiformaterkennung werden nicht unterstützt – und sind entweder '.fits' für das FITS Format oder '.xml' für eine Definition im XML Format. Das bevorzugte Format ist dabei das XML Format.

3.2.1.1 Pipelinedefinition in FITS

Bei der Definition der Aufgaben wird unterschieden zwischen dem Ziel (goal), der Arbeitsaufgabe (job) und den einzelnen Aufrufen. Die Unterteilung in Ziele und Arbeitsaufgaben hat keinen tiefergehenden Grund, hilft aber dem Nutzer des Programms, den Überblick über das Gesamtprogramm zu behalten und erscheint mir daher sinnvoll. Beispiel für ein solches Ziel könnte 'process eventlist' sein, welches die einzelnen Arbeitsaufgaben 'add offset map', 'aply badpixel map' und 'find split events' zusammenfassen könnte. Die dazugehörigen Operationen werden in den entsprechenden Fenstern dann gruppiert angezeigt.
In der Definitionsdatei werden Ziele durch den Namen der FITS-Erweiterung (extensionname) festgelegt. Alle Ziele müssen unterschiedliche Namen haben. Die weiteren Definitionen werden in einer Binary-Table abgelegt. Diese hat folgende Einträge:

job – In dieser Spalte steht der Name der Arbeitsaufgabe. Der ist nicht nur für den Benutzer wichtig, wie es bei der Unterteilung in Arbeitsaufgaben und Ziele der Fall war, sondern zeigt den Steuermodulen z.B. an, welche Startaufrufe zu welchen Beendigungsaufrufen gehören. Die hier vergebenen Namen müssen eindeutig sein. Die Ausnahme sind 'set'-Aufrufe.

status – Hier war geplant, dass der Benutzer bestimmte Arbeitsaufgaben als aktiv und manche Arbeitsaufgaben als passiv kennzeichnen kann, die dann nicht gestartet werden.
Obwohl diese Eigenschaft tatsächlich nicht in die Steuermodule eingebaut wurde, habe ich die entsprechende Kennzeichnung in den Definitionsdateien und den dazu gehörenden Einleseroutinen belassen, damit später, wenn sich herausstellt, dass dies doch eine sinnvolle Eigenschaft wäre, sie ohne Änderung der Definitionsdateien eingebaut werden könnte.
Bis dahin sollten 'set'-Aufrufe den Status 1 erhalten, alle anderen den Status 0.

type – Erlaubte Einträge in dieser Spalte sind 'single', 'loopstart', 'loopstop' 'fpipeserverstart', 'fpipeserverstop' oder 'set'. Diese Werte zeigen an, ob in der Spalte 'call' ein Aufruf steht, der bei jedem Durchlauf ausgeführt werden soll, oder ob es sich dabei um Aufrufe han-

delt, die entweder nur zu Begin des Programms, nur bei Beendigung des Programms oder nur zur Änderung bestimmter Parameter aufgerufen werden sollen. Die Werte ‘fpipeserverstart’ und ‘fpipeserverstop’ legen darüber hinaus noch fest, dass kein ‘Readers-Writer-Locking’ durch die Steuermodule vorgenommen werden soll, sondern dass dies vom Arbeitsmodul selbst durchgeführt wird.

call – Hier ist der Aufruf angeführt, den die Steuermodule dem System übergeben. Bei ‘set’- und ‘single’-Aufrufen wird dies direkt über einen ‘`system()`’-Aufruf gemacht, alle anderen werden über einen ‘`fork()`’- und ‘`exec()`’-Befehl ausgeführt. Vorher wird überprüft, ob noch andere Werte im Aufruf ersetzt werden müssen. Bei ‘loopstop’, ‘fpipeserverstop’ und ‘set’-Aufrufen, kann ein ‘##pid##’ enthalten sein. Dies weist die Steuermodule an, die Prozessidentifikationsnummer des zu stoppenden Programms hier einzutragen. Die Arbeitsmodule, die selbstständig Dateien sperren wollen, benötigen meist noch Informationen (siehe Abschnitt 3.2.6), die über ‘##*variable*##’ mit folgenden Variablen erhalten werden können:

- SEMID
  Die Steuermodule tragen hier die Identifikationsnummer des Semaphorenfeldes ein.
- KEY
  wird durch den Schlüssel ersetzt, über den ein Programm die Identifikationsnummer des Semaphorenfeldes bekommen kann.
- WLOCK
  An dieser Stelle wird der Index des Semaphores eingetragen, das von dem Programm zum Aufbauen einer Schreibsperre benutzt wird. Wenn dieser Eintrag mehrmals vorkommt, wird der Index des Semaphores der nächsten Datei eingetragen, falls das Programm in mehrere Dateien schreibt.
- WLOCKS,
  Nach dem Komma dieses Eintrags erwarten die Steuerprogramme ein Trennsymbol, beispielsweise ein ‘/’, mit dem die Indices der Semaphoren getrennt werden, die für Schreibsperren dieses Programms zuständig sind. Dies ist äquivalent zu mehreren WLOCK-Einträgen.
- RLOCK
  Wie WLOCK, nur wird hier der Index eingetragen, der für Lesesperren dieses Programms und dieser Datei zuständig ist.
- RLOCKS,
  Siehe WLOCKS, auch hier entsprechend für Lesezugriffe.
- N_O_R
  steht abkürzend für ‘number of other readers’ und wird durch die Anzahl der Programme ersetzt, die lesend auf eine Datei zugreifen wollen, in die dieses Programm schreibt.
- N_O_W
  bedeutet ‘number of other writers’ und ist nur der Vollständigkeit halber eingebaut. Es ist in der jetzigen Implementierung, bei der nur ein schreibender Prozess pro File möglich ist, immer gleich 1.

- O_R,
  Dies sind die Indices der durch N_O_R bestimmten Semaphore.
- O_W,
  ist immer der Index des Semaphors, das den Schreibzugriff regelt, siehe N_O_W.
- NSEMS
  wird durch die Anzahl der Semaphore im Semaphorenfeld ersetzt. Dies wurde nur der Vollständigkeit halber hinzugefügt, der Wert wird nicht benötigt, um Dateisperren aufzubauen. Er kann aber genutzt werden, um Sicherheitsabfragen über die Korrektheit der anderen Angaben zu machen.

3.2.2 Benutzerkommandos als Set-Aufrufe

Bei ‘set’-Aufrufen ist die Situation etwas komplizierter.
Hier sind im Normalfall Informationen des Benutzers, zum Beispiel Energiegrenzwerte, nötig. Diese Informationen müssen von der Benutzeroberfläche geholt werden. Zu diesem Zweck sind weitere Angaben an dieser Stelle in die Pipelinedefinition eingebaut. Wie schon bei ‘fpipeserver’ fangen die zu ersetzenden Einträge mit ‘##’ an und hören damit auch auf. Dazwischen stehen bei ‘set’-Aufrufen mehr Informationen, getrennt durch Kommas:
`##name,type,default,min,max,prompt##`
Der Aufbau der Paramterangabe ist der ‘Parameter Interface Library’ des ISDC abgeschaut, aber nicht identisch. An erster Stelle steht der Name des Parameters, dieser ist für den Benutzer interessant, nicht aber für das Programm selbst. Danach muss der Datentyp des Parameters genannt werden, die hier möglichen Einträge sind:

- s für string
- r für real
- i für integer
- b für boolean
- f für file

Da die Steuermodule in *Perl* geschrieben sind, wäre dieser Eintrag eigentlich nicht nötig – in der Skriptsprache *Perl* gibt es den Unterschied zwischen verschiedenen Skalaren wie ‘string’ oder ‘real’ nicht –, die graphische Benutzeroberfläche kann dem Benutzer damit aber die Eingabe der Werte erleichtern, indem beispielsweise ‘Schieberegler’ für Zahlen angezeigt werden.
Danach kann ein Standardwert eingetragen werden, der dem Benutzer angeboten wird. Die zwei folgenden Einträge sind die minimalen und maximalen Werte, die der Parameter annehmen darf. Sind diese nicht bekannt oder ergeben sie – wie bei einem ‘string’ – keinen Sinn, wird dieser Eintrag einfach weggelassen.
Der nächste Eintrag stellt nochmals eine Benutzerhilfe dar. Der hier eingetragene Wert kann von der Benutzeroberfläche als Hilfe eingeblendet werden, und er könnte Erklärungen zu diesem Parameter enthalten. Die Einträge ‘default’, ‘min’, ‘max’ und ‘promt’ sind optional, aber zumindest der ‘promt’-Eintrag ist für Benutzer sicher sinnvoll und sollte nicht weggelassen werden.

Ein Beispiel für einen ‘set’-Aufruf[6]:

```
xpaset -p FPPlot EMIN ##Emin,r,0,0,64,Die untere Grenze
   fuer die Anzeige des Spektrums in ADU.##
```

Bei Ausführung dieses Aufrufs würde der Benutzer nach einem Wert für ‘Emin’ gefragt, idealerweise mit dem Hinweis ‘Die untere Grenze für die Anzeige des Spektrums in ADU.’ Außerdem kann das Programm einen Schieberegler präsentieren, der auf 0 voreingestellt ist und von 0 bis 64 reicht. Der tatsächlich an die Shell übergebene Aufruf wäre dann z.B.:

```
xpaset -p FPPlot EMIN 5
```

Da ‘set’-Aufrufe aus beliebig vielen einzelnen Aufrufen bestehen können, kann hiermit auch ein Zurücksetzen von Teilen der Pipeline bei Änderung einzelner Parameter erreicht werden. Sind mehrere ‘set’-Aufrufe mit gleichem job-Eintrag vorhanden, dann werden diese nacheinander ausgeführt, dafür müssen sie aber mit einem ‘;’ enden. Ausserdem müssen die RESET-Aufrufe, wie später in den Beispielpipelines gezeigt wird, auch in der Reihenfolge in der Definitionsdatei aufgeführt werden, in der die Programme die Daten bearbeiten. Sonst kann es zu Lücken in den Daten kommen.

3.2.3 Pipelinedefinition in XML

Die zweite Möglichkeit, Pipelinedefinitionen abzuspeichern, ist die Speicherung in Form einer XML-Datei. Hierbei wird eine extrem einfache Form benutzt, so dass auf eine DTD (Document Type Definition) verzichtet werden kann.
Stattdessen werde ich hier das verwendete Format mit einfachen Worten beschreiben und eine minimale Pipelinedefiniton darstellen (siehe Abbildung 3.11), anhand derer das Erstellen einer neuen Definitionsdatei keine Probleme bereiten sollte.
Der Name des Rootelements hat keine Bedeutung, im Beispiel heisst es *pipelinedefinition*, es sollte aber vorhanden sein. Darauf folgt eine beliebige Anzahl an *goals*, diese werden einfach durch ihren Namen festgelegt: `<mygoal1>...</mygoal1>`.
Zwischen diesen, ein *goal* umschließenden Einträgen, stehen dann beliebig viele *jobs*. Auch hier wird einfach der Name des *jobs* zum Umschliessen benutzt: `<myjob1>...</myjob1>`
Jetzt folgen festgelegte Elemente. In jedem *job* muss enthalten sein: `<status>...</status>`, wobei die Punkte hier für eine ‘0’ (aktiv) oder ‘1’ (nicht aktiv) stehen. Ein weiterer Eintrag muss aus einem oder mehreren `<infiles>...</infiles>` Einträgen bestehen. Dazwischen muss je eine Datei angegeben werden, aus der der Prozess lesen soll. Für jede Datei wird ein einzelner Eintrag benötigt. Wird aus keiner Datei gelesen, dann muss trotzdem ein leerer Eintrag angegeben werden. Praktisch identische Einträge muss es auch mit *outfiles* geben.
Ein weiterer Eintrag bezieht sich auf den Typ des Aufrufs. Wie bei der Beschreibung der

[6] Die hier im Beispiel verwendete‘XPA’ wurde genauer beschrieben unter Abschnitt 3.1.3

```
<pipelinedefinition>
  <GetEvents>
    <Events>
      <status>0</status>
      <fpipeserverstart>./FPEventFeeder geiger=n
        outfile=shmem://h0 infile=myeventlist.fits
        semid=##SEMID## wlock=##WLOCK## nor=##N_O_R##
        or=##O_R,;## nsems=##NSEMS##</fpipeserverstart>
      <infiles></infiles>
      <outfiles>shmem://h0</outfiles>
    </Events>
  </GetEvents>
  <DisplayEvents>
    <DisplaySpectrum>
      <status>0</status>
      <loopstart>./FPPlot "infile=shmem://h0" inmode=4
        servermode=yes xcol=RAWY ycol=RAWX</loopstart>
      <loopstop>xpaset -p FPPlot EXIT</loopstop>
      <single>xpaset -p FPPlot PROCESS_DATA</single>
      <infiles>shmem://h0</infiles>
      <outfiles></outfiles>
    </DisplaySpectrum>
  </DisplayEvents>
</pipelinedefinition>
```

Abbildung 3.11: Minimale Pipelinedefinitionsdatei in XML

Pipelinedefinition im FITS-Format gibt es auch hier die Möglichkeiten *single*, *serverstart*, *serverstop*, *fpipeserverstart*, *fpipeserverstop* und *set*. Wie zu erwarten, muss also der so definierte Aufruf in einen entsprechend qualifizierenden Eintrag gestellt werden, beispielsweise: `<set>xpaset -p FPPlot EXIT</set>`. Es können mehrere Einträge der gleichen Art (beispielsweise zwei ‘set’-Aufrufe) für einen einzelnen *job* angegeben werden. Wird dies gemacht, dann werden alle Einträge nacheinander ausgeführt. Dies ist nur sinvoll, wenn die Aufrufe nicht getrennt werden können.
Die Reihenfolge der Einträge spielt für das Programm keine Rolle, sicher aber für denjenigen, der solch eine Datei anschaut und verstehen will, was darin enthalten ist oder fehlen könnte. Eine einheitliche Abfolge innerhalb einer Datei ist also sicher wünschenswert.
Anzumerken ist noch, dass im GUI die Möglichkeit eingebaut ist, diese Dateien zu erstellen und abzuspeichern.

3.2.4 Das Setzen von Umgebungsvariablen

Die Steuermodule bieten eine Schnittstelle, über die die graphische Benutzeroberfläche mit Hilfe von Benutzerangaben pipelinespezifische Header-Einträge koordinieren kann. Dies geschieht

über Umgebungsvariablen.
Den Steuermodulen ist es nicht möglich, Keywords direkt in die zu erstellenden Dateien der Arbeitsmodule zu schreiben, da diese ja von separaten Programmen bearbeitet werden und besonders bei externen Programmen nicht beeinflussbar sind.
Da aber die Arbeitsmodule dieser Pipeline über die Möglichkeit verfügen, in den Parameterdateien anstelle von direkten Eingaben Umgebungsvariablen anzugeben, kann so von den Steuermodulen indirekt ein Keyword in allen Arbeitsmodulen der Pipeline auf einen Schlag gesetzt werden.
Voraussetzung dafür ist, dass die Arbeitsmodule ihre Parameterdateien noch nicht gelesen haben.

3.2.5 Das Starten der verschiedenen Arbeitsmodultypen

Nachdem die interne Pipeline-Struktur erstellt wurde, kann die Pipeline gestartet werden.
Dazu werden die zu startenden Arbeitsmodule in drei Gruppen, wie in der Pipeline-Definitionsdatei angegeben, unterteilt. Die drei Gruppen sind ‘fpipeserver’, ‘server’ und ‘single’. Der ‘fpipeserver’-Typ bezeichnet dabei ein Programm, das nur gestartet und beendet werden muss. Sowohl die Dateisperren als auch das Timing dieser Sperren wird vom Programm selbst übernommen. Dieser Typ ist in der Beispiel-Pipeline in Abbildung 4.1 nur durch das FPEventFeeder-Programm vertreten, da er für die meisten Einsatzmöglichkeiten unnötig schwer zu designen und zu programmieren ist. Das Sperren von Dateien ist aber bereits in komfortabler und schneller Weise in den Steuermodulen implementiert. Nur Programme, die noch zusätzlich – z.B. mit ADC-Karten oder anderen externen Geräten – kommunizieren müssen, sind sinnvollerweise mit einer weiteren Schleife ausgestattet, die den Zustand dieser Ressource überprüft, und dann ist es ratsam, selbst das Locking zu übernehmen.
Der zweite Servertyp (‘server’) ist unter den für dieses Softwarepaket geschriebenen Arbeitsmodulen am häufigsten vertreten. Dieser Typ kann gleich gestartet werden, da er nicht sofort anfängt, die Daten zu bearbeiten, sondern wartet, bis er von einem Client, in dieser Pipeline also einem Steuermodul, benachrichtigt wird. Diese Benachrichtigung kann im Idealfall einer von vielen verschiedenen Befehlen sein. Beispiele für den Betriebsmudus als Server sind von den für diese Arbeit erstellten Programmen folgende: FPOmap, FPSpectrum, FPIntensity, FPPlot und FPCopyControl. Auch das externe Programm ds9 verfügt durch sein XPA-Interface über eine gute Möglichkeit, es in diesem Modus zu betreiben.
Der dritte Typ der Arbeitsmodule (‘single’) wird jedesmal gestartet, wenn Daten zur Bearbeitung zur Verfügung stehen. Er verfügt dementsprechend nicht über die Möglichkeit, mit externen Programmen während des Betriebs zu kommunizieren. Diese Art ist unter den externen Programmen sicher am häufigsten vertreten. Da sie aber sehr wahrscheinlich nicht auf Geschwindigkeit optimiert ist, ist sie für den Einsatz in einer Echtzeitpipeline nicht geeignet.
Die Beispiel-Pipeline enth"lt diesen Typ nicht, obwohl anfangs geplant war, ein externes xy-Plotprogramm auf diese Weise einzubinden (zu den Gründen hierfür siehe Abschnitt 3.1.1.1).
In Abbildung 3.12 ist die Reihenfolge des Starts der Arbeitsmodultypen schematisch dargestellt.

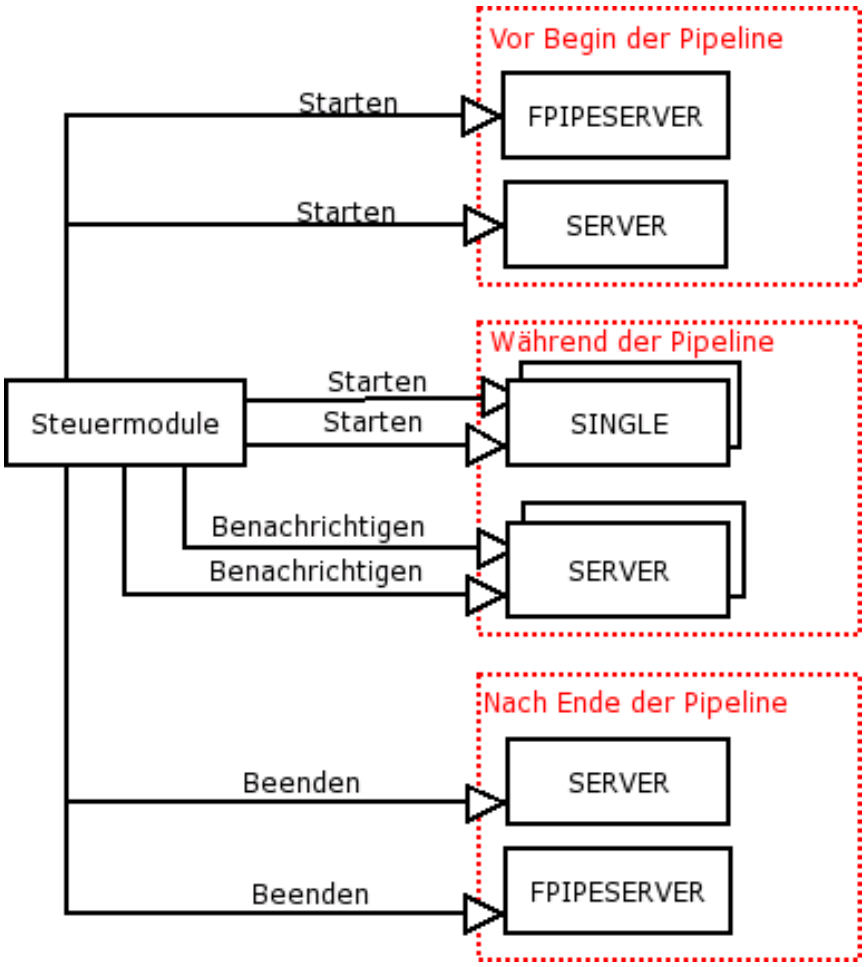

Abbildung 3.12: Schematische Darstellung des Startens und Benachrichtigens der verschiedenen Arbeitsmodultypen. Die Anzahl der Programme und Benachrichtigungen ist der Einfachheit halber stark reduziert.

3.2.6 Zur Funktionsweise der Dateisperren (Readers-Writer-Locking)

Eine wichtige Aufgabe der Steuermodule liegt in der Koordination von Dateisperren (‘file locking’).
Wie schon erwähnt, muss beim Feststellen, ob ein Dateizugriff gerechtfertigt ist (d.h., ob bereits neue Daten in der Datei vorhanden sind, ob bei einem Lesezugriff kein anderes Programm in die Datei schreibt oder ob bei einem Schreibzugriff kein anderes Programm aus der Datei liest) darauf geachtet werden, dass keine zyklische Abfrage (‘polling’) stattfindet. Zyklische Abfrage bezeichnet hier einen Vorgang, bei dem ein Programm in einer Schleife immer wieder einen Zustand (z.B. die Existenz eines Softlinks, die Größe einer Datei oder das Vorhandensein einer ‘Lock-Datei’) abfragt und dabei CPU-Zeit verbraucht, die von anderen Programmen sinnvoller genutzt werden könnte. Um beim ‘polling’ weniger CPU-Zeit zu beanspruchen, könnte das Programm zwar nach jeder Abfrage durch einen ‘`sleep`’-Befehl für eine kurze Zeit keine

CPU-Zeit benutzen, dennoch wäre auch in diesem Fall der Verbrauch an CPU-Zeit größer, als erwünscht.
In diesem Programmpaket wurde ein anderer Weg beschritten und kein ‘polling’ durchgeführt. Die Programme, die nicht arbeiten sollen, werden komplett in Schlaf versetzt und erst dann vom Kernel selbst geweckt, wenn es relevante Daten für sie zu bearbeiten gibt. Dies ist möglich durch Semaphore. Im folgenden Abschnitt werden ich darauf eingehen, welche großen Vorteile Semaphore bieten. Durch den Einsatz von Semaphoren kann nämlich komplett auf eine eigene Steuerlogik verzichtet werden, da diese stattdessen im UNIX System verankert ist.

3.2.6.1 Semaphore und System V IPC

Die ursprüngliche Version der drei IPC (Inter Process Comunication) Varianten – gemeinsamer Speicher (‘shared memory’), Nachrichtenwarteschlangen (‘message queues’) und Semaphore (‘semaphores’) – stammt aus den sibziger Jahren des letzten Jahrhunderts und war in einer internen Version von Unix (‘Columbus Unix’) implementiert. Später wurden die drei Varianten in UNIX *System V* aufgenommen, daher auch die Bezeichnung ‘SysV IPC’ [41, S. 449].
Diese Formen der IPC erstellen systemweit zugängliche Strukturen, auf denen definierte Funktionen ausgeführt werden können. Jede Struktur hat dabei eine nicht negative ganzzahlige Kennung (‘identifier’), über die die Programme auf diese Struktur zugreifen können. Diese Kennung wird bei jedem Erzeugen einer neuen SysV IPC um eins erhöht, bis die maximale Größe für ganze Zahlen (‘integer’) auf dem System erreicht ist. Danach wird wieder bei 0 angefangen.
Der größte Nachteil der SysV IPC ist, dass diese Formen keine Datei-Deskriptoren (‘file descriptors’) benutzen, und deswegen nicht mit den *poll*- und *select* Funktionen, mit denen die Existenz anderer systemweit zugänglicher Ressourcen abgefragt werden kann, zusammenarbeiten. Für die SysV IPC wurden daher neue Funktionen in den Kernel eingebaut (`msgget`, `semop`, `ipcs`, `ipcrm`, ...).
Die für die Steuermodule des Programmpakets benutzten SysV IPC sind Semaphore (der gemeinsame Speicher wird an einer anderen Stelle benutzt (siehe Abschnitt 3.1)).
Ein Semaphor besteht aus:

- einem Semaphor oder mehreren Semaphoren (es müssen weniger sein als ein systemweit festgelegter maximaler Wert)
- einer nicht negativen ganzen Zahl (kleiner als ein systemweit festgelegter maximaler Wert)

Wenn ein Semaphor andere Semaphore enthält, wird auch von Semaphorenfeldern (semaphorearrays) gesprochen. Es gibt eine systemweit maximale Anzahl von Semaphorenfeldern und Semaphoren.
Auf den Semaphoren lassen sich bestimmte Operationen ausführen. Eine wichtige Besonderheit dieser Semaphorenoperationen ist, dass sie nur ‘atomar’ ausgeführt werden, d.h. entweder die komplette Operation – die ja aus mehreren einzelnen Operationen bestehen kann – wird ausgeführt oder nicht. Es ist nicht möglich, dass nur Teile ausgeführt werden. Mit diesem

Mechanismus werden ‘race-conditions’ vermieden, die entstehen könnten, wenn zwei Prozesse versuchen, ohne atomare Operationen den Zustand einer Ressource abzufragen, und danach – bei entsprechendem Ergebnis – eine Dateisperre aufzubauen wollen. (siehe Abbildung 3.13). Die Semaphoroperation besteht aus drei Werten.

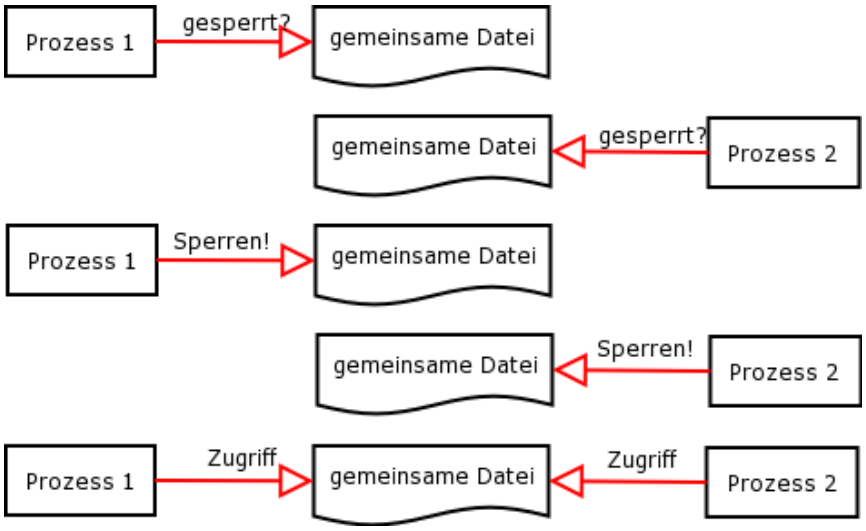

Abbildung 3.13: Wenn zwei Prozesse gemeinsam genutzte Ressourcen durch nicht-atomare Prozesse sperren wollen, kann es zu ‘race-conditions’ kommen, bei denen beide Prozesse trotz vorheriger korrekter Abfrage, ob eine Ressource genutzt wird, gleichzeitig auf diese zugreifen.

Die erste Zahl legt den Semaphorindex fest, die letzte besteht aus einem Flag, das angibt, ob die Funktion bei nicht erfolgreicher Ausführung sofort mit einem Fehler zurückkehren soll. Der mittlere Wert enthält die eigentliche Operation.
Diese läßt sich in eine der folgenden Kategorien einordnen, die jeweils unterschiedliche Verhaltensweisen zeigen:

- == 0
  dies bedeutet, dass die Operation schläft, bis das Semaphor den Wert 0 erreicht
- > 0
  diese Operation erhöht den Wert des Semaphors um den Wert der Operation, dies ist immer möglich
- < 0
  hier schläft die Operation, bis der Wert des Semaphors größer als der Absolutbetrag der Operation ist, und erniedrigt den Semaphorwert dann entsprechend

Ein Beispiel für eine Semaphoroperation ist in Abbildung 3.14 dargestellt.

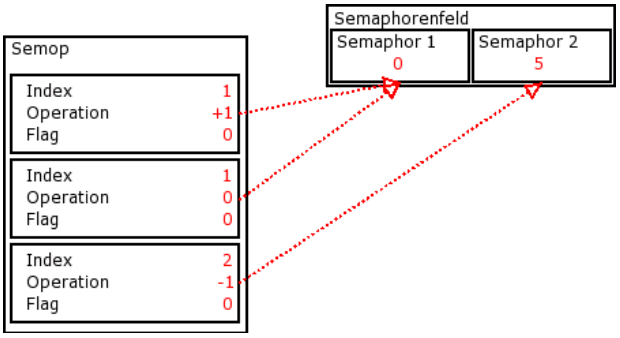

Abbildung 3.14: Beispiel für eine Semaphoroperation mit drei Unteroperationen; die hier gezeigte Situation ist ohne Schlafen des Prozesses sofort ausführbar

Die dort gezeigte Operation besteht aus drei Teilen:

- der erste Operationsteil erhöht Semaphor 1 im Feld um 1.
- der zweite Operationsteil erniedrigt Semaphor 2 um 1, wenn der Wert des Semaphors größer als 0 ist.
- der dritte Operationsteil wartet, bis Semaphor 3 den Wert 0 hat.

Ist eine dieser Operationen nicht möglich, dann schläft der Prozess, bis alle Operationen möglich sind und der Kernel den Prozess weckt. Wird der Prozess auf diese Art geweckt, dann sind alle Operationen ausgeführt.

3.2.6.2 Maximale Werte von Semaphorparametern

Für alle SysV IPC Strukturen gibt es maximale Werte, was Anzahl oder Größe von Teilen dieser Strukturen angeht.
Wie diese (vom Systemadministrator veränderbaren) Werte eingestellt sind, kann durch den Befehl ‘`ipcs -s -l`’ festgestellt werden. ‘`ipcs`’ gibt Auskunft über alle Sys V IPC Komponenten, also über gemeinsamen Speicher, Semaphorenfelder und Nachrichtenwarteschlangen. Der Parameter ‘-s’ beschränkt diese Auskunft auf Semaphore und ‘-l’ gibt die Einstellung der maximalen Werte (*legal values*) aus.
Auf einem heutigen Notebook (mit Linux-Kernel 2.6.8-24.11) sind die folgenden Werte voreingestellt:

Tabelle 3.10: Maximale Werte für Semaphore

Parameter	Beschreibung	typischer Wert	nach Stevens	maximale Werte
SEMMNI	Anzahl von Semaphorenfeldern	128	10	<= IPCMNI
SEMMSL	Semaphore pro Semaphorenfeld	250	25	<= 8 000
SEMMNS	Semaphore systemweit	32000	60	<= INT_MAX
SEMOPM	Operationen pro Semop-Aufruf	32	10	<= 1 000
SEMVMX	Semaphorwert	32767	32767	<= 32767

Die hier gezeigten Werte geben typische heutige Werte für Semaphore wieder. Die nach Stevens zitierten Werte stellen typische Werte von 1993 dar [41, S. 459]. Zu finden sind diese Werte im Include File*sem.h*. Um abschätzen zu können, was dies für mögliche Pipelines bedeutet, und ob diese Werte begrenzende Faktoren für die Pipeline darstellen, hier ein Blick auf deren typische Werte.
Pro Pipeline ist ein Semaphorenfeld nötig. In diesem Feld wird für jeden Lesezugriff und jeden Schreibzugriff ein Semaphor benötigt. Für die in Abbildung 4.1 dargestellte Pipeline werden also 13 Semaphore benutzt.
Für die Operationen pro Semop-Aufruf gilt: ein Prozess, der einen Block aufbaut, benötigt eine Operation je Schreibzugriff auf eine Datei und eine weitere für jeden Prozess, der diese Dateien auslesen möchte. Desweitern benötigt er eine Operation pro Lesezugriff und eine weiter für jeden Prozess, der auf eine dieser Dateien schreibend zugreifen möchte. Für die in Abbildung 4.1 dargestellte Pipeline werden maximal vier Semaphoroperationen in einem Aufruf ausgeführt.
Der größte Semaphorwert entspricht der Anzahl der lesenden Prozesse.
Es kann also gesagt werden, dass keiner der oben genannten Werte einen begrenzenden Faktor für Aufbau und Betrieb einer solchen Pipeline darstellt.

3.2.6.3 Vergleich mit anderen IPC Formen

Eine Alternative zu den Semaphoren wäre das sogenannte '*record locking*' gewesen.
Auch hier sind sowohl geteilte Lesesperren (shared read locks) als auch komplette Schreibsperren (exclusive write locks) möglich. Zusätzlich lassen sich hier Teile von Dateien sperren. Um *record locking* als Mechanismus zum Sperren von gemeinsamen Speichersegmenten zu nutzen, müsste eine Datei erzeugt werden, in diese Datei müssten Bytes geschreiben werden, die Bytes zeigen die Nutzung von Dateien an, und werden dann mit *record locking* gesperrt.
In Stevens (1993) ist ein Vergleich zu finden, bei dem sowohl mit Semaphoren, als auch mit *record locking* 10 000 mal eine Sperre aufgebaut und wieder abgebaut wurde [41, S. 463]. Die dazu benötigten Zeiten sind in Tabelle 3.11 zu finden.

Tabelle 3.11: Vergleich der zum 10 000maligen Sperren und Entsperren benötigten Zeiten nach Stevens (1993) [41, S. 463]. Diese Zeiten zeigen einen bis zu 50% größeren CPU-Zeitverbrauch bei Benutzung von ‘record locking’.

	SPARC, SunOs 4.1.1			80386, SVR4		
Operation	User $[s]$	System $[s]$	Clock $[s]$	User $[s]$	System $[s]$	Clock $[s]$
semaphores with undo	0.9	13.9	15.0	0.5	13.1	13.7
advisory record locking	1.1	15.2	16.5	2.1	20.6	22.9

Wie in Tabelle 3.11 zu sehen ist, ist die Benutzung von Semaphoren zeitökonomisch von Vorteil. Da die schon genannten Nachteile der Sys V IPC, in der Art, wie sie von den Steuermodulen genutzt wird, nicht wirksam werden, sind Semaphore in meiner Software sinnvoll.
Die Steuermodule sorgen dabei für die Kommunikation mit den Semaphoren, führen die ‘lock’- oder – nachdem die Datei vom Arbeitsmodul bearbeitet wurde – die ‘unlock’-Operation durch. Dazwischen schicken sie Befehle an die Arbeitsmodule, dass jetzt an der Datei gearbeitet werden kann, falls die Arbeitsmodule über einen Servermodus verfügen, ansonsten starten sie die Arbeitsmodule.
Durch diese Art der Arbeitsaufteilung ist es ohne Probleme möglich, externe Programme einzubinden, ohne dass an deren Code eine Veränderung vorgenommen werden müsste, damit diese die ‘locking’-Operationen selbst durchführen.

3.2.6.4 Das Readers-Writer-Locking im Detail

Wenn ein Arbeitsmodul auf eine Datei oder auf eine andere Ressource zugreifen möchte – die Art der Ressource spielt für das ‘locking’ keine Rolle – dann muss dies in der Pipelinedefinition (siehe Abschnitt 3.2.1) festgelegt sein. Das Steuermodul erstellt dann nach diesen Angaben und den Angaben aller anderen Programme die benötigten Semaphoroperationen.
Wie bereits gesagt, besteht eine Semaphoroperation aus drei Werten: der erste gibt den Semaphorindex im Feld an, der zweite die eigentliche Operation und der dritte sagt, ob der Prozess schlafen soll, bis die Operation möglich ist oder ob – wenn dies nicht der Fall ist – sofort ein Fehler zurückgemeldet werden soll (siehe Abbildung 3.14). Dies ist die Stelle, an der eingestellt ist, dass der Kernel selbst die Ressourcenverwaltung übernimmt, und nicht ein selbstgeschriebenes Programm ‘polling’ durchführt. Dies spart der Pipeline wertvolle CPU-Zeit. Dadurch, dass die gesamte Organisation der kompletten Pipeline durch den Einsatz von Semaphoren dem UNIX-Kernel übergeben wird, kann nicht nur auf eine eigene Steuerlogik verzichtet werden, sondern auch die von den Steuermodulen verbrauchte CPU-Zeit wird deutlich reduziert. Dies ist einer der großen Vorteile, den meine Steuermodule bieten.
Die in meinem Softwarepaket verwendete Art des ‘Readers-Writer-Locking’ macht es möglich, dass mehrere Prozesse zur gleichen Zeit lesen (deswegen ‘Readers’ im Plural) aber nur ein Prozess schreiben kann (‘Writer’ im Singular). Während des Schreibens darf natürlich auch kein Prozess lesen.

Um dies zu erreichen, benötigt jede Datei, in die geschrieben werden soll, ein Semaphor für den Schreibzugriff und jeder lesende Prozess ebenfalls ein Semaphor, über das kontrolliert wird, dass dieser Prozess die für ihn bestimmten Daten auch wirklich gelesen hat und dass diese überschrieben werden können. Ich werde bei der weiteren Beschreibung und in den dazugehörenden Graphiken dem Semaphor, das für den Schreibzugriff zuständig ist, den Index W geben, die Semaphore für die Lesezugriffe erhalten die Indizes R_i für den i-ten Prozess.
Der einfachste Fall, der trotzdem alle Möglichkeiten dieser Art des Readers-Writer-Lockings zeigen kann, ist folgender: ein Prozess schreibt in eine Datei, aus der zwei andere Prozesse lesen. Die dazu nötigen Operationen und die Zustände, bei denen diese Operationen möglich sind, werden in Abbildung 3.15 dargestellt.

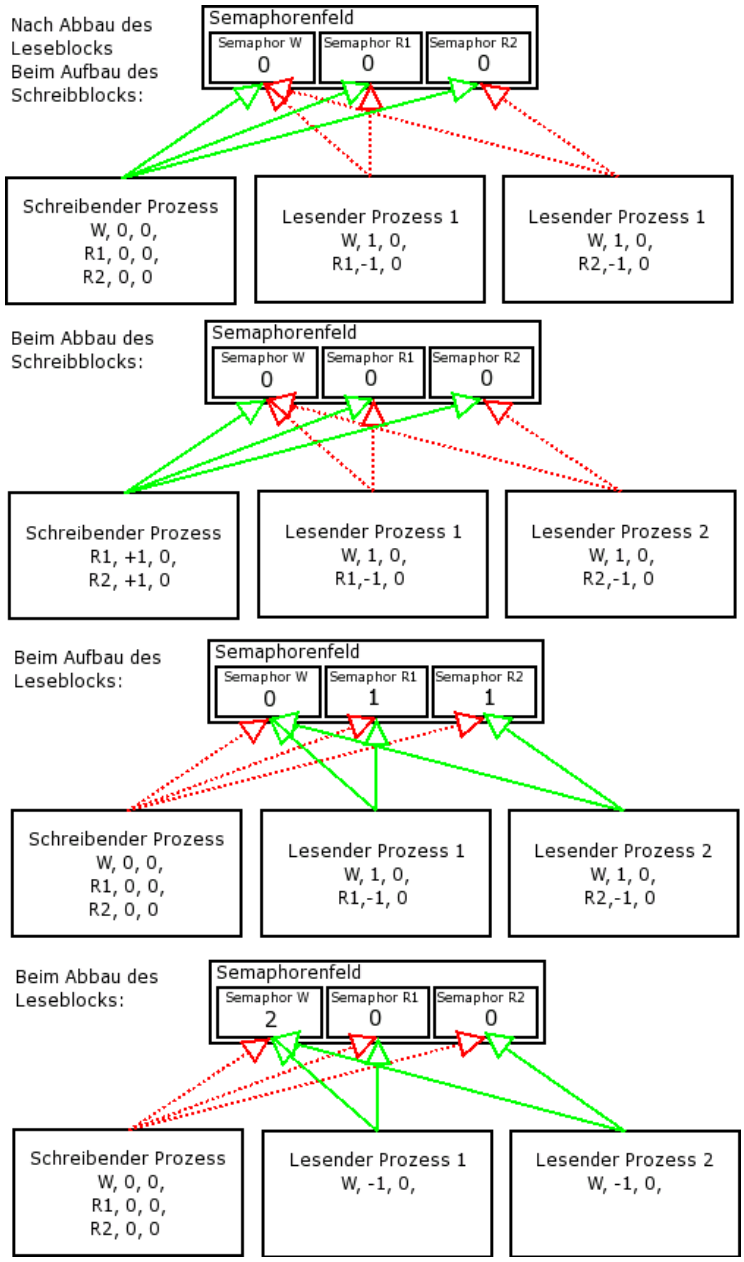

Abbildung 3.15: Dieses Beispiel zeigt die Funktionsweise des Readers-Writer-Lockings anhand eines schreibenden und zweier lesender Prozesse. Grüne Pfeile deuten an, dass diese Operation möglich ist. Rote Pfeile zeigen einen wegen nicht möglicher Operation schlafenden Prozess an.

Diese Art des Readers-Writer-Lockings erlaubt den Aufbau und Betrieb auch von komplexeren Pipelines. Daher kann auf die Implementierung der Option, dass mehrere Prozesse schreibend auf eine Datei zugreifen, verzichtet werden. Die Möglichkeit, dass dies zu einer Deadlock-Situation führt, besteht mit der bisherigen Implementierung nur in zirkulären Pipelines. Wäre die Option mehrerer Schreiber eingebaut, genügten dafür aber schon bei wesentlich einfachere Pipelines für eine mögliche Deadlock-Situation. Zusätzlich lässt sich sagen, dass ich beim Design der Beispielpipelines nie auf eine Situation gestoßen bin, in der es wünschenswert gewesen wäre, dass mehrere Prozesse auf eine Datei schreibend zugreifen.
Für den Umgang mit Semaphoren in meiner Software habe ich zwei kleine Bibliotheken geschrieben. Die in C geschriebene Bibliothek, die von FPEventFeeder für das mit meiner Software kompatible Readers-Writer-Locking benutzt wird, steht natürlich auch anderen Programmen zur Verfügung. Sie befindet sich in den Dateien `fpRWlocking.c` und `fpRWlocking.h` und enthält folgende Unterprogramme und Strukturen:

- `semaphore_infos`
  enthält Informationen über die Semaphor-ID, ein eindeutiger Wert, der dem Semaphorenarray vom System oder vom Benutzer bei der Erzeugung zugeordnet wurde, über die Anzahl der Semaphore im Semaphorenarray, die Anzahl und genauen Indices der Semaphore, die für Schreib- und Lesezugriffe zuständig sind sowie über die Anzahl der Operationen und die Operationen selbst, die beim Lese- oder Schreibzugriff ausgeführt werden müssen.
- `void create_flag_ops(semaphore_infos *sem)`
  dieses Unterprogramm füllt eine bereits vorausgefüllte '`semaphore_infos`'-Struktur vollständig aus, so dass sie von '`lock_flag`' und '`unlock_flag`' benutzt werden kann.
- `int lock_flag(semaphore_infos *sem)`
  diese Funktion greift auf die '`semaphore_infos`'-Struktur zu, um alle benötigten Information entweder für eine absolute Dateiblockade bei einem Schreibzugriff oder für eine selektive Dateiblockade bei einem Lesezugriff aufzustellen.
- `void unlock_flag(semaphore_infos *sem)`
  hier wird die von '`lockflag`' erstellte Dateiblockade wieder gelöst, entweder vollständig, falls eine absolute Dateiblockade erstellt wurde, oder nur partiell, falls eine selektive Dateiblockade erstellt wurde (siehe Abschnitt 3.2.6).
- `void get_flag_control(semaphore_infos *sem)`
  die Funktion sucht über den Semaphorenschlüssel die Systemkennung für das Semaphor.

Eine vergleichbare Bibliothek gibt es in *Perl* für Readers-Writer-Locking. Sie wird von den ebenfalls in *Perl* geschriebenen Steuermodulen benutzt. Die darin enthaltenen Funktionen sind:

- `create_flags("job_structure"=>$js ,"file_flag_hash"=>$ffh)`
- `lock_flags ("flag"=>$f, "lock_op"=>$lo)`
- `unlock_flags("flag"=>$f, "unlock_op"=>$ulo)`
- `remove_flags("flag"=>$f)`
  `remove_flags("file_flag_hash"=>$ffh)`
- `create_flag_ops("job_structure"=>$js, "file_flag_hash"=>$ffh)`

Diese Funktionen erwarten als Parameter sogenannte ‘hashes’ mit den angegebenen Schlüsseln. Die dazugehörigen Werte sind natürlich nur Stellvertreter für die wirklich im Programm verwendeten Namen.
Zu erwähnen bleibt noch, dass bei der hier vorgestellten Benutzung der Semaphore keinerlei Kenntniss der Steuermodule über die tatsächliche Reihenfolge der Arbeitsmodule innerhalb der Pipeline nötig ist. Das einzige, was die Steuermodule über den Ablauf wissen müssen, ist, welche Ressourcen die einzelnen Programme zu Lese- oder Schreibzwecken benötigen, nicht aber deren Reihenfolge innerhalb der Pipeline. Dass die Pipeline mit dem richtigen Programm startet, ist durch die Initialisierung des Semaphorenfeldes mit 0 in jedem Semaphorwert gewährleistet. Dadurch können nur Programme ausgeführt werden, die in eine Ressource schreiben, nicht aber auf eine Ressource der Pipeline lesend zugreifen müssen.

3.3 Die graphische Benutzeroberfläche und ihre Bedienung

Obwohl das GUI (Graphical User Interface) keineswegs das Herzstück dieses Softwarepaketes ist, und nur sehr wenige Funktionen vom GUI selbst ausgeführt werden – es wurde möglichst viel Arbeit in die Steuermodule verlegt – ist dies doch der Teil der Software, der für den Benutzer sichtbar ist und somit auch der Teil, aufgrund dessen der Benutzer sein erstes Urteil über die Brauchbarkeit der Software fällt. Deshalb habe ich mich bemüht, diese Oberfläche möglichst benutzerfreundlich zu gestalten. Festzuhalten bleibt jedoch, dass es gewiss nicht vom optischen Eindruck der widgets (Window Gadgets) abhängt, ob die Software wirklich gut arbeitet und auch für Echtzeitanwendungen im Labor brauchbar ist.
Das Hauptfenster besteht aus drei Teilen, dem Menü, einem kleinen Informationsteil und einer Statusleiste.

- In der Statusleiste sollte immer ein Kommentar der momentan durchgeführten Aktion zu sehen sein.
- Der Informationsteil in der Mitte des Fensters zeigt den Namen der Pipeline-Definitionsdatei an, sofern eine Datei geöffnet wurde. Die zweite Information gibt an – allerdings erst nach Beenden der Pipeline – wie lange diese gelaufen ist, also die Zeit zwischen den Befehlen START und STOP.
- Die Menüs ‘File’, ‘Actions’, ‘Windows’ und ‘Help’, sind sozusagen abnehmbar (‘tearoffs’) und können somit als eigenständige Fenster aufgeklappt gehalten werden. Dies wird in Abbildung 3.16 gezeigt.

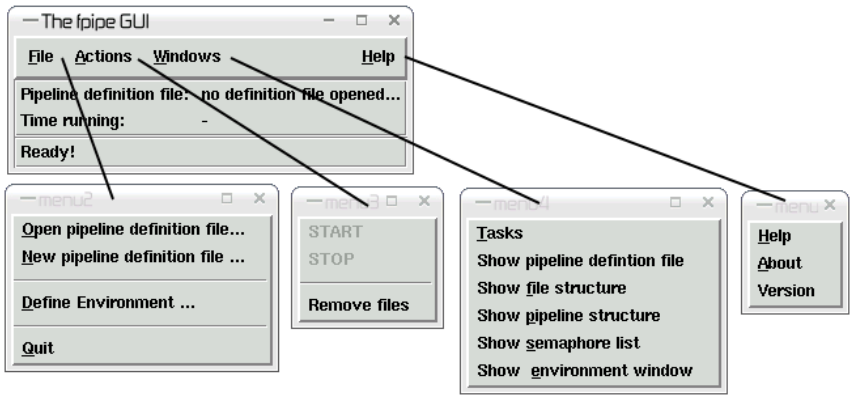

Abbildung 3.16: Das GUI Hauptfenster mit losgelösten Menüs

Über das ‘File’-Menü kann in einem Öffnen-Dialog eine Pipeline-Definitionsdatei geladen werden. Ein alternativer Weg die Datei zu öffnen ist, sie als ersten Parameter beim Aufruf der GUI anzugeben.
Ab diesem Zeitpunkt ist auch der Start-Knopf im Actions Menü anklickbar. Der ‘New Pipeline definition file…’-Knopf öffnet ein Fenster, das bei der Erstellung von Pipeline-

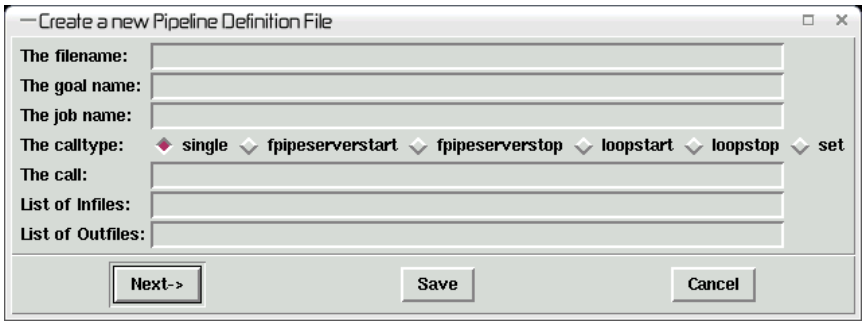

Abbildung 3.17: Der ‘Wizard’ zum Erstellen einer Pipeline-Definitionsdatei

Definitionsdateien helfen soll. Da besonders das XML-Format aber mit jedem besseren Editor ganz einfach von Hand erstellt werden kann (mit der Beschreibung und Beispiel-Pipelinedefinition als Hilfe), empfehle ich nicht das Erstellen mit diesem ‘Wizard’. Trotzdem hier noch ein paar Bemerkungen:
der ‘Next->’-Knopf verankert die bisherigen Angaben in einer Struktur. Ich habe dies so programmiert, dass ein Rückgängigmachen nicht möglich ist. Wenn der Eintrag also nicht übernommen werden soll, dann sollte dieser Knopf nicht gedrückt werden. Soll anstelle eines weiteren Eintrags das bisherige gespeichert werden, dann ist der ‘Save’-Knopf die richtige Wahl und nicht der ‘Next->’-Knopf, da dieser einen weiteren Eintrag erwartet.
Wenn die Pipelinedefinition im XML-Format gespeichert werden soll, dann muss der Dateiname auf ‘.xml’ enden, beim FITS-Format muss dieser auf ‘.fits’ enden. Diese Einschränkungen könn-

ten sicher ohne Probleme, wegen der graphischen Benutzeroberfläche aber nur mit einigem Zeitaufwand, behoben werden.
Über den ‘Define Environment. . . ’-Knopf lässt sich das entsprechende Fenster öffnen. In diesem

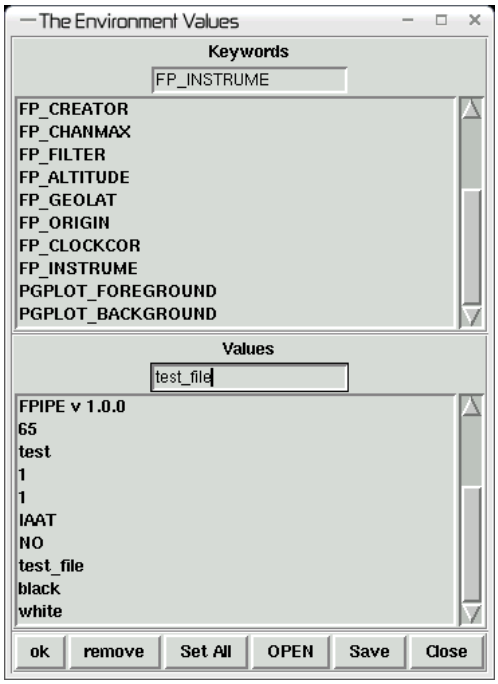

Abbildung 3.18: Das Fenster zum Setzen von Umgebungsvariablen

lassen sich Umgebungsvariablen setzen. Dies kann insofern zum Setzen von Header Keywords genutzt werden, als alle in dieser Diplomarbeit erstellten Arbeitsmodule die PIL (Parameter Interface Library) benutzen. Durch diese Bibliothek lassen sich in der Parameter-Datei anstelle von Eingaben auch die Werte von Umgebungsvariablen einlesen. Dies wird standardmässig von den Arbeitsmodulen für das CREATOR, INSTRUME, FILTER, usw. Keyword genutzt.
Der Grund, warum die Keywords nicht direkt gesetzt werden können, ist einfach die Trennung der Arbeit auf den Daten und die Koordination der Pipeline, was, wie gesagt, wichtig für die Integrierbarkeit externer Programme ist.
Die so eingetragenen Variablen lassen sich im XML-Format speichern und lesen, so dass sie nicht jedesmal eingetippt werden müssen. Ausserdem ist zu erwähnen, dass die Umgebungsvariablen erst durch den Knopf ‘Set all’ gesetzt werden. Dies ist so eingestellt, damit Dateien geöffnet werden können, ohne dass bisherige Keywords überschrieben werden. Durch die Angabe einer solchen Datei als zweiten Parameter beim Aufruf des GUI lassen sich die darin enthaltenen Variablen schon beim Startup des Programms setzen.
Das ‘Actions’-Menü hat die zwei Knöpfe START und STOP, die die Pipeline starten und stoppen oder genauer gesagt die dazugehörigen Steuermodule aufrufen.
Unter dem ‘Help’-Menü verbergen sich die kleinen Knöpfe ‘Help’ – sagt nur, dass die Hilfe in

dieser vorliegenden Arbeit oder direkt bei mir zu suchen ist, ‘About’ – sagt, woher dieses Programm kommt und ‘Version’– zeigt die Version dieses Programmes an.
Über das ‘Windows’-Menü lassen sich verschiedene Informationen über die Pipeline anschauen. Viele dieser Informationen sind eher zum Testen einer neu aufgesetzten Pipeline von Nutzen als bei einer bereits komplett funktionierenden.

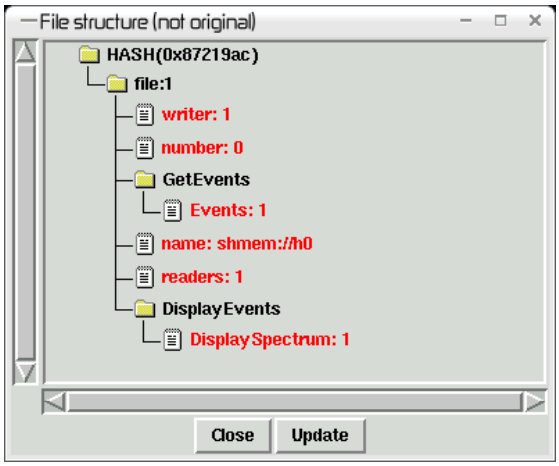

Abbildung 3.19: Das Informationsfenster für die Dateienbelegung innerhalb der Pipeline

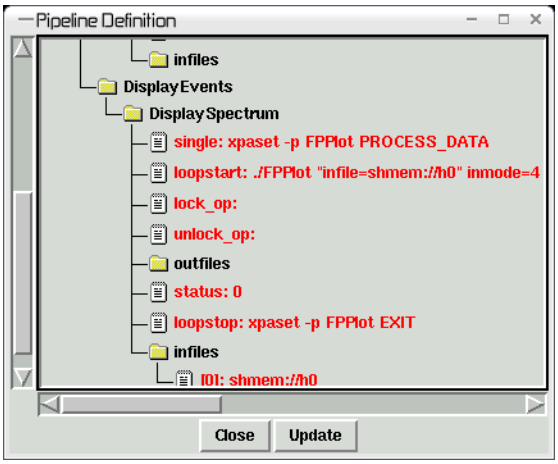

Abbildung 3.20: Das Informationsfenster für die eingelesenen Pipelinestrukturen

Die Fenster sind im einzelnen:

Pipeline definition file – öffnet die gewählte Datei entweder mit emacs, falls sie im XML-Format ist, oder mit fv, falls es sich dabei um eine FITS-Datei handelt.

File Structure – stellt die Struktur dar, die die (internen) Informationen über Dateien enthält. Je Datei gibt es eine Unterliste, die Informationen enthält über die Anzahl der schreibenden und der lesenden Prozesse, deren Namen, über den Namen der Datei und die Positionen der Semaphore, die zum Sperren der Datei benutzt werden.

Pipeline Definition – zeigt alle Informationen, die nach dem Einlesen und Bearbeiten der Pipelinedefinitionsdatei in der internen Struktur enthalten sind.

Semaphore Information – zeigt die Werte des Semaphorenfeldes, seine ID, die Anzahl der Programme, die momentan auf einen bestimmten Wert der Semaphore warten und die Anzahl der Semaphore, die auf den Semaphorenwert Null warten; gibt ausserdem noch einige zusätzliche Informationen, die mit dem Semaphorenfeld assoziiert sind.

Define Environment – öffnet das gleiche Fenster, das durch ‘File->Define Environment…’ geöffnet wird.

Das weitere Fenster, das im Windows Menü geöffnet werden kann ist das ‘task’-Fenster. Dieses Fenster ist für die Pipeline besonders wichtig.

3.3.0.5 Das Task Fenster

Dieses Fenster zeigt als Namen alle Tasks – d.h. alle Programme – an, die in der Pipeline laufen. In einem zweiten Teil des Fensters ist je ein Knopf pro ‘set’-Aufruf der Pipelinedefinition ent-

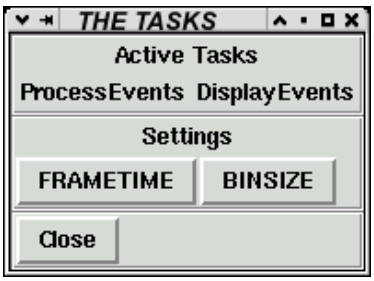

Abbildung 3.21: Das Tasks Fenster

halten. Diese ‘set’-Aufrufe sind dazu da, um die Pipeline, abgesehen vom Starten und Beenden, zu steuern.
Über diese Befehle können beispielsweise die Bingrösse von Lichtkurven, die Framezeiten oder Dateinamen geändert werden. Die Set-Aufrufe in der Pipelinedatei müssen dazu bestimmte Informationen enthalten, die das GUI interpretieren kann. Diese Informationen wurden in Abschnitt 3.2.2 näher beschrieben. Im folgenden zeige ich, wie diese Informationen im GUI in eine Benutzeroberfläche übersetzt werden.
Wird im ‘set’-Aufruf eine Eingabe im Integerformat gefordert, dann wird im dazugehörigen Set-Fenster ein Schieberegler und ein Eingabefeld angezeigt, zusammen mit dem Namen der einzutragenden Information und einem vom Benutzer mitgegebenen Zusatztext.

Abbildung 3.22: Eine Informationsanforderung im Integerformat wird in Form eines Schiebereglers dargestellt

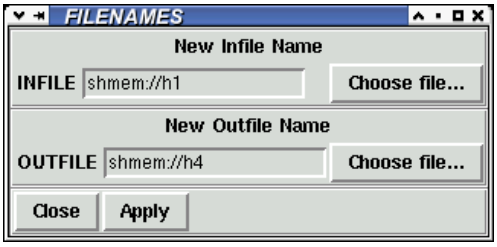

Abbildung 3.23: Eine Informationsanforderung im Dateiformat

Werden dagegen Informationen im Float- oder String-Format angefordert, werden diese in einem Eingabefeld – auch wieder mit Namen der Information und einem Zusatztext – angezeigt.
Wenn bei einem ‘set’-Aufruf mehrere Informationen vom Benutzer angefordert werden, sind diese in einem einzelnen Fenster dargestellt. Durch den jeweils vorhandenen ‘Apply’-Knopf kann dieses Kommando dann ausgeführt werden.

Kapitel 4

Momentaner Stand der Entwicklung: Tests und abschließende Bemerkungen

4.1 Allgemeines zu den Tests

Um abschätzen zu können, wie gut der Ansatz eines allgemeinen Pipelinedrivers mit variablen Arbeitsmodulen unter den gegenwärtigen Hardwarebedingungen funktioniert, habe ich einige Messungen mit verschiedenen Pipelines durchgeführt. Für Testzwecke ist es immer sinnvoll, den Testaufbau (in meinem Fall eine Abfolge von hintereinandergeschalteten Arbeitsmodulen, die durch Steuermodule dirigiert werden) so überschaubar wie möglich zu halten. Nur dann sind die Tests kontrollierbar und Fehlerquellen leicht identifizierbar. Vereinfachungen zu Testzwecken können im Fall meines Softwaretests einerseits über diesen absichtlich unkomplizierten Aufbau der Pipeline erfolgen, andererseits aber auch über die Zusammensetzung der Events und über die Rate, in der sie dem Programm zur Verfügung gestellt werden.
Allen drei Testpipelines, die ich hier aufgestellt habe und weiter unten beschrieben werde, wurden vereinfachte Daten zur Bearbeitung übergeben, deren Entstehung man sich folgendermaßen vorstellen muß: auf einem (nur fiktiven) CCD werden immer eine gewisse Zeit lang Daten bzw. Events gesammelt. Die Zeitspanne hängt normalerweise vom Betriebsmodus des CCDs ab, ist aber dann im Betrieb immer gleich lang und wird Framezeit genannt. Dieser Parameter kann im FPEventFeeder, der ein solches CCD simuliert, eingestellt werden. Ich benutze diese Größe aber nur, um regeln zu können, wie viele Events pro Sekunde ankommen. Dazu gebe ich dem FPEventFeeder eine konstante Abfolge von genau einem Event pro Frame. Auch das erfolgt aus Gründen der Vereinfachung, die der Kontrollierbarkeit dient. Interessant für die Frage nach der Funktionstüchtigkeit der Pipeline ist nun, wie sie sich verhält, wenn ihr Daten mit unterschiedlicher Geschwindigkeit übergeben werden. Deshalb wird in den Tests die Eventrate variiert und untersucht wie die Pipeline mit immer höheren Eventraten, also mit immer mehr Events pro Sekunde, zurecht kommt. Für die Beurteilung der Pipeline ist dann auch von Bedeutung, wie die Systemauslastung (der Load) ansteigt.
Während des Durchlaufens der Pipeline dürfen bei der Datenverarbeitung keine Fehler auftreten, es sollten zum Beispiel keine Datenlücken vorkommen oder unerklärlich abweichende Werte auftauchen. Im Voraus sollte deshalb im besten Fall bereits klar sein, welches Ergebnis bei

erfolgreichem Durchlauf zu erwarten wäre. Dadurch, dass ich immer ein Event pro Frame vorgebe, kann genau dies überprüft werden: die Lichtkurve sollte am Ende eine Gerade darstellen.
Ich werde jetzt kurz die drei Testpipelines und die interessanten Messgrössen vorstellen.
Die drei Pipelines, die ich für die Messungen aufgestellt habe, bauen folgendermaßen aufeinander auf:

1. Die erste Testpipeline ist noch verhältnismäßig einfach aufgebaut und erfüllt sozusagen die Mindestanforderungen. Sie bekommt vom FPEventFeeder Daten zur Verfügung gestellt, leitet diese weiter an FPOmap und stellt dem Benutzer am Ende ein Spektrum und eine Lichtkurve dar. Außerdem liegt ein Intensitätsbild als Datei vor.
2. Für die zweite Pipeline habe ich dem Aufbau der ersten Testpipeline ein weiteres Programm – FPCopyControl – hinzugefügt, da dieses für viele Benutzerinteraktionen benötigt wird, bei denen die Pipeline neu anfangen muss zu arbeiten. In dieser Pipelinevariante sind aber noch keine Interaktionen in der Pipelinedefinition enthalten.
3. Die Möglichkeit zur Benutzerinteraktion wird dann in der dritten Testpipeline eingeführt.

Bei einer Pipeline, die in einem Labor in Echtzeit arbeiten soll, gibt es verschiedene Größen, die man messen kann, um zu sehen, wie gut die Anforderungen erfüllt werden.
Eine interessante Größe ist der zeitliche Abstand zwischen der Übergabe eines Events an die Pipeline und der Darstellung der aus diesem Event abgeleiteten Informationen (ich nenne dies die Durchlaufdauer). Da es Ziel dieser Software ist, der Echtzeit möglichst nahe zu kommen, wurde zu Beginn der Diplomarbeit eine Grenze von höchsten 2 Sekunden für einen Durchlauf festgelegt.
Die Durchlaufdauer kann über die Anzahl der Events innerhalb der Pipeline bestimmt werden, da ja die Rate der Events eingestellt werden kann. Ein stark vereinfachtes – und was die Eventrate angeht deutlich reduziertes – Beispiel die Durchlaufdauer zu ermitteln wäre folgendes: hält man die Pipeline im Test z.B. zu irgendeinem Zeitpunkt an und es befinden sich noch 40 Events in ihr, dann beträgt bei einer eingestellten Rate von 10 Events pro Sekunde die Durchlaufdauer für ein Event 4 Sekunden. Für die erhoffte maximale Durchlaufdauer von 2 Sekunden dürften sich aber bei dieser Rate höchsten 20 Events in der Pipeline befinden. Im FPEventFeeder wird die Rate über die Framezeit eingestellt. Sind z.B. 8 Events pro Eventschub in einer Datei enthalten und gibt es 2 dieser Dateien in der Pipeline, dann folgt daraus, dass die Durchlaufdauer etwa $(2 + 0,5) * 8 * frametime$ ist ('+0,5' kommt daher, dass das FPEventFeeder-Programm einen internen Buffer hat, in dem ähnlich viele Events stehen können, wie in einem Schub an die Pipeline übergeben werden, im Mittel aber eher die Hälfte).
Die Testergebnisse zur Durchlaufdauer sollen zeigen, ob der selbstauferlegt zeitliche Rahmen bei variierender Eventrate eingehalten werden kann. Deshalb wurde sowohl die erste als auch die zweite Testpipeline auf ihre Durchlaufdauer hin untersucht. Damit besteht die Möglichkeit, die Testergebnisse einer relativ einfachen Pipeline mit den Testergebnissen einer komplizierteren Pipeline zu vergleichen.
Eine weitere Größe ist die Systemauslastung. Eine Abschätzung für die Systemauslastung ist der Load, also die Anzahl der Prozesse, die gleichzeitig die CPU benutzen wollen.
Der Load kann entweder über das Programm 'top' beobachtet oder mit dem Programm 'xload'

dargestellt werden. Beide Möglichkeiten werde ich bei den Tests der Beispielpipelines nutzen. Wie schon in der Einleitung erwähnt, wäre ein Load von unter 1 wünschenswert. Dann hätte man noch Spielraum, um zusätzliche Programme einzubinden oder kurzzeitig höhere Eventraten verarbeiten zu können.
Der Load spielt auch für Benutzerinteraktionen eine Rolle, da bei einigen Parameteränderungen alle bisherigen Events neu verarbeitet werden müssen, zusätzlich zum normalen Betrieb. Ein hoher Load heisst aber nicht automatisch, dass die Pipeline nicht mehr Events verarbeiten könnte, da die benötigte Rechenleistung nicht nur von der Anzahl der Events abhängt. Viele der Operationen, die in der Pipeline durchgeführt werden sind Lese-, Schreib- und Darstellungsoperationen. Diese sind pro Datei (also pro Eventschub) etwa konstant oder sie steigen – wegen der FITS-Dateigröße, die nur in Blöcken von 2880 Bytes (2880 mal 8-bit) anwächst – in Stufen an. Wenn also mehr Events pro Schub verarbeitet werden, sinkt die benötigte Rechenleistung pro Event.
Ausserdem sorgen die dezentrale Dateisperrung und das Buffering (das Sammeln von Events) des FPEventFeeders automatisch dafür, dass die Größe der Eventschübe der momentan verfügbaren Rechenleistung angepasst wird.
Der Load wurde in den Messungen bei allen drei Pipelines beobachtet. Die Ergebnisse zur Durchlaufdauer und zum Load der Pipelines werden später in den Tabellen zusammengefasst.
Eine dritte für die Tauglichkeit einer Pipeline wichtige Größe ist die Verteilung der Rechenzeit innerhalb der Pipeline.
Wird besonders viel Rechenzeit von einem einzelnen Programm verbraucht, dann führt dieses entweder besonders schwierige Rechnungen durch oder es ist nicht optimal programmiert. Die Verteilung der Rechenzeit habe ich für Pipeline Nummer 1 und 2 mit dem Programm ‘`top`’ beobachtet.
Die für die Messungen verwendete Hardware ist ein AMILO Pro mit Intel Pentium Centrino (1,5 MHz) und einem Arbeitsspeicher von 512MB. Die Tests wurden ohne besondere Konfiguration gemacht, die XPA_METHOD-Einstellung ist ‘unix’.
In den Erklärungen und den Tabellen zu den Messergebnissen tauchen einige Größen mehrfach auf, die ich hier abschließend deswegen noch einmal zusammenfassen möchte:

Zeit – stellt die real verstrichene Zeit (im Gegensatz zur ‘system’ oder ‘user’ Zeit) zwischen dem Drücken des Start- und des Stop-Knopfes dar. Diese wird durch die GUI gemessen.

Events – ist die Anzahl der in der Lichtkurve, dem Spektrum und dem Intensitätsbild verarbeiteten Events. In der Pipeline stehen noch weitere, unverarbeitete Events, da sie abgeschaltet wurde, solange FPEventFeeder noch Events weitergab. Ansonsten würden natürlich alle Events verarbeitet.

Frametime – ist der Parameter des FPEventFeeder Programms. Da in der von diesem Programm eingelesenen Beispiel-Eventliste ein Event pro Frame enthalten ist, stellt dies den reziproken Wert der Anzahl der Events pro Sekunde dar, und stimmt (grob[1]) mit den aus Zeit und Events berechenbaren Werten überein.

Eventrate – ist über die Frametime einstellbar und ergibt sich, wegen der benutzen Eventliste

[1] Hier spielen die Events in der Pipeline und ähnliches eine Rolle.

bestehend aus einem Event pro Frame, aus dem Kehrwert der Frametime.

maximaler Load – gibt einen Überblick über den auf dem System herrschenden Load. Dieser wurde mit Hilfe von ‘top’ abgelesen und stellt den Höchstwert dar, wenn nicht anders angegeben. Die Graphiken des Loads wurden von dem Programm xload erstellt.

Eventschub – gibt an, wie groß die Anzahl der Events ist, die vom FPEventFeeder-Programm an die Pipeline übergeben wird. Diese Anzahl gibt außerdem an, wie groß ein Buffer mindestens sein müsste, der die vom Detektor kommenden Events zwischenspeichert, bevor die Pipeline diese verarbeiten kann. Die hier angegebenen Werte stellen nur Momentaufnahmen dar, die sich beim Beenden der Pipeline zeigten.

4.2 Erstes Beispiel für eine mögliche Pipeline

Das erste und einfachste Beispiel für einen möglichen Aufbau der Arbeitsmodule ist in Abbildung 4.1 dargestellt:
Der Aufbau ist auch als Definitionsdatei in meinem Softwarepaket enthalten (‘pipe_def.xml’),

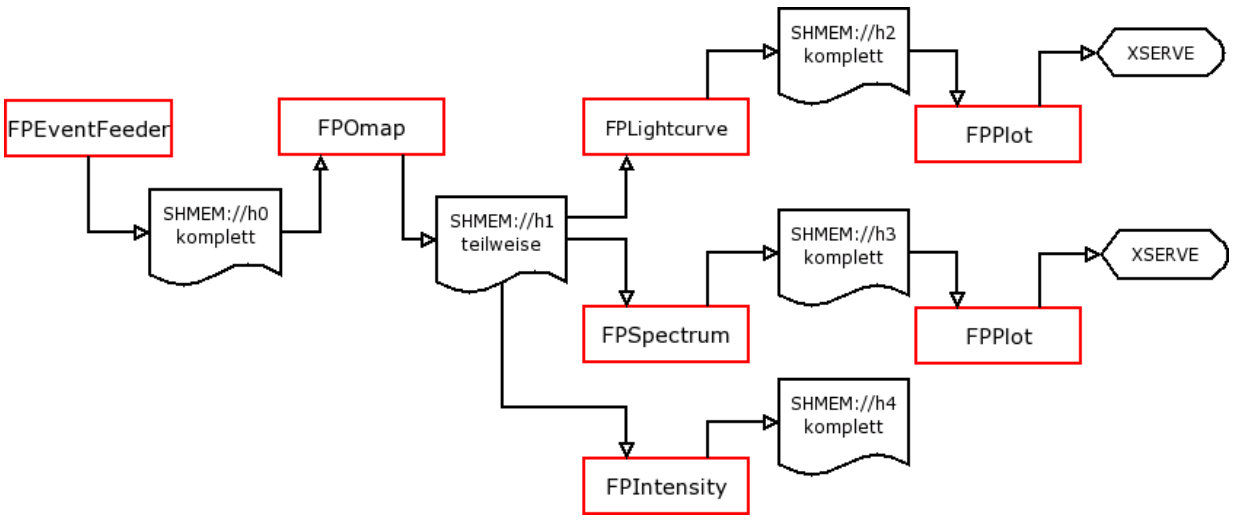

Abbildung 4.1: Die hier dargestellte Pipeline ist als Datei in meiner Software enthalten.

und ich werde ihn hier genauer beschreiben.
Das erste Arbeitsmodul FPEventFeeder liest in dieser Version der Software noch nicht von einem Detektor ein, sondern öffnet eine vorhandene Eventliste einer realen Beobachtung und schickt die Daten in Echtzeit (d.h. in den Abständen, in denen die Events in der Liste durch die FRAME Einträge mit fester Framezeit vermerkt sind) weiter.
Das darauf folgende Arbeitsmodul FPOmap steht stellvertretend für einige andere grundlegende Analyseschritte. Dabei liest es immer nur die neuesten Events ein, und überschreibt die Datei, die es im letzten Pipelinedurchlauf erstellt hat.

Die weiteren Arbeitsmodule (FPIntensity, FPSpectrum und FPLightcurve) verarbeiten die ankommenden Events in neue Formate, in ein Spektrum, eine Lichtkurve und ein Intensitätsbild. Die Lichtkurve und das Spektrum werden dann auf dem Bildschirm durch FPPlot ausgegeben.
Das Intensitätsbild ist in dieser rudimentären, nicht interaktiven[2] Pipeline nur als Datei vorhanden[3].
Die folgende Tabelle enthält die Messergebnisse zu dieser ersten Pipeline, die Anzahl der Eventschübe in der Pipeline ergab sich aus 20 Messungen zu 1,14. Dazu kommt noch eine Anzahl von Events im FPEventFeeder Buffer (zwischen 0 und 1 Eventschübe), diese setze ich als mittleren Wert bei 0,5 an. Damit folgt die Durchlaufdauer als $1,64 * Evenschub/Eventrate$:

Tabelle 4.1: Messergebnisse der ersten Beispiel-Pipeline

Events	Zeit [s]	Eventschub	Eventrate [Events/s]	Durchlaufdauer [s]	Load
5281	388,32	3	13,7	0,4	$\leq 2,2$
11879	475,60	6	25	0,4	$\leq 2,6$
39034	390,75	20	100	0,3	≤ 3
76506	306,43	51	250	0,3	≤ 3
343409	344,01	243	1000	0,4	≤ 3

Zur Durchlaufdauer von Pipeline Nummer 1 lässt sich sagen, dass die Werte alle im erhofften Bereich liegen und zumindest bei den getesteten Eventraten – die ja immerhin stark variieren zwischen 13,7 und 1000 Events pro Sekunde[4] – sogar konstant sind.
Dies zeigen auch die aufgezeichneten Loadkurven. Die hier abgebildete wurde mit einer Eventrate von 100 Events pro Sekunde aufgenommen. Die horizontalen Linien haben voneinander den Abstand von einem Load:

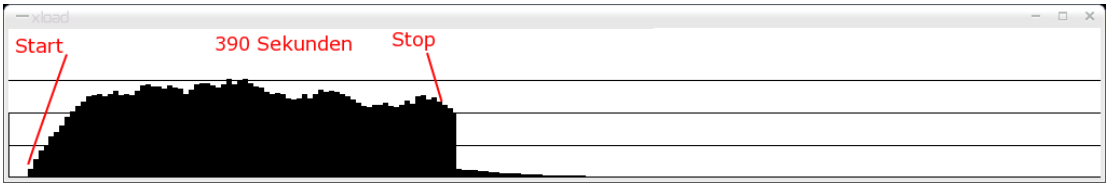

Abbildung 4.2: Loadverlauf

Die Load-Werte und vor allem die Durchlaufdauer zeigen, dass diese einfache Pipeline, was die Verarbeitungszeit angeht, mit hohen Eventraten gut umgehen kann. Dabei werden zu hohen Raten hin, immer größere Eventpakete auf einen Schlag verarbeitet.

[2] Die entsprechenden Befehle habe ich nicht in die Definitionsdatei geschrieben.

[3] Für eine Erklärung, warum ds9 noch nicht eingebaut wurde, um diese Datei darzustellen, siehe Abschnitt 3.1.1.2.

[4] für heutige abbildende Hochenergiedetektoren wie z.B. die pn-CCDs auf XMM-Newton stellen 1000 Events pro Sekunde extrem hohe Evennraten dar. Schon viel früher wird ein unerwünschter Effekt, der Energiepileup, bei Punktquellen groß.

Die Verteilung der Rechenzeit ergibt folgendes Bild:

Tabelle 4.2: Verteilung der Rechenzeit

Process	CPU-Zeit (100 Events/s)	CPU-Zeit (1000 Events/s)
FPOmap	$\sim 85,4\%$	$\sim 35,7\%$
FPPlot (1)	$\sim 2,6\%$	$\sim 52,8\%$
FPPlot (2)	$\sim 0,3\%$	$\sim 0,3\%$
FPIntensity	$\sim 1,0\%$	$\sim 1,6\%$
FPLightcurve	$\sim 0,8\%$	$\sim 0,9\%$
FPSpectrum	$\sim 0,6\%$	$\sim 0,6\%$
FPEventFeeder	$\sim 0,6\%$	$\sim 0,6\%$
fpipeGui.pl (Summe)	$\sim 0,6\%$	$\sim 1,2\%$

Dies zeigt, dass FPOmap – zumindest bei niedrigen Eventraten – noch zuviel Rechenzeit verbraucht, was verbessert werden könnte.

4.3 Zweites Beispiel für eine mögliche Pipeline

In einer weiteren denkbaren Beispielpipeline ist noch zusätzlich das FPCopy-Control Programm zwischen FPEventFeeder und FPOmap geschaltet (Datei: '`pipe_def_copycontrol.xml`').
Für diesen Aufbau ergab sich aus 20 Messungen eine Anzahl von 1,71 Eventschüben innerhalb der Pipeline und damit die hier aufgeführte Durchlaufdauer ($2,21 * Evenschub/Eventrate$):

Tabelle 4.3: Messergebnisse der zweiten Beispiel-Pipeline

Events	Zeit [s]	Eventschub	Eventrate [Events/s]	Durchlaufdauer [s]	Load
12617	922,33	11	13,7	1,8	$\simeq 4$ (stabil)
14562	876,62	8	16,7	1,1	$\simeq 3$ (stabil)
19490	977,11	9	20	1,0	$\simeq 3$ (stabil)
15812	633,22	10	25	0,9	$\simeq 3$ (stabil)
27343	274,01	31	100	0,7	$\simeq 4$ (stabil)
53496	428,49	42	125	0,7	$\simeq 3$ (stabil)
43237	173,46	59	250	0,5	≤ 3 (stabil)
128420	257,55	151	500	0,7	$\simeq 3,5$ (stabil)
172929	174,58	381	1000	0,8	$\simeq 3,6$ (stabil)
1118748	895,63	628	1250	1,1	4 bis 2
1268677	766,34	2275	1666,7	3,0	3,5 bis 1,5
2341080	947,99	6581	2500	5,8	3,3 bis 1,5

Die Messungen zeigen, dass selbst bei einem 1,5 GHz Prozessor zumindest die mittleren Werte bis zu Eventraten von 1250 Events pro Sekunde brauchbar erscheinen. Diesen Eventraten lässt sich eine Durchlaufzeit zwischen 0,5 und 1,1 Sekunden zuordnen.
Zu den Messungen bei 13.7, 1250, 1666 und 2500 Events pro Sekunde habe ich zusätzlich die Kurven der Systemauslastung mit xload aufgenommen.
Sie zeigen, dass bei der momentan zur Verfügung stehenden Rechenleistung – wie erhofft – tatsächlich Pipelines dieser Art verwendet werden können. Das gilt vor allem für Eventraten ab etwa 1250 Events pro Sekunde, dann scheint sich die Systemauslastung einem Wert von 1,5 oder kleiner anzunähern, wird also – wie erwünscht – gering, allerdings mit einer etwas zu großen Durchlaufzeit.
Die folgende Graphik zeigt die Loadkurven bei verschiedenen Framezeiten bzw. Eventraten (von oben nach unten: 0,073 s/Frame; 0,008 s/Frame; 0,006 s/Frame; 0,004 s/Frame):

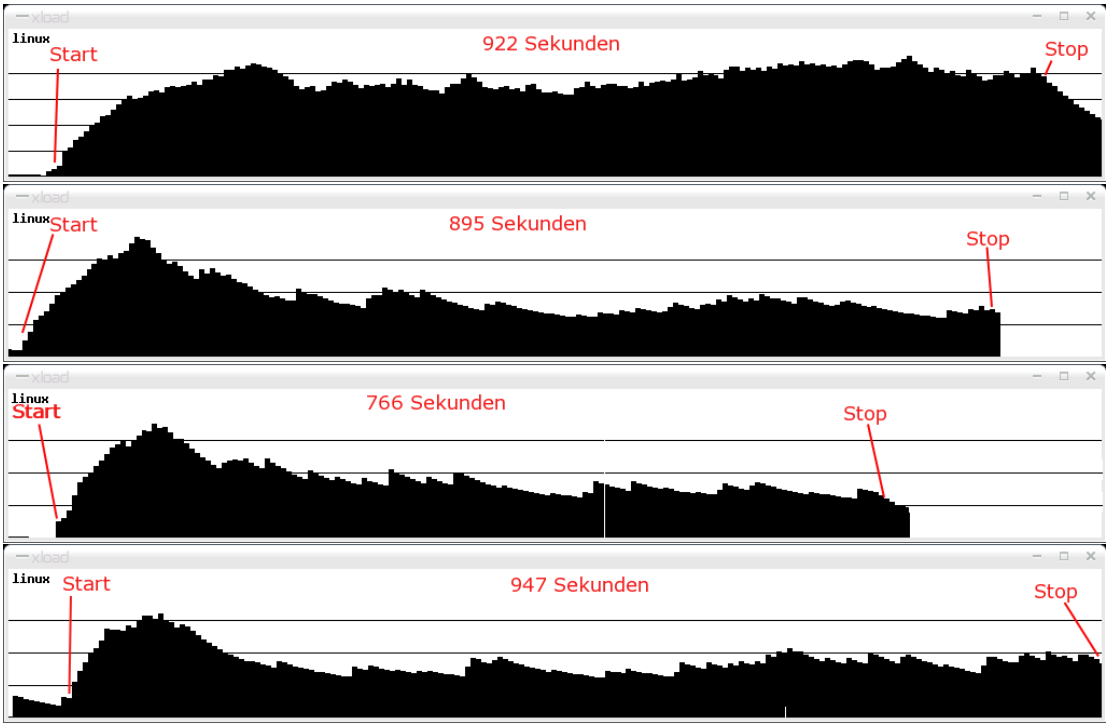

Abbildung 4.3: Loadverlauf bei verschiedenen Eventraten

Bei manchen dieser Messungen habe ich ausserdem noch die Verteilung der CPU-Zeit auf die verschiedenen Prozesse beobachtet.
Im Folgenden sind typische Werte bei Framezeiten bis 0,002 s/Frame angegeben. Die Displayangaben beziehen sich dabei auf die Größe des Plotfensters für den Spektrum-Plot und den Lichtkurven-Plot:

Tabelle 4.4: Verteilung der Rechenzeit

Process	CPU-Zeit ($1/4$ Display)	CPU-Zeit (je 1 Display)
FPOmap	$\sim 34\%$	$\sim 34\%$
FPCopyControl	$\sim 34\%$	$\sim 34\%$
FPPlot (1)	$\sim 13\%$	$\sim 12\%$
FPPlot (2)	$\sim 0,7\%$	$\sim 0,8\%$
FPIntensity	$\sim 2\%$	$\sim 2\%$
FPLightcurve	$\sim 0,9\%$	$\sim 1,1\%$
FPSpectrum	$\sim 0,3\%$	$\sim 0,6\%$
FPEventFeeder	$\sim 0\%$	$\sim 0\%$
fpipeGui.pl (Summe)	$\sim 2,6\%$	$\sim 2,4\%$

Es zeigt sich, dass FPOmap und FPCopyControl sehr viel Rechenzeit beanspruchen im Vergleich zu beispielsweise FPIntensity und FPSpectrum. Die Größe der Darstellung scheint dabei kaum Einfluß auf die Rechenleistung zu haben.
Diese Werte stellen natürlich nur Momentaufnahmen im Vergleich zu wirklichen Werten dar, die im Laufe des Betriebes immer schwanken, ihre Größenordnung ist aber stabil.
Diese Verteilung der CPU-Zeit veränderte sich aber schlagartig beim Übergang von 500 auf 1000 Events pro Sekunde:

Tabelle 4.5: Verteilung der Rechenzeit

Process	CPU-Zeit (je 1 Display)
FPOmap	$\sim 10,2\%$
FPCopyControl	$\sim 14,7\%$
FPPlot (1)	$\sim 41,6\%$
FPPlot (2)	$\sim 0,7\%$
FPIntensity	$\sim 2,7\%$
FPLightcurve	$\sim 1,5\%$
FPSpectrum	$\sim 0,9\%$
FPEventFeeder	$\sim 0,6\%$
fpipeGui.pl (Summe)	$\sim 2,4\%$

Dasjenige FPPlot-Programm, das weniger CPU-Zeit benötigt, stellt die Lichtkurve dar.
Betrachtet man die Entwicklung, die sich in den beiden vorangehenden Tabellen andeutete, weiter in Richtung noch höherer Eventraten, dann übernehmen bei einer Eventrate von 2500 Events pro Sekunde FPPlot (mit ca. 65%) und FPEventFeeder und FPLightcurve (mit je ca. 20%) die ‘Führung’.
Aus den bisherigen Messergebnissen läßt sich in erster Linie ableiten, dass FPCopyControl und FPOmap optimiert werden sollten, da diese bei kleinen Eventraten den CPU-Verbrauch dominieren. Bei höheren Eventraten würde sich sicher auch jede weitere Optimierung von FPPlot bemerkbar machen.

Die erhoffte Systemauslastung von unter 1 ist leider mit der hier verwendeten (bestenfalls durchschnittlichen) Hardware nicht zu erreichen. Echtzeit – im Sinne einer Durchlaufzeit von unter zwei Sekunden – ist aber offensichtlich bis zu sehr hohen Eventraten (bis mind. 1250 Events pro Sekunde) auch mit langsamer Hardware möglich.

4.4 Drittes Beispiel für eine mögliche Pipeline

Die bisher vermessenen Pipelines sind nicht interaktiv. Das läßt sich aber ändern durch einfügen von ‘set’-Aufrufen in die Pipelinedefinition (‘`pipe_def_copycontrol_interaktiv.xml`’).
Hier möchte ich betonen, dass auch bei kompakten Echtzeitprogrammen Interaktionen des Benutzers schwierig sind, da auch bei diesen Programmen oft die komplette Eventliste neu bearbeitet werden muss oder alle bisherigen Ergebnisse gelöscht werden müssen, wenn Parameter der Pipeline durch den Benutzer geändert werden.
In der dritten getesteten Pipeline wird die Möglichkeit einer Benutzerinteraktion eingebaut, durch die die untere Energieschwelle für das Spektrum verändert werden kann. Wie sinnvoll es ist, eine derartige Änderung auf diese Art einzubauen ist hier nicht die Frage, sondern es ist interressant zu beobachten, wie die Pipeline reagiert, wenn sie mitten im Betrieb eine Pause für Parameteränderungen und RESET-Callbacks einlegen und danach alle bisherigen Events mit veränderten Parametern erneut bearbeiten muss.
Um zu zeigen, wie ‘set’-Aufrufe in die Pipeline konkret eingebaut werden können, habe ich die Eintragungen aus der Pipelinedefinition hier abgebildet:

```
<interaktiv>
  <SpectrumEmin>
    <status>1</status>
    <outfiles></outfiles>
    <infiles></infiles>
    <set>xpaset -p FPCopyControl RESET;</set>
    <set>xpaset -p FPOmap RESET;</set>
    <set>xpaset -p FPLightcurve RESET;</set>
    <set>xpaset -p FPIntensity RESET;</set>
    <set>xpaset -p FPSpectrum INFILE "shmem://h1[PHA >##Emin,i,0,0,65,Enter
      Energy Minimum for the Spectrum.##]";</set>
    <set>xpaset -p FPSpectrum RESET;</set>
    <set>xpaset -p FPPlot RESET;</set>
    <set>xpaset -p FP2Plot RESET;</set>
    <set>killall -USR1 FPCopyControl</set>
  </SpectrumEmin>
</interaktiv>
```

Abbildung 4.4: ‘set’-Eintragungen in der Definitionsdatei

Wird während des Betriebs der Pipeline nun dieses Kommando ausgeführt, dann ergibt sich in der Loadkurve folgendes Bild[5]:

[5] wichtige Einstellungen: frametime=0,073, maxrows=20000

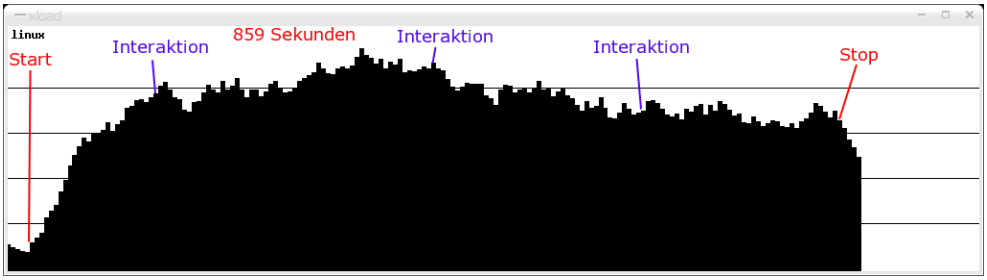

Abbildung 4.5: Loadverlauf der interaktiven Pipeline

Dies zeigt, dass die Benutzerinteraktion kaum Einfluss auf die Systemauslastung hat, zumindest nicht bei dieser kurzen Beobachtungsdauer.
Die folgende Abbildung zeigt, was der Benutzer am Ende auf dem Bildschirm vor sich hat. Die vier kleinen Fenster sind von oben nach unten: das Hauptfenster (‘fpipeGui.pl’), die beiden Fenster, über die die Interaktionen gesteuert werden können – in dieserm Fall die Festlegung der unteren Grenze des Spektrums – (‘The Tasks’ und ‘Sinteraktiv’) und noch ein Informationsfenster, im hier abgebildeten werden Informationen über die Semaphore gegeben (‘Semaphore Information’). Außerdem sind in den beiden großen Fenstern ein Spektrum und eine Lichtkurve zu sehen, wie sie von FPPlot erstellt werden können.

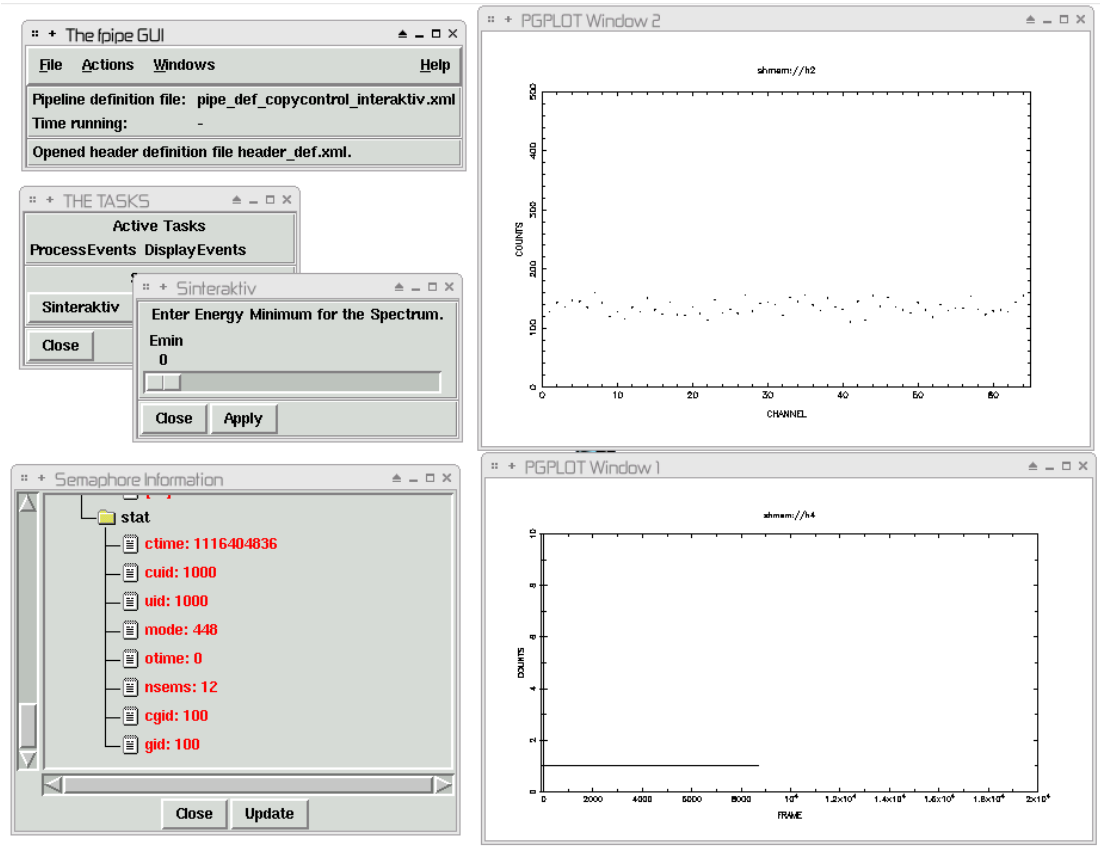

Abbildung 4.6: Screenshot des Programms mit interaktiver Pipelinedefinition

4.5 Abschließende Bemerkungen

4.5.1 Was es noch zu tun gibt

Software wird nie fertig. Zumindest kenne ich – außer TEX[6] – kein Programm, dessen Entwicklung eingestellt wurde, weil es perfekt ist. Neben diesen üblichen andauernden Veränderungen, die hoffentlich auch an dieser Software vorgenommen werden, fehlen aber noch einige relevante Teile, auf die ich jetzt eingehen möchte.
Wie bereits angedeutet, funktioniert die Einbindung von ds9 nicht so, wie erwünscht.
Dies ließe sich aber sicherlich ändern, wenn in Kooperation mit den Entwicklern von ds9 die sicherlich selten benutzten Eigenschaften des Shared Memory Zugriffs in Hinblick auf Speicherbelegung, Fokusparameter und Regionsauswahl verändert würden. Das erscheint mir trotz der vielen Probleme, die sich im Laufe der Diplomarbeit für mich aufgrund dieser Eigenschaften ergaben, immer noch die sinnvollste Lösung zu sein.
Die Alternative, ein Darstellungsprogramm selbst zu schreiben, scheint mir nicht nur zu langwierig, es widerspricht auch dem Ziel dieses Programms, außer einem allgemeinen Pipelinedriver auch externe, bereits existierende Programme, zu nutzen. Würde dieser Ansatz wieder darauf hinauslaufen, dass alle Programme selbst geschrieben werden müssten, ginge ein Teil des Nutzens, den der Standard FITS mit sich bringt, verloren.
Neben der Darstellung von Bildern fehlt es bisher auch noch an einer ausgearbeiteten Software zur Darstellung von xy-Plots. Auf vielfachen Rat hin habe ich kaum mehr Zeit als nötig war in die Arbeit an FPPlot investiert, um ein funktionierendes Programm zur Darstellung von xy-Plots und auch von Scatterplots in der Hand zu haben. Die Plotmöglichkeit von ds9 scheint mir auch hier der richtige Weg zu sein, der eingeschlagen werden sollte, um dies zu ändern. Im ds9 Plotfenster stehen nicht nur alle wichtigen Funktionen wie Zoom und ähnliches zur Verfügung, es lässt sich – wie alle ds9 Funktionen – auch über XPA Kommandos steuern. Das einzige Problem, dass die Daten nicht in Form einer FITS Datei vorliegen dürfen, sollte sich eigentlich einfach beheben lassen.
Bei den bisher erstellten Arbeitsmodulen gibt es auch noch Verbesserungsbedarf:

- Ein bekannter Fehler liegt im FPLightcurve-Programm, das an manchen Stellen, z.B. bei Eintrag 1,5001500E+04 in der Zeitspalte, eine falsche Eintragung in die ‘Counts’-Spalte vornimmt (90010 statt 1). Dies scheint mir darauf hinzudeuten, dass an dieser Stelle der Datentyp einer Variablen zu klein für die benutzten Werte ist, aber ich konnte diesen Fehler noch nicht beheben.
  Um weitere Fehler dieser Art zu finden, sind auf jeden Fall noch mehr ausgedehnte Tests notwendig.
- Die bisherigen Messungen zeigten deutlich, dass FPOmap und auch FPCopyControl noch verbessert werden könnten, da sie momentan im Vergleich zu den anderen Programmen noch zuviel Rechenzeit beanspruchen.

[6] Der Befehlsumfang von TEX steht schon seit 1985 fest. Es werden nur noch Fehlerkorrekturen vorgenommen. Dabei strebt die Versionsnummer von TEX gegen Pi.

- Der letzte Punkt betrifft die Verwendung der Pipeline an einem echten (also nicht von FPEventFeeder simulierten) Detektor. Dazu wäre es notwendig, FPEventFeeder so umzubauen, dass er anstelle einer FITS Datei als Input die Daten (also die Eventpakete) von der Verbindungsstelle der Detektorauslesehardware mit dem Computer holt. Dabei wird es sich um eine ADC Karte, ein serielles oder um ein USB Interface handeln. FPEventFeeder ist aber so aufgebaut, dass die Struktur der Abfrage, ob neue Daten vorhanden sind, diesen Bedingungen ohne allzugroße Codeänderungen angepasst werden kann.

4.5.2 Zusammenfassung

Alles in allem ergaben die Messungen sehr positive Resultate und die Machbarkeit solch eines neuen Softwaretyps kann damit meiner Meinung nach als erwiesen gelten.
Bereits mit einem modernen aber einfachen Betriebsystem, das nicht speziell auf den Echtzeitbetrieb ausgerichtet ist, und mit nur mittelmäßiger Hardware lässt sich mein Programm äußerst zufriedenstellend betreiben. Sogar bei hohen Eventraten (bis 1250 Events pro Sekunde) funktioniert die Darstellung einwandfrei innerhalb einer sehr kurzen Zeitspanne von nur 0,5 bis 1,1 Sekunden.
Da für die Inbetriebnahme meines Programms keine teure Laborsoftware wie LabView angeschafft werden muss, ist sie eine günstige Alternative zu bisher existierenden Programmen. Ökonomischer als diese ist sie auch im übertragenen Sinne, insofern, als dass tatsächlich mit ihr ein Instrument zur Verfügung steht, das für mehrere Detektortypen und in verschiedenen Phasen einer Satellitenmission eingesetzt werden kann.
Insgesamt erwies sich der beschrittene Weg als nicht immer ganz einfach, dennoch bin ich fest davon überzeugt, dass er gangbar ist. Eine neue ‘ökonomischere’ Art von Software, wie sie hier vorgestellt wurde, hat, wenn die Idee weiterverfolgt wird, unbedingt berechtigte und hoffentlich gute Aussichten irgendwann innnerhalb von Satellitenmissionen – oder auch im Rahmen der Forschung mit erdgebundenen Teleskopen – eingesetzt zu werden.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Ostermann, Dietmar: Streit um US-Mondprogramm. Weltraumträume könnten Bush eine ‘Saat der Revolte’ bescheren. *Frankfurter Rundschau online. Erscheinungsdatum: 16.01.2004*.

[2] Bush, George W.: Let us continue the journey. Rede vom 14. Januar 2004 im NASA-Hauptquartier. *Ressort Reden der ZEIT www.zeit.de*.
Online Referenz: *www.whitehouse.gov/news/release/2004/01/20040114-3.html*.

[3] Frankfurter Rundschau: Bush sucht Halt auf Mars und Mond. US-Präsident bereitet Grundsatzrede zum Raumfahrtprogramm vor / Mondstation und bemannter Marsflug als Ziele. *Frankfurter Rundschau online. Erscheinungsdatum: 10.01.2004*.

[4] von Randow, Gero: US-Präsident Bush stößt neue Raumfahrtpläne zu Mond und Mars an. Die Gründe sind eher politisch als wissenschaftlich. *Die ZEIT (zeit.de)*, 04/2004, 2004.

[5] Drösser, Tobias / Beck, Christoph: Rot ist die Zukunft. *Die ZEIT*, 15/01/2004 Nr.4, 2004.

[6] Ostermann, Dietmar: Kommentar: Rückkehr zum Mond. Cleverer Wahlkämpfer. *Frankfurter Rundschau online. Erscheinungsdatum: 10/01/2004.*

[7] Fritz-Vannahme, Hans / Schuh, Joachim: Gesucht: Europas Rolle im All. *Die ZEIT*, 10/04/2003 Nr.16, 2003

[8] Waloschek, Pedro: Wörterbuch der Physik. München: Deutscher Taschenbuch Verlag (dtv), 1998.

[9] Padmanabhn, T.: Theoretical Astrophysics: Vol I. Astrophysical Processes. Cambridge University Press, 2000.

[10] Peterson: An Introduction to Active Galactic Nuclei. Cambridge University Press, 1997.

[11] European Space Agency (ESA). XMM-Newton Webpage.
Online Referenz: *http://xmm.esa.int*.

[12] European Space Agency (ESA). XEUS Webpage.
Online Referenz: *http://www.rssd.esa.int/index.php?project=XEUS*.

[13] NASA. DUO - The Dark Universe Observatory.
Online Referenz: *http://duo.gsfc.gov/*.

[14] GDL - The Gnu Data Language.
Online Referenz: *http://gnudatalanguage.sourceforge.net*.

[15] Wells, D. C. / Greisen, E. W. / Harten, E. H.: Fits: A flexible image transport system. *Astronomy & Astrophysics Supplement Series*, No. 44, June 1981. 363pp.

[16] Harnish, Robert J. / Pence, William D. / Schlesinger, Barry M. / Farris, Allen / Greisen, Eric W. / Teuben, Peter J. / Thompson, Randall W. / Warnock, Archibald III: Definition of the flexible image transport system (fits). *Astronomy & Astrophysics*, May 2001.

[17] Greisen, E. W. / Harten, R. H.: An Extension of FITS for Groups of Small Arrays of Data. *Astronomy & Astrophysics Supplement Series*, No. 44, June 1981. 371pp.

[18] IAU: *Information Bulletin*, No. 49. 1983.

[19] IAU: *Information Bulletin*, No. 61. 1988.

[20] McNally, D.: *Transactions of the IAU. Proceedings of the Twentieth General Assembly*. Dortrecht: Kluwer, 1988.

[21] Wells, D. C. / Grosbol, P. / Greisen, Eric W. / Harten, E. H.: The fits tables extension. *Astronomy & Astrophysics Supplement Series*, No. 73, June 1988. 365pp.

[22] Wells, D. C. / Grosbol, P.: Floating point agreement for fits. 1990. *(Available electronicaly from ftp://nssdc.gsfc.nasa.gov/pub/fits/fp_agree.ps)*.

[23] Ponz, J. D. / Thompson, R. W. / Munoz, J. R.: The FITS image extension. *Astronomy & Astrophysics Supplement Series*, No. 105, May 1994. 53pp.

[24] Cotton, W. D. / Tody, D. / Pence, W. D.: Binary table extension to FITS. *Astronomy & Astrophysics Supplement Series*, No. 113, October 1995. 159pp.

[25] Greisen, E. W. / Calabretta, M. R.: Representations of world coordinates in FITS. *Astronomy and Astrophysics*, No. 395, December 2002. 1061pp.

[26] Calabretta, M. R. / Greisen, E. W.: Representations of celestial coordinates in FITS. *Astronomy and Astrophysics*, No. 395, December 2002. 1077pp.

[27] Angelini, L. / George, I. M.: Standard Strings for Mission, Instrument, Filter, Detektor & Grating Names for OGIP FITS files. *Ogip memo / 93-013 (missions, instruments, detektors & gratings). 06/03/1995.*

[28] Arnoud, Keith A. / George, Ian M. / Tennant, Allyn F.: The OGIP Spectral File format. *Ogip memo / 92-007 (spectral file format). 12/4/2004.*

[29] HFWG: On the use of the CREATOR keyword. November 1993.

[30] Pence, William D.: The fits support office at nasa/gsfc. 2004.
Online Referenz: *http://fits.gsfc.nasa.gov/fits_home.html*.

[31] HEASARC: CFITSIO User's Reference Guide. An Interface to FITS Format Files for C Programmers. July 2004.

[32] George, Ian M. / Rots, Arnold / Mukai, Koji: Guidelines for defining FITS formats for Event Lists. *Ogip memo / 94-003 (guideline for event lists). 01/12/1994.*

[33] HFWG: On the OGIP data quality flag convetion. July 1994.

[34] Angelini, L. / Pence, W. / Tennant, A. F.: The Proposed Timing FITS File Format for High Energy Astrophysical Data. *Ogip memo / 93-003. 30/11/1994*

[35] HFWG: On the Keywords and definitions denoting the channel and energy boundaries. July 1994.

[36] HFWG: On the minimum and maximum actual and legal values within columns of FITS tables. August 1993.

[37] Joye, W. A. / Mandel, E.: New Features of SAOImage DS9. *ASP Conf. Ser. 295: Astronomical Data Analysis Software and Systems XII*, 2003. 489pp.

[38] XPA: Puplic Access to Data and Algorithms, September 2003.

[39] Stevens, W. Richard: UNIX Network Programming. Boston; San Francisco; New York: Addison-Wesley, 2004.

[40] Borkowski, J. / Lock, T. / Walter, R.: Parameter Interface Library Users Manual. INTEGRAL Science Data Center, September 2002.

[41] Stevens, W. Richard: Advanced programming in the UNIX environment. Boston; San Francisco; New York: Addison-Wesley, 1993.

Danksagungen

Hiermit möchte ich mich bei allen bedanken, die mich während meiner Diplomarbeit unterstützt haben. Insbesondere seien hier erwähnt:

Prof. Dr. R. Staubert, für die Vergabe und Korrektur dieser Diplomarbeit.

Dr. J. Wilms, für die sehr gute Fernbetreuung während der gesamten Arbeit und die Woche kompakte Vorortunterstützung in Warwick.

Dr. E. Kendziorra, für die hilfreichen Anregungen und die erste Hiwi Stelle, durch die ich ans Institut gefunden habe.

Judith Benz, ohne die ich niemals die Arbeit hätte pünktlich abgeben können.

meine Eltern, die mich immer unterstützt haben und ebenfalls viel Zeit mit Korrekturlesen verbracht haben.

alle Mitglieder des Instituts, die mit mir Fussball, Magic und Frisbee gespielt haben und immer zur Verfügung standen, wenn ich Fragen hatte. Viele Grüße an Slawo in den USA und Kolja am Südpol!

Erklärung

Hiermit erkläre ich, Stefan Schwarzburg, dass ich meine Diplomarbeit mit dem Titel: *Eine Software zur Echtzeitanalyse von experimentellen Daten im Flexible Image Transport System (FITS)* nur mit den darin angegebenen Mitteln angefertigt habe.

Tübingen, den 31. Mai 2005