Forschungseinrichtung Satellitengeodäsie
Technische Universität München

„Verbesserung des Datenmanagements in inhomogenen Rechnernetzen geodätischer Messeinrichtungen auf der Basis von Middleware und Dateisystemen am Beispiel der Fundamentalstation Wettzell“

Alexander Norbert Josef Neidhardt

Vollständiger Abdruck der von der Fakultät für Bauingenieur- und Vermessungswesen der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. M. Schilcher

Prüfer der Dissertation:

1. apl. Prof. Dr. rer. nat., Dr.-Ing. habil. U. Schreiber
2. Univ.-Prof. Dr. rer. nat. J. Schlichter
3. Univ.-Prof. Dr. phil. nat. M. Rothacher, Technische Universität Berlin

Die Dissertation wurde am 06.04.2005 bei der Technischen Universität München eingereicht und durch die Fakultät für Bauingenieur- und Vermessungswesen am 27.06.2005 angenommen.

Kurzfassung

Geodätische Messsysteme, welche auf Weltraumtechniken aufbauen, haben seit dem Beginn der Weltraumgeodäsie die Grundlage für globale Referenznetze geschaffen. Die Erbringung von Daten durch flächendeckend verteilte, permanente Messeinrichtungen ist dabei wesentlicher Bestandteil. Die Anforderungen hierzu werden jedoch immer höher. Eine höhere Anzahl von Messdaten müssen in immer kürzerer Zeit von der Messeinrichtung zum Auswertezentrum gelangen, ohne dass hierbei größere Ausfallzeiten entstehen dürfen, da sonst die Kontinuität von Zeitreihen beeinflusst werden könnte.

Da das Hauptinteresse der Geodäten den geodätischen Messdaten gilt, wurde der Aspekt der Verwaltung und Verfügbarkeit dieser bislang unterbewertet. Tadellos funktionierende, informationsverarbeitende Einrichtungen werden als gegeben angenommen. Mittlerweile stoßen jedoch die historisch gewachsenen Systeme an ihre selbst auferlegten Grenzen. So war es Ansporn für das Forschungs- und Entwicklungsprogramm 2001 bis 2005 der Forschungsgruppe Satellitengeodäsie (FGS) eine Projektstudie mit der Aufgabe ins Leben zu rufen, welche die Datenhaltung grundsätzlich untersuchen sollte. Hieraus begründet sich der Inhalt dieser Arbeit, welche interdisziplinär sowohl in der Geodäsie, als auch in der Informatik angesiedelt werden muss. Durchdachte Ideen werden hierbei exemplarisch für die Fundamentalstation Wettzell angewandt.

Eine eingehende Analyse brachte zu Tage, dass die Probleme viel tiefgründiger angenommen werden mussten, als in der Formulierung zur Ankündigung der Studie angedacht war. Bereits die genutzten Übertragungsverfahren und -abläufe verursachen ein erhebliches Maß an Unsicherheit. Die stufenweise durchgeführte Zusammenfassung der gewonnenen Informationen führte zu einem 3-Stufen-Plan, welcher sich mit der schrittweisen Abstrahierung der Problemfelder und ihrer technischen Realisierung befasst.

Grundlegendes Element ist hierbei die Schaffung eines verlässlichen, flexiblen, transparenten und stabilen Übertragungsverfahrens. Der hierunter entwickelte Extended CORBA File Transfer (ECFT) bietet einen Großteil von Dienstgütekriterien an, welche in bisherigen Systeme nicht oder nur teilweise existieren und wenn, dann nicht unbedingt über Systemgrenzen hinweg kombiniert werden können. Konkurrenzkontrolle, zustandsbasierte Verarbeitung, Fehlermanagement und eine flexible Schichtarchitektur bei Verdeckung der Eigenschaften zugrundeliegender Techniken sind nur einige Elemente des neuen Systems, mit welchem sowohl im lokalen als auch im weltumspannenden Netz geodätische Daten transferiert werden können.

Eine weitere Stufe beschäftigt sich schließlich mit den Daten selbst. Die ursprünglich angedachte Schaffung von einheitlichen Metadaten wurde hier nicht als wirklich sinnvoll erachtet. Fremdbezüge und historisch gegebene Anteile wären so nur schwer zu berücksichtigen. Die Arbeit versucht stattdessen die Individualität der Daten und ihrer Formate zu erhalten und nur eine homogenisierende Transformationsmöglichkeit zu schaffen. Diese auf der Extensible Markup Language (XML) aufsetzende Beschreibung von semi-strukturierten und strukturierten Datenformaten zur Konvertierung ist bisher als einzigartig anzusehen, da sich andere Verfahren immer den Möglichkeiten von weiteren Programmiersprachen bedienen

müssen, um Daten erstmalig in eine Markup-Schreibweise zu wandeln. Durch die geschaffenen Konstrukte können beliebige, geodätische Daten beliebig umgewandelt werden.

Letzte Stufe der Studie bildet letztendlich die übergeordnete Strukturierung der realen Gegebenheiten der Fundamentalstation Wettzell. Die Einteilung von handhabbaren Verwaltungsobjekten mit in sich abgeschlossenen Kompetenzen und deren strukturiertes Zusammenspiel anhand abgeleiteter Schnittstellenzugangspunkte und hierarchisch strukturierter Datenverarbeitungsflüsse fügen die bislang erbrachten Ideen zu einem harmonischen Ganzen zusammen.

Der besondere Charakter der Arbeit ist dabei vor allem in ihrem starken Bezug zur technischen Realisierung zu erkennen. Hauptaugenmerk lag immer auf einer realen Umsetzung der Theorien und Planungen. Weshalb für die Arbeit auch die strikte Vorgehensweise anhand eines Softwareprozesses eingehalten wurde. Deshalb werden die erstellten Anteile auch über diese Arbeit hinaus in Teilen, z.B. im neuen Satellite Observing System Wettzell (SOS-W), weiterentwickelt und umgesetzt werden.

Abstract

Geodetic observatories for space techniques are the fundament for global reference systems since the beginning of space geodesy. The producing of data by these area-wide distributed, permanent data acquisition sites is therefore a main part. However, the requirements increase. A higher amount of measured data have to be transported to the analysing centres without long lasting down times during a more and more reduced time period. Interferences in these data workflows would cause affects in the continuity of the generated time series.

Because of the fact that the main interest for the geodesists applies to the measured geodetic data the aspect of administration and availability for that geodetic data is being more or less underrated. Properly working information processing facilities are supposed to be simply existent. But meanwhile the historically crown systems reach their self set limits. This fact was the cause for a project study defined at the research and development program 2001 to 2005 by the German Research Group for Satellite Geodesy (Forschungsgruppe Satellitengeodäsie (FGS)) with the challenge to analyse the data management basically. This defines the frame of the present work, which is an interdisciplinary approach affecting geodesy as well as computer science. Found ideas are applied to the current situation at the geodetic observatory at Wettzell as an exemplary realisations.

A profound analysis showed, that the problems are more basical than they were written at the studies notification. The used transfer mechanisms and flows cause a remarkable amount of unstableness. The incrementally accomplished generalisation of the collected information leaded to a three step method, which addresses the abstraction of problem categories and their technical realisation.

A basic element is therefore the creation of a reliable, flexible, transparent and stable transfer mechanism. The developed Extended CORBA File Transfer (ECFT) offers most of the criteria for quality of service, which are not or only partially existent in current solutions. And when they are offered they are restricted by system borders. Concurrency control, state driven processing, error or fault management and a flexible system architecture with hidden technical attributes are only a few elements of the new system. With it geodetic data can be transported as well in local area networks as over world wide existing nets.

At least the next step deals with the data handling itself. The originally wished creation of a consistent meta data set was not really considered as reasonable. External procurements and historically given parts would hardly be integrated with this regulation. The present work keeps the individuality of data and their formats instead and offers a homogeneous transformation possibility by defining a description language based on the Extensible Markup Language (XML). This method for conversions of semi-structured and structured data formats can be seen as unique at the moment, because all other techniques uses the abilities of foreign programming languages to convert the original data into a markup set. With the created constructs in the present work any geodetic data sets can be transformed into any wished format.

The final step of the three phases of abstraction creates a superordinated configuration for the currently existing conditions at the geodetic observatory Wettzell.

The classification into manageable administration zones with dedicated competencies and their structured interaction over flexible interface access points with a hierarchically structured data processing workflow combines all of the so far developed ideas to a harmonical unit.

The specific character of the present work is therefore especially given by the strong reference to technical realisations. Main focus always laid on a real implementation of the theories and strategies. So the strict compliance of a software process was selected for the structure of the present work. Because of that the developed parts will also be enhanced partially for specific application sets after the ending of this work, e.g. for the new Satellite Observing System Wettzell (SOS-W).

Erklärung

Ich erkläre an Eides statt, dass ich die der Fakultät für

Bauingenieur- und Vermessungswesen

der Technischen Universität München zur Promotionsprüfung
vorgelegte Arbeit mit dem Titel:

„Verbesserung des Datenmanagements in inhomogenen Rechnernetzen geodätischer Messeinrichtungen auf der Basis von Middleware und Dateisystemen am Beispiel der Fundamentalstation Wettzell“

in der

Forschungseinrichtung Satellitengeodäsie

unter der Anleitung und Betreuung durch

Univ.-Prof. Dr. phil. nat. M. Rothacher und apl. Prof. Dr. rer. nat., Dr.-Ing. habil. U. Schreiber

ohne sonstige Hilfe erstellt und bei der Abfassung nur die gemäß §6 Abs. 5 angegebenen Hilfsmittel benutzt habe.

(X) Ich habe die Dissertation in keinem anderen Prüfungsverfahren als Prüfungsleistung vorgelegt.

( ) Die vollständige Dissertation wurde in .... veröffentlicht. Die Fakultät für ..... hat der Vorveröffentlichung zugestimmt.

(X) Ich habe den angestrebten Doktorgrad noch nicht erworben und bin nicht in einem früheren Promotionsverfahren für den angestrebten Doktorgrad endgültig gescheitert.

( ) Ich habe bereits am ..... bei der Fakultät für ..... der Hochschule ..... unter Vorlage einer Dissertation mit dem Thema .... die Zulassung zur Promotion beantragt mit dem Ergebnis: .....

Die Promotionsordnung der Technischen Universität München ist mir bekannt.

München, den ....
Unterschrift

Gewidmet meinem verstorbenen Großvater, Xaver Stöberl,
meiner über alles geliebten Schwester, Diana Neidhardt,
und meiner Familie

Inhaltsverzeichnis

Kapitel 1

Einführung

Schwerpunkt des Kapitels:

Als Einstieg werden nachfolgend die charakteristischen Messsysteme einer geodätischen Fundamentalstation, wie z.B. Wettzell, in ihrer Vielfalt aufgezeigt, weil sich diese Arbeit damit beschäftigt, ein übergreifendes Konzept für die allgemeine Datenhaltung dort zu entwickeln. Der Auslöser dafür ist das Forschungs- und Entwicklungsprogramm 2001 - 2005 der Forschungsgruppe Satellitengeodäsie (FGS). Über die allgemeine Zielsetzung in dieser Arbeit wird weiter auf die interdisziplinäre Eingliederung übergeleitet. Zum Zwecke des Managements wurden beim Design und der Entwicklung gezielt Methodiken und Techniken der modernen Softwareentwicklung eingesetzt, welche anschließend dargelegt sind. Dazu zählen der verwendete Software-Entwicklungsprozess, der auch Grundlage für den Aufbau der Arbeit ist, die Nutzung der graphischen Modellierungssprache Unified Modelling Language (UML) und Festlegungen zur Programmierung an sich.

1.1 Begründung

Erfassung von geodätischen Daten nimmt an Bedeutung zu

Unser Heimatplanet Erde unterliegt als komplexes System zahlreichen Einflüssen sowohl aus dem Universum als auch aus seinem Inneren selbst. Sie bestimmen einen Großteil der globalen und lokalen Gegebenheiten auf der Erdoberfläche. Gezeiten, Plattentektonik und auch Wetterverhältnisse, um nur einige zu nennen, verändern den Planeten stetig. Das Wissen um solche Geschehnisse und die Kenntnis darüber, in welchem Umfang sie unser tägliches Leben beeinflussen, ist nicht mehr nur im Interesse der Kartographen oder einzelner Wissenschaftler, sondern dringt Dank der Nutzung von moderner Technik mehr und mehr in unser alltägliches Leben vor. Denkt man z.B. nur an das `Global Positioning System (GPS)` welches heutzutage in immer mehr Verkehrsmitteln Einzug hält und die weltweite Positionsbestimmung via Satelliten mit geodätischen Empfängern bereits auf wenige Millimeter genau erlaubt, wird schnell klar, daß Geodaten immer wichtiger werden.

Einsatz von moderner Messtechnik und Großgeräten

Die Geodäsie (= „'Wissenschaft von der Ausmessung und Abbildung der Erdoberfläche'“ sowie des Meeresbodens bzw. anderer Himmelskörper unter Berücksichtigung zeitlicher Veränderungen[1]) beschäftigt sich mit der Erbringung von solchen Vermessungsdaten und den durch Auswertung erbrachten Folgeinformationen. Dabei sind nicht nur die von Vermessern bekannten Ausrüstungen, wie z.B. Theodolit und Messstange, im Einsatz. Um die weltweiten Refernzsysteme auf der Erde exakt bestimmen zu können, werden flächendeckend Großverfahren eingesetzt. Dazu zählen u.a.[2]:

- `Very Long Baseline Interferometry (VLBI)`:

  Wie in der Astronomie auch werden dabei Radioteleskope eingesetzt, welche Mikrowellen von mehrere Millionen Lichtjahre entfernten Quasaren empfangen. Durch die enormen Distanzen zu den Mikrowellensendern können diese als unbeweglich angesehen werden und die abgestrahlten Wellen bilden beim Eintreffen auf der Erde eine gerade Front. Empfangen mehrere Radioteleskope parallel Signale von der selben Quelle können über Laufzeitunterschiede sog. Basislinien zwischen den Empfängerteleskopen ermittelt werden, wodurch u.a. die Bewegungen der tektonischen Platten aber auch die Veränderungen der Poolkoordinaten und der Drehgeschwindigkeit der Erde messbar sind. In Wettzell wird zu diesem Zweck ein Radioteleskop mit einem 20-Meter-Spiegel betrieben.

- `Satellite Laser Ranging (SLR)` und `Lunar Laser Ranging (LLR)`:

  Spezielle Teleskope senden einen Laserstrahl zu künstlichen Satelliten oder zum Mond, wo die Lichtpulse von Retroreflektoren gespiegelt und zur Erde zurück gelenkt werden. Dieser Reflektionsstrahl wird auf der Erde wieder detektiert und zur Ermittlung der Laufzeiten zwischen Sender und Reflektor genutz. Dadurch können Stationskoordinaten und deren Bewegungsvektoren abgeleitet werden. Des Weiteren können die Lage des Geozentrums ermittelt, die Bahn eines Satelliten oder die Polkoordinaten bestimmt werden. In Wettzell wird dafür ein 75cm Teleskop und ein Nd:YAG-Pulslaser verwendet.

[1] vgl. [TOR03] a.a.O. S. 1

[2] vgl. dazu die Erklärungen in [BKGW04]

- GPS :

  Spezielle Satelliten senden zeitkodierte Signalkennungen. Mittels einer Referenzzeit und des daraus generierten Refernzsignals können bei den Empfängern über Signalvergleiche Laufzeiten ausgemessen werden, was dem Schneiden von Kugelhälften um die Satelliten entspricht. Somit ist anhand dieser Ein-Weg-Streckenmessungen eine exakte Positionsbestimmung möglich. In der Geodäsie wird mit speziellen Empfängern, welche die Signalträgerphase berücksichtigen, ein Referenznetz zur Ableitung von Stationskoordinaten und Bewegungsvektoren bestimmt. Ferner können Atmosphären- und Ionosphärenparameter abgeleitet und die Polkoordinaten sowie die Bahnen der GPS -Satelliten bestimmt werden. Von Wettzell aus werden ca. 40 permanente GPS -Empfangsstationen auf der ganzen Welt (u.a. in der Antarktis oder in Lhasa/Tibet) verteilt betrieben.

- Unterstützungsdienste:

  Alle genannten Systeme arbeiten in einem Verbund aus zusätzlich benötigten Meßsystemen. So werden ein Zeit- und Frequenzsystem (z.B. Maser und Cäsiumuhren zur Bildung der Zeitskala UTC (Wettzell)), eine meteorologische Station mit Wasserdampfradiometer (zur Refraktionskorrektur), aber auch verschiedenste Datendienste (z.B. mit den erforderlichen Informationen zur Bahnberechnung eines Satelliten beim SLR) benötigt und von zentralen Diensten zur Verfügung gestellt. Verschiedene andere geodätische Geräte (z.B. ein supraleitendes Gravimeter oder ein Breitband-Seismometer, etc.) oder lokale Vermessungsnetze liefern wichtige Zusatzinformationen zur vergleichenden Messung.

- Forschungssysteme:

  Zu den im täglichen Betrieb eingesetzten Verfahren werden zur Erforschung neuer Meßtechniken auch Experimentalsysteme installiert. So werden zur Zeit Ringlaser zur Vermessung der relativen Veränderung der Erdrotation erprobt. Dabei werden zwei gegenläufige Laserstrahlen (He-Ne Laser) in einem „Ringresonator“ (16 m^2) erzeugt. Die Veränderung der Sagnac-Frequenz aus der Überlagerung beider Strahlen gibt Auskunft über die Veränderungen der Drehgeschwindigkeit der Erde. Da es sich um ein optisches Interferometer handelt sind exakte Temperatur- und Umweltverhältnisse von entscheidender Bedeutung, weshalb solche Systeme in natürlichen oder künstlichen Höhlen installiert sind. Lokale Einflüsse werden mittels zusätzlicher Neigungsmesser registriert.

Fundamentalstation Wettzell als Bündelung aller Verfahren

Alle diese fundamentalen geodätischen Messsysteme zusammengefaßt werden auf der Fundamentalstation Wettzell betrieben. Sie ist eine von weltweit sieben Stationen mit allen elemtaren Verfahren, welche durch eine Vielzahl kleinerer Einrichtungen mit jeweils einem System oder einigen wenigen Systemen flächendeckend unterstützt werden. Betrieben wird die Station vom Bundesamt für Kartographie und Geodäsie (BKG) und von der Forschungseinrichtung Satellitengeodäsie (FESG) der Technischen Universität München.

Problematik durch Heterogenität der Messsysteme untereinander fordern neue Lösungsansätze

Aufgrund der Vielzahl und Verschiedenheit der Systeme mit ihren speziellen Anforderungen und den daraus resultierenden, teilweise immensen Datenmengen in jeweils verschiedenen Formaten und Ausprägungen ergeben sich für heterogene Systeme typische Problematiken. Die auf verschiedene Arten gespeicherten

Informationen (z.B. in Dateisystemen, Datenbanken oder auch Rohdatenquellen) mit ihren unterschiedlichen Zugangsschnittstellen verfügen über wenig oder keine Zusatzinformationen bzgl. Datenstrukturen und eingesetzter Auswertemethoden. Zentralistische Vorgaben versuchen zumindest einige Formate zu fixieren. Die trotzdem aus der Heterogenität resultierenden Probleme und die somit auch wenig automatisierbaren Abläufe binden einen erheblichen personellen Aufwand. Deshalb wird in der Fundamentalstation seit längerem über verschiedene Möglichkeiten einer verbesserten Datenhaltung nachgedacht.[3]

Auslöser für eine Projektstudie

Die im Bereich der Datenhaltung angestoßenen Diskussionen mündeten in der Festlegung einer Projektstudie im Forschungs- und Entwicklungsprogramm 2001-2005 der `Forschungsgruppe Satellitengeodäsie (FGS)`, zu der die Fundamentalstation Wettzell gehört, mit folgendem Ziel: „In einer Projektstudie sollen die Datenmengen, die Datenströme, Archivierungs- und Bereitstellungsmethoden mit dem Ziel analysiert werden, Metadaten in erforderlichem Umfang zu erfassen und zu verwalten und eine effektive und wirtschaftliche Datenhaltung für alle Messverfahren von der Datengewinnung über die Auswertung bis zur Produkterstellung zu entwickeln und einzusetzen.“[4]

Dabei ist die Nutzung von leistungsfähigen Datenbanksystemen zur Archivierung und Bereitstellung der Daten und der Einsatz von standartisierten Metadatenformaten vorgesehen.[5] Unter Berücksichtigung aller Gegebenheiten ergeben sich grob einige Zielvorgaben.

1.2 Zielsetzung

Allgemeingültige, übergreifende Lösungen unmöglich, d.h., sensible Neugestaltung der lokalen Beziehungen untereinander

Betrachtet man sich die Angaben zur Projektstudie genauer, ergeben sich eine Vielzahl von Teilgebieten aus der Informatik, begonnen von der elementaren Sicherstellung der Datenübertragung bis hin zur Archivierung in komplexen Datenbanken mit ihren Managementsystemen. Bei diesem Ansatz müßten alle Spezifika der Einzelsysteme einbezogen werden, die durch internationale Partnerverbünde mitbestimmt sind. Da dies unmöglich ist, geht deshalb die Arbeit hier einen anderen Weg. Die individuellen Charakteristiken und Softwarelösungen (sog. Insellösungen) der Einzelsysteme bleiben wie bisher erhalten. Diese individuellen Lösungsansätze sind nämlich flexibel und handhabbar genug, sich den steigenden, spezifischen und heterogenen Bedingungen anzupassen. Ziel muss es deshalb sein, im Umfeld eines Messsystems weitreichende Umstrukturierungen zu vermeiden. Somit liegt das Hauptaugenmerk auf der Gestaltung der Beziehung zwischen den Messsystemen untereinander und zu zentralen Einrichtungen (z.B. dem lokalen Rechenzentrum mit den Daten der Unterstützungsdienste). Grundlage ist eine Vereinheitlichung aufgrund von existierenden Gemeinsamkeiten beteiligter Komponenten.

Datenbanken für epochenbezogene, geodätische Daten nicht optimal

Allerdings kann hier schon erwähnt werden, dass die Forderung nach Einsatz von leistungsfähigen und eher komplexen Datenbanksystemen allgemein nicht unbedingt optimal ist. Da geodätische Daten zumeist chronologisch geordnet und

[3] vgl. [FGS01] a.a.O. S. 59
[4] [FGS01] a.a.O. S. 59
[5] vgl. [FGS01] a.a.O. S. 59

epochenbezogen sind, haben einzelne Datensätze alleine keine oder nur geringe Aussagekraft. Eine Verarbeitung der Daten findet deshalb für die logisch zusammenhängenden Abschnitte meist blockorientiert statt, was der Bedeutung von „File“ (entsprechend einer Karteikarte oder Akte) sehr gut entspricht und damit durch eine Dateistruktur ideal umgesetzt werden kann. Zudem spricht dafür auch das Grundverständnis der Anwender und Systembetreuer, welche die Datenmenge „Datei“ als logische Einheit besser verwalten können, als komplexe mengentheoretische Abfragekombinationen. Damit soll es auch nicht Ziel der Arbeit sein, alle Datentypen auf Datenbanken abzubilden, sondern allgemeine Zugangspunkte zu schaffen, welche die Speicherung abstrahieren.

Aufteilung des Verarbeitungsprozesses in Produktion und Dienstleistung

Grundsätzlich kann man die Verarbeitung von geodätischen Daten in zwei Bereiche aufspalten: dem Produktionsprozess und dem Dienstleistungsprozess. In der Produktion werden sämtliche Daten erbracht, d.h., es werden Messdaten ermittelt, ausgewertet und als Resultat persistent abgelegt (= Archivierung). Somit ist hier also ein sicheres Datenflusskonzept (= Dataflow) und ein nahezu automatisch ablaufender Arbeitsfluss (= Workflow), in dem zustandsbasiert ohne Datenverlust agiert werden kann, zu fordern. Die letztendlich entstandenen Informationen für die Geodäsie werden als Dienstleistung einer Vielzahl von Abnehmern zur Verfügung gestellt (= Bereitstellung), wodurch ein effektives Finden und Erlangen der Daten über entsprechende Zusatzinformationen (= Metadaten) mit allgemein üblichen Zugangsmethoden (z.B. über `Hypertext Transfer Protocol (HTTP)`) anzustreben ist.

Zusätzlich existieren zyklische Prozesse

Demgegenüber steht z.B. die Verwaltung von Landvermessungsdaten zur Vermarkung, wie sie u.a. von Abteilungen des `BKG` erbracht wird. Dabei handelt es sich um keinen linearer Ablauf. Er bezieht sich auf wechselnde Geoinformationen (z.B. für Landkartendaten), welche kontinuierlich angepaßt werden müssen. Dieses Feld ist ideal auf Datenbanken abzubilden. Somit kommen zu den linearen Workflows zyklische hinzu.

Konzentration der Arbeit auf Anforderungen eines Datenlieferanten

Da die Fundamentalstation als reiner Datenlieferant hauptsächlich im Produktionsprozess angesiedelt ist und nur am Rande Daten als Dienstleister z.B. ins Internet stellt (wie z.B. bei den meteorologischen Messungen), werden sich die Lösungen hauptsächlich auf die Anforderungen auf das Umfeld der Erzeugung und Lieferung von epochenbezogenen Daten konzentrieren. Für reine Datendienstleistungen laufen parallel vergebene Aufträge (wie z.B. für den `International Earth Rotation and Reference Systems Service (IERS)` mit Sitz beim `BKG` in Frankfurt), welche sich im Speziellen mit der Auswahl und Bereitstellung von Daten und zugehörigen Metadaten befassen und diese auf geeignete Weise z.B. über Datenbanken zur Verfügung stellen.

Kombination von bekannten Einzellösungen mit neuen Ideen

Obwohl grundsätzlich die meisten der Probleme einzeln gesehen in der Informatik gelöst sind, wirft die Kombination dieser Fragestellungen und Lösungsmöglichkeiten immer noch Probleme auf. Zudem sollen erweiterte Denkansätze entwickelt und integriert werden, welche nicht nur theoretische Gebilde bleiben, sondern am konkreten Fall eines Prototypen in der Praxis Erprobung finden. Der entstehende Prototyp ist dabei pflegbar, erweiterbar und für zukünftige Anforderungen geeignet. Generell steht das lokale Verbundnetz der Fundamentalstation zum Test zur Verfügung, welches sowohl mit qualitativ schlechten Datenübertragungsstrecken, z.B. zu einer Station in Lhasa (Tibet), als auch mit modernen In-

ternetdiensten, Clustercomputing und browserunterstützten Abfragen bzw. Steuerungstechniken agiert.

Alles in allem ergibt sich ein immenses Aufgabengebiet, das in viele Bereiche der Informatik eindringt und damit verschiedenste Richtungen zu vereinen versucht.

1.3 Interdisziplinäre Eingliederung der Arbeit

Lösung in erster Linie auf Fundamentalstation Wettzell bezogen, jedoch verallgemeinerbar

Die Arbeit ist von der Natur der Sache her stark von Informationstechnik und Informatik im Allgemeinen geprägt. Trotzdem muss sie als fächerübergreifend betrachtet werden, da sie in erster Linie Problemlösungen für Teile aus der Geodäsie zu finden versucht. Im Besonderen werden hier Umsetzungen für konkrete Anwendungsfälle auf der Fundamentalstation Wettzell gesucht und entwickelt. Da aber dort erkannte Probleme zumeist von genereller Natur sind, welche in ähnlicher Weise auch in anderen informationsverarbeitenden Bereichen auftreten, können Lösungsansätze und erbrachte Ergebnisse auf diese übertragen werden.

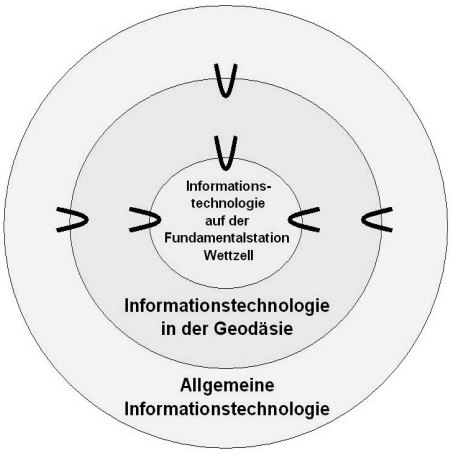


Abbildung 1.1: Eingliederungs- und Verallgemeinerungskette für diese Arbeit

Verallgemeinerungsansatz

Somit ergibt sich eine Verallgemeinerungskette für die Lösungsansätze nach Abb. 1.1 auf Seite 6. Der erstellte Prototyp wird in erster Linie der Fundamentalstation Wettzell zu Gute kommen, in der sukzessive die angedachten Verbesserungen einfließen können. Entwickelte Grundideen, z.B. bzgl. Strukturierungen und Arbeitsabläufe, sind hauptsächlich auf geodätische Daten bezogen und können damit ebenso auf die ähnlichen Abläufe in übergeordneten Bereichen der Informationsverarbeitung in der Geodäsie angewandt werden. Eine generelle Verallgemeinerung auf allgemeine Daten und Abläufe wiederum schafft den Bezug und Einfluss in die allgemeine Informationstechnologie und Informatik.

1.4 Eingesetzte Arbeitstechniken

1.4.1 Der Softwareprozess

Moderne Softwareprozesse integrieren in einen Grobablauf zahlreiche Iterationsschritte zur Implementation

Die Auswahl eines geeigneten Softwareprozesses[6] und dessen Ausgestaltung mit Umsetzungsinhalten ist der erste Teil einer Pogrammentwicklung. Die früher bekannten Top-Down-Verfahren, wie z.B. das „Wasserfallmodell“, mit strikten, azyklischen Phasen von der Analyse über das Design zur Implementierung und schließlich zum Gesamttest werden dazu längst nicht mehr in dieser Form eingesetzt. Moderne Prozesse , wie z.B. auch der Rational Unified Prozess (vgl. Abb. 1.2 auf Seite 7), beinhalten diese Modelle nur noch in Form eines Grobablaufs, welcher die Hauptelemente Einstieg (Grobanalyse), Ausarbeitung (Grobdesign), Konstruktion (Implementation) und Überleitung (Test und Übergabe) beinhaltet. Der entscheidende Unterschied ist jedoch der, dass die Konstruktionsphase ihrerseits wieder aus zahlreichen Iterationen aufgebaut ist, welche aus ähnlichen Phasen bestehen und nach Abschluss jedes Iterationsschrittes für den realen Betrieb nutzbare und geeignete Software liefern. Jede Iteration enthält somit wieder einen Lebenszyklus mit den Teilphasen Analyse, Entwurf, Implementierung und Test. Nachfolgend wird anhand allgemeiner Vorgehensweisen am Beispiel der vorliegenden Projektstudie der umgesetzte Softwareprozess aufgezeigt.

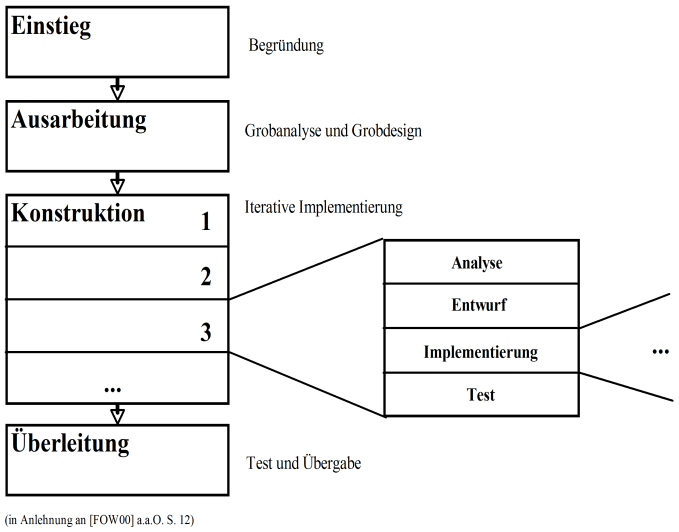


Abbildung 1.2: Aufbau des Softwareprozesses

Der Einstieg als Begründung eines Vorhabens

Der erste Einstieg ist als Grobanalyse die eigentliche Begründung des Vorhabens. Im vorliegenden Fall wird hier auf Erkenntnisse aufgebaut, welche sich aus dem Betrieb der Messsysteme ergeben.

Die Ausarbeitung als Plan einer übergeordneten Konstruktion

Bei der Ausarbeitung werden Anforderungen gesammelt. Die daraus abgeleitete erweiterte Analyse und das Grobdesign erstellen damit einen Plan für die übergeordnete Konstruktion auf hoher Ebene. Er ist vergleichbar mit dem Grundriss eines Hauses. Man muss wissen, welche Zimmer man in welcher Form haben möchte und welche Funktionen sie einnehmen werden, damit sie entsprechend miteinander kombinierbar sind. Auch entsprechende Außenwirkungen, wie Anschlüsse etc.,

[6] vgl. dazu [FOW00] a.a.O. S. 11 ff.

können hier bereits grob festgehalten werden. Dasselbe gilt auch für die Software. Es entstehen eine grobe Konstruktionszeichnung und erste Ideen für die Umsetzung und deren Realisierbarkeit. Kenntnisse über interne Berührungspunkte und die Außenwirkung (Schnittstellen), sowie ein erster Zeitplan für die iterativen Phasen. Ziel ist die Minimierung von Risiken vor allem bzgl. der geforderten Merkmale einer Software, den technologischen Herausforderungen und dem eigenen Know-how. Im vorliegenden Projekt wäre dies das Kapitel zur Analyse und der erste grobe Aufriss einer Inhaltsangabe.

Die Planung der Iterationen

Der nächste Schritt ist die Planung der Iterationsschritte der Konstruktionsphase. Da es sich grundlegend um ein Ein-Mann-Projekt handelt, müssen zeitliche Planungen, eingesetzte Techniken usw. auch von einem Mann zu bewältigen sein. Dieser erste Plan steckt im Pflichtenheft, welches jedoch nur ein Rahmenkonzept vorgibt. Wichtig hierbei ist, dass risikoreiche Umsetzungen an den Beginn der Arbeiten gesetzt werden. Deshalb bestehen die ersten Kapitel in der vorliegenden Arbeit vorwiegend aus dem Umsetzen neuer Technologien, während Teile aus den späteren Iterationen im Wesentlichen nur noch Anwendungscharakter dieser Entwicklungen haben.

Vergleich von wissenschaftlichen und kommerziellen Projekten

Ein kennzeichnender Charakterzug von wissenschaftlichen Projekten ist die Erforschung von Neuartigem und damit bisher Unbekanntem. Erforschen bedeutet jedoch immer, sowohl negative wie positive Erfahrungen zu sammeln. In wissenschaftlichen Projekten ist damit ein neuer Erkenntnisstand, egal ob positiv oder negativ, ein Fortschritt. In kommerziellen Entwicklungen sind hier die Regeln etwas strenger. Fortschritt und Erfolg hängen von vielen Faktoren ab: Funktionalität, Zeitplanung/Time-To-Market, Kundenbewertung, usw. Im vorliegenden Fall liegt eine Mischung der beiden Extrema vor. Um dem gerecht zu werden, sind nur Zeitplanungen und Phasen für die nächsten ein bis zwei Iterationen als vertrauenswürdig anzusehen. Je später eine Phase vom Planungszeitpunkt entfernt ist, desto weniger Wissen existiert darüber und desto unsicherer sind Einschätzungen dazu. Dies ist insbesondere deshalb so, weil aufgrund des Forschungscharakters neue Techniken eingesetzt werden, welche in sich bereits ein erhöhtes Risikopotential tragen.

Flexibilität durch handhabbare Einheiten

Deshalb kommt auch in der vorliegenden Projektstudie den Iterationen besondere Bedeutung zu. Sie markieren die Meilensteine des Projekts, teilen den Gesamtaumfang in handhabbare und beherrschbare Einzelbereiche auf und liefern jeweils bereits nutzbare Software. Jegliche Weiterentwicklung ist damit ein Fortschritt sowohl im wissenschaftlichen als auch im kommerziellen/betrieblichen Sinn. Insbesondere die Analyse zu Beginn jeder Phase hat hier gehobenen Stellenwert. In dieser Teilphase kann über gemachte Erfahrungen reflektiert werden, welche zukünftige Entscheidungen maßgeblich beeinflussen. In ihr sind genaue Kenntnisse für die nachfolgenden Entwicklungsschritte zu erwerben. So kann auch kurzfristig auf technische Neuerungen oder Umsetzungsprobleme reagiert werden. Oder um bei der Metapher des Hausbaus zu bleiben: Für jedes Zimmer wird individuell entschieden, wie es gestaltet wird. Der Grobplan kann zwar nur noch schwerlich umgeworfen werden, wenn sich das Gebäude bereits im Rohbau befindet. Innere Verteilungen, Zuordnungen, begrenzte Außenwirkungen oder Konstruktionen bleiben aber flexibel.

Vermeidung von Entwicklungsschleifen

Das gilt auch für die Software. Wichtig ist nur, dass Rückschritte und Phasenwiederholungen über eine möglichst genaue Analyse zu Beginn eines Entwicklungsschritts vermieden werden. Dann spielt es auch keine Rolle mehr, wenn Implementierungen nach dem aufgezeigten Schema weiter in Unterphasen des gleichen Lebenszyklusses unterteilt werden. Diese selbstähnlichen Zyklusstrukturen fördern nachhaltig die Handhabbarkeit durch begreifbare Einheiten.

Qualitätskriterien: Stabilität, Funktionalität, Codeaufbau und Leistung

Allerdings muss für die vorliegende Arbeit eingestanden werden, dass die Tests aus Zeitgründen nicht in Form von definierten Testsuiten umfangreich gestaltet werden können. Zumeist werden deshalb die entwickelten Teile parallel zu den existierenden Produktionsmechanismen installiert und so unter realen Bedingungen verwendet. Qualitätsmerkmale sind somit Stabilität und Funktionalität. Weitere Bewertungskriterien sind Codebewertungen und Leistungsmessungen.

Stufenweise Übergabe zur Überleitung der Software

Letztendlich ist so auch nach jedem Zyklus die Möglichkeit gegeben, entstandenes einzusetzen und bereits parallel zur Weiterentwicklung Phasen der Überleitung und Übergabe der Software einzuleiten.

Insgesamt ergibt sich ein flexibler Optimierungsprozess

Insgesamt ergibt sich also eine Möglichkeit, einen Optimierungsprozess am Laufen zu halten, welcher von Meilenstein zu Meilenstein die weiteren Entwicklungspfade auswählt und damit sowohl neue Erkenntnisse im wissenschaftlichen wie auch im technisch-betrieblichen Sinn erbringt. Deshalb ist auch die Gliederung der Arbeit an dieser Vorgehensweise orientiert (vgl. Abschnitt 1.5 auf Seite 15). Generalisiert kann somit auch die der Arbeit zugrundeliegende Projektstudie als Ausarbeitung einer übergeordneten Entwicklung gesehen werden, welche Ideen für zukünftige Weiterentwicklungen auf der Fundamentalstation Wettzell und evtl. darüber hinaus vorstellt.

1.4.2 Unified Modelling Language (UML)

UML als standardisierte, graphische Modellierungssprache

Die `Unified Modelling Language (UML)` ist eine von der `Object Management Group (OMG)` standardisierte Modellierungssprache, welche durch ihre vorwiegend graphische Notation von Entwicklungsmethoden verwendet wird, um Entwürfe auszudrücken[7]. Zusammen mit dem Softwareprozess bildet sie die Grundlage der Entwicklungsmethode. Die `UML` ist somit ideal geeignet, um über Entwürfe zu diskutieren und zu entscheiden.

Abstrakte Beschreibung eines Metamodells

Mit `UML` werden keine formalen Methoden umgesetzt. Vielmehr intuitive und informelle Beschreibungen eines Designs gegeben, welche in eine graphische Notation eingebettet sind. Was daraus entsteht ist das Metamodell, welches objektorientierte Zusammenhänge zwischen den einzelnen Objekten darstellt, ohne dass bereits Code erzeugt werden muss. So dient sie während der Codeerzeugung als Regelwerk und später als Dokumentation (vgl. dazu Abb. 1.3 auf Seite 10.

Am Anfang stehen die Anwendungsfälle

Eine erste Herangehensweise ist die Festlegung von Anwendungsfällen. Sie beschreiben in Stichpunkten, welche Interaktionen mit einem Nutzer stattfinden, wer beteiligt ist und welche weiteren Abhängigkeiten daraus entstehen. Eine graphische Umsetzung ist das Anwendungsfalldiagramm.

[7] vgl. zum nachfolgenden Abschnitt [FOW00] a.a.O. S. 1 ff.

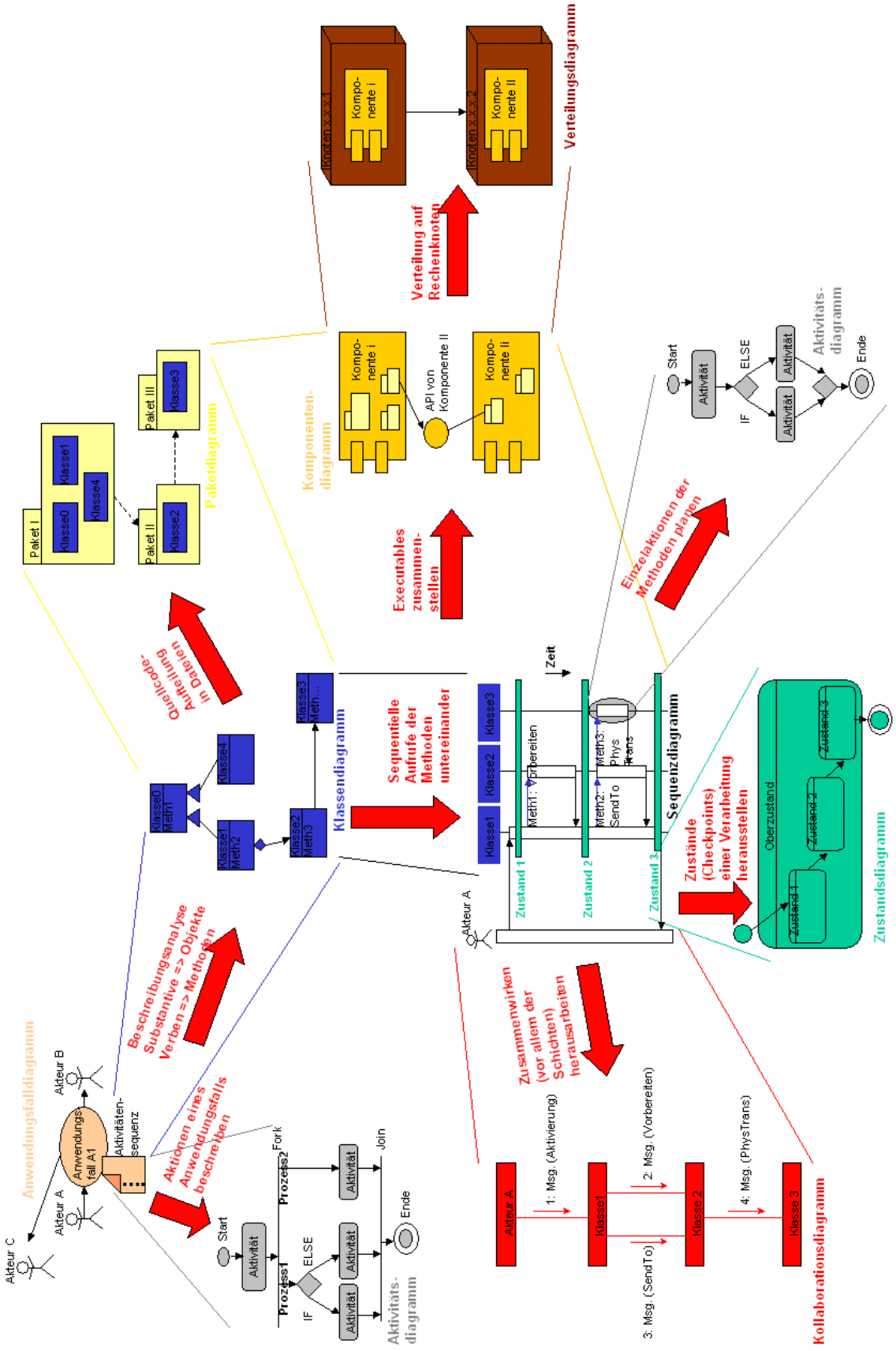


Abbildung 1.3: Zusammenhänge der UML-Diagramme

Erstellung des Klassendiagramms als eines der Basisdiagramme

Die aus diesen stichpunktartigen Beschreibungen abgeleiteten realen Objekte (Substantive) können mit bestimmten Eigenschaften (Attributen) versehen werden. Um ihre Aktionen ausführen zu können, benötigen sie Methoden (Verben), welche von anderen Interaktionspartnern aufgerufen werden. Verwandte Objekte können voneinander abgeleitet werden, was in dieser Stufe als Generalisierung bezeichnet wird. Zusammengesetzte Objekt (Container) können aus anderen statisch aufgebaut sein (sog. Einbettung). Zudem können Objekte dynamische Interaktionen betreiben, so dass sie voneinander wissen müssen (sog. Assoziation). So ergeben sich die Grundzusammenhänge für das objektorientierte Klassendiagramm, welches die Objekte und ihre Beziehungen zueinander beschreibt. Sie werden für die Umsetzung der Prototypen in dieser Arbeit hauptsächlich eingesetzt.

Einsatz der weiteren Diagramme zu Dokumentationszwecken in der Studie

Alle weiteren Diagramme beschreiben entweder interne Abläufe (Aktivitäten, Sequenzen, Kollaborationen, Zustände), abbildende Vorschriften auf die Strukturen von Programmdateien (Pakete) oder der ausführbaren Codes auf die Recheneinheiten (Verteilung). Sie wurden während der Studie mehr oder weniger dann verwendet, wenn dadurch eine dokumentierende Möglichkeit zur Darstellung interner Zusammenhänge sinnvoll war.

1.4.3 Objektorientierte Programmierung, Designregeln und Versionskontrolle

Objektorientierte Programmierung (OOP)

Überführung des Objektmodells der Realität in objektorientierten Code

Durch Einsatz vorstehender Methoden wird ein Modell geschaffen, welches ein vereinfachtes Abbild der zu entwickelnden Wirklichkeit ist[8]. Dieses Modell muss nun über mehrere Transformationsstufen in eine als Code implementierte Abstraktionsebene überführt werden[9]. Da sich die Welt in der Softwareentwicklung ideal mit Hilfe von Objekten modellieren lässt (vgl. dazu den vorherigen Abschnitt zu `UML`), gilt es, diese auch in Form der Programmierung nachzuempfinden. Mittel zum Zweck ist deshalb die `Objektorientierte Programmierung (OOP)`.

Schreiben von qualitativ guten, wiederverwendbaren und übersichtlichen Programmen

Dabei handelt es sich um ein Paradigma zum Schreiben von qualitativ guten, wiederverwendbaren und übersichtlichen Programmen[10]. Die Sprachen dafür unterstützen direkt die Vorgehensweise, wie sie in der Entwicklungsmethode vorgegeben wird. Der Unterschied zu prozeduralen, strukturierten Programmen liegt darin, dass dort die Eigenschaften der Objekte (Variablen bestimmter fester oder generischer Datentypen) getrennt von ihren Aktionen (Funktionen, Prozeduren) betrachtet werden, die sie nutzen. Somit muss jede Aktion individuell entscheiden, welche Eigenschaften die momentan genutzten Objekte haben. Im Umfang großer Programme ergeben sich so zahlreiche Verstrickungen und Verwirrungen[11].

[8] vgl. dazu [LIB00] a.a.O. S. 689
[9] vgl. dazu [LIB00] a.a.O. S. 712
[10] vgl. dazu [STRO92] a.a.O. S. 18 ff.
[11] vgl. dazu [STRO92] a.a.O. S. 25 ff.

Beispiel 1.1 für „Strukturierte Programmierung“
Im Falle des Zeichnens von Objekten, z.B. Kreis, Rechteck, usw., muss eine Zeichenfunktion intern entscheiden, was die Charakteristika sind und wie diese ausgegeben werden müssen. Obwohl alles verwandte Objekte sind. Sie alle bestehen aus Linien, die wiederum aus Punkten (Pixel) aufgebaut werden.

Gekapselte Klassen mit benötigten Variablen und Methoden als Basis

Die OOP geht einen anderen Weg. Sie definiert Programm-/Speicherstrukturen, die sog. Klassen, die ein Abbild realer Objekte sind. Sie beinhalten die benötigten Variablen und Methoden, welche zur Erbringung der Eigenschaften eines Objekts notwendig sind. Durch die im Abschnitt zu UML erwähnten Beziehungen (vgl. Abschnitt 1.4.2 auf Seite 9) können somit Generalisierungen entstehen, welche in der OOP als Ableitungen bezeichnet werden. Diese Strukturierungen in der OOP sind wohl die wichtigsten Merkmale.

Beispiel 1.2 für „Objektorientierte Programmierung“
Im obigen Beispiel bedeutet das, dass eine gerade Linie und eine gebogene Linie Ableitungen von Pixelausgaben sind. Ein Zeichenobjekt besteht aus solchen Linien (Container) und hat als Ableitungen die echten Erscheinungsbilder Kreis, Rechteck, usw.

Weitere Vorteile ergeben sich durch explizite und implizite Typ- und Zugriffsregelungen

Aufgrund des Klassenkonzepts sind zudem weitere Vereinfachungen möglich. Eine Klasse kann allgemein typunabhängig definiert sein und erst bei der Instanzierung mit bestimmten Eigenschaften eines Typs versehen werden (Templates), d.h., dass z.B. eine Vektorklasse allgemein mathematische Vektorrechnung umsetzt und erst bei der Nutzung der Typ der Zahlen (integer, float, etc.) gesetzt werden muss. Implizite Typumwandlungen können aufgrund Vererbungsbeziehungen realisiert werden. Diese können sogar so aufgebaut sein, dass eine Klasse von mehreren anderen Eigenschaften ererbt. Und letztendlich ist durch das Klassenkonzept zwangsläufig eine Kapselung des Codes in Sinneinheiten gegeben, wodurch sogar eine implizite Speicherverwaltung und Zugriffsregelung (welches Objekt kann welche Datenbereiche eines anderen Objekts einsehen) möglich wird.

Standardisierung durch Nutzung der STL

Eine weitere Vereinfachung wird durch die Nutzung von standardisierten Bibliotheken erreicht. Für die Programmiersprache C++ sind diese z.B. in der Standard Template Library (STL) zusammengefasst[12]. Grundbestandteile der STL sind drei fundamentale Elemente: Container, Algorithmen und Iteratoren. Erstere sind Objekte, welche andere Objekte in sich aufnehmen können. Dazu zählen sequentielle Container, wie z.B. dynamische Felder („vector“), Warteschlangen („queue“) und Listen („list“), aber auch assoziative Container, welche den Zugriff mit Schlüsselelementen erlauben, wie z.B. Abbildungen mit eindeutigen Schlüsseln („map“). Die Algorithmen agieren auf den Elementen in den Containern und initialisieren, sortieren, suchen oder transformieren sie. Die Iteratoren wiederum sind verfeinerte Zeigerobjekte, die es erlauben, direkt auf Elemente in den Containern zyklisch zuzugreifen. Sie können wie die Zeiger inkrementiert, dekrementiert oder dereferenziert werden. Zu diesen Bestandteilen ergänzen sich zusätzlich Allokatoren zur Speicheranforderung für die Container und Prädikatoren, um Aussagenlogik zu betreiben (größer, kleiner, usw.).

[12] vgl. dazu [SCHI98] a.a.O. S. 626ff.

In der Regel wird die STL von jedem gängigen Compiler unterstützt. Als erstes entscheidet man sich für die Art eines Containers, der zur Problemlösung ideale Eigenschaften aufweist. Zur Nutzung werden dann seine Member-Methoden verwendet, um Elemente einzufügen oder zu entfernen. Alle weiteren Aufgaben, z.B. der Speicherverwaltung, übernimmt der Container. Zusätzlich können Iteratoren auf Elemente platziert werden, welche sich am Anfang oder am Ende befinden oder mittels Suchmethoden gefunden wurden. Ist einmal ein Container entsprechend gefüllt, kann sein Inhalt mittels der Algorithmen manipuliert werden.

Breite Unterstützung und kompaktes Design der STL

Diese standardisierten Bibliotheken erlauben es somit, Entwicklungsaufwand einzusparen und bieten eine gut durchgetestete Basis. Beachtet werden muss hier nur, dass Container nicht mit herkömmlichen Zeigern zurechtkommen und deshalb mit entsprechenden Typdefinitionen gearbeitet werden muss. Das liegt daran, dass für sie kein Speichermanagement vorliegt und so bei Kopieraktionen der pointierte Speicherplatz nicht freigegeben werden kann.

Standardisierte Bibliotheken sparen Entwicklungsaufwand

Es wird damit klar, dass die OOP (zusammen mit komfortablen Makefiles, welche automatisch Modulabhängigkeiten verwerten) insgesamt ein exaktes Abbild der objektorientierten Entwicklungsmethoden darstellt. Allerdings hat sie auch ihre Tücken. Bindeglieder zwischen den Klassen sind die Beziehungen. Dahinterliegende Zusammenhänge können von außen als Black Box betrachtet werden. Ändern sich jedoch diese Beziehungsschnittstellen, ist ein erheblicher Umbauaufwand erforderlich. Die OOP ist nicht aus sich heraus ein Garant für Übersichtlichkeit und Funktionalität. Die Planung durch Softwareentwicklungsmethoden und eine gewisse Disziplin bei der Umsetzung sind hier genauso wichtig.

Aus der OOP ergeben sich zahlreiche Vorteile, jedoch nicht eo ipso

Design- und Umsetzungsregeln

Deshalb wird in der vorliegenden Arbeit nicht nur die OOP genutzt, sondern zusätzlich ein Regelsatz an Design- und Umsetzungsvorgaben aufgestellt. Sie regeln die direkte Umsetzung in Quellcode. Unterstützung finden sie durch automatische Erzeugungstools, wie sie in UML -Werkzeugen Anwendung finden (z.B. Rational Rose), um von den Klassendiagrammen direkt das Skelett der Klassen zu generieren. Erweitert ist in diesen Regeln auch festgesetzt, wie Kommentare eingebracht werden müssen, welche Strukturen die Quelldateien und zugehörigen Pfade besitzen und welche Programmierkonstrukte wie zu verwenden sind (z.B. Zeiger, Sprungbefehle usw.).

Design- und Umsetzungsvorgaben als Regelsatz für die eigentliche Programmierung

Eine weitere Regelung betrifft die Benennung von Bezeichnern. Um schon aus den Namen Rückschlüsse auf die Semantik zu erhalten und dadurch Fehler zu vermeiden (z.B. Zuweisung von vorzeichenbehafteten Werten an vorzeichenfreie) wird hierfür die Hungarian Notation von Charles Simonyi genutzt[13]. Sie wird in einer geeigneten Abwandlung verwendet (Grundlage der allgemeinen Designregeln für die Fundamentalstation Wettzell, siehe weiter unten), um durch Voranstellen von kennzeichnenden Präfixen über drei Stufen hinweg Eigenschaften zu beschreiben.

Einsatz der Hungarian Notation von Charles Simonyi

[13] vgl. dazu [MAG93] a.a.O. S. xxv-xxviii

1. **Notationserweiterung:** Präfix steht direkt vor Bezeichner
   An dieser Stelle werden Typinformationen durch vorangestellte eindeutige Buchstaben codiert (z.B. „c“ für Character, „i“ für Integer, „S“ für Structure, „C“ für Class usw.)

2. **Notationserweiterung:** Präfix steht vor 1. Notationserweiterung
   An dieser Stelle werden Referenzierungseigenschaften gekennzeichnet (z.B. „a“ für Array, „az“ für zero-terminated Array, „p“ für C-Pointer, so dass zusammen mit Präfix 1 z.B. „azc...“ für ein herkömmliche Character- Zeichenkette mit Nullabschluss steht)

3. **Notationserweiterung:** Präfix steht vor 2. Notationserweiterung
   An dieser Stelle werden Speicherinformationen angezeigt (z.B. „m_“ für Member-Variablen, „g_“ für globale Variablen, so dass zusammen mit Präfix 1 und 2 z.B. „m_pi...“ für einen Member-Zeiger in C-Notation auf einen Integerwert steht)

Die Hungarian Notation erzwingt präzises Arbeiten

Zudem wird durch die Hungarian Notation die Vergabe von präzisen Namen vorgeschlagen (Indexvariablen heißen somit nicht mehr nur „i“ sondern z.B. „iIndex“). Ein gewisser Nachteil ist jedoch, dass Typänderungen eine Vielzahl von Folgeveränderungen mit sich bringen, was ja eigentlich Sinn und Zweck für die qualitativ hochwertige Programmierung ist, keine schnellen und evtl. fehlerträchtigen Veränderungen zuzulassen.

Allgemeingültige Designregeln für die Fundamentalstation als ein Ergebnis

Dabei kann bereits der erste Erfolg dieser Arbeit genannt werden. Die im Rahmen der Arbeit erstellten Design- und Umsetzungsvorgaben wurden als Grundlage der „Design-Rules für die objektorientierte Programmierung in C++ und die strukturierte Programmierung in C“[14] verallgemeinert und erweitert und werden in dieser Form zukünftig auf alle Softwareprojekte mit C und C++ in der Fundamentalstation Wettzell Anwendung finden.

Versionskontrolle

Kontrolle von konkurrierenden Parallelversionen einer Entwicklung mittels CVS

Versionskontrolle entstand als Koordinierungsschnittstelle bei paralleler Softwareentwicklung sowohl bei industriellen Großprojekten als auch für Projekte der freien Softwareentwicklung („Open Source“). Gerade dort, wo zahlreiche Beteiligte auf der ganzen Welt an denselben Quellen arbeiten und Änderungen einbringen, ergeben sich unumstrittene Vorteile aus einer Versionskontrolle. Die Organisation zum Einbringen dieser Rückmeldungen ist manuell unüberwindbar. Zudem ergibt sich der Bedarf, eingebrachte Veränderungen wieder rückgängig zu machen, weil sie evtl. fehlerbehaftet sind. In diesem Rahmen entstand der elementare Einsatz von Dateivergleichsprogrammen (z.B. „diff“) und des `Revision Control System (RCS)`. `RCS` hatte jedoch einige Nachteile. Es stellte keine Verzeichnisstrukturen bereit, erlaubte nur strikte Sperren kompletter Dateien und war nicht netzwerkfähig. Die Lösung ergab sich durch das `Concurrent Versions System (CVS)`. Es setzte auf `RCS` auf und löste dessen Probleme. Dadurch können Dateibäume als komplette Projektbereiche betrachtet und verwaltet werden. Es ist möglich gleichzeitig an einer Datei zu arbeiten, deren Historie aufgezeichnet und Veränderungen meist automatisch zusammengemischt („merging“)

[14] vgl. dazu die erstellten Designregeln für die Fundamentalstation Wettzell [DASS04]

werden. Und CVS ist netzwerkfähig, so dass Quellen auch über das Internet bearbeitet werden können[15].

Historienbezogenes, strukturiertes Ein-/Auschecken von Revisionen

Zusammengefasst betrachtet beinhaltet damit CVS zahlreiche nützliche Eigenschaften, welche auch bei Ein-Mann-Entwicklungen nützlich sind[16]. Dateien eines Projekts können logisch strukturiert eingecheckt, verwaltet und von jedem Rechner eines Netzes abgerufen (checkout) oder mit einem Update versehen werden. Gerade bei Entwicklungen einer Software für verschiedene Plattformen mit gleichbleibenden Anteilen (wie in dieser Arbeit gegeben), kommen auch die Prinzipien paralleler Entwicklung zum Tragen, bei denen Änderungen automatisch zusammengefasst werden. Die Verwaltung der Änderungen (Revisionen) schreibt in kurzen Stichpunkten mit, wer was wie und wann verändert hat. Diese Historieninformationen können direkt in die Quellcodes eingeblendet werden, so dass auch dort die Entwicklungsgeschichte der Datei ersichtlich ist. Rückschritte zu früheren Versionen sind einfach möglich. So ist es auch möglich, Momentaufnahmen durch Zeitstempel und Marken zu definieren, welche die einzelnen Releases der Meilensteine markieren. Von dort ab lassen sich sogar getrennte Entwicklungszweige verwalten, sofern die Software bereits eingesetzt wird, jedoch Teile davon in neueren Versionen verbessert wurden (d.h., es wird der Softwareentwicklungsprozess unterstützt).

Resümee::
Techniken fördern von Grund auf Qualitätsmerkmale, um ein umfangreiches Aufgabenfeld managen zu können

Mit all diesen Methoden und Techniken als Stand der Kunst ist es möglich, komplexe Aufgaben zu lösen und fortschrittliche Software zu entwickeln. Gerade darin unterscheidet sich auch diese Arbeit von anderen auf spezielle Teilbereiche ausgelegten, wissenschaftlichen Arbeiten. Ihr Umfeld ist so breit gesteckt, dass brauchbare Lösungen ohne methodisches Vorgehen, wie sie in kommerziellen Entwicklungen üblich sind, nahezu unmöglich wären.

1.5 Aufbau der Arbeit

Gliederungsstruktur ist der Softwareprozess

Die Arbeit strukturiert sich generell nach Art des in Abschnitt 1.4.1 auf Seite 7 beschriebenen Softwareentwicklungsprozesses. Ihre Kapitel und Unterkapitel folgen seinen Prozessphasen. Der Aufbau ist in Abb. 1.4 auf Seite 16 graphisch dargestellt.

Widerspiegelung der Lebenszyklen der Entwicklungsphasen

In der Einführung wird im Prinzip der Einstieg dargelegt. Die folgende Grobanalyse ist die erste Ausarbeitung. Alle weiteren Kapitel gliedern sich in die Lebenszyklen der Konstruktionsphasen. Der Anfang ist die Analyse mit Ausgangsüberlegungen, Einordnungen und theoretischen Grundlagen. Die ausgewählte Vorgehensweise wird in einem Entwurfsteil beschrieben, in dem auch speziell zu erwähnende Implementierungsansätze aufgezeigt sind. Zum Abschluss folgen Erfahrungen, Erkenntnisse und Bewertungen, welche als eine Art Test angesehen werden können.

[15] vgl. zum vorhergehenden Abschnitt [FOG02] a.a.O. S. 35 ff.
[16] vgl. dazu im Allgemeinen [FOG02]

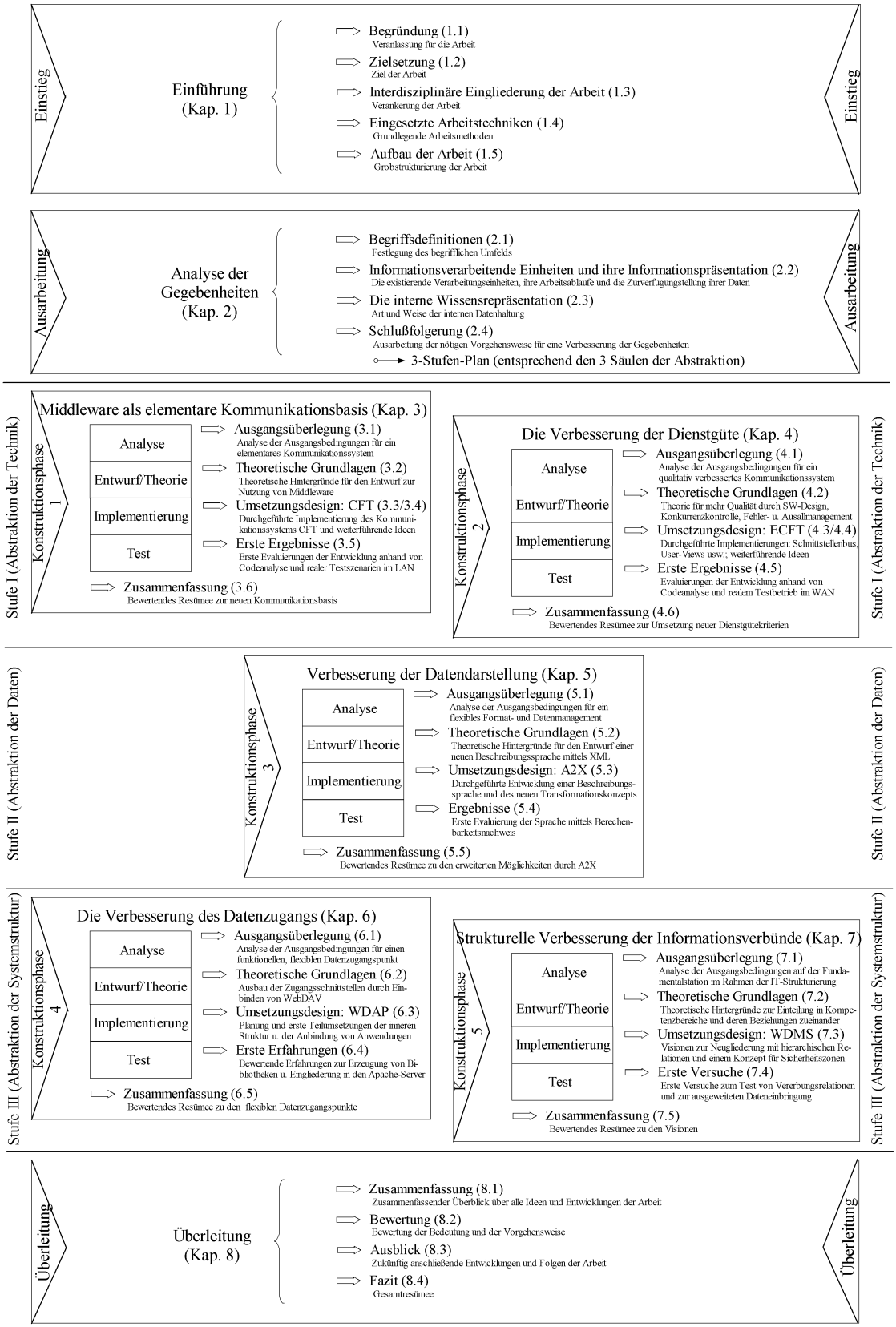


Abbildung 1.4: Aufbau der Arbeit

Zusammenfassung und Bewertung als Überleitung

Zum Abschluss der Arbeit wird als Zusammenfassung eine Gesamtbewertung geliefert, welche die Überleitung im Softwareprozess repräsentiert. Im Anhang befinden sich schließlich detailiertere Informationen zu ausgewählten Themengebieten bzw. weiterführende oder erklärende Informationen.

Nutzungsumfeld

Die eigentliche Software ist vorerst in Form von „Open Source“ - Code entwickelt, jedoch so strukturiert, dass freie Codeanteile auf einfache Weise durch kommerzielle ersetzt werden können. Die Software soll nicht vertrieben und nur im eigenen Betrieb genutzt werden. Gewonnene Erkenntnisse stehen jedoch der Allgemeinheit, z.B. durch diese Ausfertigung, zur Verfügung.

Kapitel 2

Analyse

Schwerpunkt des Kapitels:

Dieses Kapitel beschäftigt sich mit einer groben Untersuchung zu den Datensystemen der Fundamentalstation Wettzell. Es baut auf Erfahrungswerte auf, welche aufgrund der existierenden Verfahren zusammengetragen wurden. Da positive wie negative Erkenntnisse dazu auf der Fundamentalstation als weitgehend erkannt angesehen werden können, soll hier nur ein grober Abriss dargelegt werden. Zum einen werden die existierenden Abläufe und ihre Datenflüsse kurz aufgezeigt. Zum anderen werden die darin enthaltenen Knoten mit der jeweils eingesetzten Datenspeicherung sowie die verwendeten Übertragungsmechanismen anschließend als weitere Punkte einer Betrachtung unterzogen. Aus den aufgezeigten Zusammenhängen und den daraus resultierenden Problemen, können Schlussfolgerungen gezogen werden, welche Grundlage für die Lösungen in den weiteren Kapiteln sind.

2.1 Begriffsdefinitionen

Abstecken des begrifflichen Umfelds innerhalb der Disziplinen

Vor Beginn aller Arbeiten ist es notwendig, das übergeordnete, begriffliche Umfeld abzustecken. Zahlreiche Wissenschaften nutzen dieselben Begriffe, um verschiedenartige Aussagen zu umschreiben, worin sich bereits im interdisziplinären Umfeld die Grundproblematik widerspiegelt, welche auch die Informationsverarbeitung maßgeblich tangiert: wann werden aus schlichten Daten brauchbare Informationen. Nachfolgend sollen deshalb die Bedeutungen von Begriffen definiert werden, auf welche die Arbeit generell aufbaut[1].

Begriffsdefinition DEF2.1 „Daten" nach [LEX04][2]:
Daten (aus der Übersetzung „Gegebenes") sind in der elektronischen Datenverarbeitung maschinenlesbare und -bearbeitbare Repräsentationen von Informationen, welche zu diesem Zweck in Bitfolgen codiert sind und strengen Regeln, der Syntax, folgen.

Daten sind allgemeine, maschinenlesbare Bitsequenzen

Daten sind damit nach Definition maschinenlesbare Bitsequenzen. Sie können von elektronischen Verarbeitungseinheiten über Abarbeitungsvorschriften (Computerprogramme) grundsätzlich genutzt und über elektronische Netzwerke ausgetauscht werden. Dies bedeutet noch nicht, dass ihr informeller Inhalt verwertet werden kann. Daten alleine sind wertlos, solange keine Möglichkeit existiert, die umgesetzten Informationen zu entnehmen. Hauptaugenmerk bei elektronischen Daten liegt somit auf der Anwendung von strukturierenden Regeln, der sog. Syntax.

Begriffsdefinition DEF2.2 „Syntax" nach [LEX04][3]:
Die Syntax (aus der Übersetzung „Satzlehre") beschreibt Muster und Regeln, mit denen einzelne Wörter zu größeren funktionellen Einheiten und Abhängigkeiten verknüpft werden können.

Die Syntax dient als Bindeglied zwischen den Codeelementen

Die Syntax ist also das Bindeglied einzelner codierter Elemente untereinander, so dass sie von elektronischen Systemen nutzbar werden. Sie alle folgen mehr oder weniger strengen Regelmäßigkeiten und Abhängigkeitsbeziehungen zueinander, welche durch die formale Grammatik definiert werden.

Begriffsdefinition DEF2.3 „Formale Grammatik" nach [LEX04][4]:
Im mathematischen Sinne besteht eine formale Grammatik aus einer endlichen Anzahl von Terminalsymbolen, einer endlichen Anzahl an Nichtterminalsymbolen, einer endlichen Anzahl von Produktionsregeln (Ersetzungsvorschriften) und einem Startsymbol. Die Anwendung einer Regel (sog. Ableitung) auf ein Wort aus Terminal- und Nichtterminalsymbolen ersetzt dabei die linke Seite der Regel durch die Rechte. Die Anwendung aller Produktionsregeln ausgehend vom Startsymbol erzeugt hierbei alle Wörter aus Terminalsymbolen und ihre funktionellen Einheiten (Sätze) einer sog. formalen Sprache als eine Art Syntaxbaum.

Daten können strukturiert, semistrukturiert oder unstrukturiert vorliegen

Da sich diese Arbeit nicht mit Sprachentheorie beschäftigt, sollen diese Definitionen zu funktionellen Zusammenhängen ausreichen. Es soll nur noch erwähnt werden, dass sich die Komplexität einer Sprache nach Chomsky von komplex nach

[1] Grundlage der Erklärungen sind hauptsächlich Beiträge aus [LEX04]
[2] vgl. [LEX04] unter . . . /Daten.html (21.06.2004)
[3] vgl. [LEX04] unter . . . /Syntax.html (21.06.2004)
[4] vgl. [LEX04] unter . . . /FormaleGrammatik.html (21.06.2004)

einfach in die vier Kategorien frei, kontextsensitiv, kontextfrei und regulär einteilen lässt[5]. Im übertragenen Sinn bedeutet dies, dass Daten strukturiert (strenge Anordnung gemäß kontextfreien und linearen Grammatiken), semi-strukturiert (unterschiedlich strenge Anordnungen, welche gemäß Kontextsensitivität vom Umfeld abhängig sind) und unstrukturiert (keine erkennbare Struktur im Sinne von freien Grammatiken) auftreten können. Das bedeutet jedoch nicht, dass weniger strukturierte Daten bedeutungslos wären. In einem anderen Zusammenhang (Kontext, z.B. unter Anwendung der Grammatik für die menschliche Sprache) können sie durchaus nutzbar werden. Nur der Rechner kann ohne die entsprechende Grammatik keine logischen Strukturen erkennen. Somit können als Beispiele für stark strukturierte Daten Datenbankanfragen genannt werden. Wohingegen unter den Begriff der unstrukturierten Daten die allgemeinen Schriftdokumente fallen.

Information setzt ein Verständnis von Bedeutungen voraus

Aus der beschriebenen Nutzbarkeit von elektronischen Dokumenten erwächst aber nicht gleichzeitig der Nutzen in Form von Information, welcher erheblich auf inhaltlich verwertbaren Elementen beruht. Deshalb kommt hier der Begriff der Semantik ins Spiel.

Begriffsdefinition DEF2.4 „Semantik“ nach [LEX04][6] ***und*** [7]***:***
Die Semantik (Bedeutungslehre) befasst sich mit der Bedeutung einzelner Zeichen und über eine Syntax aufgebaute Zeichenfolgen (Sprachenbedeutungen). Die formale Semantik regelt dies über vollständig mathematische Methoden.

Information ist unterschiedlich interpretierbar, jedoch immer auf Wissen basierend

Erst wenn die Bedeutungen einzelner in einer Zeichenfolge beinhalteter Elemente zugänglich sind, können sie der Informationsgewinnung zugeführt werden. Da die Definition von Information immer noch Diskussionsgrundlage in den kognitiven Wissenschaften, wie z.B. der Philosophie, ist, soll nachfolgend eine hier ausreichend erklärende Zusammenfassung dafür gegeben werden.

Begriffsdefinition DEF2.5 „Information“ nach [LEX04][8] ***und*** [9]***:***
Für die Information (in der Übersetzung „durch Unterweisung Gestalt geben“) aus Sicht der Informationswissenschaft ist die Bedeutung von Wissen zentral, welches aus einem Wissensvorrat (z.B. eine Datenbank) in einen anderen (z.B. Gehirn eines Menschen) übertragen wird. Wesentlich sind hierbei der Wert an Wiedererkennbarkeit sowie der Gehalt an Neuigkeiten eines bestimmten Musters in einem bestimmten Kontext. Durch Kombination verschiedenartigen Wissens kann somit neuartige Information entstehen.

Repräsentation und Präsentation eines Wissensvorrats

Während bei Shannon der Informationsbegriff in der Nachrichtentechnik jeglicher Bedeutung entzogen ist und nur auf die Unterscheidung von Zuständen aufsetzt, soll im Umfeld der Arbeit Information als Wissensrepräsentation auf einem Datenträger angesehen werden, welche zum Zwecke weiterer Informationsgewinnung (Erlangung neuer Kenntnisse) erstellt und zwischen informationsverarbeitenden Einheiten ausgetauscht wird. Diese Komponenten eines Verarbeitungsprozesses müssen hierbei dieselbe formale Sprache sprechen. Das Wissen selbst wird daher intern repräsentiert und vorgehalten. Die resultierende Information hingegen

[5] vgl. dazu [SCHOE97] a.a.O. S. 17
[6] vgl. [LEX04] unter .../Semantik.html (21.06.2004)
[7] vgl. [LEX04] unter .../FormaleSemantik.html (21.06.2004)
[8] vgl. [LEX04] unter .../Information.html (21.06.2004)
[9] vgl. [LEX04] unter .../Information-Informationswissenschaft.html (21.06.2004)

muss (für den Menschen) verständlich präsentiert werden. Zusammengefasst setzt sich also Informationsverarbeitung aus den ineinander verschachtelten Bereichen Syntax, Semantik und Pragmatik (Neuigkeitengehalt in der Information oder Update an Wissen) zusammen.

Auswahl eines geeigneten Kommunikationsmodells

Eine weitere Grundlage für den Informationsaustausch ist das angewandte Kommunikationsmodell. In der Datenverarbeitung werden dazu das Sender-/Empfänger-Modell (die Information fließt von einer Informationsquelle zu einer Senke) und das „Beobachtungs"-Modell (ein Empfänger beobachtet Zustandsveränderungen z.B. zur Informationsgewinnung über den aktuellen Verlauf eines Verarbeitungsprozesses) eingesetzt.

Das Kommunikationssystem ist zentrales Bindeglied für jegliche Informationsverarbeitung

Im übertragenen Sinn auf diese Arbeit bedeutet dies konkret: Ein funktionierendes Informationssystem hat dafür zu sorgen, dass der Informationsaustausch störungsfrei zwischen den Kommunikationspartner von statten geht. Ferner muss dafür gesorgt sein, dass Kommunikationspartner (Maschine - Maschine sowie Maschine - Mensch) eine ihnen verständliche Syntax nutzen, welche eine für den Vorgang verständliche Sprache bildet. Diese sollte allgemein gültig sein und ist mit semantischer Bedeutung so zu füllen, dass der Informationsgehalt für die Weiterverarbeitung generell nutzbar wird.

Steigerung der Verständlichkeit durch Metadaten

Um dies über Systemgrenzen hinweg zu gewährleisten und den Systemen eine Möglichkeit der übergreifenden Strukturierung zu geben, werden die eigentlichen Daten durch sog. Metadaten erweitert, welche eine Interpretation und Bewertung des Informationsgehalts und der Bedeutung für den Wissensaustausch erleichtern.

Begriffsdefinition DEF2.6 „Metadaten" nach [LEX04]*[10]*:
Metadaten sind Zusatzinformationen über andere Daten, welche z.B. in einem Katalog zusammengefasst sind und weiterführende Informationen zu den Daten liefern, um eine bessere Klassifizierung dieser zu gewährleisten.

Metadaten unterstützen die Repräsentation und die Präsentation

Über Metadaten lassen sich damit nicht nur übergeordnete Strukturen in Form einer überlagerten Syntax (vgl. `Extensible Markup Language (XML)`) in ein Dokument aufnehmen, es lassen sich auch Erscheinungsformen und Kennzeichnungen einzelner Elemente bewirken, welche zu einer verbesserten Informationspräsentation führen (vgl. z.B. Texthervorhebungen in dem weitverbreiteten Textsatzsystem LaTeX).

Qualitätssicherung in Informationssystemen muss bereits beim Kommunikationssystem beginnen

Trotz aller Strukturierung lassen sich daraus wieder heterogene, komplexe Formate entwickeln. Um nun ein informationsverarbeitendes System, wie es die Fundamentalstation Wettzell als solches darstellt, zu redesignen reicht eine alleinige Betrachtung nur der Informationen und ihrer Darstellungen an sich nicht aus, auch wenn Parser (siehe Abschnitt 5.2.1 auf Seite 133) bereits in der Lage sind, Validierungen an den Daten vorzunehmen. Vielmehr müssen auch Zusammenhänge und Verständigungspunkte analysiert werden, um Fehlerursachen möglichst am Punkt ihrer Entstehung zu erkennen. Wie kann z.B. dafür gesorgt werden, dass bruchstückhafte Daten in der Interpretation nicht zu falschen Informationen werden, was eine Anforderung an die Qualitätssicherung darstellt.

[10] vgl. [LEX04] unter . . . /Metadaten.html (21.06.2004)

Alle genannten Anteile zusammengefasst bilden damit ein sog. Informationssystem, welches die Daten rechnergestützt verwaltet.

Begriffsdefinition DEF2.7 „Informationssystem“ nach [LEX04]*[11]*:
Ein Informationssystem besteht aus Hardware und Software um Daten rechnergestützt erfassen, speichern, verarbeiten, pflegen, analysieren, benutzen, verbreiten, dispositionieren, übertragen und anzeigen zu können.

Hauptaufgabe der Analyse ist es nun, informationsverarbeitende Einheiten des Informationssystems für die Fundamentalstation Wettzell zu identifizieren und ihre internen Abläufe zu charakterisieren. Dabei ist auch zu betrachten, wie sie ihre Informationen präsentieren und welche Kommunikationsmechanismen sie dafür einsetzen. Eine weitere Betrachtung stellt sich der internen Wissensrepräsentation.

2.2 Informationsverarbeitende Einheiten und ihre Informationspräsentation

Heterogene Datendienstleistungen mit festen, externen Vorgaben

Die Fundamentalstation Wettzell bietet aufgrund ihrer unterschiedlichen Messsysteme verschiedene Datendienstleistungen an. Sie alle sind eingebunden in internationale, geodätische Dienste, wie z.B. dem „International VLBI Service“ oder dem „International GPS Service“. Jeder dieser Dienste nutzt für seine Datenzentren eigene Datendarstellungen, welche als interne Standards spezifiziert sind und von den datenliefernden Stationen eingehalten werden müssen. Zudem ist auch festgesetzt, welche Zugangspunkte für den Datenaustausch zur Verfügung stehen. Diese von außen gegebenen Vorgaben können somit nicht beeinflusst werden.

Messsysteme konsumieren und produzieren Daten

Jedes Messsystem ist individuell und nutzt Daten, welche generell in zwei Gruppen aufgeteilt werden können: Daten, welche für die Messung benötigt werden und solche, die durch die Messung entstehen. Nachfolgend werden die Abläufe anhand der Hauptsysteme kurz umrissen.

Die Abläufe beim Radioteleskop Wettzell zur Gewinnung von VLBI-Messdaten

Beim `Radioteleskop Wettzell (RTW)` für `VLBI` ergibt sich damit folgender, grober Zusammenhang (vgl. Abb. 2.1 auf Seite 24). Aktiv wird über die Koordinierungsdatenbasis `Crustal Dynamics Data Information System (CDDIS)` der für das Experiment gültige Ablaufplan („Schedule“) über das `File Transfer Protocol (FTP)` geholt. Kurzfristige Änderungen erreichen das System via Email. Zusammen mit lokal vorgehaltenen Daten (z.B. Standortinformationen) und Informationen von lokalen Zusatzdiensten (z.B. Meteorologie, Zeit und Frequenz) fließen sie in den Beobachtungsprozess ein. Dieser läuft vollautomatisch ab. Die daraus gewonnenen Ergebnis sind zum einen die Messdaten (binär), welche im Umfang solche Ausmaße annehmen, dass sie direkt mit dem Aufzeichnungsmedium (Magnetband bei Mark IV, Festplatte bei Mark V) über einen Kurierdienst verschickt werden. Zum anderen werden Log-Dateien (textbasiert) zu den Messungen erzeugt, welche erneut per `FTP` versandt werden. In den neuen Versuchen zur `Electronic Very Long Baseline Interferometry (eVLBI)` sollen auch kurzzeitige Experimente (sog. Intensives) mittels `FTP` übertragen werden. Alleine aufgrund der enormen Datenflut ist dies jedoch

[11] vgl. [LEX04] unter . . . /Informationssystem.html (21.11.2004)

nicht ohne zusätzliche Maßnahmen zu gewährleisten (z.B. schnelle Standleitungen und Punkt-zu-Punkt-Routing bis zum Empfänger). Ziel all dieser Datenflüsse sind Korrelatoren, welche die Auswertung übernehmen und aus den aufgezeichneten Signalen brauchbare Informationen (z.B. Erdrotationsparameter) für die Geodäsie erzeugen. Der Versand geschieht je nach Experiment manuell zu unterschiedlichen Auswertezentren. Diese so gewonnene Information fließen wieder in das CDDIS der National Aeronautics and Space Adminstration (NASA) ein.

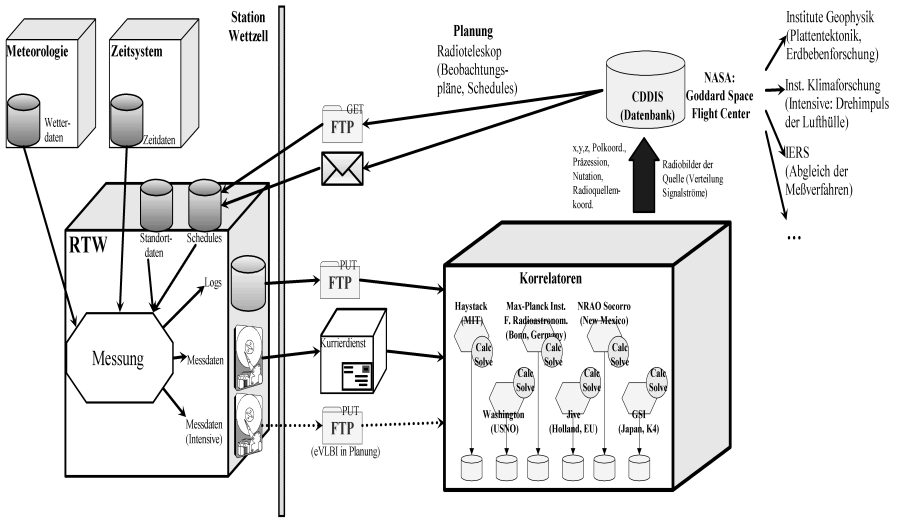


Abbildung 2.1: Schematischer Datenfluss beim Radioteleskop Wettzell (RTW)

Die Abläufe beim Wettzell Laser Ranging System zur Gewinnung von SLR-Messdaten

Da beim Wettzell Laser Ranging System (WLRS) die Datenmengen durch Vorauswertung wesentlich kleiner sind, können in diesem Umfeld generell elektronische Wege zum Austausch genutzt werden. Der Ablauf ist dabei ähnlich zum RTW (vgl. Abb. 2.2 auf Seite 25). Von einem Datendienst (z.B. CDDIS oder EUROLAS Data Center (EDC)) werden die für eine Messung benötigten Informationen per FTP geholt und teilweise kurzfristig über Emails mit Teildatenbeständen aktualisiert. Zusammen mit den lokal errechneten Predictions (Bahnvorhersagen), den vorgehaltenen Parametern und den Informationen der lokalen Zusatzdienste fließen sie in den Messprozess ein, der über einfache Sockets seine Zustände (EUROLAS-Status) an andere Beobachtungsstationen meldet und deren Statusdaten über den selben Weg erhält. Die Beobachtung erzeugt in einem internen XML -Format eine Observationsdatei, welche alle Treffer-, Kalibrations- und Refraktionsinformationen aus der Messung enthält. Sie wird in einer Auswertung statistisch untersucht, was als Ergebnis die Bahnsegmente in Form von charakteristischen Normalpunkten (Textdateien) erbringt. Diese Ergebnisdateien werden anschließend zum EDC versandt. Die nicht versandten, lokalen Informationen (z.B. die für die Normalpunkteberechnung benötigten Kalibrationen) werden in eigene Datenbanken gespeichert und dienen der Systemkontrolle. Zusätzlich werden die XML - Beobachtungsdateien archiviert. Alles in allem ergibt sich ein stark manuell geprägter Ablauf.

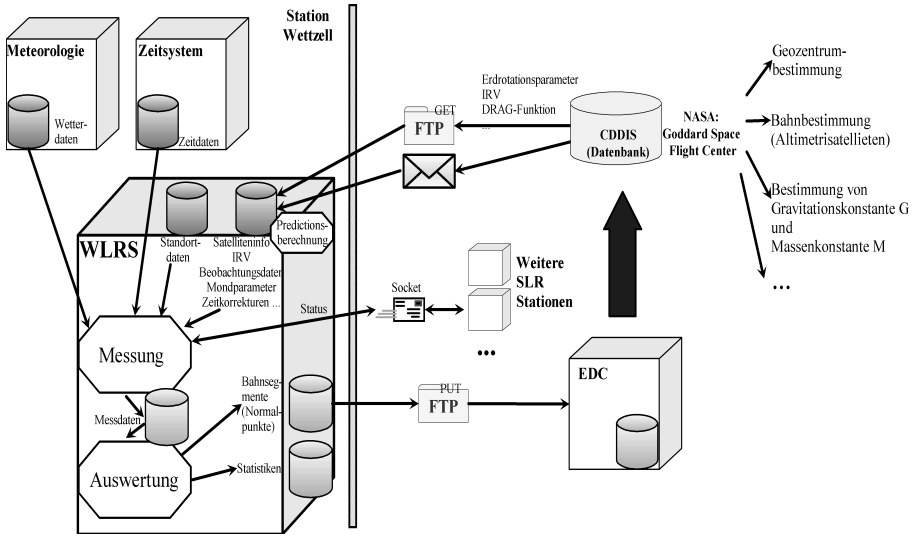


Abbildung 2.2: Schematischer Datenfluss beim Wettzell Laser Ranging System (WLRS)

Die Abläufe bei der klassischen Gewinnung von GPS-Messdaten durch permanente GPS-Stationen

Im GPS -Verbund existiert dagegen ein komplett anderer Ablauf. Die Fundamentalstation Wettzell betreibt über 50 permanente, auf der ganzen Welt verteilte GPS -Stationen. Sie bestehen im klassischen Netz (kein Echtzeitversand der Daten) aus den Empfängern mit Antenne und einem Messrechner. Die empfängerspezifischen Bitströme werden dabei im Rechner zu standardisierten Dateien (Receiver Independent Exchange Format (RINEX)) im Stunden- und Tagestakt umgewandelt. Die komprimierten Dateien können dann anschließend an die Fundamentalstation stunden- und tagesweise verschickt werden, wo sie gesammelt und nach Qualitätskriterien überprüft werden. Zusätzlich kann man auch Datensätze von vor Ort installierten Wetterstationen mitübertragen. Der Versand erfolgt über eine Vielzahl unterschiedlicher Wege (vom Satellitenlink bis hin zu Modem-Modem-Verbindungen) meist über FTP (abgesehen von wenigen, älteren Stationen, die mit Mailbox-Abfragen arbeiten). Die gesammelten RINEX -Dateien werden anschließend an das GPS /GLONASS Data Center weitergeleitet. Dort werden die Informationen für den German Reference Frame (GREF) , European Reference Frame (EUREF) und International GPS Service for Geodynamics (IGS) vorgehalten.

Neuere Entwicklungen zur Echtzeitgewinnung von GPS-Messdaten

In den neueren Entwicklungen gehen Bestrebungen dahin, den Mess-Personal Computer (PC) vor Ort einzusparen und schnelle Datenleitungen direkt zur Übertragung der empfängerspezifischen Binärdaten zu nutzen (Echtzeitnetz). Diese werden dann zentral gesammelt und in RINEX umgewandelt. Sinn und Zweck ist die Realisierung eines Echtzeitnetzes für GPS -Korrekturparameter.

Die Abläufe bei Zusatzdiensten am Beispiel der Gewinnung meteorologischer Daten

Die weiteren Zusatzdienste erzeugen lokal die jeweiligen Daten. Ein Beispiel hierfür sind die Wetterstationen. Sie liefern zeitgetaktet die jeweiligen meteorologischen Informationen (z.B. Temperatur, Luftdruck, Feuchte, etc.). Ein „Wetterprozess“ im Hauptrechnersystem schreibt diese Werte in eine meteorologische Datenbank (Socket mit Textanfragen), wo die Informationen über Common Gateway Interface (CGI) -Skripts nutzerfreundlich über das HTTP präsentiert werden

können. Abfragen über einen längeren Zeitraum werden in Stapelverarbeitung abgehandelt und die Ergebnisse, welche nach angegebenen Kriterien vorausgewertet und sortiert wurden, werden als Email-Versand an den Auftraggeber geleitet.

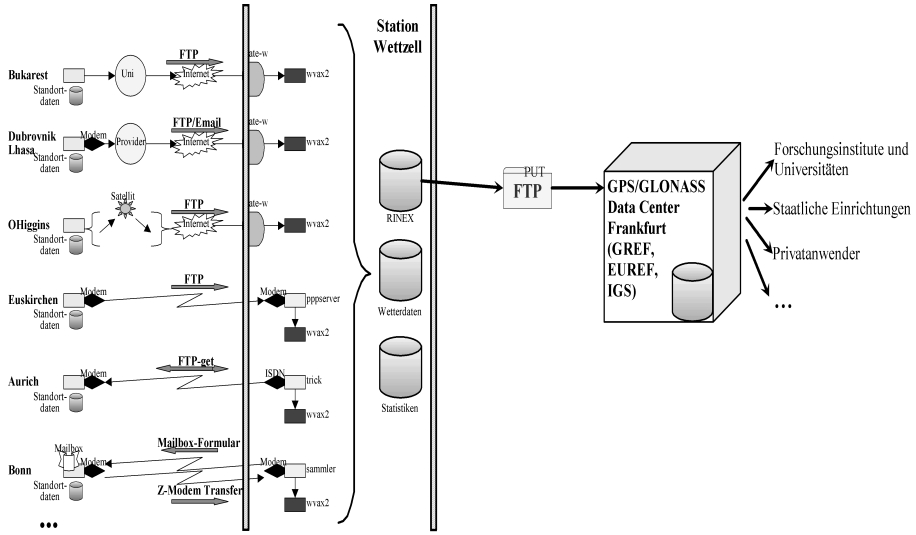


Abbildung 2.3: Schematischer Datenfluss im klassischen GPS-Netz (mit Stunden-/ Tagesdateien)

Zusatzdienste und Bürokommunikation basieren auf linearen Verarbeitungsstrecken

Aus solchen ähnlichen, linearen Verarbeitungsstrecken bestehen die weiteren Zusatzdienste. Allerdings nutzen sie zur Übertragung verschiedenartige, interne Verfahren (Samba, `Network File System (NFS)` , eigene Sockets). Diese werden auch für die herkömmliche Bürokommunikation eingesetzt, welche hauptsächlich aus Dokumenten besteht.

Zusammenfassung der Gegebenheiten

Zusammenfassend lässt sich damit feststellen, dass vorgegebene äußere Bedingungen vorherrschen, welche Datenformate und Zugangsmechanismen festschreiben. Für die externen Datenzentren sind dies beim Uplink (das Aufspielen von Daten) zumeist `FTP` -Server und Email-Dienste. Gerade aber `FTP` ist im Bezug auf Firewalltechnologie ein Kandidat für Unsicherheitsfaktoren (breite Portstreuung etc.). Für den Downlink von ausgewerteten Informationen (das Abholen von Daten) existieren zusätzlich `HTTP` -Server. Die Datenzentren sind entweder hierarchisch gegliedert oder gleichwertig auf einer Ebene. In allen Fällen befindet sich eine datenerzeugende Station als „Blattknoten“ am Rand des Verarbeitungsbaums. Bei `GPS` setzt sich auch innerhalb der Stationsverantwortung ein hierarchischer Ablauf fort.

Eingliederung neuer Entwicklungen in die internen Eigenheiten der Fundamentalstation

Alle Aktivitäten dieser Arbeit beschränken sich damit vorerst auf das stationsverantwortete Geschehen. Die Fundamentalstation ist ein reiner Datenlieferant und kümmert sich damit also hauptsächlich um die Gewinnung neuer Daten und der Repräsentation, wobei zwangsläufig auch neues Wissen entsteht. Die Präsentation der Informationen hingegen ist entweder in starre Schranken gelegt (äußere Vorgaben) oder nur ein Nebenprodukt (Wetterdatenabfragen etc.). Primärziel muss es daher sein, die internen Zusammenhänge auf ein stabiles Fundament zu stellen und dabei momentan existierende Probleme zu beheben.

Problem:
Interne Probleme ergeben sich in erster Linie aus dem Übertragungssystem

Interne Probleme ergeben sich dabei dadurch, dass die auf eigenen Sockets basierenden Übertragungen nicht weiter überwacht werden, wodurch es zu störenden Fehlinformationen kommen kann. Bei kompletten Ausfällen von Teilkomponenten kann sich ein Schneeballeffekt ergeben, welcher nur mehr durch manuellen Eingriff korrigierbar wird. Daten werden nicht übertragen oder bleiben nach einem Verbindungsabbruch als Fragmente übrig. Dieses Problem existiert gerade auch mit FTP .

Problem:
Zustandslose Verfahren und fehlende Überwachungsmechanismen

Die Vielzahl intern genutzter Protokolle und Verfahren sind zudem nicht kombinierbar. Da aber übergeordnete Informationsstrukturen auf die Vollständigkeit der Informationen angewiesen sind, müssen als erste Maßnahme diese Dienstgüteeffekte verbessert werden. Da alle Verfahren auf dem Transmission Control Protocol (TCP)/Internet Protocol (IP)-Standard aufsetzen, ist eine sichere Übertragung gewährleistet. Jedoch sorgen die Verfahren nicht dafür, dass blockweise übertragene Datenmengen auf Vollständigkeit überwacht werden. Solche zumindest fehlererkennenden und bestenfalls durch zustandsbasierte Transaktionen bestimmten Verfahren sind aber gerade für hierarchische Produktionssystem von großer Bedeutung. In diesen ist ein folgender Arbeitsschritt darauf angewiesen, dass sein vorhergehender korrekt abgewickelt wurde.

Problem:
Fehlende, allgemeingültige, semantische Klassifizierbarkeit der Daten

Des Weiteren muss dafür gesorgt werden, dass über allgemeine Verfahren Daten klassifizierbar und zuordenbar werden. Ohne das Wissen über semantische Zusammenhänge (was bedeutet z.B. ein Wert mit der Bezeichnung „Epoche“) und die eingesetzte syntaktische Kombinationen in den Formaten (welches Format ist wie aufgebaut) sind die Daten unbrauchbar. Dies wird insbesondere deutlich, wenn Softwareupdates zu neuen internen Formaten führen und die bisherigen Strukturen nicht mehr verwendbar sind. Somit sind verallgemeinernde Beschreibungen vorzusehen, mit denen man extern festgelegte Formate beschreiben und umwandeln kann.

2.3 Die interne Wissensrepräsentation

Zentrale und teilweise dezentrale Datei- und Datenbanksysteme

Die Wissensrepräsentation selbst ist eine Mischung aus stark zentralistisch (Meteorologie, GPS) und teilweise dezentral (WLRS , RTW) geprägten Datenspeichern. Dies ist zum Teil auch aus einer Verkehrsanalyse des Netzwerks ersichtlich. Dort werden einige wenige Hauptrechner auf denen sich die Datenbanken oder die Dateisysteme für die Büroanwendungen oder die Messsysteme befinden stark frequentiert. Damit sind auch schon die zwei in der Station eingesetzten Repräsentationsarten genannt: Dateien und Datenbanken.

Sequentielle Messdaten bilden sich ideal auf Dateistrukturen ab

Aus der vorhergegangenen Analyse ist ersichtlich, dass der Großteil der Daten in Form von Dateien verwaltet wird. Diese wohl einfachste und älteste Möglichkeit hat besonders großen Wert bei sequentiell angeordneten Informationen. Sie liegen im Allgemeinen bei zeitbezogenen, auf Epochen basierenden Messreihen aus der Geodäsie vor. Das Auslesen erfolgt deshalb zumeist als ganzer Datenblock.

Problem:
Individuelle und damit heterogene Datenstrukturen je Messsystem

Die Daten selbst sind nach eigenen, programmspezifischen und messsystembezogenen Gesichtspunkten strukturiert und entweder binär oder als Text aufgebaut. Binäre Daten sind hierbei äußerst anfällig für Veränderungen der Software oder der Hardware. Handelt es sich um endgültige Messdaten, welche auch an die Datenzentren verschickt werden, so sind deren Formatstandards eingehalten. Jede Ver-

arbeitungseinheit hat jedoch seine eigenen Ablagepunkte (Speicherstellen), -arten (Aufbau der Dateisysteme) und -strukturen (interne Formate).

Dateisysteme versus Datenbanksysteme

In der Fundamentalstation Wettzell wurden auch Versuche unternommen, solche sequentielle Daten in Datenbanken unterzubringen (z.B. Wetterdatenbank und `GPS` -Datenbank, sowie Kalibrationsdatenbank des `WLRS`). Generell stellt dies kein Problem dar und hat bzgl. indexorientierter Suchzugriffen sogar Vorteile. Jedoch verliert die Datenbank mit ihren indexorientierten Zugriffen an Bedeutung, wenn einzelne Werte alleine mehr oder weniger wertlos sind und nur in einer kontinuierlichen Abfolge wirklich Informationen beinhalten. Dies ist bei Zeitreihen der Fall. Ein weiterer Nachteil ist der extreme, zusätzliche Arbeitsaufwand zur Verwaltung von Datenbanken und deren tabellaren Strukturen. Ein spezialisierter Administrator muss diese durchführen, welcher die Eigenschaften der Datenbank kennt. Das eingesetzte, freie `Database Management System (DBMS)` PostgreSQL bietet zudem nicht so komfortable Managementstrukturen, wie sie bei kommerziellen Produkten üblich sind. Mit Dateisystemen hingegen kann jeder Nutzer ohne viele Vorkenntnisse arbeiten. Er kann dazu auch allgemein übliche Zugriffstechniken (z.B. Netzwerkdateisysteme) nutzen.

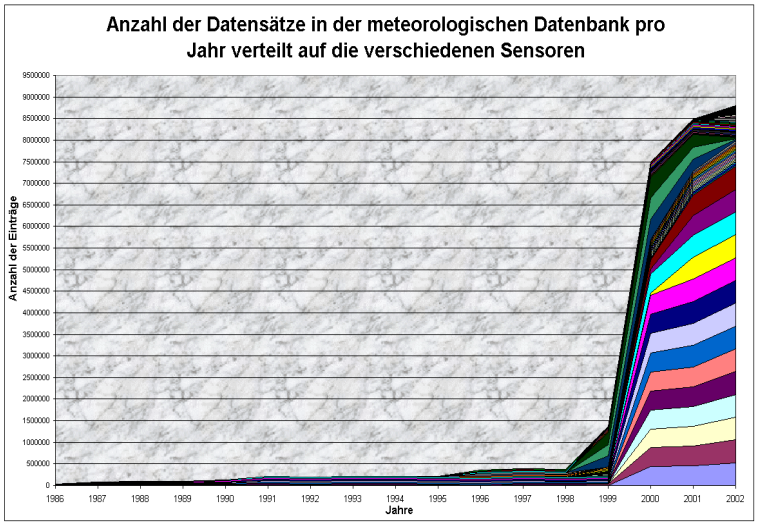


Abbildung 2.4: Zunahme des Datenaufkommens im zentralen DBMS für meteorologische Daten

Problem:
Zu wenig DBMS-optimierte Zugriffs- und Weiterverarbeitungsmechanismen

Die Daten in der Datenbank selbst sind nach bekannten Datenbankstrukturen einzelnen Schlüsselbegriffen in Tabellenform zugeordnet und meist epochenbezogen angeordnet. Spezielle Werte dienen hierbei als Bezugspunkte (Relationen) für andere Tabellen (Entitäten), so dass mengentheoretische Zusammenordnungen möglich sind. Aufgrund der Forcierung von Datenbanken in den letzten Jahren nahm auch das Datenaufkommen darin zu (vgl. Abb. 2.4 auf Seite 28). Jedoch sind die Zugriffsmechanismen nicht dafür ausgelegt und so ergeben sich trotz der relativ wenigen Zugriffe über die Zeit hinweg zahlreiche Probleme. Als Beispiel dazu kann die Wetterdatenbank herangezogen werden. Darauf finden relativ selten Abfragen statt (Caching bringt damit nur bedingt Vorteile), welche zudem als sequentielle Zugriffe ablaufen (Indexzugriffe über Schlüssel bleiben weitestgehend ungenutzt). Die entnommenen Daten müssen zudem weiterbearbeitet werden, so

dass sie erst dann als echte Informationen vorliegen. Auch Vorteile aus schnellen Sortiervorgängen bleiben weitestgehend bei epochenbezogenen Daten ungenutzt, da diese dann meist auch in zeitlicher Ordnung verarbeitet werden müssen.

Problem:
Für Datenbankzugriffe sind zusätzliche Kenntnisse erforderlich

Ein weiteres Problem ist der gehobene Administrationsaufwand gegenüber einem Dateisystem. Für Datenbankzugriffe muss der Nutzer spezielle Kenntnisse (z.B. Mengentheoretische Beschreibungen in `Structured Query Language (SQL)`) haben. Auf ein Dateisystem kann ohne großes Zusatzwissen zugegriffen werden. Durch den Mangel dieser Kenntnisse beim Standardnutzer ist wohl auch die fehlende Optimierung der Zugriffe erklärbar, so dass Fehlerzustände und zahlreiche Ein- und Auswahlvorgänge auftreten und trotz der relativ geringen Aktivitäten nicht mehr wirklich überwachbar und bewertbar sind. Das zeigte sich durch das in der Analyse erstellte Perl-Logbuch für die zentrale Meteorologiedatenbank (vgl. dazu Anhang A auf Seite 205).

Die Datenbank ist momentan ein überqualifiziertes Datenarchiv

Zusammengefasst zeigt sich so, dass sich der Umfang der Daten in der Datenbank aufbläht, während nur wenige nutzende Zugriffe stattfinden, welche die Vorteile dieser Datenspeicherung nicht wirklich ausnutzen. Damit wird die Datenbank zu einem überqualifizierten Datenarchiv degeneriert.

Hybridsysteme und standardisierte Metadaten

Sinnvollere Hybridsystem, welche sich aus einer Mischung von verallgemeinerten Tabellenstrukturen in Datenbanken für Zusatzinformationen (Metadaten) und den heterogenen Nutzdaten in Dateisystemen zusammensetzen, sind noch nicht im Einsatz. Ursache ist wohl der fehlende Satz an Metadaten und die Unterschiedlichkeit der Anforderungen. Generell sind nämlich sowohl hochgradig strukturierte Messdaten als auch unstrukturierte Daten in Schriftdokumentenform in verschiedene Verarbeitungsprozesse eingebunden. Abhilfe wird dabei momentan im internationalen Verbund durch Standardisierungen für Metadaten (z.B. vom Open GIS Consortium durch die Spezifikation von OpenGIS(tm) Metadaten (ISO/TC 211 DIS 19115) für `Geographic Information Systems (GIS)` [12]) geschaffen. Diese werden sich wohl in den nächsten Jahren auch etablieren.

Problem:
Redundante Daten

Ein weiteres Problem der Wissensrepräsentation ist die Existenz von Replikaten. Redundant existieren unterschiedliche Versionen derselben Daten. Besonders betroffen sind hier Dokumente aus der Bürokommunikation und Programmcodes.

Komplikationen durch die Problemfelder entstehen erst bei der Informationspräsentation

Alles in allem entstehen aus den aufgezählten Charakteristika keine Probleme, solange das jeweils gegebene Umfeld der Messsysteme nicht verlassen wird. Sobald aber mehrere verarbeitende Einheiten zugreifen können und somit Informationen präsentiert werden müssen, kommt es zu unüberschaubaren Zusammenhängen. Diese äußern sich z.B. dadurch, dass ein in einem System veränderter Datumsbezeichner zahlreiche Problemstellen in anderen Systemen bewirken kann, weil die Zusatzinformation bzgl. der Bedeutung nicht vorliegt.

[12] vgl. dazu [OGIS04]

2.4 Schlußfolgerungen

Im Forschungs- und Entwicklungsprogramm 2001-2005[13] wird in der Planung einer Projektstudie zur Datenhaltung ein erster großangelegter Vorstoß zur Planung zeitgemäßer Datensysteme angeregt. Allerdings wird dort der Hauptschwerpunkt der Problematik in der Strukturierung, Formatierung und Ablage der Daten gesehen. Die Untersuchungen in den letzten Abschnitten zeigen aber, dass alleinige Untersuchungen von Datenmengen und Datenströmen zur Generierung von Zusatzinformationen (sog. Meta-Informationen) nur einen Teil des gewünschten Erfolgs erbringen können. Diese erweiterte Informationsbasis erbringt spezielle Vorteile beim Finden und einfache Handhaben von Informationseinheiten.

Tiefergehender Ansatz bis in den Bereich des Datentransports

Der gegebene Einblick in die Abläufe und die dafür genutzten Komponenten in der Fundamentalstation Wettzell zeigt jedoch schnell auf, dass ein Lösungsansatz tiefergehend durchgeführt werden muss, der schon Problematiken beim automatischen Datenabgleich und den heterogenen Komponenten zu lösen vermag. Natürlich kann an den von außen gegebenen Grundsätzen, welche aus Dienstleistungsverbunden resultiert, nichts verändert werden. Aber es lohnt eine Betrachtung der stationsinternen Abläufe, was mit einer Kapselung aller externen Kommunikationspartner durch eine gegebene, abstrakte Schnittstelle in Einklang gebracht werden kann.

Zwei Arten der Datengewinnung: komplett lokal oder verteilt sammelnd

Bei den internen Abläufen kann man zwischen zwei Arten der Datengewinnung unterscheiden:

1. die Daten werden komplett durch ein lokal auf der Station angesiedeltes Messsystem gewonnen
2. die Daten werden von weltweit verteilten Messsystemen aufgezeichnet und in der Fundamentalstation Wettzell zusammengeführt und gesammelt

Kommunikationsbeziehungen bei Abläufen unter 1. verlassen im Laufe einer Messung außer in der Vorbereitungsphase zur eigentlichen Datengewinnung das lokale Stationsnetz nicht (gegeben bei den örtlichen Großsystemen `VLBI` und `SLR`). Demgegenüber werden unter 2. zwar die Daten von unabhängigen Permanentstationen aufgezeichnet, für das Anbieten des Datendienstes ist aber ein zentrales Sammeln und Verarbeiten in der Fundamentalstation notwendig, was einer Verbindung über das Internet bedarf (gegeben bei `GPS` -Permanentstationen).

Problem:
Arbeitszeitintensive Betreuung und Weiterentwicklung der Systeme

Die generelle Erkenntnis, dass verschiedene Übertragungsmechanismen und Datenformate genutzt werden, welche nicht ohne Schwierigkeiten für automatische Abläufe geeignet sind, erfordern zusätzlichen Aufwand an Kontrolle. Die meist ungenügenden Teillösungen werden dabei um aufwändige Überwachungskomponenten erweitert, denen eine manuelle Kontrolle überlagert ist, durch welche Fehlzustände korrigiert werden können. Diese Fehlersituationen entstehen aus den undefinierten Zwischenzuständen einer teilweise erfolgten Übertragung. Aber gerade die Vielschichtigkeit und Heterogenität der Systeme führt zu beliebig vielen Möglichkeiten von undefinierten Zuständen, was in letzter Konsequenz ein eigentlich „unproduktives“ Ausmaß an Arbeitszeit bindet. Diese unüberschaubaren

[13] vgl. [FGS01] a.a.O. S. 59 f.

Zustandskopplungen können auch bei Weiterentwicklungen zu schwer behebbaren Fehlerfällen und massivem Arbeitsaufwand führen.

Problem: Disharmonie von Zugriffsmechanismen untereinander

Ein weiterer Teilaspekt ergibt sich aus der Disharmonie der technischen Realisierungen selbst. Unterschiedliche Netzdateisysteme, Datenbanken und ähnliche Speichereinheiten können z.B. nicht miteinander in Einklang gebracht werden. Unterschiedliche Zugriffsmechanismen bei Datenbanken (mengenorientiert) und Dateisystemen (sequenziell blockweise) sowie in verschiedenen Dateiverwaltungen eigene Zugriffskontrollen, welche untereinander nicht kompatibel sind, sind erkannte Ursachen davon. Es kommt damit über kurz oder lang entweder zu nicht kontrollierbaren, kritischen Abschnitt (mehrere Dateisysteme greifen gleichzeitig auf den selben Pfad zu), oder es sind umständliche Lösungen zur Umgehung dieser Probleme (sog. „Work Around"-Lösungen) zu entwerfen, sollen Daten über unterschiedliche Zugriffsverfahren bearbeitet werden.

Problem: Heterogenität im allgemeinen

Des Weiteren setzt sich die Heterogenität auch bei den Rechensystemen fort. Unterschiedliche Rechnertypen mit unterschiedlichen Betriebssysteme (z.B. über 8 unterschiedliche alleine auf der Fundamentalstation Wettzell, von MacOS über diverse Linux-Derivate bis hin zu unterschiedlichen Windows-Systemen) erfordern es, dass entsprechend unterstützte Zugriffsmechanismen eingesetzt werden müssen. Dies ist zum Teil auch eine Ursache für die vielschichtigen, heterogenen Zugriffsverfahren. Letztendlich geht dies sogar soweit, daß sich die Zahlendarstellungen in Binärformaten unterscheiden können.

Transparenzkriterien zwischen den Systemen und zum Nutzer

Allgemein betrachtet ist somit festzustellen, dass in den aktuell genutzten Anwendungen die Kriterien zur Transparenz zwischen den Systemen und zum Nutzer nur ungenügend eingehalten werden. Diese sind vor allem[14]:

1. **Transparenz des Zugriffs:**
   Bedeutung: Jeder Zugriff auf Funktionalität und Daten ist von der selben Art und Weise, egal ob er auf entfernten oder lokalen Ressourcen aufbaut.
   Ausgangssituation Wettzell: Der Anwendungsprozess muss genau wissen, wie er auf die einzelnen Systemteile zugreifen kann. Der Zugang zu einer Datenbank (meist einfache Sockets) unterscheidet sich von dem zu einem Dateisystem (oft Netzwerkdateisysteme). Zudem wird zwischen lokalen (Standardzugriffe über Betriebssystem) und entfernten Quellen (Netzwerkzugriffe) unterschieden.
   Resultierende Schwierigkeiten: Heterogenität bei den Zugriffen bewirkt, dass die Eigenheiten der Zugangsschnittstelle bis in die Anwendung sichbar werden (u.a. Funktionsumfang und -art). Kleine Änderungen führen zu Folgeanpassungen in weiten Teilen anderer Anwendungen.

2. **Transparenz der Technologie:**
   Bedeutung: Die genutzte technologische Umsetzung bleibt der Anwendung verborgen.
   Ausgangssituation Wettzell: Jede Anwendung ist individuell bzgl. des Designs und der Umsetzung in einer Programmiersprache. Ihr zugrunde liegen unterschiedliche Betriebssysteme und Hardwarekomponenten mit ihnen eigenen Charakteristika.

[14] vgl. dazu [SING94] a.a.O. S. 10

Resultierende Schwierigkeiten: Die Eigenheiten sowohl von Hard- als auch von Software spiegeln sich in der Anwendung wider. Alleine die Berücksichtigung von \ (Windows) und / (Unix) in den Pfadangaben für den Dateizugriff zeigt die Vielschichtigkeit dieser Problematik. Des Weiteren steht Funktionalität (z.B. Zugriff nach Indizes) nicht allgemein zur Verfügung, da sich die technologische Umsetzung unterscheidet.

3. **Transparenz bei konkurrierendem Verhalten:**
   Bedeutung: Die Anwendung agiert mit Ressourcen ohne Berücksichtigung von Konkurrenzanwendungen.
   Ausgangssituation Wettzell: Konkurrierende Zugriffe verschiedener Dateisysteme untereinander sind nicht möglich. Jede Anwendung ist in ihrer Betrachtungsweise abgeschlossen, ohne andere Anwendungen zu berücksichtigen. Bei Bedarf herrscht eine strenge Serialisierung durch Warteschlangenverfahren.
   Resultierende Schwierigkeiten: Es existieren nicht oder nur schwer kombinierbare Systeme, welche prinzipiell die selben Aufgaben erfüllen.

4. **Transparenz bei Ausfall:**
   Bedeutung: Der Ausfall von einer definierten Anzahl von Systemanteilen ist für die Anwendung nicht sichtbar.
   Ausgangssituation Wettzell: Die zentralisierten Komponenten werden durch Redundanzsysteme mehrfach ausgelegt.
   Resultierende Schwierigkeiten: Ausfälle zentraler Teilkomponenten sind in der gegebenen Realisierung aufgrund der Redundanz nicht unmittelbar sichtbar (sondern nur durch manuelle Kontrolle). Meist existiert eine nicht mehrfach ausgelegte Teilkomponente als Schwachstelle, wie z.B. das Kommunikationsmedium (Teilpfad des Netzwerks, mit dem die Einheiten über ihre Netzwerkkarte verbunden sind). Nach einem kompletten Stillstand einer Anwendung ist ein „weicher“ Wiederanlauf (Recovery) mit Herstellung des Zustands vor einem Absturz meist nicht vorgesehen, was oft zu Datenverlust führt.

5. **Transparenz des Ortes:**
   Bedeutung: Das Wissen um den Ablageort von Funktionalität und Daten wird verdeckt.
   Ausgangssituation Wettzell: Die einzelnen Programme und Daten sind an fest definierten Orten abgelegt. Diese werden direkt angesprochen.
   Resultierende Schwierigkeiten: Die Eigenheiten des Ablageorts spiegeln sich in der Anwendung wider. Da der Ort der Ablage absolut angegeben ist, führen Verlagerungen von Daten zu Änderungen in den Anwendungen.

6. **Transparenz bzgl. der Daten:**
   Bedeutung: Die Repräsentation von Daten ist allgemein gültig verständlich.
   Ausgangssituation Wettzell: Jede Anwendung hat ihren individuellen Datenumfang in ihrer eigenen Art und Weise dargestellt.
   Resultierende Schwierigkeiten: Jede Anwendung benötigt zum Lesen ihrer Daten Importiermechanismen, welche sich individuell unterscheiden.

Beispiel 2.1 für „die Anwendung von Transparenzkriterien“
Angenommen eine Bibliothek umfasse zahlreiche Bücher an verschiedenen Standorten mit unterschiedlichen Ablagesystemen und Magazinen. Die Verbesserung der Transparenz in diesem Zusammenhang würde in letzter Instanz bedeuten, dass ein Bibliothekar beschäftigt wird, der eine den Bücherentleihern verständliche Sprache spricht, alle Anfragen bearbeitet und die Bücher verwaltet. Durch ihn wird das in der Bibliothek gelagerte Wissen transparent nutzbar. Die Ausleihaktionen sind intuitiv begreifbar. Es gibt einen einheitlichen Zugriff auf die Systeme. Die technologische Umsetzung und der Ort der Ablage bleiben für den Leihvorgang unwesentlich. Konkurrierende Ausleiher eines Buches werden sequentiell bedient. Um ein gewisses Maß an Ausfallsicherheit zu erzielen, ist darüber hinaus die Beschäftigung mehrerer Bibliothekare ratsam. Schließlich sollten die Bücher auch noch einen einheitlichen Aufbau aufweisen (Leserichtung, Abfolge der Abschnitte, etc.) und in den gängigen Landessprachen vorliegen, um das Wissen transparent zu repräsentieren.

Erweiterung der Transparenzkriterien

Alle weiterführenden Kriterien sind zumeist Spezialisierungen der genannten. Trotzdem seien hier noch zwei erwähnt:

1. **Transparenz bzgl. einer logischen Ordnung:**
   Bedeutung: Die Aktivitätsschritte einer Verarbeitungskette und ihre jeweiligen Ausgangs- und Resultatdaten unterliegen einer logischen Ordnung im Sinne einer Anordnungsreihung. Einzelne Aktionen bedingen vorweg andere und/oder bilden Ausgangspunkte für weitere. Die jeweiligen Daten können damit entsprechend ihrer Entstehungs- oder Nutzungsreihenfolge angeordnet werden. Diese Ordnung ist von der Anwendung transparent nachvollziehbar.
   Ausgangssituation Wettzell: Verarbeitungsketten werden entweder manuell oder mit Hilfe von eigenen Skripten aufgebaut und aufrechterhalten. Die Ordnung der Daten bzgl. ihrer Entstehung ist an die Uhrzeit gekoppelt oder wird mittels eigener, logisch geordneter Sequenznummern aufgebaut.
   Resultierende Schwierigkeiten: Abläufe können schnell zu komplexen, unstrukturierten Gebilden aus Einzelaktionen werden. Die Anordnung anhand von Zeitkriterien ist schwierig, da globale Uhren in der Regel nicht vorliegen, auch wenn in einem begrenzten Umfeld bestimmte Zeitkriterien eingehalten werden können. Sequenznummern bilden zwar eine logische Anordnung bzgl. eines Verarbeitungspfades, lassen jedoch keine Aussagen über echte Gleichzeitigkeit bzgl. paralleler Verarbeitungsstrecken zu. Zur Verhinderung eines übermäßigen Aufwands und aufgrund der Außenbeziehungen werden sich diese Mechanismen nicht ganz vermeiden lassen.

2. **Transparenz bzgl. Innovationen:**
   Bedeutung: Innovationstransparenz sorgt dafür, dass ein Nutzer bei einem Innovationssprung eines Systems auf den vertrauten Umgang und die bewährten Abläufe nicht verzichten muss, ihm aber trotzdem die innovativen Neuerungen zur Verfügung stehen. Sie ist in der Regel ein Spezialfall der Technologietransparenz, da sie aus der Unvereinbarkeit von unterschiedlichen, technologischen Lösungen herrührt, was komplette Systemwechsel erfordert.

Ausgangssituation Wettzell: Jede Neuerung führt meist zu kompletten Systemwechseln. Die Unvereinbarkeit der Systeme ist dabei die Ursache zahlreicher paralleler Anwendungen (z.B. verschiedene, verteilte Dateisysteme) mit unterschiedlichen Nutzerinteraktionsmöglichkeiten .
Resultierende Schwierigkeiten: Eine nicht zu unterschätzende Problematik besteht in psychologischen Effekten bei den Anwendern. Eine generelle Innovationsscheu bzgl. Neuentwicklungen ist die häufigste Ausprägung, da zusätzliches Wissen zur Bedienung neuer Komponenten erworben werden muss. Dieser psychologische Effekt wird durch anfängliche Schwierigkeiten aufgrund von Softwarefehlern und Konfigurationsproblemen verstärkt.

Die in dieser Arbeit entworfenen Lösungsansatze versuchen schrittweise diese Transparenzkriterien in die täglichen Abläufe speziell auf der Fundamentalstation Wettzell zurück zu bringen. Generell kann aber festgestellt werden, dass ähnlich gelagerte Probleme im Allgemeinen in der datenverarbeitenden Informationstechnologie existieren.

Die Säulen der Abstraktion des Datenmanagements

Die Einhaltung der Kriterien zur Transparenz fordern im Grunde eine generelle Abstraktion des Datenmanagements, welche auf drei Säulen aufbaut (vgl. dazu Abb. 2.5 auf Seite 34). Eine verallgemeinerte Technikschnittstelle (I.) dient zum Zugriff auf Daten in einer abstrahierten Form (II.) durch beliebige Anwendungen mit ihnen eigenen, kombinierten und allgemein zusammengefassten Abläufen in einer gemeinsamen Strukturierung (III.). Somit ergeben sich drei große Aufgabenfelder mit nachfolgenden Lösungsschritten entsprechend der Säulen der Abstraktion:

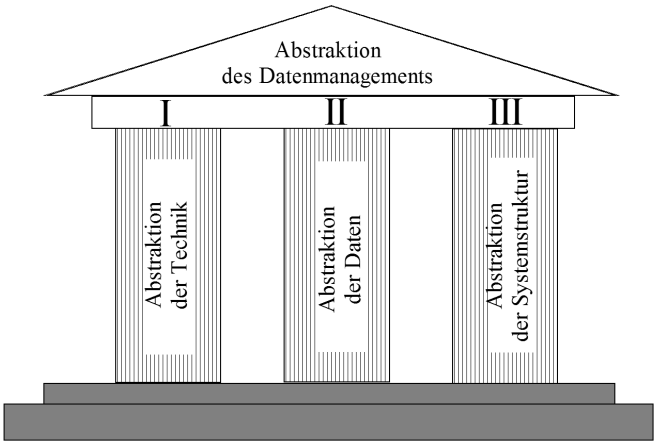


Abbildung 2.5: Drei Säulen der Abstraktion beim Datenmanagement

1. Verbesserung des Datenzugangs (I.)
   (a) Schaffen einer flexiblen, portablen, von der Hardware unabhängigen und einfachen Kommunikationsbasis
   (b) Verbesserung der Dienstqualität hin zu einem in sich stabilen System

2. Verbesserung der Datenrepräsentation (II.)
   (a) Verbesserung der Datendarstellung unter Einbeziehung aktueller Strukturen
3. Verbesserung der Informationspräsentation und Informationsverarbeitung (III.)
   (a) Schaffung eines kombinierten Datenzugangs
   (b) Strukturelle Verbesserung von Verbünden und Verarbeitungsabläufen

Kapitel 3

Middleware als elementare Kommunikationsbasis

Schwerpunkt des Kapitels:

Nachfolgend werden grundlegende Elemente eines neuen, auf Standardmiddleware basierenden Übertragungsmechanismus diskutiert. Ausgangspunkt sind die Analyse der aktuellen Datentransportmechanismen bzw. -komponenten und die daraus resultierenden Probleme im vorausgegangenen Abschnitt. Aufbauend auf der Überlegung zur Abstraktion von Heterogenität und Schnittstellen werden unterschiedliche Zwischenschichtkonzepte untersucht. Die als am geeignetsten erachtete `Common Object Request Broker Architecture (CORBA)` dient für die weiteren Entwicklungen als Basis. Erkenntnisse über die unterschiedlichen Implementierungen bzgl. der `CORBA` -Services und die freie Verfügbarkeit der Quellen führen zur Nutzung der `CORBA` -Implementierung `The ACE Object Request Broker (TAO)` der `Adaptive Communication Environment (ACE)` , auf dem die definierte, abstrakte Dateitransferschnittstelle aufsetzt. Die dazu entwickelte Client-Serverentwicklung `CORBA File Transfer (CFT)` bildet die Grundlage für alle weiteren, komplexeren Anwendungen. Der Client simuliert den kommandozeilenbasierten Dateizugang, wie er bei der Nutzung des `FTP` üblich ist.

3.1 Ausgangsüberlegungen

Schaffung eines flexiblen, portablen und einfachen Kommunikationssystem

Im Sinne der in Abschnitt 1.4.1 auf Seite 8 aufgezeigten Vorgehensweise anhand eines Entwicklungsprozesses wird in einem ersten Iterationsschritt die Implementierung eines flexiblen, portablen und grundlegenden Kommunikationssystems geschaffen. Es soll eine Abstraktion der Technik entsprechend der ersten Säule zur Abstraktion in Abschnitt 2.4 auf Seite 34 schaffen. Aus der Analyse der Gegebenheiten im vorhergehenden Abschnitt wird deutlich, dass es so zu gestalten ist, dass es betriebsystemunabhängig funktioniert und sich in bestehende Strukturen einfügt. Grundidee ist das Einführen einer Abstraktionsschicht zwischen Anwendung und zugrundeliegender Betriebssystemfunktionalität. Dazu setzt das Kommunikationssystem auf einer Zwischenschicht (Middleware) auf, um von Grund auf bestehende Problematiken (z.B. durch Systemheterogenitäten) zu vermeiden. Der so entstehende, elementare Prototyp wird in nachfolgenden Phasen weiter für den Produktionsbetrieb ausgebaut und erweitert.

Einhaltung von Transparenzkriterien

Die realisierte Abstraktionsschicht verbessert vorwiegend die Zugriffs- und Technologietransparenz, wobei Teilaspekte der Transparenz bzgl. konkurrentem Verhalten (z.B. unteilbare Einzelaktionen) einfließen. Dabei sollen die Kriterien bzgl. der Innovationstransparenz Beachtung finden (vgl. dazu die Transparenzkriterien in Abschnitt 2.4 auf Seite 31). Es stellt sich somit die Frage, durch welche Maßnahmen eine Vereinheitlichung des Datentransports nahezu ohne Einschränkungen möglich wird. Gerade in den internen Datengewinnungsabläufen steckt hierzu ein großes Potential an Möglichkeiten, da sie eigenverantwortlich betrieben und verändert werden können. Während sich spätere Kapitel mit Teilaspekten der Daten an sich beschäftigen, soll zunächst tiefergehend schon bei den Übertragungsmechanismen angesetzt werden, da gerade auch komplexe Systeme nur so wirkungsvoll sind, wie ihr schwächstes Glied.

Bedeutung für die FS Wettzell: Vereinheitlichung, Eigenverantwortung über den Programmcode und Effizienzgewinn

Bedeutung für die Geodäsie: effektivere Ablaufstrukturen

Bedeutung für die Informatik: allgemeine Vereinheitlichung

Lösungen in diesem Bereich haben für die Fundamentalstation Wettzell neben Vorteilen durch die Vereinheitlichung von Abläufen über die Messsystemgrenzen hinweg vor allem auch Vorzüge im Bereich der Eigenverantwortung über den Code und die Systeme selbst. Dies kann zu kostensparenderen Strukturen führen. Für den Bereich der Geodäsie sind nachfolgende Überlegungen dahingehend interessant, da Strukturen, wie sie in der Fundamentalstation vorliegen, auch in anderen internationalen Messstationen vorzufinden sind bzw. sogar auf die hierarchischen Abläufe in den globalen Diensten projiziert werden können. Nutzbare Verbesserungen könnten hier zu effektiveren Ablaufstrukturen führen. Die Überlegungen können aber auch auf die Informationsverarbeitung im allgemeinen übertragen werden. Dort gibt es seit langem von einigen Seiten die Bestrebungen, Vereinheitlichungen u.a. im Bereich der Abstrahierung von Datenzugangsschnittstellen zu schaffen und dabei Systemcharakteristika zu verbergen.

3.2 Theoretische Grundlagen

Die zu realisierende Abstraktionsschicht dient vor allem dem sicheren Datentransfer zwischen Client und Server. Sie basiert auf einem Zwischenschichtkonzept, welches als Middleware bezeichnet wird. Sie verdeckt die Heterogenität betriebssystemnaher Kommunikationsanteile vor dem Zugriff aus der Anwendung.

3.2.1 Die Idee hinter einer Middleware

Grundlegenden Kommunikationsschnittstelle zur Abstraktion

Erster Schritt auf dem Weg zu einem neuen Übertragungsverfahren ist die Entwicklung einer grundlegenden Kommunikationsschnittstelle, welche von den rechner- und kommunikationstypischen Charakteristika abstrahiert. Gerade in heterogenen Umgebungen, so wie sie fast in allen Netzen und auch auf der Fundamentalstation Wettzell zu finden sind, kann eine einheitliche Schnittstelle zum Kommunikationsmedium erheblich den Programmieraufwand von Applikationen verringern und zu einem hohen Maß an Portierbarkeit führen. Deshalb entstanden die ersten Ideen zu solchen Verallgemeinerungen im Zusammenhang mit verteilten Systemen.

Begriffsdefinition DEF3.1 „Verteiltes System“ nach [SING94][1] und [PUD01][2]:
Ein verteiltes System besteht aus einer Anzahl von Rechnern mit jeweils eigenem Betriebssystem, welche zum gemeinsamen Erreichen eines angestrebten Ziels gekoppelt werden, dabei keinen gemeinsamen Speicher und keine gemeinsame Uhr nutzen und durch Nachrichtenaustausch über ein Kommunikationsnetzwerk untereinander kommunizieren

Allgemeine, vereinfachte und übersichtliche Zugangsschnittstelle zu allg. Netzen

Der Grund des Zusammenschlusses in verteilten Systemen ist stets die Erfüllung einer gemeinsamen Aufgabe. In allgemeinen, heterogenen Netzen können Rechner nicht zwangsläufig einem höheren Ziel zugeordnet werden. Trotzdem wird auch hier von verallgemeinerten und damit vereinfachten, übersichtlichen Zugangsschnittstellen profitiert. Es gibt deshalb verschiedene Ansätze zur Abstrahierung, welche allgemein unter dem Begriff „Middleware“ zusammengefasst werden können.

Begriffsdefinition DEF3.2 „Middleware“ nach [PUD01][3]:
Middleware kann als Verteilungsplattform beschrieben werden, welche homogenisierende, generische Dienstleistungen zwischen Anwendung und Netzwerk definiert und realisiert.

Grundelemente für mehr Transparenz

Dabei handelt es sich um eine grundlegende Definition, welche vor allem die Kommunikationsschnittstelle betrachtet. In nachfolgenden Kapiteln wird diese Definition noch erweitert und dadurch verallgemeinert. Vorerst reicht diese grundlegende Abstrahierung bzgl. Kommunikation, so dass Middleware für eine individuelle Anwendung auf einem individuellen Rechensystem einen einheitlichen Netzwerkzugang liefert. Dabei werden von dieser Zwischenschicht die zur Erfüllung der Kommunikation benötigten Betriebssystemzugangspunkte verdeckt. Des Weiteren werden im Rahmen von verteilten Systemen meist auch Zusatzdienste realisiert, welche das Management von verteilten Anwendungen vereinfachen und verfeinern. Grundelemente hierbei sind das Auffinden von gewünschten Partnerknoten, die Konvertierung von Datentypen (sog. Marshalling), sowie die Überwachung der Kommunikation und der verteilten Abläufe. Eine Middleware bietet damit bereits Lösungen für mehr Transparenz im Sinne der Transparenzkriterien bzgl. Zugriff und Technologie (siehe dazu Abschnitt 2.4 auf Seite 31).

[1] vgl. [SING94] a.a.O. S. 71
[2] vgl. [PUD01] a.a.O. S. 7
[3] vgl. [PUD01] a.a.O. S. 2

Aufrufsemantik der Middleware

Grundlage dafür ist unter anderem ein geeignetes Fehlermodell in Form einer Aufrufsemantik . In den gängigen Middleware-Lösungen sind hier zwei Möglichkeiten[4] gegeben, welche in Anlehnung an die Semantik in den zugrundeliegenden Fehlermodellen (siehe weiter unten) umgesetzt sind:

- **„Gößtmögliche Bemühungen“ („Maybe“):** Aufruf erfolgt ohne Erwartung einer Antwort; das zugrundeliegende System bemüht sich nach Möglichkeit um Ausführung, garantiert sie aber nicht.
- **Höchstens einmal („At most once“):** Korrekter Lauf erzeugt einen Durchlauf, wobei ein Fehlerfall keinen oder einen Durchlauf bedeuten kann, d.h. jeder korrekte Lauf ist nur einmal ausgeführt worden

Synchrone, auftragsorientierte Kommunikation

Die Kommunikation in den gängigen Middlewaresystemen ist üblicherweise auftragsorientiert. Das bedeutet, dass der Auftraggeber mittels einer Nachricht einen Auftrag an einen Auftragnehmer erteilt und daraufhin eine Antwortnachricht erwartet. Erst danach gilt die Kommunikation als erfolgreich beendet. Synchrone (Auftraggeber wartet blockiert auf Antwort) und asynchrone (Auftraggeber ist nicht blockiert und kann nach Bedarf die Antwort entgegennehmen) Kommunikationsmodelle sind möglich. Die meisten Middleware-Systeme nutzen jedoch die synchrone, auftragsorientierte Methode der Remote Procedure Calls, welche intuitiv in den normalen Arbeitsfluss integriert werden kann[5].

Begriffsdefinition DEF3.3 „Remote Procedure Call (RPC)“ in Anlehnung an [SING94][6]:
Remote Procedure Calls sind in Anlehnung an Prozeduraufrufe in einem Programm prozedurale Kontroll- und Datenflüsse über ein Kommunikationsnetzwerk, die als Interaktion zwischen einem anfordernden Klienten und einem Serviceanbieter stattfinden.

Asynchrone, auftragsorientierte Kommunikation

Eine asynchrone, auftragsorientierte Kommunikation wird teilweise als Zusatz angeboten, da dadurch die natürliche Parallelität des Auftragerstellens, Versendens, Bearbeitens und Beantwortens automatisch genutzt wird. Diese Möglichkeit wird u. a. als `Remote Service Invocation (RSI)` [7] oder auch als Asynchronous Methode Invocation[8] bezeichnet.

Andeutung des Client-Server-Modells

Die Abbildung eines `Remote Procedure Call (RPC)` auf ein Netzwerk[9] wird in einer Anwendung über eine Stellvertreterprozedur (Client Stub) realisiert, welche transparent die Kommunikation aktiviert und einen Vertreter (Server-Stub) auf der entfernten Maschine mit dem eigentlichen Programmcode kontaktiert. Dieser bearbeitet die Aufgabe und gibt über denselben Weg seine Ergebnisse zurück. So deutet sich hier bereits das später in diesem Abschnitt beschriebene Client-Server-Modell an.

Abstrahierung der elementaren Datendarstellung für den transparenten Datentransfer

Konvertierungen von Prozedurparametern werden transparent mittels einer standardisierten Datenrepräsentation `External Data Representation (XDR)` abgewickelt[10]. Die Transparenz endet aber schon bei Call-By-Referenz-Parametern

[4] vgl. [PUD01] a.a.O. S. 39 und [PUD01] a.a.O. S. 15 ff.
[5] vgl. [PUD01] a. a. O. S. 12 f.
[6] vgl. [SING94] a.a.O. S. 88
[7] vgl. [PUD01] a.a.O.S.12
[8] vgl. [ORYAN02]
[9] vgl. zum nachfolgenden Abschnitt [STEV92] a.a.O. S. 723 ff.
[10] vgl. [WEB304]

(= Zeiger), welche über Rechnergrenzen nur als Datenkopie handhabbar sind. Einige Middlewarelösungen versuchen deshalb den entfernten Speicherzugriff zu gewährleisten (siehe die Beschreibung von „Message Passing Interface“ im Anhang B auf Seite 209). Über die Bindungsinformation kann ein Auftraggeber seinen zugehörigen Auftragnehmer finden. Zudem wird dadurch zwischen den beiden Vertretern eine Kompatibilitätsprüfung der Parameter und Prozedurstruktur ermöglicht. Als Transportprotokolle werden meist mehrere unterstützt (z.B. `TCP` , `User Datagram Protocol (UDP)`). Die Ausnahmebehandlung für entfernte Aufrufe ist über zusätzliche Signalkanäle gegeben.

Aufrufsemantiken beim RPC

Da durch den Netzwerktransfer mögliche Ausnahmesituationen und Fehlerquellen vermehrt auftreten, ist auch die Aufrufsemantik bei `RPC` von entscheidender Wichtigkeit. Sie regelt, wie oft eine Prozedur aktiviert werden darf (z.B. bei Nachrichtenverdopplung durch Wiederholung des Aufrufs nach einem Timeout), wovon die Art der Prozedur abgeleitet werden kann. Folgende Semantikregeln sind im Rahmen von `RPC` denkbar und spiegeln sich auch in die Middleware wider[11]:

- **Exakt einmal („Exactly once“):** Durchlauf genau einmal => schwer zu gewährleisten
- **Höchstens einmal („At most once“ oder „Zero-or-one“):** Korrekter Lauf erzeugt einen Durchlauf; Fehlerlauf bedeutet kein oder ein Durchlauf
- **Mindestens einmal („At least once“):** Durchlauf mindestens einmal => einfach zu gewährleisten; am besten für idempotente Prozeduren (= mehrere Ausführungen verändern das Ergebnis nicht weiter) geeignet

Beispiel 3.1 für „Semantik beim Methodenaufruf“
Eine beliebige, in einem Durchlauf abgeschlossene Berechnung kann mehrfach aktiviert werden und verlangt maximal nach „At least once“, während ein blockweise schreibender Dateizugriff genau einmal ausgeführt werden darf und dadurch „Exactly once“ erfordert.

Abbildung auf Kommunikationssockel

Die `RPC` -Kommunikation selbst wird letztendlich auf sog. Kommunikationssockel (= Sockets) abgebildet. Sie stellen die Betriebssystemschnittstelle zum internen Protokollstack für die Kommunikation.

Begriffsdefinition DEF3.4 „Socket“ nach [WEB404]*[12]*:
Sockets sind Softwareobjekte, welche eine Applikation mit einem Netzwerkprotokoll verbinden und dabei die Grundaufgaben zum Öffnen, Schließen, Anmelden und Verbinden des Sockets sowie die Handhabung einer Verbindungsanfrage und das Senden und Empfangen von Daten erfüllen.

Zusammenfassend kann somit von mehreren Abstrahierungsschichten ausgegangen werden, von denen die Middleware direkt unterhalb der eigentlichen Anwendung als allgemeinste und komfortabelste fungiert (vgl. Abb. 3.1 auf Seite 42).

[11] vgl. [STEV92] a.a.O. S. 727 f. und [SING94] a.a.O. S. 90
[12] vgl. [WEB404]

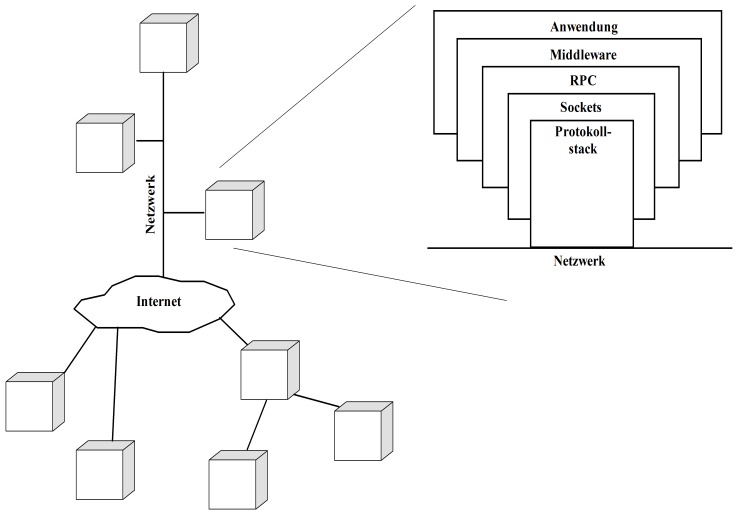


Abbildung 3.1: Abstraktionsschichten im Zusammenhang mit Middleware

Grundprinzip Client-/Server-Modell

Wie bereits in den Erläuterungen zum RPC erkannt, kristallisiert sich ein grundlegendes Modell für die Auftragsabwicklung heraus, welches als elementar für heutige Kommunikationssysteme angesehen werden kann: das Client-Server-Modell. Ein aktiver Auftraggeber (Dienstnehmer, Client) erteilt einen Auftrag an einen wartenden Auftragnehmer (Dienstanbieter, Server), welcher diesen ausführt und mit dem Ergebnis der Arbeiten antwortet.

Begriffsdefinition DEF3.5 „Client-/Server-Modell“ in Anlehnung an [SING94][13]***:***
In einem Client-/Server-Modell erteilt ein Dienstnehmer (Client) an einen Diensterbringer (Server) über Nachrichtenaustausch einen Auftrag, den dieser bearbeitet und mittels einer Antwortnachricht quittiert, wobei die Initiierung immer vom Client ausgeht.

Folgender in Abb. 3.2 auf Seite 43 graphisch dargestellter Ablauf eines entfernten Aufrufs kann in Anlehnung an den RPC [14] grob aufgezeigt werden:

1. Die Anwendung aktiviert mittels eines Methodenaufrufs (Methode-Call) die clientseitige Vertreterprozedur (Client-Stub), wodurch zum einen der RPC - Fluß und zum anderen die Middlewareaktivitäten (Konvertierungen, Finden des Kommunikationspartners etc.) und Überwachungsabläufe starten.

2. Der Client-Stub aktiviert letztendlich über einen Systemaufruf den Protokollstack des Betriebssystems, durch welches eine Nachricht über das Netzwerk versandt wird.

3. Diese Nachricht aktiviert wiederum auf der Serverseite den wartenden Vertreter (Server-Stub) mittels einer Systemaktivierung. Dort wird die Rückkonvertierung durchgeführt und durch die Laufzeitumgebung ein geeigneter Ablauf paralleler Clients angestoßen.

4. Ist der Vertreter eingeplant, aktiviert er die eigentliche Aufgabenlösung im Anwendungsteil des Servers als Methodenaufruf.

[13] vgl. [SING94] a.a.O. S. 80
[14] vgl. [STEV92] a.a.O. S. 723 f.

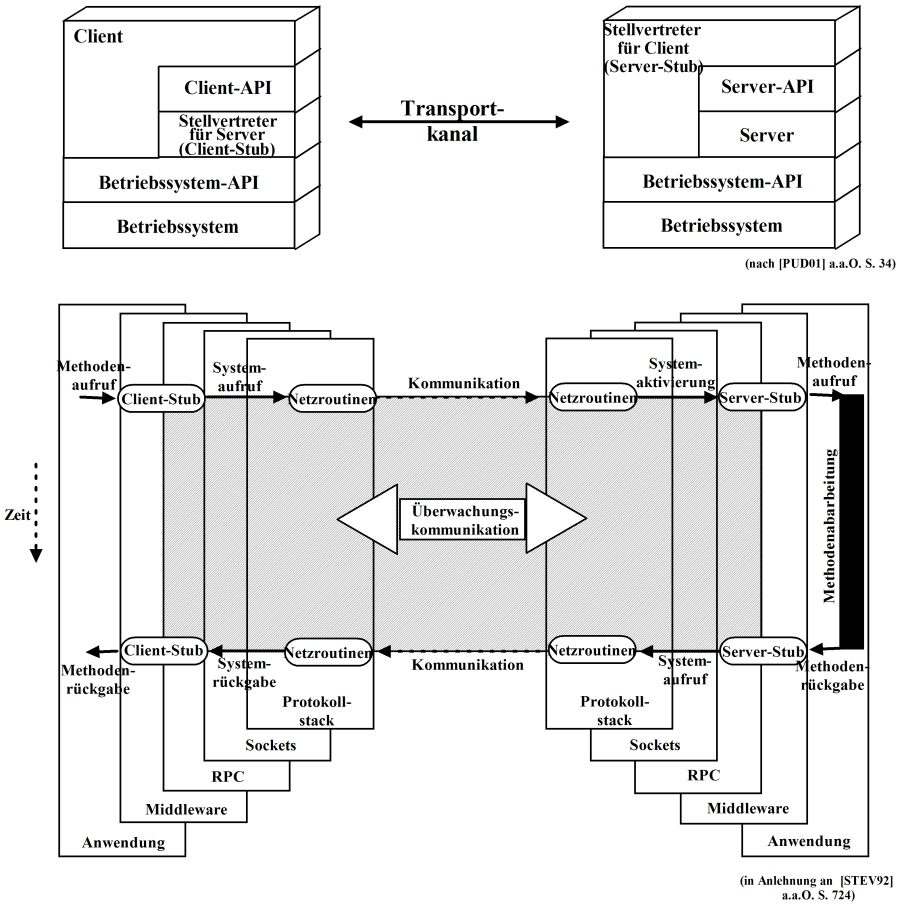


Abbildung 3.2: Schematisches Client-/Server-Modell mit RPC-Middleware

5. Nach Bearbeitung der Aufgabe erfolgt eine Rückgabe der Ergebniswerte an den Server-Stub, welcher nun wie zuvor der Client-Stub die Aufgaben der Middleware wahr nimmt.

6. Letztendlich wird anschließend durch einen Systemaufruf eine Antwortnachricht über das Netzwerk zurück an den Client-Stub geschickt, welcher seinerseits die Rückkonvertierung vornimmt und wie ein lokaler Prozedurenaufruf das Ergebnis an die Aufrufende Anwendung liefert.

Während des Ablaufs sorgt die Middleware dafür, dass verschiedene Fehlerfälle abgefangen werden und überwacht die Kommunikation bzw. die Ausführung der entfernten Funktionalität. Das alles ist aber für den Anwendungsprogrammierer nicht sichtbar, wodurch der übergeordnete Gesamtzusammenhang nicht durch unüberschaubare Feinheiten verdeckt und damit transparenter wird.

Middleware bedient grundlegende Transparenzkriterien

Aktuelle Middleware löst die Anforderungen bzgl. Client-Server-Anbindung aufgrund ihrer internen Strukturierung auf geeignete Weise. Dass diese Grundidee des Datentransports größerer Datenmengen auf Basis von RPC funktioniert, zeigt sich im Network File System. Es basiert grundsätzlich auf diesen Mechanismen und kann als eines der gängigen Datenzugangssysteme in der UNIX-Welt angesehen werden. Der Einsatz von Middleware bietet aber den enormen Vorteil an Transparenzgewinn. Durch ihre abstrahierende Wirkung wirkt sie sich insbe-

sonders positiv im Bezug auf Zugriffstransparenz aus. Auch Teile bzgl. der Transparenz der Technologie (Zugang zur Kommunikationsschicht wird vereinheitlicht und dabei Verwaltungsaufwand von der Anwendung verlagert) und des konkurrierendem Verhaltens (Prozeduraufrufe sind aus externer Sicht nicht unterbrechbar) werden bereits zufriedenstellend abgedeckt. Somit ist die Idee zum Einsatz einer gängigen Middleware als Kommunikationsschicht mehr als legitim, auch wenn hier evtl. zusätzliche Erweiterungen zum Erlangen einer ausgeweiteten Transparenz notwendig sind.

Es stellt sich damit die Frage, welche Middleware aktuell als „Stand der Kunst“ bezeichnet werden kann und welche für den Einsatz zum Datentransport für Datendienste im allgemeinen im `Local Area Network (LAN)` und `Wide Area Network (WAN)` gleichermaßen geeignet ist.

3.2.2 Die Wahl einer Middleware

Zahlreiche Middleware-Lösungen für verteilte Systeme

Middleware ist keine Entwicklung, welche erst vor kurzem begonnen hat. Seit einigen Jahren gibt es von verschiedenen Seiten die Bestrebungen, die Probleme in heterogenen Systemen zu lösen. Da die Entwicklung von Middleware hauptsächlich für verteilte Systeme betrieben wird, haben sich eine Vielzahl von Ideen und Lösungen im Laufe der Jahre entwickelt und teilweise durchgesetzt.

Eine Wahl kann deshalb nur nach entsprechender Analyse der verschiedenen Ansätze und ihrer Spezifika erfolgen, welche im Anhang B auf Seite 209 angefügt ist. Hier soll nun nur noch in einem Überblick eine zusammenfassende Bewertung und Auswahl durchgeführt werden.

Extrem spezialisierte oder ähnliche Lösungen

Die im Anhang B vorgestellten Middleware-Lösungen zeigen ganz deutlich, dass sie entweder extrem spezialisiert sind oder auf ähnlichen Konstruktionen basieren. Letzteres führt glücklicherweise dazu, dass sie entweder bereits über Schnittstellen zueinander verfügen oder leicht als Ganzes untereinander ausgetauscht werden können.

Lösungen für paralleles Rechnen hier ungeeignet: PVM, MPI, Agentensysteme

Zusammenfassend spalten sich damit die auf paralleles Rechnen (`Parallel Virtual Machine (PVM)` , `Message Passing Interface (MPI)` , Agentensysteme) ausgelegten Ansätze von den restlichen für allgemeine verteilte Anwendungen ab. Interessant sind aber die Prozesskontrolle mit Synchronisationsmechanismen bei `PVM` (evtl. für den Datenworkflow) und vor allem synchronisierte Dateioperationen bei `MPI` (für zustandbasierten Zugriff auf Dateien). `MPI` wird dabei u.a. auch für das „Grid-Computing“ (= „Gitter-Berechnung“) eingesetzt, bei dem die Rechenleistung vieler Computer eines Netzwerks zur Berechnung von rechenintensiven Problemen zusammengefasst wird[15]. Trotz der Tatsache, dass virtuelle Agenten auch auf schlechter Infrastruktur agieren können, sind sie für Datensysteme relativ wenig geeignet, da dann zentralistische Ansätze schon aus Gründen der Datensicherung besser sind. Wird jedoch der komplette Datenumfang mit den Agenten übertragen, so ist der Vorteil zunichte und im Gegenteil ein extremer Overhead an Verwaltung zu erfüllen.

[15] vgl. dazu [LEX04] unter . . . /Grid-Computing.html und [FRI04] a.a.O S. 62

Datenbankorientierte Ansätze ungeeignet: ODBC, JDBC

Des Weiteren kann auch von den rein datenbankorientierten Ansätzen (Open Database Connectivity (ODBC) und Java Database Connectivity (JDBC)) Abstand genommen werden. Dateien sind als handhabbare Einheiten allgemein verständlich und individuell flexibel für alle Arten von Daten einsetzbar. Wie in der Zielsetzung in Abschnitt 1.2 auf Seite 5 ausgeführt, ist gerade die Datei für die Aufgabenstellung notwendig, da hier zeitlich sortierte Daten als Einheitenblöcke zu verstehen sind. Nachteile aber sind fehlende indexorientierte Zugriffe, wie sie bei Datenbanken üblich sind, wodurch bei Dateien das Finden von Informationen nur sequentiell möglich ist.

Auch Hybridsysteme erfordern nicht zwingend datenbankorientierte Ansätze für das Kommunikationssystem

Dies kann letztendlich über Hybridsysteme gelöst werden, bei denen z.B. die eigentlichen Daten in herkömmlichen Dateisystemen abgelegt sind und alle Zusatzinformationen zur Verwaltung in eine angegliederte Datenbank ausgelagert werden. Dadurch werden die Vorteile der Datenbanken (einfache Suche über Metainformationen und schneller Zugriff auf indizierte Einzelinformationen) mit denen der Dateisysteme (sequentielle Ablage von zusammenhängenden Dokumentstrukturen) vereinigt. Dies kommt vor allem beim sog. Content Management durch ein Content Management System (CMS) zum Einsatz, das zur Präsentation unterschiedlich strukturierter Daten über Web-Schnittstellen eingesetzt wird. Mehr oder weniger statische Inhalte aus Dateien, wie z.B. Menüs oder Dokumente, werden durch das CMS mit dynamischen und flexiblen Inhalten und zusätzlichen Metadaten aus einem DBMS kombiniert[16]. Gerade für die Präsentation von Daten sind solche kombinierten Systeme ideal geeignet, was z.B. für die Umsetzung des Datendienstes im IERS auch in einer ähnlichen Form zur Anwendung kommen soll. Da aber bei diesen Systemen die Datenbankanfrage im Grunde lokal durch das CMS stattfindet, werden somit für das Kommunikationssystem nicht zwingend datenbankorientierte Ansätze gefordert.

Weniger systemflexible Lösungen ungeeignet: DCE, COM-Familie

Damit verbleiben die allgemeineren Lösungen, welche stark mit den unter Distributed Computing Environment (DCE) genutzten Definitionen verwandt sind. Sie alle basieren meist auf einer eigenen, wenn auch unterschiedlichen Sprache zur Definition von Schnittstellen (Interface Definition Language (IDL)), organisieren die Verteilung durch handhabbare Strukturen und bieten häufig Zusatzdienste zur Verwaltung und für immer wiederkehrende Aufgaben an. Hauptkriterium ist deshalb, das flexibelste und unter allgemeinen Gesichtspunkten umfangreichste, offene Konzept herauszufinden. Unter diesem Gesichtspunkt scheiden DCE (Server nicht für alle Systeme) und die Component Object Model (COM) -Familie (hauptsächlich für Microsoft-Systeme nutzbar) aus, obwohl DCE auch über ein nützliches, verteiltes Dateisystem verfügen würde. Dieses ist allerdings an NFS orientiert und somit nicht zustandsbasiert.

Weniger sprachflexible Lösungen ungeeignet: Java

Auch Java mit den Java Beans kann ausgeklammert werden. Java bietet zwar viele nützliche Eigenschaften und wäre trotz ihres Interpretercharakters durch die Just In Time (JIT) -Compilertechnologie bzgl. des Laufzeitverhaltens vermutlich schneller oder zumindest äquivalent zu herkömmlichen Compilersprachen[17], schränkt jedoch auf eine Programmiersprache ein. Damit würde zusätzlich zu den in der Fundamentalstation Wettzell hauptsächlich eingesetzten Standardsprachen, welche auch nicht einfach ersetzt werden können, eine weitere Sprache eingeführt, welche damit die Gesamtkomplexität einer Einrichtung nur erhöht.

[16] vgl. auch [KIR04] a.a.O. S. 9
[17] vgl. dazu [MANG05]

Gegenseitige Zugangspunkte und Ähnlichkeiten verweisen auf CORBA

Alle genannten Allgemeinlösungen weisen jedoch in einem Punkt eine Gemeinsamkeit auf: Sie alle besitzen Zugangspunkte zu CORBA . CORBA selbst bietet nahezu alle gewünschten Eigenschaften, auch wenn Dateisystemdienste etc. nicht in der gewünschten Form vorgesehen sind. Natürlich ist anzumerken, dass die Weiterentwicklungsprozesse in den CORBA -Standards relativ blockierend ablaufen, was einen gewissen Unsicherheitsfaktor birgt. Gerade auch in Bezug auf neue Technologien, welche von vielen Stellen im Rahmen der Web-Services forciert werden. Diese neuen Technologien stecken jedoch in den Kinderschuhen und sind stark von verschiedenen Firmenpolitiken geprägt. Dabei lösen sie auch nur die Dienstzugangsproblematik und nicht Fragen bzgl. der Dienste selbst (z.B. konsistente Dateioperationen im parallelen Mehrnutzerbetrieb). Eines ist aber klar herauszustellen: Web-Services bieten durch ihre offene Schnittstellenkonzeption eine gute Möglichkeit, bestehende Middleware nutzbar zu machen.

Zu neue, unausgereifte Techniken ungeeignet: Web-Services

Entscheidung:
Gekapseltes CORBA mit eigenen Erweiterungen

Damit kann zusammenfassend CORBA als Basismiddleware aufgegriffen werde. Dazu sind einige Erweiterungen (Dateizugang, Authentifizierung, etc.) nötig, um mit rudimentären, herkömmlichen Techniken gewünschte Dienste zu schaffen. Zudem ist darauf zu achten, dass die Middleware gekapselt in eine Anwendung eingegliedert wird, so dass durch Austausch von wenigen Komponenten ein Wechsel der zugrundeliegenden Zwischenschicht zu neuen Techniken umsetzbar ist.

3.2.3 CORBA genauer betrachtet

Middleware als Black Box

Diese Arbeit beschäftigt sich nicht mit der Middleware an sich. Die Zwischenschicht ist vielmehr ein Mittel zum Zweck und kann in den weiteren Abschnitten als „Black Box“ angesehen werden, welche untergeordnete Aufgaben unter Einhaltung von Transparenzkriterien löst. An gegebener Stelle werden deshalb nur die von der Middleware zur Verfügung gestellten Zugangsschnittstellen erläutert, sofern sie zum Einsatz kommen. Trotzdem ist es an dieser Stelle der Entwicklung eines Kommunikationssystems wichtig, gewisse Einblicke in die Interna von CORBA zu erhalten.

Verteilungsplattform

Softwarebus

CORBA bildet durch ihren Aufbau eine klassische Verteilungsplattform für objektorientierte Strukturen, welche mittels des Object Request Broker (ORB) eine Art Softwarebus ermöglicht (vgl. dazu erneut Abb. 3.4 auf Seite 48). Das bedeutet, dass die Zwischenschicht der Anwendung transparent das Vorhandensein von Methoden und Datenstrukturen widerspiegelt, obwohl diese sowohl lokal als auch entfernt über ein Netzwerk angesprochen werden (vgl. dazu Abb. 3.3 auf Seite 47[18]. Die elementare Kommunikationsfunktionalität steckt dabei im sog. „ORB Core“, der zentralen Einheit der Middleware, welche auch die Datenkonvertierung (Marshalling) übernimmt[19]. Entfernte Objekte können dabei über eine eindeutige Referenz identifiziert werden[20]. Dieser Netzzeiger wird als Interoperable Object Reference (Interoperable Object Reference (IOR)) bezeichnet

Eindeutige Objektreferenzen

[18] in Anlehnung an [PUD01] a.a.O. S. 25
[19] vgl. [SELL00] a.a.O. S. 75
[20] vgl. für nachfolgenden Abschnitt [MERKL02]

und beinhaltet unter anderem den Rechnernamen, den Port und einen eindeutigen Schlüssel.

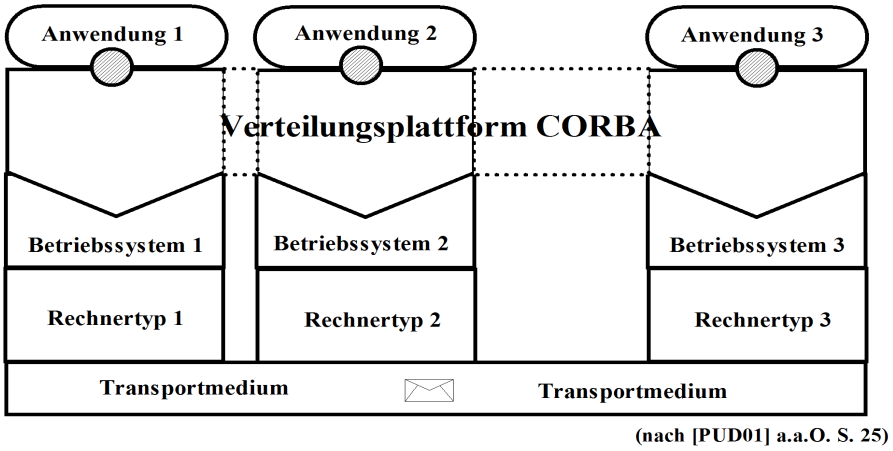


Abbildung 3.3: Schematische Darstellung einer Verteilungsplattform

Um ein Objekt ansprechen zu können, wird diese `IOR` vom Client vor dem ersten Kontakt in das System benötigt. Dieser erste Zeiger kann dabei über „string_to_object“-Routinen direkt eingegeben werden. Dies ist für direkte Verbindungen in kleineren Umgebungen geeignet. Für weitläufigere Anwendungen stehen Verzeichnisdienste zur Verfügung. Der sog. Naming Service beinhaltet wie ein Telefonbuch unter eindeutigen Dienstbezeichnungen die zugehörigen Referenzen. Der Trading Service fungiert des Weiteren wie die „Gelben Seiten“, wobei Zusatzangaben (z.B. Leistungskennzahlen) eine gezieltere Auswahl des Diensterbringers erlauben. Für die Referenzierung zum Kontakt dieser zentralen Komponenten existiert in CORBA eine Methode “ORB::resolve_initial_references“. Jegliche weitere Referenzierung ist von da ab für die Anwendung transparent.

Nutzung von Verzeichnisdiensten

Gerade solche zentralen Verzeichnisdienste können ideal für die Datenanbietersuche eingesetzt werden, wo verschiedene Anbieter in unterschiedlicher Qualität verschiedene Daten anbieten.

Die Architektur

Zentrale Komponente der Verteilungsplattform ist der sog. `ORB` , dessen wesentliche Aufgabe das Management der Methodenaufrufe (also die Abbildung auf das zugrundeliegende `RPC` -Kozept) ist. Für immer wiederkehrende Anforderungen in einer verteilten Anwendung wurden auf den `ORB` aufbauende Dienste (Namensdienst, Ereignisdienst, Traderdienst, etc.) in der `Common Object Service Specification (COSS)` zusammengefasst. Auf diese setzen weitere „Common Facilities“ zur Administration der vernetzten Anwendung und die Frameworks für spezielle Anwendungsfälle auf.

Verteilungsplattform mit Diensten und darauf aufbauenden Erweiterungen

`CORBA` ist eine klassische Middleware (vgl. dazu Abb. 3.4[21] auf Seite 48) mit Abstraktionsschichten in denen eigene „Repositories“ (Verwaltungsstrukturen mit Konvertierungsinformationen) für entfernte Objekte die einzelnen Vertreter

[21] in Anlehnung an [PUD01] a.a.O. S. 41

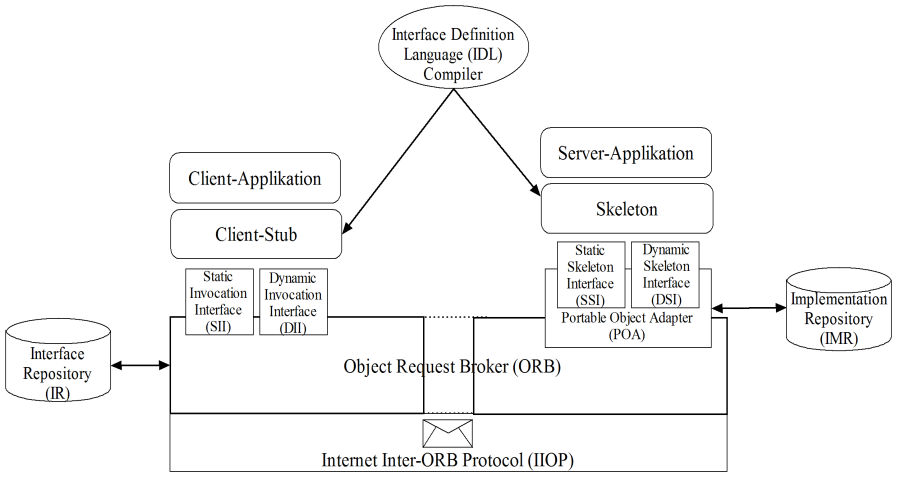


Abbildung 3.4: CORBA-Architektur

Interface Definition Language

(Client- und Server-Stubs (Skeletons)) verwalten. Die Schnittstellen werden sprachunabhängig in einer eigenen Interface Definition Language (`IDL`) deklariert und mittels eines speziellen `IDL` -Compilers in den Zielcode übersetzt. Zum Kontaktieren eines entfernten Objekts werden verallgemeinerte `IOR` genutzt, welche aus Performanzgründen meist statisch Verwendung finden, aber z.B. für das `Dynamic Invocation Interface (DII)` auch dynamisch über den Namensdienst abgerufen werden können. Die Aufrufsemantik für entfernte Methoden ist „at-most-once“ (`RPC`) oder „maybe“ (`RPC` ohne Antwort) (vgl. zu Aufrufsemantik Abschnitt 3.2.1 auf Seite 40). Auf Seite des Servers verbindet ein transparenter `Portable Object Adapter (POA)` den `ORB` mit der Methodenrealisierung.

Aufrufsemantik „at-most-once“ und „maybe“

Sprachunabhängigkeit

Die Umsetzung ist sprachunabhängig und findet durch Übersetzung des Codes auf dem Zielsystem statt, indem eine Implementierung der `CORBA` -Spezifikation von einem beliebigen Hersteller individuell hinzu gebunden wird. Die definierte „InterORBability“ (Interoperable `ORB` s) sorgt dabei dafür, dass verschiedene Implementierungen, die sich an den Standard halten, miteinander kombinierbar sind. Die Kommunikation findet über ein eigens Protokoll, das `Internet Inter-ORB Protocol (IIOP)` , statt.

Die Schnittstellendefinition mit IDL

Innerer Aufbau eines Interfaces

Schnittstellen zwischen Client und Server werden in `CORBA` über die sog. Interface Definition Language (`IDL`) definiert[22]. Dabei handelt es sich um eine abstrakte Sprache, mit welcher der Aufbau, nicht aber die Realisierung eines Interface festgelegt wird. Diese Definition kann mittels standardisierter `IDL` -Sprachanbindungen (momentan für C, C++, Java, Smalltalk, Ada und Cobol) auf nahezu jede beliebige Sprache abgebildet werden, sofern diese die Grundprinzipien z.B. der `IDL` -Datentypen unterstützt. Ein `IDL` -Compiler übersetzt aufgrund der Anbindungsdefinition eine Schnittstellenbeschreibung in zwei Stellvertreter-Objekte: den Client-

Sprachanbindung und Abbildung durch IDL-Compiler

[22] vgl. für folgende Ausführungen [PUD01] a.a.O. S. 42 f. und [SELL00] a.a.o. S. 76 ff.

Stub und das Server-Skeleton (vgl. dazu Abb. 3.4 auf Seite 48). Die Clientanwendung kann so Server-Methoden aktivieren, welche vom Client-Stub vertreten werden. Die Realisierung der Server-Skeleton-Methoden erfolgt dabei separat in der gewählten Programmiersprache. Anschließend können die jeweiligen server- und clientseitigen Quellcodes mit einem Standardcompiler der jeweiligen Sprache in ein für eine Plattform ausführbares Programm übersetzt werden. So ergibt sich die komplette Sprach- und Plattformtransparenz als Teil der Technologietransparenz.

Ziel ist die Sparch- und Plattformtransparenz

Welche Probleme in diesem Vorgehen enthalten sind, ist daraus ersichtlich, dass jede zugrundeliegende Programmiersprache unterschiedliche Strategien verfolgt. Die in IDL genutzten Mehrfachvererbungen lassen sich z.B. in prozeduralen Sprachen wie C nur ungenügend nachbilden. C++ ist dabei ein relativ guter Kandidat, da nahezu eine direkte Umsetzung erfolgen kann. In Java z.B. verbergen sich hier jedoch einige Probleme. Java unterscheidet zwischen Schnittstellen (abstrakte Interfaces mit Mehrfachvererbung) und Klassen (Implementierungen mit Einfachvererbung). Eine Abbildung von IDL -Interfaces nach Java-Interfaces würde sich anbieten, wenn dort statische Methoden erlaubt wären, wie sie in CORBA z.B. für das Einfügen von Objekten in eine Instanz des „any“-Typs benötigt werden. Diese Problematiken sind nur über entsprechende Umwege, z.B. mit sog. „Helper-Classes“, zu lösen. Auch die in Java fehlende Unterstützung der Referenzübergabe von Parametern führt zu ähnlichen Konstrukten. So ist bereits daraus eine Tendenz zur Programmiersprache C++ abzuleiten.

Probleme bei der Abbildung auf verschiedene Sprachcharakteristika

Tendenz zur Nutzung von C++

Das Kommunikationsprotokoll IIOP

CORBA definiert ein eigenes Protokoll[23] zur Kommunikation zwischen den verteilten Anwendungen (zwischen ORBs): das Internet Inter-ORB Protocol (IIOP). Dabei handelt es sich um eine Spezialisierung des allgemeinen, einfach verständlichen und allgemein zugänglichen General Inter-ORB Protocol (GIOP) . In diesem werden drei Festlegungen für die Kommunikation getroffen:

IIOP als Spezialisierung des GIOP zur Kommunikation zwischen den ORBs

1. Die Common Data Representation (CDR) ist eine Transfersyntax, durch welche die IDL -Datentypen in grundlegende Datentypen für den Datenaustausch umgewandelt werden. Sie legt die Byteanordnung (Endian), die interne Speicheranordnung („Memory-Alignment“) für primitive Datentypen und eine komplette Abbildung der IDL -Typen fest. Zu erwähnen ist, dass immer die Senderangaben enthalten sind und damit immer der Empfänger für die korrekte Umsetzung verantwortlich ist („Receiver makes it right“). Die CDR sorgt insgesamt also für Transparenz bzgl. Datenformate.

2. Zudem werden sieben Nachrichtenformate festgelegt, wobei eine Nachricht immer aus Message-Header (Protokollinfos, etc.) und Service-Context (Inhalt) besteht:

   (a) Request: Zugriff auf Attribute und Operationen durch den Client (Angaben zu Aktionen werden als Strings übertragen)

   (b) Reply: Der Server antwortet auf einen Request mit einem Reply. Darin sind die Ein-/Ausgabeparameter, der Rückgabewert und evtl. aufgetretene Ausnahmezustände enthalten. Bei einer Oneway-Operation wird

[23] vgl. dazu [MINT02] und [MERKL02]

keine Antwort erwartet und so auch kein Reply gesendet. Ist ein Zielobjekt mittlerweile an einem anderen Ort, erfolgt eine LocationForward-Antwort, wodurch die Ortstransparenz gewährleistet wird.

(c) CancelRequest: Will der Client einen Zugriff abbrechen, sendet er diese Nachricht.

(d) LocateRequest: Will ein Client eine Objektreferenz auf Gültigkeit prüfen, erfolgt dieser Aufruf.

(e) LocateReply: Der Server antwortet auf einen LocateRequest mit LocateReply. Darin enthalten ist, ob die Referenz bekannt ist oder nicht und zudem gegebenenfalls der neue Ort des Objekts, sofern dieser bekannt ist.

(f) CloseConnection: Will ein Server die Abarbeitung abbrechen, meldet er an alle beteiligten Klienten die Beendigung.

(g) MessageError: Sowohl Client als auch Server können erkannte Probleme an den Partner weitermelden. Dies ermöglicht eine dedizierte Reaktion.

3. GIOP gibt zudem Vereinbarungen für den Nachrichtentransport selbst: So wird festgelegt, dass die Übertragung verbindungsorientiert und verlässlich sein muss. Der Transport selbst ist eine Übertragung einer Byte-Folge. Die Initialisierung davon ist abbildbar auf den TCP /IP -Stack. Zudem wird bei einem ungewünschten Abbruch eine Angabe der Ursache zurück gemeldet. Gerade letzteres ist für automatische Systeme zur Qualitätssicherung und Reaktionsbestimmung sehr wichtig.

Flexibles, robustes, skalierbares, transaktionsorientiertes, reihenfolgewahrendes Internetprotokoll ohne Sitzungsorientierung

Im Falle von IIOP ergibt sich damit ein flexibles Internetprotokoll, welches zeitweise als revolutionär für den allgemeinen Internetverkehr angesehen wurde, wodurch zahlreiche Anwendungen mit Unterstützung des Protokolls entstanden (z.B. IIOP -Erweiterung im Netscape Browser). Es ist robust, skalierbar und transaktionsorientiert und erhält im Gegensatz zu HTTP die Reihenfolge der Nachrichten, wodurch eine zeitliche Ereignisanordnung entsteht. Allerdings ist es leider auch nicht sitzungsorientiert (jeder neue Aufruf ist unabhängig vom vorhergehenden). Der Aufbau des Protokollblockdiagramms ist in Abb. 3.5 auf Seite 51 dargestellt.

Überwinden von Firewalls: Spezielle Proxies oder Tunnel

Die Firewallproblematik erlegt jedoch einige Schranken auf (vgl. bzgl. der Überwindung von Firewalls den Abschnitt 4.5.4 auf Seite 123). Da es sich bei IIOP um ein weiteres Protokoll handelt, welches über Sicherheitssperren geschleust werden muss, gibt es zur Überwindung dieser zum einen den Einsatz von speziellen Proxies oder zum anderen die Nutzung von Tunnelung mittels Fremdprotokollen bzw. den kompletten Wechsel zu einem anderen Protokoll. Da Proxies nicht generell vorausgesetzt werden können und die Tunnelung relativ viele zusätzliche Zwischenstufen benötigt, ist bereits hier anzumerken, dass wohl langfristig ein kompletter Wechsel z.B. zu dem auf HTTP basierenden Simple Object Access Protocol (SOAP) oder ähnlichen Protokollen nötig werden kann.

GIOP bietet durch seine Flexibilität viele Vorteile bzgl. der Einhaltung von Transparenzkriterien

Der Grundgedanke hinter GIOP -Ableitungen ist allerdings gerade im Hinblick auf Fehleranalysen und die geforderten Transparenzkriterien (vgl. Abschnitt 2.4 auf Seite 31) sehr dienlich, da diese bereits teilweise ohne weiteres Zutun gelöst

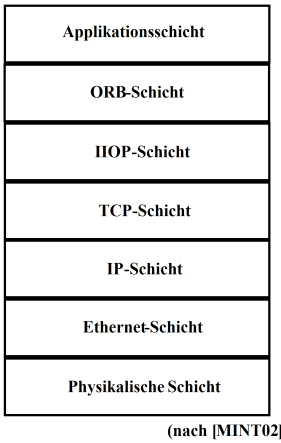


Abbildung 3.5: Blockdiagramm für IIOP

sind (z.B. Technologietransparenz für Datentypen, Orts- und Zugriffstransparenz). Dies wird zusätzlich dadurch gefördert, dass weitere Protokolle abgeleitet werden können und in der Form eines sog. `Environment Specific Inter-ORB Protocol (ESIOP)` transparent nutzbar sind (evtl. kann so auch `SOAP` adaptiert werden).

Die CORBA-Dienste (Services)

Die `CORBA` -Services[24] standardisieren eine Reihe von Dienstleistungen der Middleware, welche von vielen Anwendungen benötigt werden, ohne die Implementierung an sich festzuschreiben. So enthalten die Standardisierungen einige zusätzliche, unberücksichtigte Teilaspekte (z.B. beim „Event Service“ wird die Erhaltung der Reihenfolge der Ereignisse nicht zwingend vorgeschrieben[25]), welche für Produktionssysteme einige Schwierigkeiten bergen. So ist es auch nicht verwunderlich, dass nur die wenigsten `CORBA` -Implementierungen nahezu alle Services anbieten. Da einige Dienste eine enge Verzahnung zum `ORB` erfordern, können Service-Implementierungen von unterschiedlichen Herstellern auch nicht beliebig gekoppelt werden.

Standardisierung, jedoch mit Lücken

Die in der sog. Common Object Service Specification (`COSS`) zusammengefassten Dienste sind im Allgemeinen `IDL` -Definitionen, welche aus Vorgaben für den Server und für den Client bestehen. Nachfolgend sind die Dienste aufgelistet:

COSS

1. **Naming Service:** Dabei handelt es sich um eine Art Telefonbuch, so dass Server ihre Objektreferenzen unter charakteristischen Namen veröffentlichen können.
2. **Event Service:** Ermöglicht die Reaktion auf Zustandsänderungen mittels Ereigniskanälen ohne Nutzung einer synchronen Aktivierung durch einen Client (aber: z.B. keine Atomarität und Reihenfolgenerhaltung vorgesehen)
3. **Persistent Object Service:** Dadurch können Objektzustände transparent auf beliebige Speichermedien ausgelagert werden. (aber: z.B. Kompromiss bzgl. Gemeinsamkeiten verschiedener Speichermedien)

[24] vgl. dazu [SELL00] a.a.O. S. 83 ff.
[25] vgl. [SELL00] a.a.O. S. 89 f.

4. **Lifecycle Service:** Definiert Funktionalität zum dynamischen Erzeugen, Kopieren, Migrieren und Löschen von Objekten (aber: z.B. keine Lösung für Migration von Ressourcenzugängen)
5. **Concurrency Service:** Regelt konkurrierenden Zugriff auf Objekte (Sperrmodi; meist Kombination mit Transaction Service)
6. **Externalization Service:** Ausgabe/Eingabe des Objektzustands in/aus einem Stream
7. **Relationship Service:** Unterstützt die Verwaltung und Modellierung von Objekten untereinander
8. **Transaction Service:** Ermöglicht transaktionsorientierten, verwalteten Datenaustausch (aber: relativ komplex und umfangreich)
9. **Query Service:** Bietet Möglichkeit zu mengenorientierten Anfragen an beliebige Datenquellen
10. **Licensing Service:** Ermöglicht Verwaltung von Lizenzrechten
11. **Property Service:** Dient zur Handhabung von Objekteigenschaften
12. **Time Service:** Ermöglicht Zeitformate und Schnittstellen für Zeitstempel basierend auf einem Timer
13. **Security Service:** Regelt eine sichere Kommunikation und erlaubt Authentifizierung und Autorisierung (aber: nur bzgl. der Objekte und nicht im Rahmen des Betriebssystems)
14. **Trading Service:** Nachbildung der „Gelben Seiten“ mit mengenorientierten Zusatzinformationen zu einem Dienst
15. **Collection Service:** Definitionen für Aggregattypen und Iteratoren (Verwaltung von Datensammlungen)
16. **Persistent State Service:** Ersetzt den Persistent Object Service und erlaubt den Austausch von Zustandsinformationen

Effektiver Nutzen der Dienste unterschiedlich

Bewertend kann festgestellt werden, dass viele der Dienste nicht unbedingt notwendig sind, da sie entweder zu umfangreich, zu wenig ausgereift oder durch die enge ORB -Kopplung implementationsabhängig sind. Generell als nützlich können aber z.B. die Verzeichnisdienste angesehen werden.

Verschiedene Implementierungen

Zahlreiche Beispielimplementierungen

CORBA wurde, wie erwähnt, von der OMG als Standard herausgegeben. Allerdings stellt die OMG selbst keine Beispielimplementierung zur Verfügung, sondern weist nur auf Implementierungen hin, welche sich genau an den Standard halten. Somit gibt es zahlreiche kommerzielle, aber auch freie Produkte auf dem Markt, welche mehr oder weniger zahlreich für verschiedene Anwendungen bereits eingesetzt wurden oder werden.

Implementierungen	mico/E	MICO	TAO 1.1a	TAO 1.2a (beta) (und höher[2004])	JacORB	omniORB 3	omniORB 4.0 (prev)	OpenORB
Unterstützte Sprachen								
C++	-	Ja	Ja	Ja	-	Ja	Ja	-
Java	-	-	-	-	Ja	-	-	Ja
Python	-	-	-	-	-	Ja	Ja	-
Tcl	-	Ja	-	-	-	-	-	-
Zusätzlich unterstützte Protokolle								
COM	-	-	-	-	-	-	-	-
SOAP	-	-	-	-	-	?	-	Geplant
Grundfunktionalität (Core)								
Dynamic Infocation Interface	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Dynamic Skeleton Interface	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Interface Repository	Ja	Ja	Geplant	Ja	Ja	Ja	Ja	Ja
Basic Object Adapter	Ja	Ja	-	-	-	Ja	Ja	Ja
Portable Object Adapter	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Value Type Semantics	Ja	Ja	Ja	Ja	Geplant (Ja[2004])	-	-	Ja
Erweiterte Grundfunktionalität (Extended Core)								
Minimum CORBA	-	- (Ja[2004])	Ja	Ja	-	-	-	-
Asynchronous Messaging Interface	-	- (Geplant[2004])	Ja	Ja	- (Ja[2004])	-	Geplant	Geplant (-[2004])
CORBA Component Model	Geplant	Geplant (Ja[2004])	Geplant	Geplant	-	-	Geplant (-[2004])	Geplant (-[2004])
Quality of Service	-	-	Ja	Ja	-	-	-	-
Fault Tolerance	-	- (Geplant[2004])	Geplant	Geplant	-	-	-	Geplant
Realtime	-	-	Geplant	Ja	-	-	-	-
Realisierte CORBA-Dienste								
Naming Service	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Trading Service	Ja	Ja	Ja	Ja	Ja	-	-	Ja
Event Service	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Transaction Service	-	-	-	- (Geplant[2004])	Ja	-	-	Ja
Security Service	-	Ja	Geplant	Ja	Geplant	-	-	Geplant (-[2004])
Time Service	-	Ja	Ja	Ja	-	-	-	Ja
Persistent State Service	Geplant	-	-	- (Geplant[2004])	-	-	-	Ja
Concurrency Service	-	-	Ja	Ja	Ja	-	-	Ja
Property Service	Geplant	Ja	Ja	Ja	-	-	-	Ja
Relationship Service	Geplant	Ja	-	-	-	-	-	-
Notification Service	Geplant	Geplant	Ja	Ja	- (Ja[2004])	Ja	Ja	Ja

nach [PUDPRO01]
und [PUDPRO04]

Tabelle 3.1: CORBA Produktprofile für „Open-Source“-Implementierungen (in den Jahren 2001 und 2004)

Kriterium: Anzahl umgesetzter Komponenten

Damit stellt sich die Frage, auf welcher CORBA -Implementierung aufgesetzt wird. Dies spielt grundsätzlich zwar nur eine untergeordnete Rolle, da meist verschiedene, standardkonforme Implementierungen austauschbar sind, trotzdem gibt es kleine Unterschiede (z.B. bzgl. der Anzahl umgesetzter Dienste, Methoden- und Klassenaufbau, Aufrufparameter des ORB , etc.). So sollte ein Kriterium auch der gebotene Umfang an standardisierten Einzelkomponenten sein.

Kriterium: Open Source

Ein entscheidendes Kriterium im Rahmen dieser Arbeit ist die freie Verfügbarkeit des Codes im Sinne von „Open-Source“, da damit der komplette Quelltext zur Verfügung steht. Aufgrund dieser Festlegung kann bereits eine Auswahl getroffen werden, bei welcher die kommerziellen Produkte (z.B. ORBacus, VisiBroker, Orbix, etc.) ausscheiden. Die verbleibenden freien Produkte werden mittels nachfolgender, im Jahre 2001 vorliegender Tabelle 3.1 auf Seite 53 hinsichtlich der umgesetzten Zusatzdienste und Standardkonformität bewertet[26]. Mit eingetragen sind die im Jahre 2004 erneut überprüften Vermerke[27], wodurch die Weiterentwicklungen sichtbar werden (eine komplette Übersicht befindet sich im Anhang C auf Seite 223).

Kriterium: Unterstützung von C++

Die meisten Implementierungen unterstützen die Programmiersprachen C++ und Java. Da die OMG bei ihren Vorgaben in etwa die Strukturen von C++ als Grundlage nahm und so auch die Sprachabbildung ideal umgesetzt ist, bietet sich der Einsatz von C++ an (vgl. Abschnitt 3.2.3 auf Seite 49). Bei heute gängigen C/C++-Compilern handelt es sich um sehr hoch entwickelte und gut funktionierende Werkzeuge. Im Hinblick auf Umfang und benötigte Funktionalität sind insbesondere MICO Is CORBA (MICO) und ACE /TAO zu nennen, welche in den höheren Versionen als am besten geeignet erscheinen.

MICO: Minimalistischer Implementierungsansatz

MICO stand ursprünglich für „Mini CORBA“. Es wurde federführend von Kay Römer und Arno Puder entwickelt, um eine Grundlage für ein praxisnahes Lehrbuch zum Thema Middleware für verteilte Systeme zu schaffen. Durch die Open-Source-Mitarbeit entstand aus dem minimalistischen Ansatz eine komplette CORBA -Implementierung, so dass als Akronym MICO Is CORBA (MICO) gewählt wurde[28]. Es handelt sich aber immer noch um einen kompakt gehaltenen Lösungsansatz, welcher sich komplett an den Standard hält und deshalb auch als Beispielimplementierung von der OMG anerkannt wird. MICO basiert auf allgemeinen Bibliotheken in C++. Das klare Design setzt nur wirklich für CORBA benötigte Elemente um und ist ideal für erste, experimentelle Applikationen geeignet. Die Quellen selbst sind frei verfügbar und unterstützen diverse Plattformen (z.B. Sun Solaris, IBM, AIX, HP-UX, Linux, Digital Unix, Ultrix, Windows NT und Pocket PC-Systeme)[29].

TAO: Implementierung zur Lösung von Optimierungsproblemen

TAO hingegen steht für „The ACE Object Request Broker“. Die ersten Entwicklungen dazu entstanden infolge der Erstellung einer Dissertation im Zusammenhang mit dem Design und der Performanzoptimierung für parallel arbeitende Protokollstacks, wozu eine Umgebung mit Namen ACE geschaffen wurde. Mit dieser löste Douglas C. Shmidt zwei Anfang der 90er Jahre existierende Problematiken:

[26] vgl. dazu [PUDPRO01]
[27] vgl. dazu [PUDPRO04]
[28] vgl. [ROEM01]
[29] vgl. [WEB602]

- Für grundlegende Funktionalitäten auch in objektorientierten Umgebungen mussten immer noch prozedurale C-Funktionen genutzt werden.
- Für die Handhabung von allgemein existierenden Aufgaben bzgl. Netzwerkprogrammierung gab es keine allgemeine Schnittstelle.

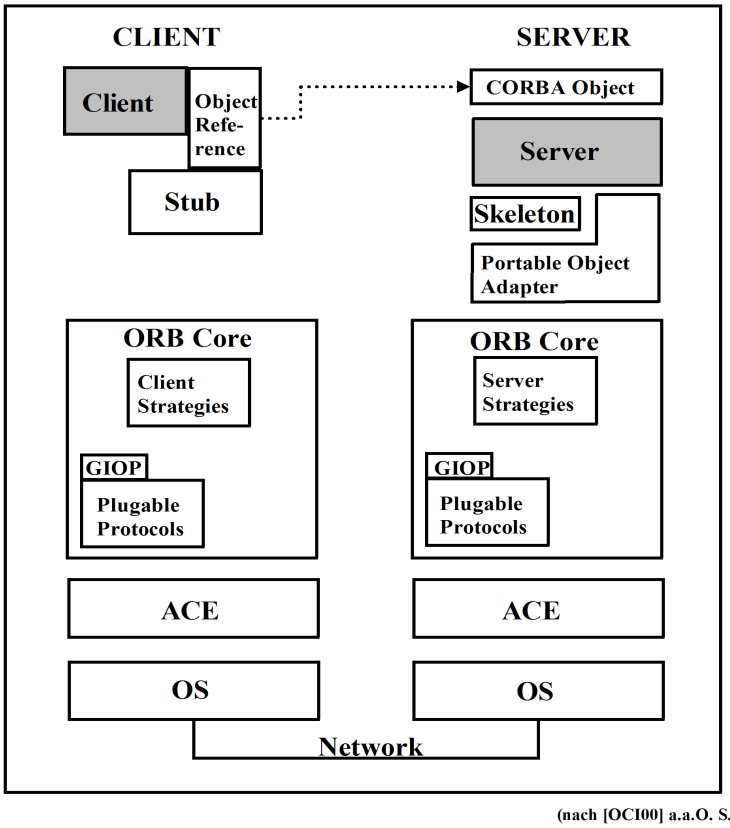


Abbildung 3.6: TAO Architektur

Unterstützung zeitkritischer Abläufe bereits durch die Middleware

Die bereits in diesem Zusammenhang entwickelten Lösungen wurden ab dem Jahr 1994 durch die immer mehr propagierten Middleware-Ansätze zu skalierbaren, objektorientierten, verteilten Systemen ausgebaut. Die breite Unterstützung durch Firmen trug schließlich auch dazu bei, dass die resultierende Implementierung TAO zahlreich eingesetzt wird. Ideen in den Bereichen ablaufkritischer Anwendungen mit Sicherung entsprechender Qualität und eines erforderlichen Durchsatzes bei gegebener Latenzzeit waren schließlich Ausgangspunkte für die Echtzeiterweiterungen von CORBA durch die OMG (Real-time CORBA)[30]. Die zugrunde liegende Architektur von TAO ist in Abb. 3.6 auf Seite 55 dargestellt[31]. Die Quellen von TAO sind für nahezu alle verbreiteten Betriebssysteme nutzbar.

TAO als Grundlage für den Prototyp

Aufgrund der guten Unterstützung von verschiedenen Seiten, den zahlreichen Informationsquellen und den umfangreichen Quellen wird TAO als Grundlage genutzt. Besonders die existierenden Möglichkeiten der asynchronen Kommunikation (Ausnutzung von natürlicher Parallelität) und der Echtzeitmöglichkeit sind hierbei interessante Aspekte. MICO dient dabei in Teilen nur zur Verifikation und als

[30] vgl. [KREKL04]
[31] vgl. [OCI00] a.a.O. S. 7

Experimentierumfeld u.a. auch deshalb, weil das Übersetzen für das Betriebssystem TRU64 UNIX von Hewlett-Packard (ehem. DEC) nicht ohne größeren Aufwand möglich war. Vergleicht man die Profilinformationen der CORBA -Implementierungen aus dem Jahr 2004[32](also fast drei Jahre nach Projektstart) im Anhang C auf Seite 223 mit den Einträgen in Tabelle 3.1 auf Seite 53, so ist aufgrund des Entwicklungsstandes festzustellen, dass die Entscheidung für TAO gerechtfertigt war.

3.3 Umsetzungsdesign: CORBA File Transfer (CFT)

Aufbauend auf die gewählte Middleware kann nun ein neues Design zur Datenübertragung speziell für Dateien entwickelt werden. Zweck dieses ersten Prototyps ist es, die Machbarkeit und Funktionalität des Ansatzes aufzuzeigen. Er ist noch nicht mit den später benötigten, komfortablen Fähigkeiten ausgestattet, kann jedoch bereits für spezielle Aufgaben eingesetzt werden. Das hier entwickelte Design wird später für Anforderungen eines realen Produktionssystems erweitert (vgl. dazu die Erweiterung zum Extended CORBA File Transfer in Kapitel 4 auf Seite 83).

3.3.1 Festlegungen

Basierend auf den vorbereitenden, theoretischen Grundlagen der letzten Abschnitte, ergeben sich Festlegungen, welche für das gesamte System bindend sind. Ergänzend dazu sind einige Vorgaben durch die bestehenden Systeme zu berücksichtigen.

Festlegung:
TRU64, Linux und Windows als Zielbetriebssysteme

Auf der Fundamentalstation Wettzell werden hauptsächlich die Betriebssysteme TRU64 UNIX (ehem. DEC-UNIX), Linux (u.a. von SuSe) und Windows (2000/NT) eingesetzt. Deshalb ist dafür zu sorgen, dass das entwickelte System auf allen diesen Systemen lauffähig ist. Dazu ist jeweils die CORBA -Plattform für die Betriebssysteme zu übersetzen und zu installieren, so dass die Bibliotheken in das Projekt eingebunden werden können.

Festlegung:
gcc und MSVC++ als Compiler für C++

Wie zuvor dargelegt, ist die Wahl von C++ als Programmiersprache durchaus legitim. Insbesondere ist diese Entscheidung auch dahingehend vorteilhaft, dass in der prozeduralen Sprache C geschriebene Programme der Messsysteme einfach eingebunden werden können. Der entwickelte Programmcode ist damit kein Interpretercode, sondern muss für jede Zielplattform explizit übersetzt werden. Auf der Fundamentalstation werden momentan die Compiler „gcc“ für UNIX-Derivate und „Microsoft Visual C++“ (MSVC++) für die Windowswelt genutzt. Der entstehende Code ist deshalb so zu standardisieren, dass eine Übersetzung mit diesen Compilern möglich ist.

Festlegung:
Minimalprinzip mit Middleware als „Black Box“

Für die Realisierung werden so viele CORBA -Dienste wie wirklich nötig aber nur so wenig wie gerade möglich in Anspruch genommen. Dieser Ansatz vereinfacht die Möglichkeit, einen späteren kompletten Middleware-Wechsel durchzuführen. Die Abhängigkeiten zu CORBA werden so entschieden vermindert. Alle nötigen Erweiterungen setzen auf der Middleware auf, d.h., es werden keine Änderungen an der Zwischenschicht selbst vorgenommen. Diese ist eine „Black Box“.

[32] vgl. [PUDPRO04]

Der gesamte Zugangspunkt dazu für Verwaltung und erstmalige Referenzierung von Basisobjekten wird in der endgültigen Anwendung gekapselt und durch verallgemeinerte Methoden aktiviert. Die Erweiterungen werden schichtenweise darübergelegt, so dass der Grad der Abstrahierung frei gewählt werden kann. Das resultierende System ist eine erweiterte Middleware, welche proprietäre Anforderungen löst.

Festlegung:
Nutzung des Naming Service von CORBA

Um dynamische Verbindungen ohne vorherige Kenntnisse der Örtlichkeiten zu ermöglichen, ist zumindest ein Verzeichnisdienst erforderlich, bei dem sich die Server registrieren lassen. Hier wird zur Vereinfachung auf den Naming Service von CORBA zurückgegriffen. Dieser kann zentral angesiedelt sein und eine Vielzahl von Klienten bedienen. Um den Austausch der Serverreferenz bei einer Punkt-zu-Punkt-Verbindung zwischen Client und Server ebenfalls möglichst einfach zu gestalten, ist auf den Plattformen der Server zusätzlich ein solcher Dienst zu realisieren. Dieser trägt im Allgemeinen nur die Referenzen der lokalen Dienstleister.

Festlegung:
Strikte Trennung des aktiven Klienten vom passiven Server

Betrachtet man sich die Abläufe in der Fundamentalstation Wettzell, ist festzustellen, dass es sich meist um zentralistische Systeme handelt. Diesem Punkt wird durch vorgenannte Festlegungen ebenfalls entsprochen. Ein zentraler Server dient als Anlaufpunkt für aktive Messklienten. Kann das Messsystem nur aktiv angesprochen werden, so muss sich dort ein Server befinden, welcher über einen Client mit dem zentralen Server Datenaustausch betreibt. Unterschiedliche Multi-Tier-Modelle (vgl. zur Erklärung ODBC in Abschnitt B.4 auf Seite 213) sind denkbar.

Festlegung:
„Lock&Feel“-Prinzip

Für die Nutzerinteraktion soll ein Konzept des „Look&Feel“ umgesetzt werden. Dieses besagt, dass genutzte Komponenten in ihren Nutzerzugangsstellen nachgebildet werden, so dass der Anwender direkt keinen Unterschied zum vorherigen System erkennen kann, was eine rudimentäre Forderung für Innovationstransparenz (vgl. dazu Abschnitt 2.4 auf Seite 31) darstellt.

Mit diesen Festlegungen kann nun zur Entwicklung der Schnittstelle zwischen Client und Server übergegangen werden.

3.3.2 Die Schnittstelle zwischen Client und Server in CFT

FTP als Ausgangspunkt der Betrachtungen

Die pragmatische Vorgehensweise beim Entwurf einer verteilten Middleware ist in erster Annäherung das Design einer Schnittstelle. Die nachfolgende Schnittstelle zum Datenaustausch entstand aus einer Betrachtung der Funktionalität herkömmlicher FTP -Clients. FTP kann als eines der meist genutzten Protokolle zum Datenaustausch angesehen werden. Die Clients implementieren darauf aufbauend unterschiedliche Befehlsfunktionalität über die entweder elementare Kommandozeileninterpreter oder graphische Werkzeuge platziert sind. Die gemeinsame Schnittmenge aller Befehle der einzelnen FTP -Programme kann somit als grundlegender Befehlsumfang für den Datenzugang angesehen werden (vgl. für nachfolgende Betrachtungen Abb. 3.7 auf Seite 58).

Grobgliederung: Rahmenanweisungen und Sitzungsanweisungen

Eine erste Betrachtung zeigt, dass es Anweisungen gibt, welche eine Verbindung initiieren oder beenden bzw. grundlegende Verwaltungsaufgaben übernehmen (z.B. „open“, „close“, „user“). Sie wirken auf den allgemeinen Kontext einer Kommunikation ein. Erst wenn diese Anweisungen ausgeführt wurden, wird ei-

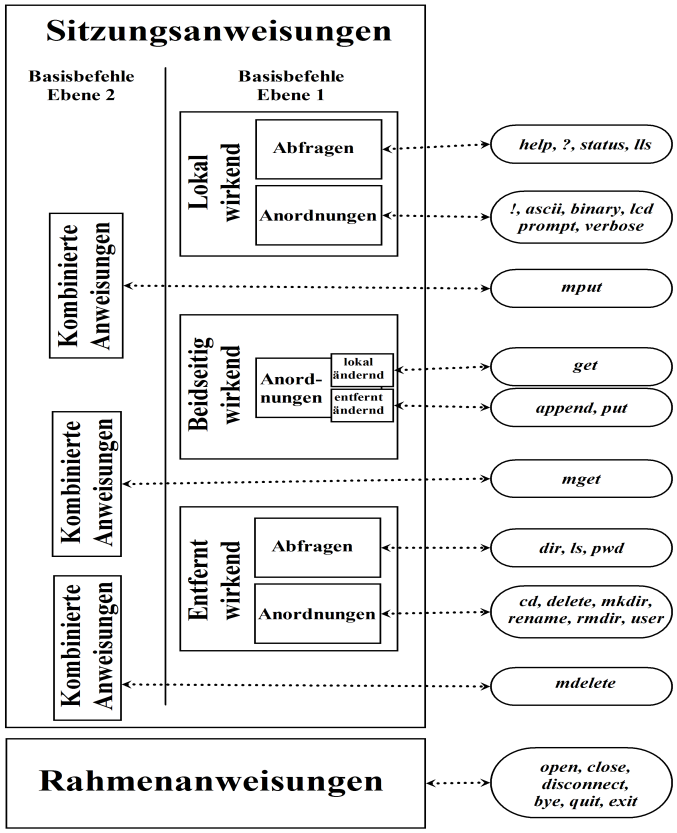


Abbildung 3.7: Strukturierung der Anweisungen

ne Vielzahl weiterer möglich. Diese erste Gruppe von Befehlen wird deshalb als Rahmenanweisungen bezeichnet. Da die weiteren Befehle den Ablauf einer Datensitzung steuern, werden sie fortan als Sitzungsanweisungen bezeichnet. Wobei hier Sitzung als Folge von getätigten Anweisungen mit dem Zweck der Erledigung einer Aufgabe und noch nicht im Sinne einer sitzungsorientierten Kommunikation verwendet wird.

Gliederung der Sitzungsanweisungen in lokal, entfernt oder beidseitig wirkend

Eine nähere Betrachtung dieser Sitzungsanweisungen eröffnet, dass die Menge generell in drei Gruppen aufgesplittet werden kann: lokal, entfernt (remote) und beidseitig wirkende Kommandos. Lokal wirkende Befehle bedürfen keiner Übertragung über ein Kommunikationsmedium. Sie agieren auf lokalen Ressourcen innerhalb der eigenen Programmverantwortung (z.B. „lcd“ für einen lokalen Verzeichniswechsel). Remote-Befehle wirken mittels Kommunikation zu Servern. Sie haben keine Auswirkungen auf lokale, durch den Client verantwortete Gegebenheiten. Die Ressourcenverwaltung obliegt dem Server, welcher meist auf einem anderen Rechner angesiedelt ist (z.B. „cd“ für einen entfernten Verzeichniswechsel). Sowohl lokal als auch entfernt wirken alle weiteren Befehle, welche zum eigentlichen Datenaustausch benötigt werden. Hierbei werden sowohl entfernte als auch lokale Ressourcen angesprochen bzw. verändert („get“ für das Anfordern einer Datenmenge in Form einer Datei). Die Bearbeitung ist dabei grundsätzlich einseitig beeinflussend, d.h., dass nur auf einer Seite eine Veränderung herbeigeführt wird (schreibender Dateizugriff), wobei die andere Seite als unveränderter Lieferant dient (lesender Zugriff).

Direkt wirkende, atomare Anweisungen vs. beidseitig wirkende, zusammengesetzte Anweisungen

Betrachtet man die so aufgeteilten Kommandos näher, sind weitere Charakteristika zu erkennen. Sowohl die lokal als auch die remote wirkenden Anweisungen sind direkt wirkende, atomare Abläufe. Sie können prinzipiell zusammenhanglos nacheinander ausgeführt werden, sofern die leicht bindenden, semantischen Voraussetzungen (z.B. Pfadangaben) stimmen. Diese atomaren Einheiten lassen sich damit auch durch `CORBA` und das `IIOP` direkt abbilden, da jeder Aufruf einer entfernten `CORBA` -Methode im Sinne eines kritischen Abschnitts atomar ausgeführt wird. Anders sieht es bei den übrigen beidseitig wirkenden Abläufen aus. Sie setzen sich aus einer Folge von mehreren lokal und entfernt abfolgenden Befehlen zusammen, welche eng gekoppelt aufeinander in zeitlicher Reihenfolge aufbauen. Die Abbildung ist deshalb schwierig, da `IIOP` nicht sitzungsorientiert arbeitet und damit Vorgängerzustände verloren gehen. Deshalb wird bereits hier ersichtlich, dass eine verteilte, sitzungsorientierte Verwaltung implementiert werden muss.

Geeignetes Aufsplitten von Schnittstellenmethoden

Grundsätzlich könnten damit die beidseitig wirkenden Befehle als eine Art übergeordnete, sitzungsorientierte Zusammenfassung von lokalen und entfernten Anweisungen gesehen werden, welche intern auf atomare Befehle abgebildet sind. Trotzdem ist eine gleichwertige Aufteilung in die oben genannten drei Gruppierungen auf selber Abstraktionsebene sinnvoll, da die eng gekoppelten, beidseitigen Befehle nur im Zusammenwirken sinnvoll genutzt werden können. Es macht z.B. keinen Sinn eine existierende, atomare Anweisung zum Dateiöffnen inmitten anderer atomarer Befehle auszuführen, wenn kein Zugriff auf die Datei erfolgt. So ist zusammenfassend zu sagen, dass für die leicht gekoppelten, atomaren und die aus eng gekoppelten, atomaren Einzelanweisungen zusammengesetzten Befehle eigene, voneinander unabhängige Schnittstellenmethoden notwendig sind.

Einseitige vs. verteilte Sitzungsverwaltung

In diesem Zusammenhang bzgl. der Aufteilung auf lokal und entfernt wirkende Anweisungen fällt zugleich eine weitere Entscheidung, nämlich ob die Sitzungsverwaltung einseitig oder verteilt stattfindet. Ein Aufruf des eigentlich entfernt wirkenden Verzeichniswechsels „cd“ kann unter Berücksichtigung der fehlenden Sitzungsorientierung von `CORBA` als lokale Veränderung einer Variablen für Pfadangaben zum entfernten System implementiert werden. Damit wäre „cd“ in einem System mit clientseitiger Verwaltung eine lokal wirkende Anweisung. Wird die Verwaltung verteilt, so ist auch dafür die Nachbildung von sitzungsorientierten Kommunikationsabläufen notwendig. Pfadangaben des betreffenden, entfernten Systems sind dabei dann auch dort abgespeichert, wodurch z.B. „cd“ als remote wirkende Anweisung gilt. Da Dateien ab einer bestimmten Größe nicht mehr mit einem Funktionsaufruf komplett übertragen werden können, bedeutet dies eine Aufsplittung der Übertragung in einzelne Blöcke. Dies hat wiederum zur Folge, dass ein Funktionsaufruf auf den anderen aufsetzt, was die serverseitige Verwaltung von Sitzungen erfordert. Da diese somit generell notwendig sind, kann auch bei der Verwaltung von Umgebungsvariablen, wie z.B. Pfadangaben, eine verteilte Kontrolle mit Sitzungsorientierung vorausgesetzt werden.

Bildung atomarer Kommandoeinheiten: Abfragen und Anordnungen

Aufbauend auf die obige Aufgliederung können fortfahrend für die lokal und entfernt wirkenden atomaren Kommandos zwei Gattungen identifiziert werden: Abfragen und Anordnungen. Abfragen dienen dazu, Informationen über entsprechende Gegebenheiten zu erhalten („ls“ für die Abfrage des Inhalts eines entfernten Verzeichnisses). Sie verändern den Zustand des angesprochenen Systems nicht.

Anordnungen hingegen wirken verändernd auf die angesprochene Umgebung („mkdir“ erzeugt ein entferntes Verzeichnis). Insgesamt bilden damit alle diskutierten Fälle eine elementare Ebene von Befehlen.

Zusammengesetzte Befehle als Ergänzung

Auf diese erste Ebene bauen weitere zusammengesetzte Befehle auf. Zu diesen zählt z.B.“mget“, wodurch mehrere Dateien hintereinander geholt werden können. Dies setzt aber voraus, dass die Dateien beim Client bekannt sind, so dass er sie anschließend einzeln anfordern kann. Somit ist „mget“ ein übergeordneter Befehl, welcher sich aus „ls“ und einer Abfolge von „get“ bildet. Solche Befehle erweitern durch Kombination die Funktionalität ohne die Basisfunktionalität zu verändern.

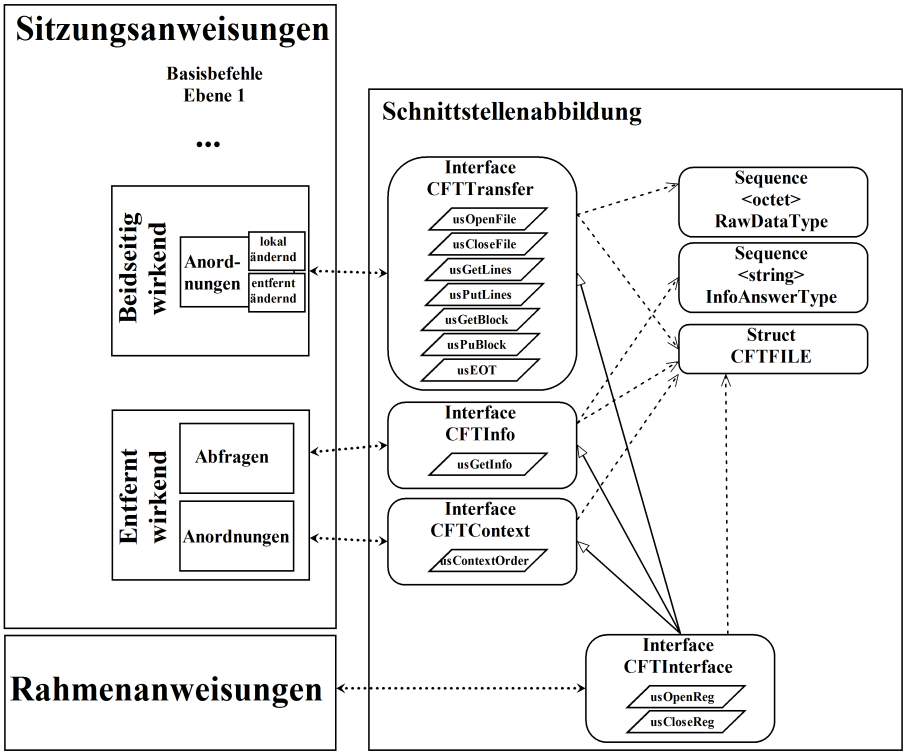


Abbildung 3.8: Schnittstellenabbildung

Fundament bilden die beidseitig wirkenden Befehle

Betrachtet man nun die erkannten Aufteilungen generell bzgl. der Erweiterung um zusätzliche Funktionalität, so wird ersichtlich, dass beidseitig wirkende Basisanweisungen (z.B. „get“) als allgemeines, unveränderliches Fundament gesehen werden können. Alle Ergänzungen erweitern entweder die Grundfunktionalität der lokal oder remote wirkenden Atomaranweisungen oder kombinieren existierende Funktionalität auf höherer Ebene. Dies ist eine wegweisende Erkenntnis für die Abbildung auf eine allgemeine Schnittstelle, da daraus die Erfordernis von starren Schnittstellenmethoden abgeleitet werden kann.

Weiche vs. starre Schnittstellenmethoden

Geeignete Abbildung der Befehle auf Schnittstellenmethoden

Nach den vorangegangenen Diskussionen kann aufgrund der getroffenen Einteilung ein geeignetes Abbildungsmodell auf Methoden einer Schnittstellendefinition stattfinden, welche in `IDL` abgefasst ist (siehe Anhang D auf Seite 227). Betrachtet werden dafür lediglich entfernt wirkende Befehle (vgl. für nachfolgende Betrachtungen Abb. 3.8 auf Seite 60). Alle Methoden liefern dabei generell einen Fehlercode zurück, welcher vom Typ „unsigned short“ ist. Dieser Fehlerco-

de ist eindeutig definiert und erlaubt direkte Rückschlüsse auf die Fehlerursache. Zudem gibt es je nach Methode Übergabe- (an die Methode übergebene Werte; „in“) oder Rückgabeparameter (von der Methode zurückgegebene Werte; „out“) bzw. eine Mischung davon (übergebene Werte, welche verändert zurückgegeben werden können; „inout“).

„CFTInterface“ mit Rahmenanweisungen

Vorweg sind nun erst die Rahmenanweisungen zu betrachten. Sie werden in der Klasse „CFTInterface“ angesiedelt, welche von allen anderen Interfaceklassen Methoden ererbt. Eine Abstraktion der Funktionalität ergibt, dass sie alle auf zwei Grundmethoden abgebildet werden können:

1. **Registrierung einer Sitzung eröffnen („usOpenReg“):**
   Alle für eine Sitzung relevanten Daten werden ausgetauscht, wodurch eine eindeutige Kennung zur Identifikation angelegt wird. Die benötigten Parameter sind:
   (a) ***Übergabe des Nutzernamens:*** „cpUserName“ vom Typ „string“
   (b) ***Übergabe des Nutzerpassworts:*** „cpUserPwd“ vom Typ „string“
   (c) ***Rückgabe der Sitzungsidentifikation:*** „SCFTID“ in Anlehnung an C vom eigens definierten Typ „CFTFILE“; dieser Typ besteht aus einer Nutzeridentifikation „usUserID“ (Typ „unsigned short“), einer Stromidentifikation „usStreamID“ (Typ „unsigned short“) und einem Zeitstempel „ulTimeStamp“ (Typ „unsigned long“), der bei jedem Aufruf zu aktualisieren ist
2. **Registrierung einer Sitzung schließen („usCloseReg“):**
   Die angelegte Identifikation wird verworfen. Benötigt wird dazu als Parameter nur:
   (a) ***Übergabe/Rückgabe der Sitzungsidentifikation:*** „SCFTID“

„CFTTransfer“ als zentrale Basis zur Datenübertragung

Als zentrale Basis der Schnittstelle können die beidseitig wirkenden Befehle zur eigentlichen Datenübertragung angesehen werden. Da sie sich aus mehreren eng gekoppelten, atomaren Einzelaktionen zusammensetzen, bilden sie ein relativ starres Gerüst. Somit können diese atomaren Funktionalitäten auf eigene Schnittstellenmethoden abgebildet werden. Sie sind in Anlehnung an die elementaren Dateifunktionen in der Sprache C folgende:

1. **Öffnen einer Datei („usOpenFile“):**
   Öffnet eine Datei entsprechend des gewünschten Modus und liefert anhand der Sitzungsidentifikation gleichzeitig einen eindeutigen Dateizeiger (d.h. zu einem Sitzungszeitpunkt kann jeweils nur eine Datei offen sein). Die nötigen Parameter sind:
   (a) ***Übergabe des Dateinamens:*** „cpFileName“ vom Typ „string“
   (b) ***Übergabe des Öffnungsmodus:*** „cpMode“ vom Typ „string“; dabei handelt es sich um eine klassische Darstellung vergleichbar zur Sprache C
   (c) ***Übergabe/Rückgabe der Sitzungsidentifikation:*** „SCFTID“
2. **Geöffnete Datei schließen („usCloseFile“):**
   Schließt eine durch die Sitzungsidentifikation zugewiesene Datei. Die Parameter sind:

   (a) ***Übergabe/Rückgabe der Sitzungsidentifikation:*** „SCFTID“

3. **Lesen einer Folge von Textzeichen („usGetLines“):**
   Liest eine angegebene Länge von Textzeichen (Format in `American Standard Code for Information Interchange (ASCII)` / `American National Standards Institute (ANSI)`) aus einer geöffneten Datei und liefert sie an den Aufrufenden. Benötigte Parameter sind:

   (a) ***Rückgabe der gelesenen Zeichenfolge:*** „cppLine“ vom Typ „string“

   (b) ***Übergabe/Rückgabe der gewünschten/tatsächlichen Anzahl von Zeichen:*** „ulpLength“ vom Typ „unsigned long“

   (c) ***Übergabe/Rückgabe der Sitzungsidentifikation:*** „SCFTID“

4. **Schreiben einer Folge von Textzeichen („usPutLines“):**
   Schreibt eine angegebene Länge von Textzeichen (Format in `ASCII` /`ANSI`) in eine geöffnete Datei. Die Parameter sind:

   (a) ***Übergabe der zu schreibenden Zeichenfolge:*** „cpLine“ vom Typ „string“

   (b) ***Übergabe der Anzahl zu schreibender Zeichen:*** „ulLength“ vom Typ „unsigned long“

   (c) ***Übergabe/Rückgabe der Sitzungsidentifikation:*** „SCFTID“

5. **Lesen einer Folge von Binärzeichen („usGetBlock“):**
   Liest eine angegebene Länge von Binärzeichen aus einer geöffneten Datei und liefert sie an den Aufrufenden. Benötigte Parameter sind:

   (a) ***Rückgabe der gelesenen Zeichenfolge:*** „CppBlock“ vom eigens definierten Typ „RawDataType“; dabei handelt es sich um eine Sequenz aus „octet“ (8-Bit-Zeichen)

   (b) ***Übergabe/Rückgabe der Anzahl gewünschter/tatsächlich gelesener Zeichen:*** „ulpLength“ vom Typ „unsigned long“

   (c) ***Übergabe/Rückgabe der Sitzungsidentifikation:*** „SCFTID“

6. **Schreiben einer Folge von Binärzeichen („usPutBlock“):**
   Schreibt eine angegebene Länge von Binärzeichen in eine geöffnete Datei. Die Parameter sind:

   (a) ***Übergabe der zu schreibenden Zeichenfolge:*** „CpBlock“ vom eigens definierten Typ „RawDataType“

   (b) ***Übergabe der Anzahl zu schreibender Zeichen:*** „ulLength“ vom Typ „unsigned long“

   (c) ***Übergabe/Rückgabe der Sitzungsidentifikation:*** „SCFTID“

7. **Überprüfung des Transferendes (= Dateiende; „usEOT“):**
   Kontrolliert, ob alle Zeichen übertragen sind und liefert „wahr“ (= 1) bei kompletter Übertragung und „falsch“ (= 0) sonst. Die Parameter beschränken sich dabei auf folgendes, da die Wahrheitsaussage im Rückgabewert der Methode steckt:

   (a) ***Übergabe/Rückgabe der Sitzungsidentifikation:*** „SCFTID“

Die weiter üblichen Funktionen, wie z.B. das Positionieren des Dateizeigers „seek“, können mittels der gegebenen Methoden nachgebildet werden (z.B. bei „seek“ durch geeignetes Schließen, Öffnen und Lesen der Zeichen bis zur gewünschten Position).

Flexible Erweiterbarkeit für restliche Anweisungen

Das Konzept für die weiteren, atomaren Aktionen sieht eine Erweiterbarkeit des Funktionsumfangs ohne Änderung der Schnittstellendefinition vor. Das ist dadurch gewährleistet, dass die Abbildung auf eine gemeinsame Methode je nach Anweisungsart stattfindet. Die einzelnen Anweisungen selbst werden durch codierte Kennungen charakterisiert.

„CFTInfo“ mit Abfragemethode

Für die Abfragen ist folgende Methode in der Klasse „CFTInfo“ angesiedelt:

1. **Abfragen einer gewünschten Information („usGetInfo“):**
   Führt die durch eine Befehlsidentifikation bestimmte Abfrage aus, wobei gegebene Abfrageparameter berücksichtigt werden und liefert die Antworten zeilenweise sortiert zurück. Die dazu nötigen Parameter sind:
   (a) ***Übergabe der Befehlsidentifikation:*** „usOrderID“ vom Typ „unsigned short“
   (b) ***Übergabe der zu berücksichtigenden Parameter:*** „cpParameterString“ vom Typ „string“
   (c) ***Rückgabe der Antwortzeilen:*** „CpAnswer“ vom eigens definierten Typ „InfoAnswer“; dabei handelt es sich um eine Sequenz aus „string“
   (d) ***Rückgabe der Anzahl der Antwortzeilen:*** „uspNumberOfLines“ vom Typ „unsigned short“
   (e) ***Übergabe/Rückgabe der Sitzungsidentifikation:*** „SCFTID“

„CFTContext“ mit Anordnungsmethode

Für das Absetzen von Anordnungen existiert folgende Methode in der Klasse „CFTContext“:

1. **Absetzen einer kontextverändernden Anweisung („usContextOrder“):**
   Führt den durch eine Befehlsidentifikation bestimmte Abfrage aus, wobei gegebene Abfrageparameter berücksichtigt werden. Der Erfolg der Anweisung kann mittels des allgemeinen Rückgabewerts ersehen werden. Die Parameter sind hierbei:
   (a) ***Übergabe der Befehlsidentifikation:*** „usOrderID“ vom Typ „unsigned short“
   (b) ***Übergabe der zu berücksichtigenden Parameter:*** „cpParameterString“ vom Typ „string“
   (c) ***Übergabe/Rückgabe der Sitzungsidentifikation:*** „SCFTID“

Flexibilität bis hin zum Austausch des Übertragungsmechanismus

Betrachtet man sich die resultierende Schnittstellenabbildung, so ist eine Allgemeinheit der Vorschriften zu erkennen. Diese ist nicht nur beliebig für andere Aufgaben erweiterbar, sondern ermöglicht auch eine Adaption anderer Übertragungsmöglichkeiten (sogar FTP könnte bei Bedarf adaptiert werden). Dazu sind letztendlich die Schnittstellenmethoden entsprechend abzuleiten und nachzubilden.

Auf diese Schnittstelle aufbauend kann nun eine Softwarearchitektur mit Komponentenaufteilung aus Modulen umgesetzt werden. Die in IDL für Stub und Skeleton definierte Schnittstelle wird über entsprechende Compiler in die Sprache C++ übersetzt und bildet eine zentrale Einheit des Umsetzungsmodells.

3.3.3 Die modulare Architektur

Abstraktes Komponentenmodell

Die im vorangegangenen Abschnitt beschriebene Schnittstelle ist eine abstrakte Definition von Zugangsmechanismen zwischen Client und Server. Nach der Definition von Komponenten ist dies eine Grundvoraussetzung für ein abstraktes Komponentenmodell. Somit können auch die im neuen System vorhandenen Komponenten identifiziert werden. Es sind (vgl. Abb. 3.9 auf Seite 64):

- Der **Client** mit seinem Stub, welcher als Service-Zugangspunkt (Service Access Point) zum Server dient.
- Der **Server** mit dem Skeleton, welches die Serverfunktionalität implementiert und dem Client als Gegenstelle dient.
- Die jeweils, elementaren **CORBA -Bestandteilen**, welche automatisch generiert oder als Bibliotheken hinzugebunden werden.
- Und zum Austausch von Objektreferenzen ist schließlich noch ein **Naming Service** zu nutzen, welcher von den CORBA-Implementierungen mitgeliefert wird und als Register für die Zugangspunkte auf die Server dient.

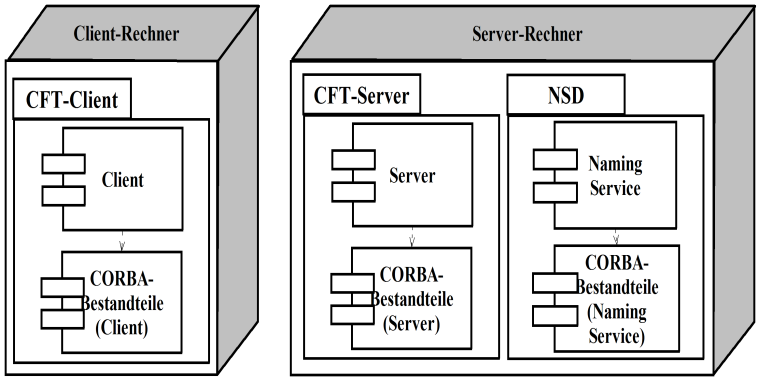


Abbildung 3.9: Komponenten und ihre Verteilung

Verteilung auf Rechnereinheiten

Nach der Festlegung von Komponenten kann auch ihre Verteilung im Gesamtsystem aufgestellt werden. Da eine möglichst exakte Nachbildung von FTP erfolgen soll, gibt es somit einen zentral auf einem Rechner laufenden Server, welcher von diversen, meist auf anderen Rechnern installierten Klienten kontaktiert wird. Eine Besonderheit beim CFT ist es aber, die Objektreferenz zum Serverzugang zu erhalten. Deshalb wird als Festlegung vorgeschrieben, dass auf jedem Server-Rechner auch ein Naming Service Daemon (NSD) angesiedelt sein muss. Damit läuft ein Verbindungsaufbau folgendermaßen ab:

Ablauf eines Verbindungsaufbaus

1. Der Klient erhält die Adresse und den Port des gewünschten Servers
2. Der Klient erstellt eine Verbindung zum Verzeichnisdienst und erfragt mittels einer Kennung aus Dienstname und Dienstart beim Naming Service die Objektreferenz (`IOR`) des Datenservers
3. Der Klient baut mittels der Referenz eine Verbindung zum Datenserver auf

Duplikation von Verzeichnisdiensten

Die angegebene Verteilung bedeutet aber nicht, dass nicht auch ein separat existierender Naming Service auf einem anderen Rechner ebenfalls als Referenzgeber dienen kann. Allerdings wird vorgesehen, dass sich die Server selbstständig nur bei ihrem lokalen Verzeichnisdienst anmelden. Alle weiteren Register werden mittels eines eigenen Klienten gefüllt, welcher eine Liste von Server-Rechnern durchläuft und die Server über einen Befehl auffordert, sich auch auf dem separaten Dienst zu registrieren. Komplette Duplikationen von zentralen Verzeichnisdiensten sind auf diese Art ebenfalls möglich, indem der Duplikator die komplette Liste eines Naming-Dienstes durchläuft und die eingetragenen Server zur weiteren Registrierung auffordert.

Architektur aus Modulen

Die Komponenten selbst sind aus einzelnen Modulen aufgebaut. Sie besitzen fest definierte, reale Schnittstellen und werden von anderen Modulen wie „Black Boxes“ behandelt. Sie ermöglichen es, auf einfache Weise modulbezogene Tests durchzuführen und erlauben die komplette Ersetzung ganzer Funktionsabschnitte. Zusammen betrachtet bilden sie die Architektur von `CFT` (vgl. Abb. 3.10 auf Seite 65)

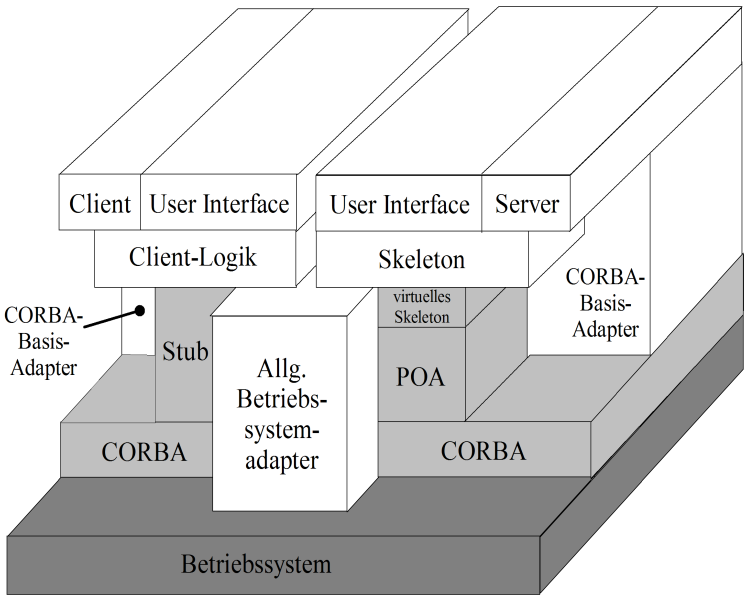


Abbildung 3.10: Die Architektur von CFT

Nachfolgend werden nun die einzelnen Module als Überblick erläutert.

Die CORBA-Module

Anteile der CORBA-Implementierung

Dabei handelt es sich um alle CORBA -gegebenen Anteile. Sie beinhalten die in den letzten Abschnitten beschriebenen Details und werden entweder über Bibliotheken von der CORBA -Implementierung zur Verfügung gestellt oder aber durch den IDL -Compiler aus der IDL -Definition erzeugt. Zu ihnen zählen der ORB -Core als elementare Verteilungsplattform mit dem serverseitigen POA und die generierten Modulteile Interface-Stub und virtuelles Skeleton (eine Art abstrakte Klasse von Schnittstellenmethoden), von dem sich die eigentliche Realisierung ableitet.

Der Middleware-Adapter

Kapselung der Middleware

Der Middleware-Adapter entstand aus der Idee heraus, die eigentliche Middleware zu abstrahieren. Aufgrund der gezeigten Vielzahl von verschiedenen Ansätzen soll damit eine Möglichkeit geschaffen werden, Basisfunktionalität der Zwischenschicht transparent von der Anwendung zu entfernen und über diesen Adapter anzubieten, der als Ausprägung in diesem Fall als CORBA -Basisadapter bezeichnet ist.

Darin wird eine generelle Umsetzung von Fehlercodes aus der Zwischenschicht realisiert. Die weitere Funktionalität besteht hauptsächlich in der Initialisierung und Aktivierung von benötigten Komponententeilen. Dabei kann zwischen allgemein gültigen und client- bzw. serverabhängigen Abläufen unterschieden werden. Beim Adapter für CORBA „CBASE“ existiert damit eine allgemeine Klasse von Methoden, welche für die Umsetzung der CORBA -Argumente aus den Aufrufparametern beim Programmstart sorgen. Des Weiteren werden die Verwaltungsstrukturen der benötigten CORBA -Komponenten (ORB , Naming-Service-Schnittstelle, etc.) angelegt und aktiviert.

Von diesen allgemeinen Teilaufgaben leiten sich die speziellen ab oder werden von ihnen ergänzt. Auf Seite des Servers sind das vor allem das „Rebind“ (eine Art Registrierung beim lokalen Namensdienst) und die Aktivierung des Schleifendurchlaufs zur Anfragebearbeitung. Der Client hingegen implementiert das „Resolve“ (die Abfrage der Registrierung zum Erhalten der Objektreferenz) und die interne konvertierte Weitergabe dieser an die abhängigen Anwendungsteile.

Das Server-Skeleton

Realisierung der angebotenen Schnittstellenfunktionalität

Im Serverskeleton, hier kurz CFT -Skeleton genannt, sitzt im Wesentlichen die Realisierung der durch das Interface angebotenen Funktionalität. Dies sind die Methoden, welche mittels IDL entwickelt wurden. Somit wird hier auch die übergeordnete Abbildung auf das Betriebssystem stattfinden, wodurch eine enge Kopplung zum allgemeinen Betriebssystemadapter vorliegt.

An dieser Stelle sind auch die Befehlskennungen und ihre Fehlerrückgaben definiert. Die während der FTP -Analyse erkannten Befehle werden hierbei umgesetzt und auf Basis der im Projekt standardisierten Betriebssystemfunktionen

implementiert. Zudem ist im Skeleton auch die später beschriebene Sitzungsverwaltung mit Registration und Authentifizierung angesiedelt. Damit ist ersichtlich, dass im Skeleton-Modul die eigentliche Funktionalität des Dienstes steckt.

Der Server und sein Nutzerinterface

Umgebende Hülle im Server

Der Server ist nur noch die das Skeleton umgebende Hülle, so dass es als eigenständiges Programm lauffähig wird. Das bedeutet, dass hier das Hauptprogramm steckt, welches die durch den Middlewareadapter gegebenen Komponenten initialisiert und auf höherer Ebene die Abarbeitungsschleife startet.

Der Server hat damit eigentlich direkt auch kein Nutzerinterface mehr, da der Nutzer nur mittels eines Clients auf den Server zugreifen kann. Allerdings sind in der trotzdem so bezeichneten Nutzerschnittstelle die Übernahme und Weitergabe der Aufrufparameter enthalten.

Die Client-Logik

Arbeitsfluss und zugehörige Logik

Im Client steckt die eigentliche Ablaufsteuerung und damit die Logik für den Arbeitsfluss (Workflow). Die einzelnen Teilaufgaben — das sind im Wesentlichen die Befehle aus der FTP -Analyse — sind spezielle Methoden, welche entweder auf den Client-Stub, den lokalen Betriebssystemadapter oder auf beide gleichzeitig zugreifen. Im rudimentären CFT wird bei der Implementierung noch streng zwischen Client und Server getrennt. Diese Trennung wird später in den Erweiterungen durch Verallgemeinerungen reduziert, so dass für beide gleichermaßen stattfindenden Aktionen nur einmal umgesetzt werden müssen.

Aufgrund der benötigten Zugriffe auf den Stub ist auch klar, dass hier die Referenz darauf angesiedelt ist. Zudem sind hier (ähnlich zum Server) die übergeordneten Aktivierungen der Initialisierungen vorhanden, welche für die Remote-Funktionalität hauptsächlich durch den „Open“-Aufruf erfolgen. Eine Besonderheit im Methodenumfang ist, dass aufgrund der Schnittstellendefinition zwischen Client und Server die Blockgröße für die Übertragung der Nutzdaten beliebig (im Rahmen der Größe von „unsigned long“) variiert werden kann. Somit existieren zusätzlich Methoden zur Manipulation dieser Einheit in Byte-Schritten.

Der Client und sein Nutzerinterface

Kommandozeileninterpreter

Die Schnittstelle zum Nutzer ist ein übergeordnetes Modul und implementiert in erster Näherung einen Kommandozeileninterpreter, wie er von FTP -Klienten bekannt ist. In einer Endlosschleife (bis zu einem Endbefehl) werden Kommandozeileneingaben erst geparst, dann interpretiert und schließlich die Kommandos einer Befehlsidentifikation und zugehörigen Methode zugeordnet. Nach der Initialisierung über die Programmaufrufparameter wird die Schleife einmalig in Gang gesetzt.

Lineare Skripts zur Automatisierung von einfachen Abläufen

Die Interpreterschleife kann durch ihre innere Umsetzung zudem dazu genutzt werden, lineare Programme, welche aus einer Reihung von CFT -Befehlen bestehen, als Skript einzulesen und zu interpretieren. Dadurch wird es möglich, vorerst einfache automatische Abläufe zum Versenden oder Empfangen zu realisieren. Die

Umlenkung der Standardausgabe in eine Datei kann hierbei als Logbuch der Aktionen dienen. Eine spätere Erweiterung der Befehlssätze kann sogar soweit gehen, eine beliebig komplexe Skriptsprache mit bedingten Abläufen als Interpretersprache umzusetzen, welche mehr oder weniger intelligent auf Fehlerzustände aus den dedizierten Rückgabecodes reagiert.

Skriptsprache beliebig komplex erweiterbar

Der allgemeine Betriebssystemadapter

Verbesserung der Kapselung von Systemaufrufen

Obwohl Middlewaresysteme oft als die Betriebssysteme der Zukunft angekündigt werden, gehen viele davon bei der Nutzung von Betriebssystemfunktionen bzgl. Abstraktion nicht weit genug. Sie verdecken zwar die für ihren speziellen Einsatzzweck notwendigen Aufrufe (z.B. Prozessgenerierung und/oder Kommunikation), bieten aber keine allgemeine Schnittstelle zu den Systemaufrufen. Der Anwendungsprogrammierer ist damit selbstverantwortlich, wie abstrakt und unabhängig von Systemspezifika er programmiert. Bei den meisten verteilten Anwendungen mag dies ausreichen, aber schon wenn es um die Manipulation von Verzeichnisbäumen geht oder eine Zeitabfrage mit höherer Auflösung nötig wird, begibt man sich wieder in spezielle Abhängigkeiten. Dies gilt es zu kapseln und zu abstrahieren.

Adapter als Abstraktion vorerst benötigter Systemaufrufe

Deshalb wird in CFT ein allgemeiner Betriebssystemadapter genutzt. In erster Näherung ist dieser ein Zugangspunkt zum Dateisystem und zu weiteren von CFT benötigten Systemaufrufen. Dazu gehören zum einen Konvertierungen der UNIX-ähnlichen Pfade von CFT („/“ wird genutzt, Großbuchstaben werden von Kleinbuchstaben unterschieden) in die betriebsystemspezifischen Pfadangaben (Windows z.B. verwendet „\“ und bewertet Groß- und Kleinbuchstaben gleich). Zum anderen werden hier auch die Systemaufrufe für Zeitangaben im Millisekundenbereich etc. transparent umgesetzt.

Klare Trennung von allgemeinem und spezifischem Code

Durch den Adapter entsteht eine klare Trennung von allgemeingültigem Code und den wenigen systemabhängigen Elementen. Flexibilität und Funktionalität bei Portierungen sind damit zu erwarten. Zudem ist darin der Grundstein für die Erweiterungen hin zu einer verallgemeinerten Zwischenschicht gelegt, welche letztendlich beliebig komplex ausgestattet sein kann.

3.3.4 Besonderheiten von CFT

Versuchsplattform für CORBA-Technologie

`CFT` diente von seiner Konzeption ausgehend zu aller erst einmal als Versuchsplattform zur Erprobung der `CORBA` -Technologie und der Umsetzung der Überlegungen dazu. Hierzu mussten erste Erfahrungen über die benötigten `CORBA` -Komponenten (z.B. Naming Service) gewonnen werden. Gewonnen wurden auch Erfahrungen darüber, die `IDL` -Datentypen richtig zu nutzen (Referenzierungen etc.) und definierte Sequenzen (z.B. für die Übertragung von Binärblöcken) bzw. eingesetzte Strings (z.B. für die Übertragung von Textblöcken) korrekt bzgl. des Speichermanagements zu nutzen.

Stufenweise kontrolliertes Wachstum

Stufenweise wurde so der Prototyp mittels Testprogrammen um Funktionalität erweitert. Von den ersten Versuchen bis hin zur Fertigstellung von `CFT` wurden in einer Art von kontrolliertem Wachstum die ersten Grundlagen für alle folgenden Abschnitte geschaffen. Dazu gehört z.B. auch die Einteilung von modulgruppenabhängigen Fehlercodes, welche bei automatischen Läufen eine Zuordnung der

Fehlerzustände zu den Modulgruppen und damit eine bessere Beurteilung erlauben. Da diese ersten Erfahrungen das weitere Projekt maßgeblich beeinflussten, sollen nachfolgend einige Details daraus explizit näher vorgestellt werden.

CFT-Befehlssatz

Alles in allem wird im ersten Prototyp folgender Befehlssatz angeboten: „ascii“, „binary“, „prompt“, „verbose“, „status“, „exit“, „quit“, „bye“, „dir“, „ls“, „pwd“, „delete“, „mkdir“, „cd“, „rmdir“, „rename“, „append“, „put“, „get“, „mget“, „mdelete“, „mput“, „lcd“, „help“, „?“. Die Rahmenanweisungen zum Verbindungsaufbau werden vorerst beim Programmstart über die Aufrufparameter aktiviert.

Die Erweiterung um Sitzungsorientierung

Fehlende Sitzungsorientierung

Wie bereits erwähnt, ist eine Client-Server-Verbindung in CORBA nicht sitzungsorientiert. Dies bedeutet, dass der Client zwar von der ersten Zuteilung der Ressourcen beim Methodenaufruf bis hin zum Entzug bei der Rückkehr eine exklusive Zuteilung ohne Unterbrechung erhält (Transparenz bei konkurrentem Verhalten ist für Einzelanweisungen gegeben), danach aber alle Einstellungen verloren sind. Jeder Aufruf müsste deshalb alle benötigten Informationen von vorher wieder mitliefern.

Undefinierte Zustände bei Dateizugriffen im Multi-User-Mode

Bezogen auf die Handhabung von Dateien ist dies eher umständlich. Aktive Dateizeiger würden verloren gehen. Zwischenspeichern nutzt nur im Single-User-Betrieb, da sonst jeder Nutzer die selbe Sicht auf die Speicherbereiche hat und somit auch auf die selben Zeiger zugreift, wodurch es zu Überschneidungen und undefinierte Zustände kommt. Im Wesentlichen ergeben sich daraus bereits erste kritische Abschnitte, wie sie in späteren Kapiteln beschrieben werden (vgl. Abschnitt 4.2.2 auf Seite 88. Noch lassen sich aber die Probleme durch die Verwendung einer nutzerbezogenen Sitzungsverwaltung lösen.

Erste Gedanken gelten Security Service

Die ersten Gedanken bzgl. der Umsetzung gelten standardisierten Komponenten von CORBA , wie z.B. dem Security Service. Da hier Konzepte zur Authentifizierung definiert sind, welche zwangsläufig während der Dauer einer Sitzung nutzbar sein müssen, gibt es somit auch eine Form der Sitzungsorientierung. Leider muss aber festgestellt werden, dass zwar einige CORBA -Implementierungen diesen Service unterstützen, jedoch oft nur teilweise umsetzen (TAO z.B. bietet nur Level 0, was der später beschriebenen Nutzung des Secure Socket Layer (SSL) entspricht[33]). Dies liegt wohl daran, dass das OMG -Papier relativ komplex und umfangreich ist. Um keine zu enge Bindung an CORBA zu fordern, wird somit auf einen Einsatz verzichtet.

Rahmenbefehle mit dynamischer Sitzungsverwaltung je Nutzer

Ausgangspunkt sind die Rahmenbefehle zum Registrieren (usOpenReg) und Deregistrieren (usCloseReg) von Sitzungen, wie sie in der IDL des Interfaces definiert sind. Hinter diesen verbirgt sich im Grunde eine dynamische Sitzungsverwaltung je Nutzer in Form einer verketteten Liste, welche etwas komfortabler als indizierte Map ausgelegt werden kann (erleichtert die Suche nach einer bestimmten Registrierung).

[33] [DOC04]

Strukturierte Verwaltungselemente

Die Listenelemente selbst bestehen in erster Linie aus den Einträgen, aus denen die Sitzungsidentifikation erzeugt wird (Zeitstempel, Nutzerkennung und Datenstromkennung). Zudem sind ein Dateizeigerobjekt und der zugehörige Öffnungsmodus enthalten, da jeder Nutzer effektiv nur eine Datei zu einem Zeitpunkt offen halten darf. Um die Verwaltung der Pfade zu erleichtern, sind dazu die Angabe zum Heimatpfad (Startpfad bezogen auf den Nutzer) und zum aktuell gültigen Pfad als Referenzpunkte für alle weiteren Angaben in den Befehlen abgespeichert. So wird am Anfang jedes Befehls aus den Pfadangaben im Befehl (absolut oder relativ) und den Angaben aus dem Verwaltungselement der gültige Pfad erstellt.

Zustandsautomat zur Kontrolle einer Befehlsfolge

Besondere Bedeutung kommt dem letzten Wert des Elements zu, welcher den aktuellen Knoten eines Zustandsautomaten (Finite State Machine) beinhaltet. Über diesen Wert ist es möglich, eine vordefinierte Abfolge von Befehlen einzuhalten (im einfachsten Fall: Datei öffnen, Datei bearbeiten, Datei schließen).

Verwaltungsablauf

Beim Registrieren wird nach entsprechender Überprüfungen des Nutzers ein Eintrag in der Liste erzeugt. Aus diesem wird die Sitzungsidentifikation extrahiert und an den anfordernden Client zurückgegeben. Ein Nutzer kann sich somit mehrfach registrieren, falls er gleichzeitig mehrere Dateien parallel nutzen will. Bei jedem weiteren Methodenaufruf wird vorweg die Sitzungsidentifikation mit den Einträgen in der Verwaltungsliste abgeglichen und der für den Nutzer gültige Zustand wieder hergestellt. Dies ist vergleichbar mit dem Vorgehen beim Prozesswechsel im Betriebssystem. Vor dem endgültigen Rücksprung an den Client, werden die aktuellen Werte wieder in die Liste übernommen und der Zeitstempel gesetzt. Aus dem aktualisierten Listenelement wird die aktualisierte Sitzungsidentifikation generiert und zusammen mit den Rückgaben an den Client zurück übertragen. Beendet dieser schließlich seine Sitzung, wird der Verwaltungseintrag gelöscht und evtl. nötige rudimentäre Aufräumarbeiten erledigt (z.B. Datei schließen).

Zeitstempel zur Verwaltung von Aufräumaktionen

Der Zeitstempel dient zum einen dazu veraltete Aufrufe zu identifizieren und um eine „Deadline“ für verwaiste Sitzungseinträge zu setzen. Ein verwaister Eintrag ist demnach dadurch gegeben, wenn ein Verwaltungseintrag seit mindestens einer Stunde nicht mehr genutzt wurde. Dies ist nötig, da der Server keine aktiven Abfragen an den Client senden kann, um dessen Existenz oder Zustand zu überprüfen. Der gesetzte „Timeout“ setzt somit Aufräumaktionen in Gang. Nachfolgende Methodenaufrufe mittels der gelöschten Sitzungskennung werden zurückgewiesen.

Zeitliche Umsetzung ohne Timer

Um sich den Zugriff auf einen Zeitsignalgeber (z.B. Timing Service) zu sparen (jede weitere externe Komponente erhöht unnötig den Verwaltungsaufwand), werden die Zeitkontrollen vor dem Abgleich der Sitzungsidentifikationen mit den Verwaltungseinträgen direkt während der Methodenaufrufe durchgeführt. Dies reicht aus, da nachfolgende Aktionen keinen Wert darauf legen, ob die Aufräumaktionen exakt zur Deadline stattgefunden haben. Für sie ist nur wichtig, dass vor Beginn der aktuell laufenden Aktion ein wohldefinierter Zustand vorliegt. Voraussetzung hierfür ist, dass keine weiteren Server auf die Daten zugreifen, bzw. dass sich diese mittels eines Masters gegenseitig synchronisieren.

Authentizität und Sicherheit

Einschränkungen durch fehlende Rechteverwaltung je Nutzer

CFT verfügt über keinen Authentifizierungsmechanismus, so dass Passwörter abgefragt werden können. Deshalb läuft der Server auch unter den eingeschränkten Rechten eines dedizierten Nutzers. Er ist damit also nur eingeschränkt für ein echtes Produktionssystem zu verwenden, da jeder Anwender mit CFT -Client auch Zugriff auf einen CFT -Server erhalten und über dessen Rechte verfügen kann. Der Einsatz ist dadurch vorerst einzelnen, ausgewählten Anwendungen im lokalen Netz vorbehalten.

SSLIOP über „Plugable Protocols“ zur Abschwächung der Einschränkungen

Um diese Einschränkung bereits an dieser Stelle zu lockern und Erfahrungen bzgl. verschlüsselter Übertragungskanäle zu sammeln, wurde an dieser Stelle mit der von TAO angebotenen Umsetzung des Security Services (Level 0) experimentiert. Die Möglichkeit bietet sich in der Nutzung von eigenen ESIOP -Definitionen. Durch sie können z.B. Protokolle zur Verschlüsselung integriert werden. TAO verfügt deshalb auch über ein Secure Socket Layer Inter-ORB Protocol (SSLIOP) , welches auf den SSL in der freien Version OpenSSL aufsetzt. Ein weiteres Plus in TAO ist der Ansatz der „Plugable Protocols“[34], welcher es ermöglicht dynamisch beim Programmstart vorübersetzte Protokolle nach dem ESIOP -Standard als eigene Schichten einzugliedern. Dazu wird ein eigenes Framework angeboten, auf welches hier aber nicht weiter eingegangen werden soll. Grundsätzlich werden über diese Methode bereits die Protokolle IIOP , Unix Domain Socket Inter-ORB Protocol (UIOP) , Shared Memory Inter-ORB Protocol (SHMIOP) und SSLIOP angeboten.

Nötige Schritte zur Nutzung von SSLIOP

Folgende Schritte sind für die Nutzung von SSLIOP durchzuführen:

1. OpenSSL übersetzen und installieren
2. Die TAO -Quellen sind entsprechend mit SSLIOP zu übersetzten.
3. Ein Authentifizierungszentrum muss ein Zertifikat erstellen[35]. Dazu wird mittels des Aufrufs „`openssl genrsa -idea -rand <datei1>:<datei2> -out key.perm 1024`“ ein Schlüsselpaar in „key.perm“ unter zufälliger Berücksichtigung des Inhalts der Dateien nach dem Parameter „-rand“ erzeugt. Anschließend kann der eigentliche „Certification Request“ mit „`openssl req -days 1000 -new -x509 -key key.perm -out cert.perm`“ aus den Schlüsseln und einigen eingegebenen Zusatzinformationen generiert werden.
4. Um SSLIOP zu nutzen benötigen sowohl Client als auch Server eine Konfigurationsdatei mit dem Inhalt (unter UNIX):

```
dynamic SSLIOP_Factory Service_Object *
    $TAO_ROOT/orbsvcs/orbsvcs/libTAO_SSLIOP.so:
    _make_TAO_SSLIOP_Protocol_Factory()
    "-SSLAuthenticate NONE
     -SSLPrivateKey PEM:client_key.pem
     -SSLCertificate PEM:client_cert.pem"
    static Resource_Factory
    "-ORBProtocolFactory SSLIOP_Factory"
```

5. Das Anwendungsprogramm muss mit den Parametern „`-ORBSvcConf <Konfigurationsdatei> -o <IOR>`“ gestartet werden (IOR wird am besten direkt angegeben, da sonst auch der Nameserver SSLIOP nutzen muss)

[34] [OCI00] a.a.O. S. 241 ff.
[35] [ENG02]

Möglichkeit zum Nachweis der Authentizität

Mit diesem Trick ist nicht nur eine sichere, verschlüsselte Verbindung gewährleistet, sondern auch die Authentifikation geregelt, da nur der Anwender Zugang erhält, welcher auch über einen gültigen Schlüssel verfügt. Allerdings erhalten alle Anwender nach wie vor nicht wie in FTP ihre Umgebungsbedingungen, sondern die expliziten Nutzerrechte des Serverprozesses.

Variabler Austausch von CORBA-Implementierungen

Grundsätzlich gilt für alle Implementierungen, welche nicht auf einer Interpretersprache, wie z.B. Java, basieren, dass das jeweilige CORBA -Paket erst übersetzt und installiert werden muss. Nach dem geeigneten Setzen der Pfadangaben können die Bibliotheken in die Anwendungsquellcodes eingebunden werden.

Wechsel zwischen standardkonformen Implementierungen bei geringen Abänderungen

Aufgrund von Übersetzungsproblemen der CORBA -Implementierung MICO unter dem Betriebssystem TRU64 UNIX von Hewlett-Packard (ehem. DEC) war es notwendig komplett auf TAO umzuschwenken. Aus der Not heraus geboren, konnte so aber gezeigt werden, dass ein kompletter Wechsel zwischen den beiden Implementierungen möglich ist und dabei nur folgende Punkte tangiert:

1. **IDL-Compiler:** Statt „`idl.exe`“ bei MICO wird „`tao_idl.exe`“ eingesetzt. Erster erzeugt eine Code- und eine Headerdatei, die sowohl für Client als auch für Server genutzt werden. Die TAO-Version erzeugt acht getrennte Dateien (2x3 für den Server und 1x3 für den Client) mit den Endungen „.h“, „.cpp“ und „.i“ (Inlinefunktionen).

2. **Include-Dateien:** Statt den allgemeinen Includes `<CORBA.h>`, `<mico/CosNaming.h>` und `<mico/poa.h>` bei MICO werden die Pendants `<tao/corba.h>`, `<tao/orbsvcs/orbsvcs/CosNamingC.h>` und `<tao/PortableServer/POA.h>` verwendet (ähnliches gilt auch für Dateien weiterer Dienste)

3. **Naming Service Daemon:** Statt „`nsd -ORBIIOPAddr inet:<Rechnername>:<Port>`“ bei MICO wird bei ACE „`Naming_Service -ORBEndpoint iiop://<Rechnername>:<Port> -m 0`“ verwendet. Dies ist aber nicht zwingend erforderlich, da alle standardisierten Services gleichermaßen genutzt werden können.

4. **Abweichungen in den Methoden:** Einige kleinere Änderungen sind noch bei der Deklaration der Methoden selbst zu beachten. TAO erweitert hier um „throw (CORBA::SystemException)“ für Systemausnahmebehandlungen.

5. **Aufrufparameter der Programme:** Die Programmparameter sind leider nicht standardisiert (trotz Ähnlichkeit; ersichtlich beim oben genannten Aufruf des Naming Service Daemon).

Einsatz kommerzieller Produkte möglich

Somit kann jede beliebige, standardkonforme und für C++ geeignete Implementierung mittels weniger Abänderungen zum Einsatz kommen. Dadurch ist die Grundlage für eine evtl. später gewünschte Nutzung von kommerziellen ORB-Produkten gewährleistet. Im Hinblick auf einen allgemeinen Standard ist diesbezüglich aber trotzdem anzumerken, dass es von erheblichem Vorteile wäre, wenn auch die genannten Unterschiede durch Standardisierung behoben werden würden.

3.4 Eine weiterführende Idee: Der Autokonverter

Unterschiedliche Binärdarstellungen in den Dateien bisher nicht unterschieden

In der bisherigen Anordnung werden Dateien ohne weitere Überwachung oder Umwandlung als Text- oder Binärfolge von Bytes übertragen. Ein großes Problem ist es aber, dass die Binärdarstellung maschinenabhängig ist. So konnten z.B. in den Systemen Anfang der 90er die von den Messsystemen der Fundamentalstation erzeugten Daten zwar auf dem Hauptrechner mit dem Betriebsystem VMS abgelegt werden, das Bearbeiten war aber nur mit erheblichem Aufwand durch eigens geschriebene Konverter möglich. Das Problem ist auch bis heute nicht vollständig gelöst, da für jede Binärdatei quasi die Information über ihre Herkunft notwendig ist.

Nutzung der impliziten Konvertierung durch CORBA

Da eine CORBA -Implementierung aber bereits für das sog. „Data Marshalling“, also die Konvertierung sorgt, könnte man durch eine entsprechende Schnittstellenmethode die Konvertierung automatisch integrieren. Voraussetzung ist, dass der Sender den strukturellen Aufbau der Datei kennt, was aber im Allgemeinen ein system-unabhängiges Programmmodul gewährleistet.

Nachfolgend ist ein Fragment einer möglichen Schnittstelle in IDL angegeben:

```
module WDMS
{
    <...>

    // Sequence of unsigned short
    typedef sequence<unsigned short> USSeqType;
    // Sequence of unsigned int
    typedef sequence<unsigned int> UISeqType;
    // Sequence of unsigned long
    typedef sequence<unsigned int> ULSeqType;
    // Sequence of short
    typedef sequence<short> SSeqType;
    <...>

    // Sequence with type-ordering
    typedef sequence<unsigned long> OrderType;

    <...>

    interface CFTTransfer
    {
        <...>

        unsigned short usGetConvertedBlock (inout CFTFILE SCFTID,
                                            out OrderType CppOrder,
                                            inout unsigned long ulpLength,
                                            out USSeqType CppUShort,
                                            out UISeqType CppUInt,
                                            out ULSeqType CppULong,
                                            out USSeqType CppShort,
                                            <...>);
        unsigned short usPutConvertedBlock (inout CFTFILE SCFTID,
                                            in OrderType CppOrder,
                                            in unsigned long ulpLength,
                                            in USSeqType CppUShort,
                                            in UISeqType CppUInt,
                                            in ULSeqType CppULong,
                                            in USSeqType CppShort,
                                            <...>);
        <...>
    };
    <...>
};
```

Ablauf von Zugriffen über den Autokonverter

Wenn der Sender die gelesenen Bytes einem Typ zuordnen kann, so trägt er dessen Wert in die entsprechende Sequenz ein. Die Sequenzen in der Parameterliste der Methoden haben virtuelle Nummern entsprechend ihrer Position im Methodenaufruf (z.B. „0“ für die erste Sequenz, welche im vorliegenden Fall mit „unsigned

short“ belegt ist; „1“ für die zweite Sequenz, welche Werte des Typs „unsigned int“ fasst, usw.). Diese Positionsangabe wird in die Anordnungssequenz (1. Parameter der Methode) eingetragen, worauf der Längenzähler zu erhöhen ist. Wird nun das Gegenstück zum Schreiben der Daten aufgerufen, kann mittels der Anordnungsinformation der entsprechende Wert aus der zugehörigen Sequenz ausgegeben werden. Die Konvertierung von Typinformationen ist an dieser Stelle transparent während der Übertragung automatisch erfolgt.

Diese Interface-Methode wurde vorerst nicht integriert, da in den nächsten Abschnitten hauptsächlich auf `ASCII` /`ANSI` -basierte Daten übergegangen werden soll, so dass generell keine dieser Problematiken mehr auftreten. Eine Umsetzung der Methode würde auch spezialisierte Adapter mit entsprechenden Modulen und Zusatzinformationen über die Binärdatei erfordern.

3.5 Erste Ergebnisse

3.5.1 Systemunabhängigkeit des Codes

Eine Masszahl für Transparenz: Anzahl der anzupassenden Codezeilen bei Portierung

Die ersten gewonnenen Erfahrungen betreffen den Code selbst. Ausgangsgedanke war die Verbesserung der Transparenz insbesondere bezogen auf die Technologie. Eine gute Maßzahl dafür ist, wenn bei wechselnden Betriebssystemen und Rechnertypen der anzupassende Code verschwindend gering ist.

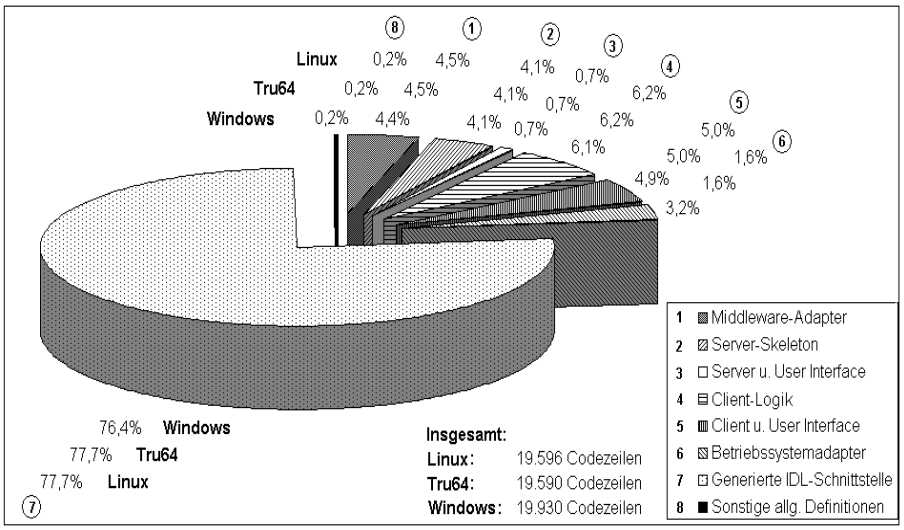


Abbildung 3.11: Verteilung der Codezeilen auf CFT-Module

75% automatisch über IDL generiert

Eine Analyse der Codezeilenzahl (ohne Kommentare) der speziell für CFT erstellten Module bringt hierbei zu Tage, dass sich die Investition in die neue Technologie durchaus lohnt (vgl. Abb. 3.11 auf Seite 74). Fast dreiviertel davon werden durch den IDL-Compiler automatisch und auf jedem System einheitlich aus der Schnittstellendefinition erzeugt. Sie bilden die Schnittstelle zu den in der Middleware umgesetzten Transportverbindungen, was eine erhebliche Erleichterung ist,

wenn man nur an die doch relativ umständliche und systemabhängige Programmierung von Sockets denkt. Das restliche Viertel musste explizit implementiert werden.

Client 10% Server 5%

Für die Kapselung der Middleware durch den CORBA -Adapter entfallen ca. 5% des Codes. Aus der Verteilung zwischen Client (ca. 10%) und Server (ca. 5%) ist bereits ersichtlich, dass sich der Hauptschwerpunkt der Ablauflogik im Client befindet. Erweiterungen diesbezüglich betreffen somit auch hauptsächlich den Client, was zu weiterem Ansteigen des Client-Codes bei nahezu unverändertem Server führen wird.

Bei Portierung anzupassender Betriebssystemadapter mit weniger als 3% Anteil an Codezeilen

Die entscheidende Erweiterung und auch einzige vom System abhängige Einheit, der Betriebssystemadapter, hat wie erhofft nur zwischen ca. 3 % (Windows) und 1,6 % (UNIX) des Codes. Dieses Ungleichgewicht verschiedener Betriebssysteme erklärt sich dadurch, dass z.B. für die CFT -Pfade eine zu UNIX ähnliche Syntax genutzt wird, so dass unter Windows erst Umwandlungen stattfinden müssen.

Transparenz des Codes gegeben

Somit ist zusammenfassend festzustellen, dass nahezu der komplette Code unverändert auf unterschiedlichen Maschinen zum Einsatz kommen kann. Lediglich ein kleiner Prozentsatz ist anzupassen. Natürlich erkauft man sich diese Freiheit mit einer gewissen Abhängigkeit von der Middleware und ihrem Hersteller. Da diese aber im Gegensatz zu Betriebssystemen vereinheitlicht als Standard vorliegt, ist ein Wechsel (wie in Abschnitt 3.3.4 auf Seite 72 gezeigt) nahezu transparent möglich. Restliche Abhängigkeiten werden zudem dadurch vermieden, dass jeweils nur Hauptbestandteile der Middleware zum Einsatz kommen.

3.5.2 Eine Beispielanwendung im GPS-Dienst

Einsatz als GPS-Sammler im LAN

Ein möglicher, zukünftiger Einsatzzweck des Prototyps ist das Sammeln von GPS -Daten der weltweit verteilten Permanentstationen, welche von der Fundamentalstation Wettzell aus betreut werden. Schon nach Abschluss des beschriebenen ersten Meilensteins ist eine Verwendung möglich. Diese beschränkt sich vorerst auf das lokale Netz der Station, wo parallel zum herkömmlichen Produktionssystem mit FTP eine Teststrecke zwischen der GPS -Permanentstation „WTZT“ zu einem Beispielserver mittels CFT aufgesetzt wurde (vgl. Abb. 3.12 auf Seite 76).

Übertragung von RINEX-Daten

Eine GPS -Station besteht aus einer Antenne, einem Receiver und einem Mess-PC (mit Windows NT 4.0) zum Aufzeichnen der Messungen. Jede der Stationen des sog. klassischen Netzes (parallel wird auch ein Echtzeitnetz betrieben, in dem die Datensätze kontinuierlich übertragen werden) senden stündlich ihre Messdaten als Stundendateien und einmal am Tag eine Zusammenfassung als Tagesdatei. Sie sind eigentlich im textbasierten RINEX (internationaler Standard zur Repräsentation von GPS -Daten) strukturiert. Zur Verkürzung der Übertragungszeiten werden sie aber gepackt und sind somit als Binärdaten zu behandeln. Zuvor werden mittels entsprechender, kommerzieller Softwarekomponenten über Perlskripte aus den proprietären Receiverdatensätzen standardisierte, RINEX -konforme Dateien erzeugt.

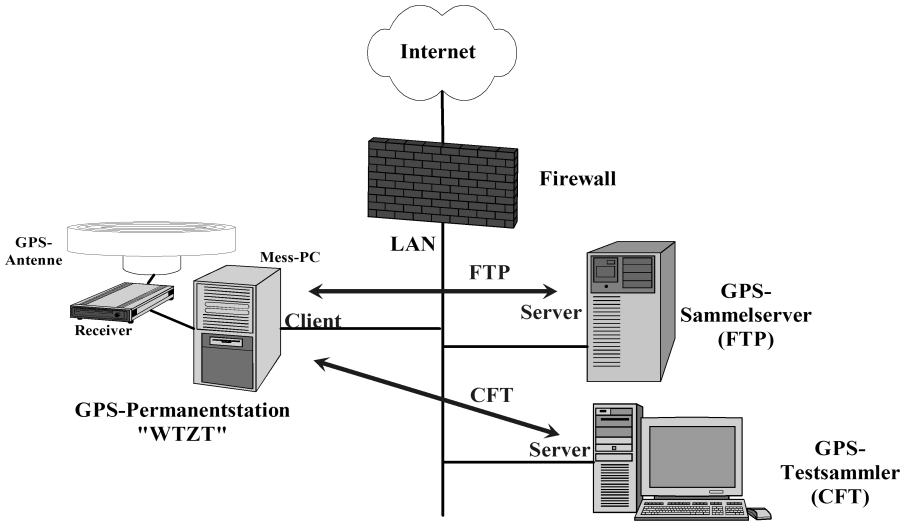


Abbildung 3.12: CFT-Beispielanwendung im GPS-Dienst

Produktionssystem basierend auf Client-Server-Modell

Jede dieser Stationen ist dabei über unterschiedliche Wege zu erreichen (Internet, direkte Modemverbindung, Satellitenlink, etc.) und nutzt unterschiedliche Übertragungsverfahren (hauptsächlich `FTP` und `Simple Mail Transfer Protocol (SMTP)`). Der Großteil beinhaltet einen Datenklienten, welcher sich in den vorgegebenen Zeitabschnitten bei einem Sammelserver in der Fundamentalstation Wettzell anmeldet und dorthin die Daten überspielt. Eine anschließende Qualitätskontrolle dient zur Überwachung der Stationen und ihrer Daten. Werden Fehlzustände erreicht oder vorgegebene Zeiten überschritten, so erfolgt eine Fehlermeldung per Email an die Verantwortlichen, welche manuell in das System eingreifen.

Einige der Messstationen befinden sich im lokalen Netz der Fundamentalstation. Ein erster, lokaler Versuch sollte deshalb zeigen, dass der entwickelte Dateitransfer `CFT` ohne große Änderungen im bestehenden System integriert werden kann, der Ablauf transparent zu den bisherigen Abläufen stattfindet und die erwarteten Vorteile durch die Middleware real erkennbar werden.

Integration ähnlich zu Produktionssystem problemlos

Die Integration in das bestehende System ist problemlos gegeben. Es muss lediglich der vorübersetzte `CFT` -Client auf den Mess-PC integriert werden. Dieser Client wird mittels des eigenen, linearen Skriptmechanismus angesteuert und über einen, dem Produktionssystem („ncftp“-Server) ähnlichen Startaufruf aktiviert. Die Umlenkung der Ausgabe liefert zugleich eine entsprechende Protokollierung. Die Abläufe zwischen herkömmlichem Verfahren und Prototyp sind vergleichbar. Der entscheidende Vorteil ergibt sich aber aus der Tatsache, dass seit einiger Zeit versucht wird, Linux-Rechner als Messrechner auszustatten und somit das selbe Programm ohne weitere, grundsätzliche Änderungen aber in beliebig komplexer Form (z.B. bzgl. Fehlerüberwachung) auch dort zum Einsatz kommen kann.

Langzeittest über mehr als 70 Tage

Die Erwartungen werden erfüllt. Der Testaufbau konnte über mehr als 70 Tage ohne nennenswerte, eigenverschuldete Unterbrechungen kontinuierlich betrieben werden.

3.5.3 Die Transferleistung anhand vergleichender Messungen in einer Testumgebung

Messung von Übertragungsraten zwischen zwei äquivalenten PCs

Eine Bewertung der Leistungsfähigkeit ist unter anderem die Messung von Übertragungszeiten und resultierenden Übertragungsraten. Dazu wurde in einer Teststellung mit zwei äquivalenten PC (Dell Optiplex GX200: Pentium III, 927 MHz, 256 KB Cache), welche über einen Hub verbunden und ans LAN (10 MBit/Sek) gekoppelt waren, mehrere Messreihen aufgezeichnet. Auf den Rechnern waren sowohl Windows 2000 als auch Suse Linux 7.2 installiert, so dass ein Vergleich zwischen den durch die Middleware abstrahierten Betriebssystemen möglich ist.

Durchführung mit abgespecktem ECFT über verschiedene Messreihen

Die Messung selbst wurde mit dem im nächsten Kapitel beschriebenen Extended CORBA File Transfer (ECFT) durchgeführt, da hier aufgrund der internen Architektur und des umfangreichen Funktionsumfangs ein besserer Vergleich möglich ist. Diese erweiterte Version wurde allerdings so eingestellt, dass sie dem eigentlichen CFT in der internen Bearbeitung ähnlich ist (z.B. Deaktivierung der automatischen Typerkennung). Durchgeführt wurden jeweils ca. 30 Durchläufe von vorgefertigten Skripten, welche jeweils mit steigender Blockgröße (10^2, ... , 10^6 Bytes) unterschiedliche Dateigrößen (10^3, ... , 10^8 Bits) sowohl als Binär- als auch als ASCII /ANSI -Übertragung holten (get) und wieder schrieben (put). Ergebnis dieser Durchläufe sind Mittelwert- und Standardabweichungsmatrizen einer grobgranularen Annäherung (evtl. existierende Sprungstellen und Zwischenplateaus bei kleineren, regelmäßigen Deltaveränderungen bleiben unberücksichtigt), welche als Graphen in Microsoft Excel dargestellt sind.

Charakteristika anhand ausgewählter Darstellungen

Nachfolgend werden charakteristische Erkenntnisse anhand ausgewählter Darstellungen aufgezeigt. Der komplette Satz an Darstellungen von Messergebnissen befindet sich im Anhang E auf Seite 231. Hervorzuheben ist, dass im Internet nahezu keine Angaben zu einer Übertragung von großen Datenmengen mit dem rudimentären TAO gefunden werden konnten. Teilweise wurde zwar ein brauchbarer Ansatz durchgeführt[36], allerdings werden verschiedene Rechenertypen und Betriebsysteme in den Messungen gemischt und Probleme mit TAO durch damals existierende Bugs genannt, weshalb nur MICO zum Einsatz kam. Andere Angaben befassen sich hauptsächlich mit spezifischen Internas von TAO , z.B. bzgl. Load-Balancing[37].

Große Leistungsunterschiede zwischen Linux und Windows

Ein Vergleich der Übertragungsraten zwischen Linux und Windows (vgl. Abb. 3.13 auf Seite 78) zeigt, dass der exponentielle Anstieg bei steigenden Dateigrößen in Linux wesentlich kürzer ist. Dies liegt daran, dass unter Linux auch bei kleinen Dateien bessere Transferraten erzielt werden als bei Windows 2000. Zudem tritt die Sättigung als Annäherung an den maximalen Idealwert etwas früher ein. Stärker ist der Effekt bei einem Anstieg der Blockgrößen. Während Linux jeweils bereits bei einer Blockgröße von 10^4 Byte in die Sättigung läuft, geschieht dies unter Windows 2000 erst bei 10^5 Bytes. Durch die längeren Anstiegsphasen in Windows ergibt sich somit auch im Graphen eine wesentlich kleinere Sättigungseintrittsfläche, ab der die Übertragungsleistung im Wesentlichen keine großen Steigerungen mehr erfährt. Bei Windows wäre dies in etwa die Querschnittsfläche bei 4,5 MBit/Sek. Unter Linux liegt dieser Querschnitt bei etwa 6,0 MBit/Sek. Daraus erkennt man auch, dass die absoluten Leistungsunterschiede zwischen den Betriebssystemen er-

[36] vgl. dazu [HOLM04]
[37] vgl. [OTH04]

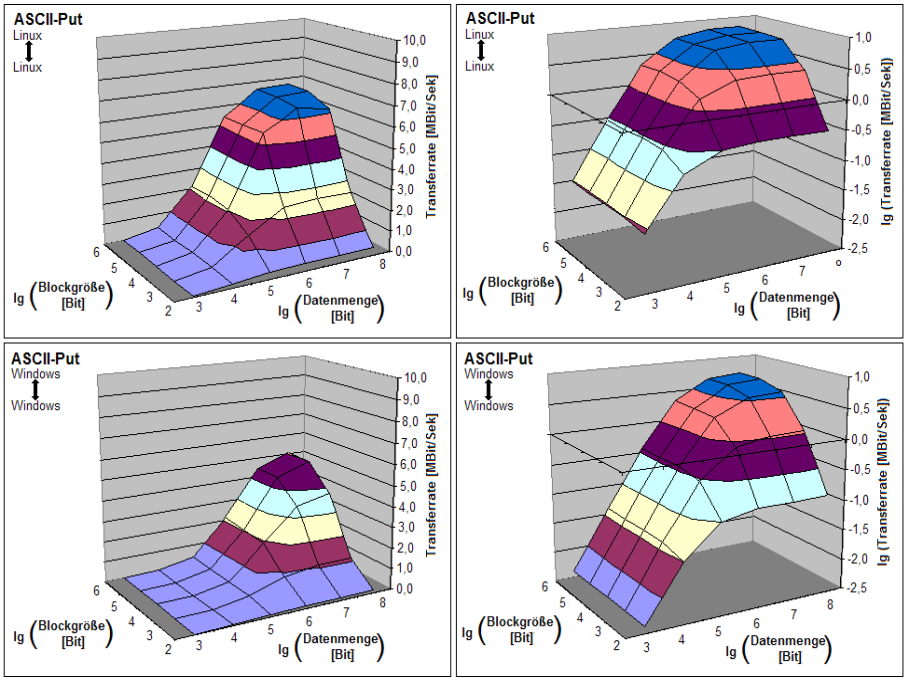


Abbildung 3.13: Transferraten beim ASCII-Put unter Linux und Windows

heblich sind. Während Linux einen Spitzenwert von 6,7 MBit/Sek erreicht, kommt derselbe Anwendungscode auf derselben Maschine unter Windows nur auf knapp 4,9 MBit/Sek.

Mittlerer Fehler zeigt ähnliche Charaktersitika ist aber absolut betrachtet bei Windows geringer

Eine zusätzliche Betrachtung der absoluten Schwankungen der Transferrate in der Stichprobe verdeutlicht, dass diese bei Übertragungen von großen Dateien und mit großen Blöcken am geringsten werden. Dies ist sowohl unter Windows 2000 als auch unter Linux gegeben. Ein deutlicheres Bild zeichnet sich bei einer Analyse der Standardabweichung (`Root Mean Square Error (RMS)`) ab (vgl. Abb. 3.14 auf Seite 79). Ein Vergleich der Betriebssysteme zeigt, dass die Kurvenformen des mittleren, absoluten Fehlers vergleichbare Charakteristika aufweisen, in denen sich wahrscheinlich die Eigenschaften des Ethernets nebst zugehörigem Protokollstack widerspiegeln. Verschlechterungen sind hier im oberen Mittelfeld der Dateigrößen und zugleich großen Transferblöcken zu verzeichnen. Allerdings sind auch hier in den Absolutwerten große Unterschiede zwischen den Systemen zu verzeichnen. Linux weist hier einen `RMS` auf, welcher mehr als doppelt so groß ist, wie bei Windows. Die Störeinflßse auf dem physikalischen Netz können hierbei zwar als ähnlich angenommen werden, jedoch ist Linux aufgrund seiner höheren Übertragungseffektivität auch stärker an Netzwerkschwankungen gekoppelt und reagiert damit darauf wesentlich sensitiver. Dies verursacht einen höheren `RMS` , da das Gesamtsystem dadurch schneller auf solche Einflüsse reagiert. Da aber die Stichprobe mit ca. 30 Werten relativ gering ist, kann daraus kein wirkliches Qualitätsmerkmal abgeleitet werden. Eine ausgedehntere Messung kann jedoch aufgrund des erheblichen Aufwands als Nebenprodukt der Entwicklung des Prototyps nicht durchgeführt werden.

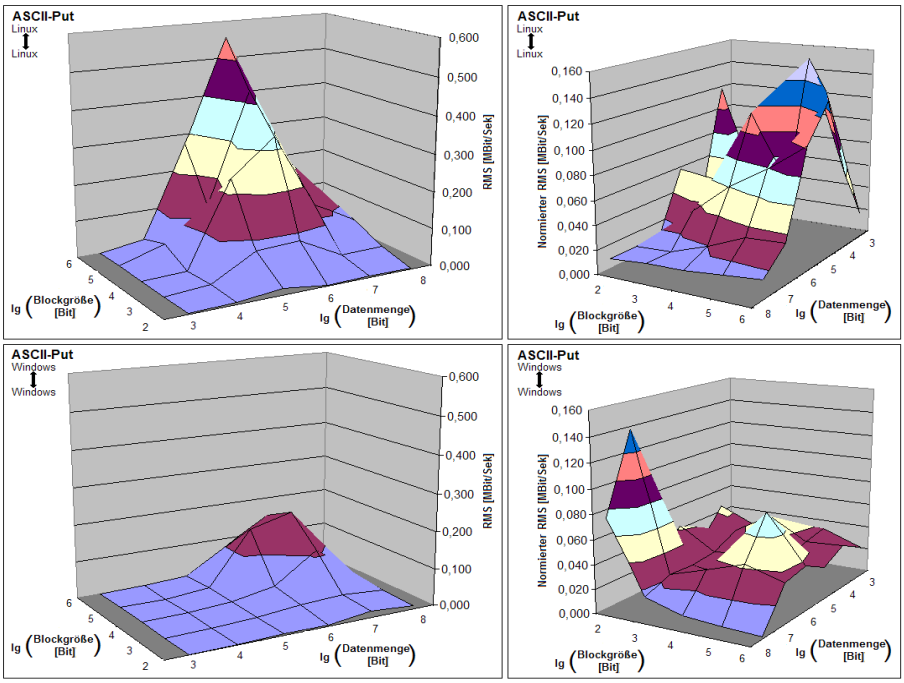


Abbildung 3.14: RMS bei Messreihen zum ASCII-Put unter Linux und Windows

Normierung zeigt, dass Vergleiche nur mit großen Übertragungseinheiten sinnvoll

Normiert man den Mittelwert der Transferrate auf 1, so wird im Vergleich erkennbar, dass die maximalen Abweichungen im mittleren Datei- und Blockbereich auftreten. Charakteristika und Systemspezifika sind allerdings nicht mehr ersichtlich. Eines kann aber festgestellt werden, dass der mittlere Fehler bei kleinen oder bei großen Übertragungseinheiten (Dateigröße, Blockgröße) die kleinsten, relativen Abweichungen annimmt. Das heißt, dass dort die Störeinflüsse entweder noch nicht oder im Mittel nicht mehr greifen. Im übertragenen Sinne auf Messungen mit verschiedenen Transfermedien angewandt, bedeutet das, dass entweder eine kurze Transferzeit oder eine ausreichend lange Zeitspanne abgedeckt werden muss, um aussagekräftige Ergebnisse auch bei kleinen Stichproben zu erzielen. Qualitative Vergleiche der Übertragungsleistungen sind wohl mit großen Dateien bei entsprechend großen Übertragungsblöcken am besten geeignet (zumindest im LAN), da hier die geringsten Abweichungen bei hoher Gesamteffizienz auftreten.

Leistungseinordnung im Vergleich zu FTP

Zur besseren Einordnung der Leistung in herkömmliche Systeme ist es sinnvoll, das gängigste Dateiübertragungsprotokoll FTP als Maßstab zu nehmen (vgl. dazu 3.15 auf Seite 80). Es wird dabei derselbe Versuchsaufbau genutzt. Server unter Linux ist wu-ftp und unter Windows die inetd-Variante der Suite Hummingbird Connectivity (Exceed). Unter Windows wird sowohl der Windows-Client als auch der in den GPS -Stationen genutzte ncftp-Client gemessen, während unter Linux nur die mitgelieferte Variante betrachtet wird. Da hier keine beliebigen Blockgrößen einstellbar sind, ergibt sich ein linearer Verlauf über unterschiedliche Dateigrößen.

CFT-Abbildung auf Windows schlechter

Festzustellen ist, dass bei den Transferraten auch hier Linux die idealere Kurvenform aufweist, welche vergleichbar zu CFT ist. Der generelle Verlauf und der Eintritt in den Sättigungsbereich sind damit betriebsystemspezifisch zu bewerten. Windows zeigt zudem einen stark clientabhängigen Verlauf, so dass wohl zusätz-

lich einige Einflüsse aus der Implementierung eine Rolle spielen. Generell tendiert aber auch Windows zum Graphen unter Linux-Bedingungen. Daraus kann geschlossen werden, dass die Einbussen im CORBA File Transfer unter Windows 2000 aus der schlechteren Implementierung herrühren und wahrscheinlich nicht von einem weniger effektiv nutzbaren Protokollstack. Die Ursache kann eine schlechtere Abbildung durch den Microsoft Compiler oder einfach eine qualitativ schlechtere Umsetzung der `CORBA` -Implementierung unter Windows sein. Darauf, dass letzteres wohl als Ursache am wahrscheinlichsten ist und z.B. nicht die anfänglich im Windows-Client vermutete Nutzung von unbekannten Zugangspunkte zum Protokollstack, deutet auch, dass ncFTP beim Empfangen von Dateien leistungsstärker ist als der Windows-Client, jedoch beim Senden ein um ein Vielfaches schlechteres Verhalten aufzeigt. Das bedeutet, dass hier vermutlich auch Implementationsunterschiede vorliegen. Letztendlich kann aber über die wahre Ursache keine direkte Aussage getroffen werden.

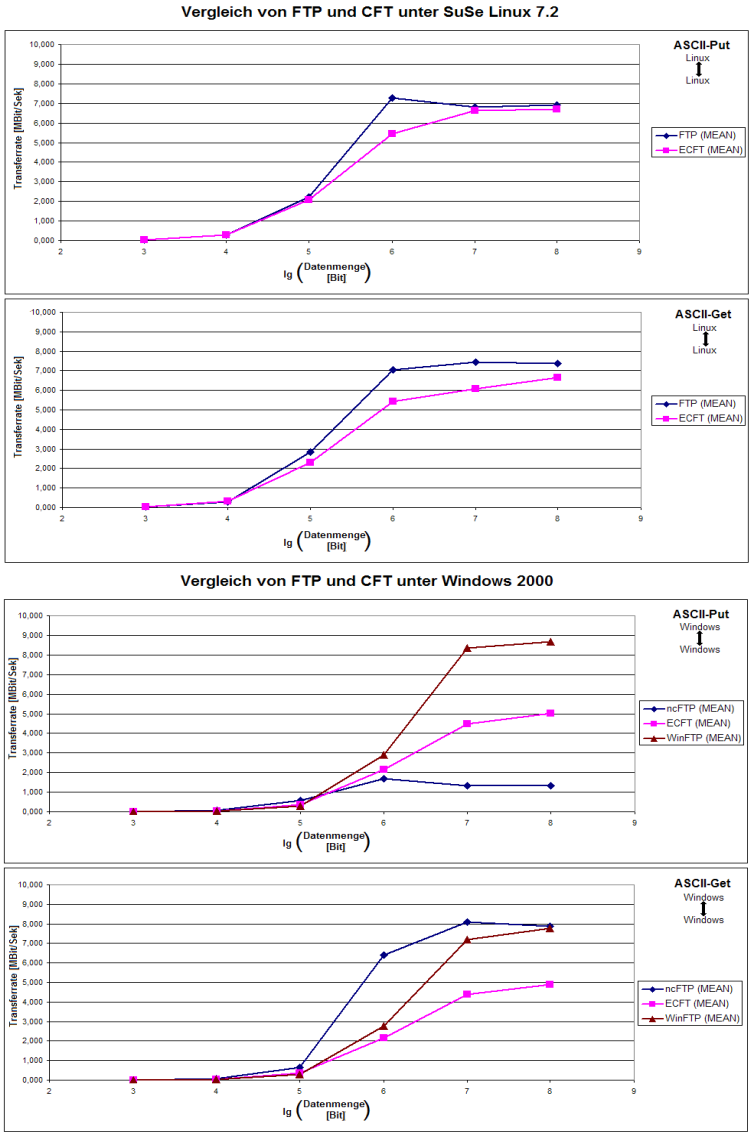


Abbildung 3.15: Vergleich der Übertragungsraten von FTP und CFT unter Linux und Windows

Betrachtet man die Standardabweichungen, so zeichnen sich zwar ähnliche Bilder wie unter CFT ab, sie sind aber durch viele zahlreiche Spitzen entstellt. Daraus ist zu schließen, dass die Stichproben mit je 10 Werten deutlich zu gering sind. Da die Messungen jedoch nur als Zusatz zur Projektstudie erstellt wurden, konnten Verbesserungen hier aus Zeitgründen nicht weiter verfolgt werden. Eines lässt sich aber trotzdem mutmaßen, dass der eigentliche Kurvenverlauf (bei kleinen und großen Dateien geringe mittlere Fehler, im Mittelfeld maximale Störeinflüsse) wohl vom Übertragungsmedium und den Protokollstacks herrührt.

Stichprobe zu klein, jedoch beste Ergebnisse bei großen Übertragungseinheiten bereits sichtbar

Abschließend kann damit festgestellt werden, dass eine Abstrahierung durch Middleware nur innerhalb der Programmierlogik gegeben ist. Die Leistung bleibt weiterhin entscheidend von zwei Dingen abhängig: der Effektivität innerhalb des Betriebssystems und der effektiven Umsetzung der Middleware als Abbildungsschicht auf das Betriebssystem. Somit ist man trotz aller Transparenz letztendlich bezogen auf Leistungsmerkmale wieder von einzelnen, individuellen Eigenschaften abhängig. Dadurch wird der Programmiervorteil bei einer Nutzung von Middleware nicht geschmälert. Allerdings kann man nicht einfach davon ausgehen, dass dieselbe Anwendung unter verschiedenen Systemen dieselben Qualitätsmerkmale aufweist.

Bewertung:
Abstrahierung bezieht sich nur auf Programmierlogik und nicht auf alle Teile der Implementierung, wie z.B. die Transferleistung

3.6 Zusammenfassung

In diesem Kapitel wurde aufgrund der Idee, eine generelle Vereinheitlichung und Abstraktion von allgemeinen Heterogenitäten zu schaffen, eine middlewarebasierte Kommunikationsschicht geschaffen. Als Middleware wurde nach dem Abwägen verschiedener Lösungsansätze CORBA ausgewählt, deren Internas eine genauere Untersuchung erfuhren. Darauf aufsetzend wurde als Kommunikationsbasis der CORBA File Transfer mit einer einfachen Schnittstelle, einer gut strukturierten, modularen Architektur und sonst in CORBA nicht existierenden Erweiterungen (Sitzungsorientierung etc.) entwickelt. Die Qualität der Umsetzung kann anhand der Codezeilen, eines Langzeittests und von Übertragungsraten bestätigt werden. Allerdings kam hierbei zum Vorschein, dass sich systemgegebene Eigenschaften als Qualitätsmerkmale nicht komplett verdecken lassen.

Da der gewählte Ansatz viele Vorteile in der Programmierung hat und die gemessenen Werte zufriedenstellend sind, wird der geschaffene Prototyp weiter ausgebaut. Mit der Software wurde ein auf Middleware basierter Dateiübertragungsmechanismus geschaffen, der dadurch flexibler und universeller sein sollte, als herkömmliche Systeme.

Lösungsansatz wird weiter verfolgt

Kapitel 4

Die Verbesserung der Dienstgüte

Schwerpunkt des Kapitels:

Das nachfolgende Kapitel beschäftigt sich mit dem Ausbau der geschaffenen Kommunikationsbasis zu einem brauchbaren Prototyp mit entsprechender Dienstgüte (`Quality of Service (QoS)`), da herkömmliche Mechanismen hierin oft einen Mangel aufweisen. Über drei Stufen soll ein annehmbares Maß an Qualität geschaffen werden. Als erstes wird das Softwaredesign so konstruiert, dass Client und Server auch weiterhin zum Großteil aus denselben Modulen aufgebaut sind, welche hauptsächlich den Betriebssystemzugang realisieren. Zudem sollen im Client beliebig viele Kombinationen von lokalen und entfernten Ressourcen über ein geeignetes „Multi-Tier“-Konzept realisierbar sein. Zur Vermeidung von übergeordneten, kritischen Abschnitten, wie sie bei gleichzeitigem Dateizugriff durch mehrere Nutzer auftreten, wird in einem nächsten Schritt ein geeignetes Sichtenmodell mit Versionssicherung entwickelt, welches die Aktionen in eine logische Ordnung bringt und jedem Nutzer die Sicht auf die Daten gewährt, die er zu Beginn einer Aktion hatte. Im letzten Schritt werden Systemausfälle durch geeignete Wiederanlaufszenarien mit Checkpointing- und Logging-Mechanismen integriert. Der resultierende „Extended CORBA File Transfer (ECFT)“ verfügt zu allen diesen Dienstgütemerkmalen zusätzlich auch über eine Nutzerauthentifizierung und -autorisierung sowie über einen umfangreichen Satz an Nutzerfunktionalität.

4.1 Ausgangsüberlegung

Ausgangspunkt ist das entwickelte Kommunikationssystem auf CORBA -Basis. Dieses lässt sich beliebig um Anteile erweitern, welche in anderen Übertragungsverfahren fehlen und oft für automatische Anlagen wünschenswert wären.

Wunsch nach flexibler Kombination von Schnittstellen

Dazu zählt eine flexible Kombination von mehreren entfernten und lokalen Schnittstellen in einem Client. Klassische FTP -Clients z.B. verbinden sich mit einem Server und agieren mit diesem im Dialog. Zur Realisierung gleichzeitiger, paralleler Automatisationsstrecken ist jedoch eine Art von „Multicast“ zu mehreren Servern wünschenswert. Deshalb ist in einer ersten Annäherung eine flexible Schnittstellenverwaltung umzusetzen.

Reproduzierbares Systemverhalten und aussagekräftige Fehlercodes

Im klassischen FTP kommt ein weiteres Problem hinzu. Dort ist es möglich, dass Fehlzustände zu Abbrüchen führen. Diese sind meist nicht mehr nachvollziehbar, da nur ungenügende Informationen zur Fehleranalyse vorliegen. Es sind ein reproduzierbares Verhalten des Programms selbst (eine Signifikanz in den Systembedingungen ist ersichtlich) und aussagekräftige Fehlercodes von Nöten. Außerdem ist dafür zu sorgen, dass Fehler nicht zu unerwünschten Zwischenzuständen führen (z.B. fragmentierte Dateien).

Konsistenter Datenzugang

Ein Qualitätsanspruch wäre deshalb auch, dass sich jeder weitere Arbeitsschritt, bevor er ausgeführt wird, darauf verlassen kann, dass sein vorhergehender komplett ausgeführt wurde. Jeder Arbeitsschritt sollte zudem eine konsistente Sicht auf die Daten haben, welche gegenseitige Konkurrenzsituationen vermeidet und seine Abläufe in einen zeitlichen Rahmen einer logischen Ordnung stellt. Dabei wäre es von Vorteil, wenn sich mehrere Konkurrenten nicht gegenseitig blockieren, wie es bei einer Nutzung von Sperrmechanismen (z.B. bei Web-based Distributed Authoring and Versioning (WebDAV)) der Fall ist. Insgesamt sollte somit eine zustandsbasierte Datenübertragung entwickelt werden, welche sogar in gängigen Netzdateisystemen, wie z.B. NFS , fehlt.

Recovery anhand von Checkpoints

Aufsetzend auf konsistenten Zustandsfolgen, welche sich durch parallele Verknüpfungen zwischen den Nutzern auszeichnen, sollte es damit auch möglich sein, an den für den Abarbeitungsprozess markanten Stellen Rücksprungmarkierungen (sog. Checkpoints) zu definieren. Diese Kontrollpunkte wären eine ideale Ausgangsbasis für ein gezieltes Wiederanlaufszenarium nach einem systembedingten Crash. Werden zusätzlich Zustandsübergänge in Form von Semi-Zuständen in Loggingdateien persistent zu den Zuständen abgelegt, ist eine gezielte Reaktion nach einem Absturz möglich. Herkömmliche Systeme fahren nach einem Wiederanlauf nicht mit vor dem Absturz durchgeführten Aktionen fort und sind damit auch nicht in der Lage, Aufräumarbeiten an ihrem Datenbestand durchzuführen. Nur in einem DBMS sind solche Mechanismen integriert. Da aber nicht alle Daten sinnvoll in Datenbanken abzubilden sind, wäre gerade auch bei der Handhabung von Dateien ähnliches wünschenswert. Schließlich sollte man die Daten einer Einrichtung, wie z.B. der Fundamentalstation Wettzell, als durchaus wertvoll und wichtig ansehen.

Da der bisherige Prototyp auch noch nicht über einen geeigneten Authentifizierungsmechanismus verfügt, der es erlaubt auch Autorisierungen, die durch das Betriebssystem auferlegt werden, für verschiedene Nutzer zu verwalten, ist auch auf diesem Gebiet eine Qualitätsanpassung nötig. Da herkömmliche Systeme darüber verfügen, muss zumindest ein ähnlicher Stand erreicht werden. Gleiches gilt auch für den möglichen Befehlsumfang, der durchaus mit Zusatzdiensten aufwarten darf.

Authentifizierung und Autorisierung

Ein weiteres Qualitätsmerkmal ist die Möglichkeit zur Verfolgung und Aufklärung von internen Fehlzuständen im Programm. Einige herkömmliche Systeme verfügen deshalb über einen Loggingmechanismus auch für diesen Bereich, welcher sich durch regelmäßige Ausgaben mit entsprechenden Dringlichkeitsprioritäten auszeichnet. Ähnliches sollte damit auch im Prototyp umgesetzt sein.

Integration eines Logbuchs für Aktionenfolgen

Alles in allem sind es wünschenswerte Eigenschaften eines praktikablen und funktionellen Datenhaltungssystem für Dateien. Deshalb werden nachfolgend Ideen und Umsetzungen zu den einzelnen Punkten dargelegt. Ähnlich zum vorherigen Kapitel soll auch der erweiterte Prototyp anschließend erneut bewertet werden.

Lösungen in diesem Bereich verbessern die im vorherigen Kapitel entworfene Kommunikationsbasis und haben somit ähnlichen Stellenwert für die Fundamentalstation Wettzell bzw. gliedern sich in ähnlicher Form auch in Bereiche der Geodäsie und der Informatik ein. Zu erwähnen hierzu ist im Speziellen, dass die Ideen und entworfenen Module unabhängig von der CORBA-Kommunikationsbasis nutzbar sind. Damit können sie auch in andere Entwicklungen integriert werden, was einen zusätzlich Verallgemeinerungswert aufweist.

Bedeutung für die Disziplinen: ähnlich wie beim vorhergehenden CFT, jedoch auch unabhängig von CORBA nutzbar

4.2 Theoretische Grundlagen

Die Verbesserung von Qualitätsmerkmalen führt im Allgemeinen zu einer Erhöhung der Dienstgüte.

Begriffsdefinition DEF4.1 „Dienstgüte“ nach [LEX04]*[1]*:
Dienstgüte (Quality of Service) ist im Bereich der Telekommunikation die Gesamtheit aller Qualitätsmerkmale, welche das ordnungsgemäße Funktionieren eines Nachrichtennetzes bezeichnen.

Um Dienstgüte zu erbringen, werden Fehlerparameter erfasst und aufgezeichnet, mit denen das System überwachbar ist. Sie sind Auslöser für angestoßene Reparaturmaßnahmen[2], welche teilweise automatisch und teilweise manuell durchgeführt werden müssen. Ein solches Konzept kann auch in die Ebene der Kommunikationssoftware integriert werden.

Dienstgüte durch Fehlererkennung

Um von Beginn an ein überschaubares Konzept zu entwickeln, wird vorerst die Softwarestruktur für eine Verbesserung der Dienstgüte vorbereitet. Dazu kommt vor allem ein geeignetes Konzept im Bereich der Konkurrenzkontrolle für mehrere, parallel agierende Nutzer und ein ausgefeiltes, teilautomatisches Fehlermanagement.

Ausbau der Softwarestruktur

[1] vgl. [LEX04] unter . . . /Dienstguete.html (23.07.2004)
[2] vgl. [LEX04]

4.2.1 Die Verbesserung des Softwaredesigns

Herangehensweise „Top Down“ oder „Bottom Up“

Zur Verbesserung der Softwarestruktur kann man grundsätzlich von zwei Richtungen an die Sache herangehen: beginnend von der Anwendungsschicht oder von der Betriebssystemschicht. Da die bisherige Software nur zeigen sollte, ob die elementaren Ideen zur Übertragung funktionieren, lohnen sich an dieser Stelle beide Wege, um eine Verbesserung im Hinblick auf ein zukünftiges Produktionssystem zu erwirken.

Bottom Up: Ausweitung des Middleware-Begriffs

Startet man von tieferen Schichten (= „Bottom Up“), ist zu erkennen, dass der Abstraktionscharakter herkömmlicher Middleware nicht weit genug geht. Im klassischen Sinn ist sie per Definition (vgl. Abschnitt 3.2.1 auf Seite 39) eine Verteilungsplattform mit generischen Dienstleistungen zwischen Anwendung und Kommunikation. Schnell zeigt sich aber, dass dies nicht ausreicht, weil auch direkte Systemaufrufe für die unterschiedlichen Betriebssysteme unterschiedlich sind. Diese Heterogenität wird damit in die Anwendung projiziert, was zu heterogenem Code führt. Deshalb sollte der Begriff Middleware um diesen Aspekt erweitert werden.

Begriffsbestimmung BEG4.1 „Verallgemeinerter Middlewarebegriff“:
Eine Middleware ist eine Zwischenschicht zwischen Anwendung und Betriebssystem, welche durch generische Dienstleistungen sowohl als Verteilungsplattform zwischen Anwendung und Netzwerk als auch allgemein abstrahierend die Systemspezifika homogenisiert.

Diese Erweiterung liefert genügend Freiraum, um eine flexible Portierbarkeit auf unterschiedliche Plattformen zu erlaubt. Natürlich wird dadurch der Funktionsumfang auf einen gemeinsamen Nenner aller gewünschten Betriebssysteme beschränkt.

Geschichtete Mehrwertdienste durch eine erweiterte, proprietäre Middleware

Im vorliegenden Fall heißt eine Erweiterung, dass die Middleware `CORBA` nun Teil einer neuen, proprietären Middleware ist, welche weitere Objekte für die in der vorliegenden Aufgabe benötigten Betriebssystemdienste bereitstellt. Diese können sich in Form eines Dienstleistungsstacks von Stufe zu Stufe im Sinne einer Schaffung von Mehrwertdiensten aufbauen. Auf unterster Ebene befinden sich somit z.B. nur Eins-zu-eins-Abbildungen der realen Systemaufrufe. Von Stufe zu Stufe kommen dann kombinierte Abläufe hinzu, bis schließlich komplette, höherwertige Dienstleistungen folgen. Der Grad der Abstrahierung ist damit frei von der Anwendung wählbar.

Top Down: Aufweichung der starren Gegebenheiten der Client-Anwendung

An dieser Stelle kann nun die Betrachtung von der Anwendung aus einsetzen („Top Down“). Im `CFT` ist sie stark von starren Gegebenheiten geprägt. Die Clientanwendung z.B. sieht genau zwei Zugangspunkte. Einer davon realisiert mehr oder weniger komplex auf eigene Weise den Zugriff auf das lokale Dateisystem. Der andere dient als Schnittstelle zum entfernten `CFT` -Server. Darüber sitzt eine Ansammlung von Methoden, welche die Client-Logik bilden und diese beiden Zugangspunkte verbinden. Die Logik wiederum wird über den Befehlsinterpreter für die Kommandozeilen angetriggert.

Der Server ist im Grunde genommen eine Verlängerung des Client-Stacks über ein Netzwerk hinweg. Im CFT sitzt deshalb hinter der aktivierenden Schnittstelle direkt die Realisierung des Dateisystemzugangs. Die Anwendungsschicht wird somit nicht lokal durch einen Nutzer bedient, sondern über das Netzwerk aktiviert. Sie realisiert den Zugang auf ein lokales Dateisystem. Ein realer Anwender beim Server initiiert nur den Start des Servers.

Somit ergibt sich eine relativ starre Konstellation. Ein flexibles Design muss deshalb dafür sorgen, dass einzelne Realisierungen flexibel ausgetauscht und kombiniert werden können. Ein Client z.B. soll auch die Möglichkeit haben, zu einem Zeitpunkt parallel mit mehreren Servern verbunden zu sein. Dadurch müssen Dateien nicht erst komplett in das lokale Verzeichnis kopiert werden, bevor man sie auf einen weiteren Server schreiben kann, sondern können in einem Arbeitsschritt von einem Server zu einem anderen übertragen werden. Ein Server wiederum soll z.B. so flexibel sein, dass er nicht nur auf lokale Dateisysteme abbildet, sondern auf entfernte weitervermittelt oder eine andere Art der Datenablage anschließt.

Grundüberlegung ist ein Multi-Tier-Konzept

An dieser Stelle kommt als Grundgedanke das „Tier“-Konzept bei ODBC ins Spiel, welches verschiedene Treiber unter einem einheitlichen Schnittstellenzugangspunkt zur Verfügung stellt, egal ob man lokal zugreift, die Verbindung zu einem Server hat oder einen clientseitigen Serververtreter anspricht. Eine Verallgemeinerung eines solchen Mehr-Klassen-Konzepts und der resultierenden Kombinierbarkeit liefert die zusätzliche Flexibilität. Deshalb ist im erweiterten Softwaredesign die Umsetzung eines „Multi-Tier-Konzepts“ durchzuführen.

Diese Erweiterung fordert damit natürlich eine generelle Umstrukturierung im Client. Er kann nun nämlich auf einen „Mehr-Klassen-Verteiler“ wieder schichtenweise seine Mehrwertdienste aufbauen, die ab dann die neue Client-Logik bilden. Zusätzlich sollte der Interpretermechanismus für Befehle so flexibel sein, auch automatische Skripte in linearer Form zu verwerten, was im Design einen zweiten Zugangspunkt im Nutzerinterface bildet.

Zusammengefasst bilden diese Grundgedanken die Grundlage für ein flexibles und kompaktes Design einer erweiterten Version des CORBA File Transfers.

4.2.2 Die Handhabung kritischer Abschnitte zur Kontrolle konkurrierender Zugriffe

Management von kritischen Abschnitten bei konkurrierenden Zugriffen auf gemeinsame Ressourcen

Ein generelles, weiteres Problem im bisherigen CFT ist, dass es keine Kontrolle darüber gibt, welcher Nutzer auf welche Dateien gleichzeitig zugreift. Wie in allen Systemen, die mit für mehrere Nutzer gemeinsamen Ressourcen (hier sind es die Dateien) arbeiten, treten aber auch beim verteilten Dateizugang kritische Abschnitte auf. Diese gilt es zu entkoppeln, um die Datenintegrität zu wahren.

Begriffsdefinition DEF4.2 „Kritischer Abschnitt“ in Anlehnung an [SING94][3]:
Ein kritischer Abschnitt (Critical Section) ist ein Codesegment eines Prozesses, welches eine mit anderen Prozessen geteilte Ressource benutzt, die zu einem Zeitpunkt nur von genau einem Prozess verwendet werden darf.

Vermeidung von Inkonsistenzen durch gegenseitigen Ausschluss

Wird eine gemeinsame Ressource von mehreren Prozessen gleichzeitig verwendet, so kann es zu Inkonsistenzen bzgl. ihres Zustands kommen. Um diese konkurrierenden Zugriffe koordiniert zu verarbeiten, sind Verfahren des gegenseitigen Ausschlusses zu implementieren.

Begriffsdefinition DEF4.3 „Gegenseitiger Ausschluss“ in Anlehnung an [SING94][4]:
Gegenseitiger Ausschluss (Mutual Exclusion) bedeutet, dass konkurrierende Zugriffe von verschiedenen, unkoordinierten Anfragen auf eine gemeinsame Ressource so serialisiert werden, dass ihre Integrität gewahrt bleibt.

Lösung in einem einzelnen Rechner durch Semaphore

In einem einzelnen Rechner ist dies relativ einfach durch sog. Semaphore zu lösen[5]. Dabei handelt es sich um gemeinsame Speicherelemente, welche die Zugriffserlaubnis symbolisieren und von den Konkurrenten abgefragt werden können. Der schnellste erhält den Zuschlag und darf das Semaphore setzen und auf die Ressource zugreifen. Entscheidend ist, dass zumindest die Semaphoreaktionen atomar (nicht unterbrechbar) sind.

Lösung in einem verteilten System durch Nachrichtenmechanismen

In einem verteilten System existieren jedoch keine gemeinsamen Speicherelemente, weil sowohl die Ressourcen als auch die Nutzer im Netzwerk verteilt sind. Somit sind Nachrichtenmechanismen anzuwenden, bei denen keine garantierten Transferzeiten gewährleistet werden können. Und auch Uhren zur verteilten Serialisierung laufen nicht synchron. Folge ist, dass keine Einzelstation eine globale Sicht über das Gesamtsystem erhalten kann.

Lösungen basieren auf Token oder auf Aussagenlogik

Die Lösung sind mehr oder weniger aufwändige Algorithmen, welche entweder auf dem Tokenprinzip (nur derjenige, der ein sog. „Token“ besitzt, darf auf die Ressource zugreifen) basieren oder auf Aussagenlogik (nur wenn gewisse Behauptungen zutreffen, darf der kritische Abschnitt betreten werden) aufbauen. Für alle jedoch gilt ein Client-Servermodell, in dem ein Client entweder einen kritischen Abschnitt betreten will, gerade einen ausführt oder außerhalb abläuft. Dies sind die clientseitigen Zustände für kritische Abschnitte (außer bei tokenbasierten: dort kann auch ein „Idle-Token“-Zustand auftreten, wenn das Token bei einer Station liegt und keine weitere einen kritischen Abschnitt betreten will).

Ein guter Algorithmus besitzt dabei die Fähigkeiten,

- Verklemmungen (Deadlocks) zu vermeiden (endloses Warten auf den Eintritt),
- Verhungerung (Starvation) zu umgehen (ein Konkurrent besetzt dauerhaft den kritischen Abschnitt),
- Fairness zu beweisen (jede Anfrage wird in einer logischen Reihenfolge abgearbeitet) und
- Fehlertoleranz zu ermöglichen (Fehler werden automatisch behoben).

[3] vgl. [SING94] a.a.O. S. 15
[4] vgl. [SING94] a.a.O. S. 121
[5] vgl. zum folgenden Abschnitt [SING94] a.a.O. S. 121 ff.

Simple Lösung in TAO: ein eigener Thread nimmt Aufträge entgegen und gibt sie sequentiell weiter

Der einfachste Weg ohne komplizierte Algorithmen gegenseitigen Ausschluss zu erreichen, ist die Nutzung eines gemeinsamen Ressourcenverwalters (Master). Eine reduzierte Version davon bezogen auf nur eine Ressource wird dabei von CORBA genutzt. Ein eigener Thread (Prozessfaden: Verwaltungseinheit innerhalb eines Prozesses) nimmt die Aufträge entgegen und gibt sie serialisiert und geordnet an den eigentlichen Ausführungsthread weiter. Jede Aktion ist atomar und kann nicht unterbrochen werden. Die Sortierung erfolgt in der Reihenfolge des Eintreffens. Da in CORBA die Ressource immer durch das RPC -Prinzip repräsentiert wird, ist für atomare Einzelaktionen das Problem der kritischen Abschnitte gelöst.

Für die Dateibearbeitung sind zusätzliche, übergeordnete Mechanismen einzuführen

Allerdings handelt es sich bei Dateizugriffen, die das Schreiben oder Lesen des Inhalts betreffen, zumeist nicht um atomare Einzelaktionen, da eine Datei ab einer bestimmten Größe blockweise verarbeitet werden muss. Somit ist ein übergeordneter kritischer Abschnitt vorhanden, da konkurrierende Zugriffe auf dieselbe Datei nicht stattfinden dürfen, solange ein Konkurrent die Ressource „Datei“ bearbeitet. Damit müssen auf höherer Ebene wieder Mechanismen ergänzt werden, welche dieses Problem lösen. Diese Lösungen können aber auf den simplen Prinzipien für atomare CORBA -Aktionen aufsetzen.

Sonderstellung der Ressource „Datei“: Duplikation in parallele Versionen möglich

Bei Dateien verhält es sich etwas anders als mit herkömmlichen Ressourcen (z.B. der Hardware), welche nur einmal existieren. Von derselben Datei können parallel beliebig viele Kopien (Replikate) existieren. Diese können parallel verarbeitet werden. Solange dafür gesorgt wird, dass für die Übernahme der Änderungen und damit neuer Versionen eine faire, logische Anordnung eingehalten ist, kann nichts gegen eine Vermehrung der zu einem Zeitpunkt mehrfach genutzten Ressource sprechen.

Lösung sind parallele Dateisichten

Die Lösung ist damit ein Algorithmus, welcher parallele Sichten auf eine Datei erlaubt, so dass jeder neue Zugriff eine konsistente Sichtweise auf das Dateisystem erhält und jeder momentan involvierte Bearbeiter während seiner Bearbeitung die Ausgangssichtweise bei Beginn der Aktivitäten vorfindet. Ein geschickt gewähltes Registrierungssystem muss dabei die parallelen Sichten verwalten und nach Abschluss konkurrierender Zugriffe eine logische Übernahme der Ergebnisse automatisch durchführen. Nach außen hin wird den Anwendern somit ein herkömmliches Dateisystem vorgespiegelt, während im Hintergrund real zahlreiche, temporäre Duplikate einer Datei existieren. Dabei erfahren die Anwender anhand der Rückmeldungen beim Öffnen, ob bereits andere Schreiber existieren. Nach dem Abschluss gibt der Rückgabewert an, ob die aktuell geschriebene Version bereits veraltet ist und ersetzt wird bzw. worden ist.

An einem Beispiel sollen noch einmal die verschiedenen Ansätze auf den konkreten Anwendungsfall des Dateizugriffs angewandt werden:

Beispiel 4.1 für „die konkurrierende Handhabung eines Dateiabschnitts“

a) ***Zugriff ohne Überwachung des kritischen Abschnitts***
Jeder Leser und Schreiber arbeitet direkt mit der Datei ohne gegenseitige Koordination. => Höchste Gefahr der Inkonsistenz und von Zugriffsfehlern

b) ***Zugriff nach exklusiver Zuweisung des kritischen Abschnitts (LOCK-Mechanismus)***
Jeder Leser und Schreiber muss vor einem Zugriff prüfen, ob bereits ein anderer Prozess eine exklusive Zuweisung gesetzt hat. Ist dies nicht der Fall, muss eine Sperre für die Datei gesetzt werden. Erst wenn diese erteilt wurde, kann ein Zugriff erfolgen. Dabei ist entscheidend, dass die ersten beiden Aktionen „Testen“ und

„Setzen“ als ein nicht-teilbarer Befehl ausgeführt werden. => Höchste Sicherheit bzgl. Datenkonsistenz, aber gegenseitige Blockade der Zugriffsprozesse

c) ***Sequenzialisierung der Zugriffe (QUEUE-Mechanismus)***
Jede Aktion auf eine Datei wird bei einem Warteschlangen-Verwalter angemeldet, der die Zugriffe zeitoptimiert einordnen und ausführen kann. => Ebenfalls höchste Sicherheit bzgl. Konsistenz, der Auftraggeber ist während der Abfrage blockiert (synchrone Kommunikation) oder kann parallel weitere Aufgaben erledigen, während er auf eine Antwort wartet (asynchrone Kommunikation)

d) ***Verwaltung separater Zugriffssichten (VIEW-Mechanismus)***
Jeder Lese- und Schreibzugriff wird getrennt verwaltet. In dieser Verwaltung wird gewährleistet, dass während einer Bearbeitung jeder agierende Prozess die Sicht auf die Datei behält, so wie er sie zu Beginn der Aktion vorfand. Aktionen sind zeitlich sortiert und es gibt mehrere, parallel existierende Versionen einer Datei. Erweitert kann sogar eine Versionskontrolle mit einer Datenzusammenführung (Merging) erfolgen(aber: evtl. manuelle Eingriffe nötig, siehe weiter unten). => Hoher Nutzerkomfort, variierbare Sicherheitsbedingungen bzgl. Datenkonsistenz, keine Blockade, höchster Verwaltungsaufwand

Das entstandene Konkurrenzkontroll-Modell erhält in seiner Grundform für jeden Nutzer eine konsistente Sicht auf die Dateien. Allerdings ist es dahingehend optimistisch, welchen Stand es den Nutzern liefert (vgl. dazu Abb. 4.1 auf Seite 90). Für einen lesenden Zugriff ist so immer die letzte, komplett existierende Version gültig. Die Übernahme von neuen Versionen hängt von zwei Einordnungszeiten ab: dem Start des Schreibvorgangs und dessen Ende. Dies stellt in der Regel keine Einschränkungen dar, da die Daten erst gültig werden, wenn sie komplett sind. Existiert mittlerweile ein weiterer Schreiber, der zudem schneller fertig ist, so ist diese Version gültig und die langsamer geschriebene Vorgängerversion wird nie sichtbar.

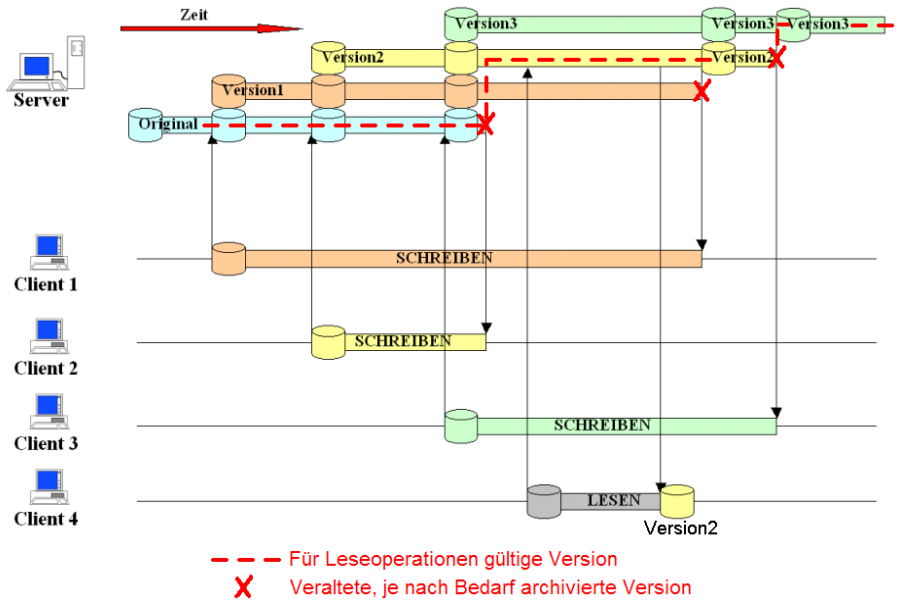


Abbildung 4.1: Beispiel zur Konkurrenzkontrolle mittels paralleler Sichten

Problem der „Lost Updates“ lösbar durch Sperren oder Merging-Mechanismen

Das Problem der „Lost Updates“, also von parallel stattfindenden Änderungen, welche durch eine neuere Version unberücksichtigt verloren gehen, bleibt durch das Vorgehen unbehandelt. Zur Lösung gibt es zwei praktikable Alternative:

1. Nutzung von exklusiven Sperren: Wird eine Version verändert, so setzt der schreibende Vorgang eine exklusive Sperre, welche alle Zugriffe verhindert. Dieses Verfahren ist vom verwaltungstechnischen Aufwand am geringsten, weshalb es sich als erste Näherung zur Verhinderung von „Lost Updates“ für die vorliegende Arbeit anbietet. Allerdings kommt es dadurch auch zu den meisten, gegenseitigen Behinderungen, da ein Klient bei konkurrierendem Zugriff generell von der Konkurrenzsituation durch eine Sperrung seines Zugriffs erfährt und aktiv darauf reagieren muss.

2. Nutzung eines „Merging“-Verfahrens: Mit Hilfe dieses Verfahrens wird anhand Gleichheiten versucht, verschiedene Versionen in Deckung zu bringen, um Veränderungen zu erkennen. Die resultierende, gültige Version wird dann aus den einzelnen Teilen alter und neuer Versionen „zusammengemischt“ (in dieser Art wird unter CVS verfahren). Dieses Verfahren fügt sich ideal in das vorhergehend beschriebene Sichtenmodell, ist jedoch in der Realisierung relativ umfangreich und beliebig komplex. Ferner ist anzumerken, dass es dann an seine Grenzen stößt, wenn in mehreren Versionen gleiche Anteile verändert wurden, da es die Semantik der Änderungen nicht wirklich versteht. Dadurch kann es nicht immer selbständig entscheiden. In diesen Situationen muss ein externer Schreiber manuell eingreifen. Trotzdem kann dieses Verfahren, langfristig betrachtet, in den Folgejahren der vorliegenden Arbeit integriert werden.

Gegenseitiger Ausschluss ist unter Einhaltung gewisser Randbedingungen erfüllbar

Die Bedingungen für den gegenseitigen Ausschluss lassen sich mit den beschriebenen Methoden erfüllen. Sofern jede Aktion auf eine erlaubte Zeitspanne begrenzt ist (Kooperatives Verhalten mit evtl. freiwilliger Selbstaufgabe der Ressource) und in keine Endlosschleife läuft, gibt es keine Lifelocks (Endlosläufer) oder Deadlocks. Jeder Konkurrent, der sich für die Datei interessiert, erhält in einer endlichen Zeit Zugang (keine Verhungerung) in der Reihenfolge des Eintreffens seiner Anfrage (Fairness). Werden Deadlines für die Kommunikation genutzt, ist die Fairness zwar etwas eingeschränkt, es werden aber evtl. Deadlocks im Client verhindert, welche durch Fehler beim Netzwerktransfer verursacht sind. Und je ausgefeilter der Algorithmus zur Übernahme und zur Kohärenzprüfung ist, desto fehlertoleranter wird die Handhabung des gegenseitigen Ausschlusses.

4.2.3 Das Fehler- und Ausfallmanagement

Handhabung von Fehlzuständen

Ein weiterer Punkt im Hinblick auf ein praktikables Produktionssystem ist die Handhabung von Fehlerzuständen. Fehlermanagement bedeutet im Großen und Ganzen das Stabilisieren eines Systems nach einer Fehlersituation. Um fehlerhaftes Verhalten einzuschätzen sind vorerst einige Begriffe zu klären[6].

Begriffsdefinition DEF4.4 „Fehler“ in Anlehnung an [SING94][7]:
Ein Fehler (Error) ist die Manifestation eines anomalen physikalischen Defekts (Fault), welcher zum Versagen (Failure) eines Systems führen kann.

[6] vgl. zum folgenden Abschnitt [SING94] a.a.O. S. 297 ff. (mit Abänderungen)
[7] vgl. [SING94] a.a.O. S. 298

Damit ist an den Begriff des Fehlers direkt die Definition des Systemversagens geknüpft.

Begriffsdefinition DEF4.5 „Versagen“ in Anlehnung an [SING94][8]:
Das Versagen (Failure) eines Systems entsteht, wenn es nicht nach Spezifikation agiert und damit trotz erlaubter Zustandsübergänge einen irrtümlichen Zustand einnimmt.

Die Ursachen für ein Versagen können generell in vier Gruppen eingeteilt werden:

1. **Prozessfehler:** Sie werden ausgelöst durch Programmfehler oder ähnliches, so dass fehlerhafte Ergebnisse geliefert werden.
2. **Systemfehler:** Sie werden verursacht durch eine Unterbrechung der Abarbeitung (Prozessabsturz, Prozessorausfall, etc.). Nach Auftreten kann es zu kompletter oder teilweiser Amnesie kommen (das Bisherige ist komplett oder teilweise unbekannt). Das System kann zudem komplett angehalten sein oder eine Art Pausezustand einnehmen (startet immer in denselben Fehlerzustand)
3. **Sekundärspeicherfehler:** Sie entstehen, wenn Daten nicht mehr zugreifbar sind (z.B. Festplattenfehler)
4. **Kommunikationsfehler:** Sie entstehen, wenn das Kommunikationsmedium nicht mehr korrekt arbeitet.

Kritische Fehler und katastrophale Alarme

Fehler lösen somit im Normalfall kritische Situationen aus und führen das System in einen instabilen Zustand. Während häufig solche Fehlersituationen selbstständig durch spezielle Fehlerbehandlungen in den Programmroutinen oder durch eigens vorgesehene Methoden des Systems wieder behoben werden können, sind Ausnahmesituationen (Exceptions) irreversibel. Sie führen zu irrtümlichen Folgezuständen (dazu zählen z.B. Ausnahmesituationen bei der Referenzierung eines Speicherelements), so dass ein System in einen katastrophalen Zustand läuft. In solchen Situationen können in der Regel keine weiteren Aktionen mehr ausgeführt werden. Es ist nur noch möglich, sie als Alarme zu melden.

Begriffsbestimmung BEG4.2 „Alarm“:
Alarme sind nicht selbstständig behebbare Ausnahmesituationen, welche einen dauerhaften Fehlzustand zur Folge haben und unweigerlich zum Versagen eines Systems führen (katastrophale Zustände).

Minimierung von Alarmen

Tritt ein Alarm auf, so muss manuell von außen in die Systemabläufe eingegriffen werden. Da in einem automatisch agierenden System solche manuellen Eingriffe zu minimieren sind, sorgt ein geeignetes Fehlermanagement für die Erkennung und Reparatur von kritischen Zuständen. Dies kann z.B. nach einer Fehlersituation im herkömmlichen Bearbeitungsfluss (quasi “on-the-fly“), z.B. durch Wiederherstellen eines stabilen, fehlerfreien Zustands, geschehen, so dass die nachfolgenden

[8] vgl. [SING94] a.a.O. S. 298

Aktionen wieder auf einer konsistenten Ausgangssituation aufbauen. Wenn z.B. eine Datei nicht zum Lesen geöffnet werden kann, weil sie nicht existiert, werden auch keine folgenden Leseoperationen gestartet.

Selbstständige, lokale Fehlerbehebung

Die einfachste Form des Fehlermanagements besteht in einem interaktiven System darin, eindeutige Kennungen für Fehler über den Client an den Nutzer zu melden. Eine zusätzliche Meldung erfolgt an den Serveradministrator. Beide sind damit bei Bedarf in der Lage, auf Fehlerzustände zu reagieren. Da jedoch in einem verteilten Client-Server-System mit lokalen und entfernten Ressourcen eine Fehlerbehebung durch einen Remoteclient nicht immer möglich ist, sollte jeder Kommunikationspartner selbstständig in der Lage sein, für sich einen stabilen Zustand herbeizuführen. Ein Ausfall eines Netzwerkabschnitts z.B. kann zu inkonsistenten Zwischenzuständen eines Kommunikationspartners führen, welche während des Ausfalls nicht durch den anderen repariert werden können. Besonderes Augenmerk liegt hierbei auf den Servern, welche als Zentralisierungspunkte eine 1:n-Beziehung mit ihren Klienten eingehen und somit als Knotenpunkte nach Möglichkeit immer konsistente Zustände erreichen müssen.

Zur selbstständigen Behebung von Fehlersituationen gibt es generell zwei grundlegende Vorgehensweisen:

1. Sind alle Fehler bekannt und in Zusammenhang zu einem korrekten Ablauf zu bringen, so können sie systematisch kompensiert werden, um dem Prozess einen erneuten, korrekten Ablauf zu ermöglichen (forward-error recovery).

2. Können die Fehlerquellen und ihre Auswirkungen nicht so einfach lokalisiert werden (in einem normalen System wohl am wahrscheinlichsten), so kann der aktuelle, fehlerhafte Prozesszustand auf einen früheren, korrekten zurückgesetzt werden (backward-error recovery).

Vorhersehbare und nicht vorhersehbare Fehler

Die erste Vorgehensweise geht von vorhersehbaren Fehlern aus (z.B. beim Versuch eine Datei zu öffnen, tritt ein Fehler auf) und ist damit ideal auf lokal gekapselte und abschätzbare Methodenaufrufe anwendbar, welche lokal begrenzt Rücknahmen bis zur Ausgangssituation durchführen können. Die zweite Vorgehensweise stellt einen allgemeinen, als stabil bekannten Systemzustand wieder her und ist deshalb ideal zur Behebung von Ausfallerscheinungen nutzbar. Beide Methoden nutzen damit Prinzipien der Wiederherstellung (Recovery) von fehlerfreien Zuständen.

Begriffsdefinition DEF4.6 „Recovery“ in Anlehnung an [SING94]*[9]*:
Wiederherstellung (Recovery) ist die Herbeiführung eines normalen Operationsmodus eines Systems durch Rückführung aus einem fehlerbehafteten in einen fehlerfreien Zustand.

Handhabung von Systemausfällen

Ein Sonderfall der Wiederherstellung ist der Wiederanlauf nach einem Komplettausfall einer Teilkomponente. Diese Wiederinbetriebnahme bis zur Erreichung eines stabilen Ausgangszustands wird vom Ausfallmanagement erbracht, welches als ein Zusatz zum Fehlermanagement angesehen werden kann und mit ihm eng kooperieren muss. Das Ausfallmanagement sorgt dafür, dass Zwischenzustände vor

[9] vgl. [SING94] a.a.O. S. 297 und [SING94] a.a.O. S. 300

einem Ausfall keine kritischen oder katastrophalen Folgezustände nach dem Wiederanlauf hervorrufen und damit zu einer inkonsistenten Ausgangssituation führen.

Realisierungen des „backward-error recovery“

Da, wie erläutert, zur Handhabung von Ausfällen die „backward-error recovery“ ideal geeignet ist, kommt sie insbesondere beim Ausfallmanagement zum Einsatz. Zur Realisierung dieses Verfahrens zur Wiederherstellung von Systemzuständen gibt es generell zwei Möglichkeiten, bzw. eine Kombination davon:

1. Zum einen können sämtliche Teilschritte einer Verarbeitung von einer Station aufgezeichnet werden (sog. “Logs“). Diese operationenbasierte Bearbeitung liefert genügend Informationen, um Aktivitäten zurückzusetzen, welche zum Fehler geführt haben.

2. Die zweite Möglichkeit definiert stabile Punkte, sog. “Recovery Points“, in denen ein kompletter Zustand mit seinen momentan existierenden Einstellungen und Eigenschaften persistent abgelegt wird. Dieses zustandsbasierte Verfahren agiert somit über “Checkpoints“, auf die das System nach einem Fehlerzustand zurückgesetzt wird (“Rolling back“).

Ziel ist die Aufhebung der Amnesie

Beiden Methoden gemeinsam ist, dass sie versuchen, die komplette oder teilweise aufgetretene Amnesie zu überwinden. Deshalb speichern sie in regelmäßigen Abständen alle benötigten Informationen persistent zwischen, welche entweder zum Zeitpunkt eines stabilen Zustands vorherrschten oder zum Erreichen dieses benötigt werden. Diese Informationen können global für ein gesamtes verteiltes System gelten (sehr aufwändig) oder nur lokal einen Knoten erfassen. Nach einer Wiederherstellung, sei es im Rahmen des Ausfall- oder des Fehlermanagements, werden alle weiteren Aktionen von dem erreichten Zustand aus weitergeführt.

Benötigte Aktionen für das Recovery

Zusammenfassend kann festgestellt werden, dass zur persistenten Ablage der Wiederherstellungsinformation stabile Speicher benötigt werden und es generell die Aktionen “Durchführen“ (“Do“), “Rücksetzen“ (“Undo“), “Erneutes Durchführen“ (“Redo“) und “Anzeigen des Zustands“ (“Display“) geben muss. Um alle Möglichkeiten der Wiederherstellung zu nutzen, ist eine Kombination von “Logs“ und “Checkpoints“ sinnvoll. Vereinfacht wird dies, wenn Aktionen idempotent sind, d.h., eine Mehrfachausführung (z.B. durch ein “Redo“) den resultierenden Zustand nicht verändert. Dadurch können Aktionen bedenkenlos nach einem Wiederanlauf erneut ausgeführt werden, wodurch gesichert wird, dass sie mindestens einmal korrekt abgearbeitet wurden. Das “Undo“ kann so feinfühliger zur Anwendung kommen.

Vermeidung des Domino Effekts

An dieser Stelle ist noch auf die Folgeproblematiken beim Wiederherstellen von früheren Zuständen hinzuweisen. Bei verschränkten Aktionen in einem verteilten System kann eine Rücksetzaktion einen lawinenartigen Rollback veranlassen, da alle beteiligten Partner ihrerseits auf frühere Zustände zurückspringen müssen, was durch die Verschränkung zu einer Folge von gegenseitig bedingten Rücksprüngen führen kann. In einem strikten Client-Server-Modell wird diese Problematik dahingehend entschärft, dass alle Aktivitäten von einem Client ausgehen müssen und ein Server nicht seinerseits einen Rollback initiieren kann. Trotzdem ist darauf zu achten, dass ein Client keine verschränkten, ändernden Aktionen auf mehreren Servern parallel vornimmt. Klienten sind gezwungen, erst eine schreibende Aktionenfolge mit einem Server abzuwickeln, bevor sie eine weitere mit einem anderen

Server starten. Eventuell nötige Rollbacks kann ein Client so mittels der Fehlercodes erkennen und nur für die aktuelle Kommunikationsbeziehung anwenden. Komplett getrennte Verarbeitungsfolgen oder lesende Zugriffe sind diesbezüglich in der Regel kein Problem. Im letzteren Fall ist hierbei nur dafür zu sorgen, dass ein lesender Client nur komplett geschriebene Daten abfragen kann. Dies ist durch die Einhaltung von Zugriffssichten zur Kontrolle von konkurrierenden Klienten aus dem vorhergehenden Abschnitt 4.2.2 gegeben.

Aktionenfolgen bilden Transaktionen

In diesem Zusammenhang wird bereits ersichtlich, dass es sinnvoll ist, einzelne, zusammenhängende, atomare Aktionen zu Sequenzen zusammenzufassen, welche nur als Ganzes ausgeführt werden dürfen. Erst wenn die komplette Aktionenfolge durchlaufen wurde, ist damit wieder ein stabiler Zustand erreicht. Solche unteilbaren Aktionenfolgen werden als Transaktionen bezeichnet und dienen als Grundlage für einen zustandsbasierten Arbeitsfluss.

Begriffsdefinition DEF4.7 „Transaktion“ in Anlehnung an [SING94][10]:
Eine Transaktion besteht aus einer atomaren Aktion oder aus einer Sequenz von Lese- und Schreibzugriffen, welche unteilbar als Einheit ausgeführt werden müssen oder insgesamt keine Auswirkungen haben. Dabei erhalten sie die Konsistenz der Daten und haben eine endliche Ausdehnung.

Rudimentäre Transaktionen werden bereits durch die Schnittstellendefinition und das Sichtenmodell der Konkurrenzkontrolle vorgegeben

Jede bisher in der Arbeit beschriebene Schnittstellenmethode ist damit eine Transaktion. Ein grundlegendes Beispiel für eine unteilbare Folge von Aktionen sind z.B. schreibende Dateizugriffe. Nur wenn die Befehlsfolge vom Öffnen, über das mehrmalige, blockweise Schreiben, bis hin zum Schließen der Datei als Einheit ausgeführt wurde, darf eine Datei als gültig übernommen werden. Diese Form rudimentärer Transaktionen wird bereits durch den Einsatz von Zugriffssichten bei der Konkurrenzkontrolle (vgl. vorhergehenden Abschnitt 4.2.2) erbracht. Umfangreichere Aktionssequenzen sind im Rahmen der vorliegenden Arbeit nicht notwendig. Allerdings ist es noch erforderlich, dass jeder Client den aktuellen Zustand einer Transaktion ermitteln kann.

Ein auf den genannten Mechanismen basierendes Fehler- und Ausfallmanagement sollte damit den Bedingungen für das vorliegende, reale System (in dem es z.B. durchaus durch Kommunikationsunterbrechungen zu einem Aufsplitten in Teilnetze kommen kann) genügen[11]:

1. Elementare Einheiten für Nutzeraktivitäten sind Transaktionen.
2. Ein Mechanismus zur Konkurrenzkontrolle wird eingesetzt.
3. Die Transaktionen werden eindeutig gekennzeichnet und sind in eine logische Reihenfolge zu bringen.
4. Stationsfehler sind zumindest beim Client über das Protokoll oder über Timeouts erkennbar. Der Server führt bekannte Aktionen komplett durch.
5. Kommunikationsunterbrechungen dürfen nur den betroffenen Abschnitt eines Netzwerks beeinflussen, nicht jedoch die funktionierenden Komponenten

[10] vgl. [SING94] a.a.O. S. 481 und [SING94] a.a.O. S. 485
[11] vgl. dazu [SING94] a.a.O. S.320 mit Abwandlungen

6. Jede Station erhält für sich konsistente Zustände. Der Klient gleicht dabei seine Zustände nach einem Fehlerfall mit dem Server ab, mit dem er während des Fehlerfalls eine Kommunikationsbeziehung eingegangen war.

Asynchrones, pesimistisches Fehler- und Ausfallmanagement

Letztendlich entsteht durch eine Kombination der genannten Mechanismen ein asynchrones Fehler- und Ausfallmanagement zwischen Klienten und Server, da jede beteiligte Komponente für sich stabile Zustände erhält, während die Kommunikationspartner eine synchrone, überwachte Verbindung nutzen. Dies ist wichtig, damit im Fehlerfall ein Client unmittelbar einen Fehlercode entweder vom Server oder von der Kommunikationsschicht mitgeteilt bekommt. Somit kann also davon ausgegangen werden, dass beim Client jede Fehlersituation erkannt wird und pesimistisch eine adäquate Vorgehensweise zur Überprüfung und Behebung auslöst. Wichtigste Aufgabe des Servers ist es dabei, dass er als zentraler Synchronisationspunkt u.a. mittels der Checkpoints konsistente Zustände erhält. Durch eine Kombination von Logs und Checkpoints ist er sogar meist in der Lage, vor einem Ausfall angestoßene Aktionen (sind in der Logdatei aufgeführt) zu vervollständigen, wodurch häufig der vom Klienten gewünschte Folgezustand eingestellt werden kann. Den tatsächlich eingenommenen Zustand einer Transaktion kann der Client z.B. nach einem Ausfall in einer Klärungsphase abfragen. Solange diese Klärung nicht eindeutig abgeschlossen werden kann, führt der Client keine weiteren Aktionen mehr durch.

Alle beschriebenen, theoretischen Grundlagen zusammengefasst bilden die Ausgangsbasis für das Design des „Extended CORBA File Transfer“.

4.3 Umsetzungsdesign: Extended CORBA File Transfer (ECFT)

Erweiterung des Prototyps

Ausgehend vom Prototyp `CFT` und den Überlegungen zu den theoretischen Verbesserungen wird ein umfassenderer Ansatz umgesetzt: der `Extended CORBA File Transfer (ECFT)`. Ziel ist die Schaffung eines erweiterten Prototyps für ein Produktionssystem. Deshalb werden z.B. auch zusätzliche Befehle im Kommandozeileninterpreter ergänzt. Mit der entstehenden Software, können in folgenden Kapiteln Vereinheitlichungen aufgesetzt werden. Das verlässliche Konzept bildet damit auch die Basis für höherwertige Datendienste aus den nachfolgenden Kapiteln, welche implizite Konvertierungen und Zusatzinformationen beinhalten.

4.3.1 Die erweiterte Architektur

Im weiteren Verlauf wird nach der Reihenfolge der vorgenannten Ideen vorgegangen. Eine erste Verbesserung betrifft damit auch die Erweiterung der Softwarearchitektur.

Umsetzung des Multi-Tier-Konzepts

Grundlegende Umbaumaßnahme ist die Umsetzung des „Multi-Tier-Konzepts“ und damit die Schaffung eines Schnittstellenbusses (über einen einheitliche Schnittstellezugangspunkt können verschiedene Schnittstellen angesprochen werden). Die

Umsetzung besteht darin, einen „Multi-Tier-Connector“ („Mehr-Klassen-Verbinder“) zu definieren.

Begriffsbestimmung BEG4.3 „Multi-Tier-Connector („Mehr-Klassen-Verbinder“)“:
Ein Multi-Tier-Connector ist ein Verbindungsstück, welches Möglichkeiten bietet, mehrere von einer allgemeinen Schnittstelle abgeleitete Zugangswege zu kombinieren und zum Zwecke der Kaskadierung selbst auch eine Ableitung dieser Schnittstelle ist.

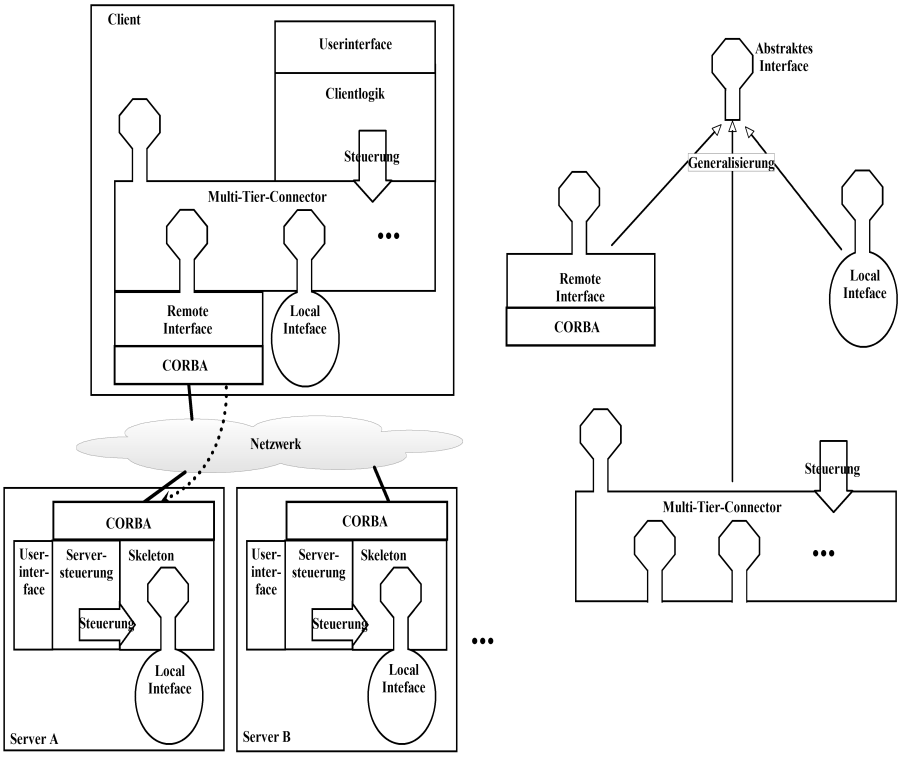


Abbildung 4.2: Der „Multi-Tier-Connector“

Schnittstellenbus

Im Klartext bedeutet dies (vgl. dazu Abb. 4.2 auf Seite 97), dass alle gleichartigen Schnittstellen von einem übergeordneten, abstrakten Interface abstammen und nur noch spezielle Realisierungen dieses sind. Ob nun ein Dateisystemzugang über remote oder lokale Wege erwirkt wird, ist somit nur noch Teil der Abbildung auf eine gewählte Schnittstellenklasse. Dort sind auch alle Mechanismen enthalten, um fehlerhafte Eigenschaften (z.B. nicht mehr existierende Verbindungen bei entfernten Zugängen) zu beheben. Der Connector selbst ist ebenfalls eine Ableitung der abstrakten Schnittstelle und ermöglicht somit eine Kaskadierung mehrerer Connectoren. Intern betrachtet handelt es sich um einen Container für abgeleitete Schnittstellen des Typus der Basisklasse. So können durch implizite Typumwandlung in die allgemeine Schnittstelle einheitliche Zugangsmethoden genutzt werden. Zusätzlich besitzt der Connector noch einen Steuerungszugang, über den gezielt Manipulationen an den aufgenommenen Interfaces ausgeübt werden können (z.B. Befehle an eine spezielle Schnittstelle senden oder Datensätze von einer zur anderen kopieren). Der Client nutzt diesen Kontrollzugang und baut darauf seine

Mehrwertdienste in Form einer verallgemeinerten Logik auf (z.B. kopiere Datenblöcke von Schnittstelle A des Connectors zu Schnittstelle B oder sende einen Befehl an Schnittstelle C des Connectors).

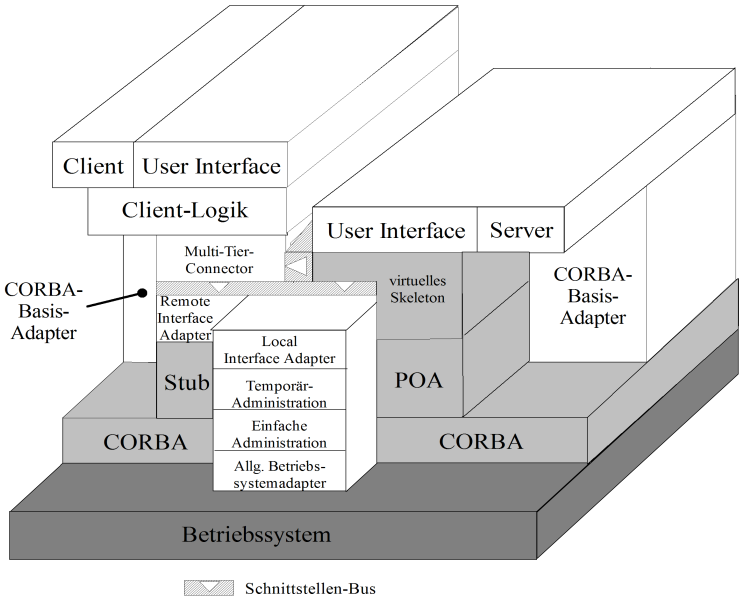


Abbildung 4.3: Die erweiterte Softwarearchitektur des ECFT

Problem bei Multicasts: Synchronisationsbarrieren sind notwedig

Zur vollen Nutzung des Schnittstellenzugangs eines Connectors im Sinne von Multicast-Verteilungen ist neben einem umfangreichen Transaktionskonzept die Realisierung von sog. Synchronisationsbarrieren nötig. An diesen definierten Stellen im Programmcode synchronisieren sich parallele Aktivitätsträger, so dass sie logisch zeitlich geordnet nach kompletter Vollendung eines Verarbeitungsschritts in den folgenden Zustand übergehen[12]. Dies ist notwendig, weil bei einer Mehrfachverteilung im Sinne eines Multicasts jeder Abarbeitungsschritt auf allen angeschlossenen Schnittstellen erst komplett ausgeführt sein muss, bevor die nächste Aktion angestoßen werden darf. Dies ist dahingehend nicht trivial, da zwar aufgrund des Client-Server-Modells die komplette Verwaltungslogik in den Multi-Tier-Connector verlagert werden könnte, es jedoch bei einem Ausfall der Kommunikation trotzdem gewährleistet sein muss, dass sich alle angeschlossenen Interfaces auf den selben Abarbeitungszustand einigen können. Da Multicasts für die vorliegende Arbeit unwesentlich sind, werden sie zwar angedacht, jedoch nicht in diesem Rahmen umgesetzt.

Schnittstellenbus in der Architektur

Die Architektur wurde bzgl. des Schnittstellenbusses so erweitert, dass dieser nicht nur im Rahmen des Multi-Tier-Connectors, für den er konzipiert wurde, im Client zum Einsatz kommt (vgl. Verbindungselemente mit Pfeilen unterhalb des Connectors in Abb. 4.3 auf Seite 98). Auch der Zugangspunkt des virtuellen Skeletons zur eigentlichen Realisierung der Serverfunktionalität basiert auf den standardisierten Methoden des Busses (vgl. Verbindungselemente mit Pfeilen ausgehend vom virtuellen Skeleton in Abb. 4.3 auf Seite 98). Da zusätzlich auch der Connector über eine entsprechende Aufrufschnittstelle verfügt (Möglichkeit

[12] vgl. [WALD95] a.a.O. S. 364

zur Kaskadierung), kann eine Serverrealisierung entweder aus einem Multi-Tier-Connector mit mehreren Schnittstellen, einer für den Server lokalen Schnittstelle zum Dateisystem oder einer Einzelverknüpfung zu einem weiteren, entfernten Interface (ECFT -Proxykonzept; vgl. Abschnitt 7.3.2 auf Seite 185) bestehen. Die Kombinationsmöglichkeit ist damit beliebig flexibel gestaltbar.

Schichtung der Zugangsmethoden zum Betriebssystem

Zusätzlich zum Schnittstellenbus, welcher sich hauptsächlich auf die Umsetzung der Clientanwendung auswirkt, kann noch der allgemeine Betriebssystemadapter aus Abb. 3.10 auf Seite 65 in ein Schichtenkonzept umgewandelt werden. Auf den rudimentären Betriebssystemdiensten setzen stufenweise höherwertige Zugangsmethoden zum Betriebssystem auf, so dass sich der Weg zum Betriebssystem wie in Abb. 4.3 auf Seite 98 dargestellt aus den Schichtebenen Local Interface Adapter, Temporär-Administration, einfache Administration und dem verbleibenden allgemeinen Betriebssystemadapter zusammensetzt.

Insgesamt ergibt sich damit die erweiterte Architektur in Abb. 4.3 auf Seite 98. Eine Darstellung des zugehörigen Klassendiagramms ist im Anhang F auf Seite 243 abgedruckt. Nachfolgend werden nun die geänderten und neuen Module in einem Überblick erläutert.

Allgemeiner Betriebssystemadapter

Bisheriger Betriebssystemadapter

Unterste Stufe der Unterteilung des Betriebsystemzugangs ist das bisher als allgemeiner Betriebssystemadapter bekannte Modul.

Einfache Administration

Bisheriges Interface der CFT-Funktionalität

Auf den allgemeinen Betriebssystemadapter setzt ein Basisdienst auf, welcher auf höherem Niveau die bisher bekannten Funktionalitäten der Schnittstelle des CFT als einfache Administration abbildet. Im Wesentlichen ist damit hier die bisherige Schnittstelle erhalten. Fehlercodes werden zwar erzeugt, jedoch nur an den Nutzer weitergereicht, ohne dass automatische Reparaturaktionen ausgelöst werden.

Temporäre Administration

Ergänzung um Konkurrenzkontrolle, Fehler- und Ausfallmanagement

Der einfachen Administration wiederum überlagert ist die Erweiterung um Konkurrenzkontrolle, Fehler- und Ausfallmanagement. Da dieser Teil maßgeblich durch das Konzept temporärer Sichten geprägt ist, wird sie als temporäre Administration bezeichnet.

Local Interface Adapter

Zusammenfassung der neuen Funktionalität zu einem erweiterten Zugangsadapter

Der so entstandene Dienstestack wird schließlich von einem Adapter für den lokalen Dateizugang übernommen und als Schnittstelle zur Verfügung gestellt. Bei diesem Filesystem-Adapter existieren zwei Ausprägungen, welche die Handhabung der Nutzerrechte regeln. Ein Client kann so die Variante nutzen, welche keine zusätzliche Autorisierung erfordert, da für ihn die Rechte des ausführenden Prozesses gelten. Der Server hingegen kann mit erweiterten Authentifikationsmechanismen

ausgestattet sein und jedem Nutzer seine individuell erlaubte Sicht durch die Nutzerrechte im Betriebsystem bieten.

Remote Interface Adapter

Remotezugang zu einem lokalen Interface

Der Adapter zum Remote Interface ist eine spezielle Ausprägung zum Transportmechanismus von CORBA . Er erfüllt die selben Eigenschaften, wie der Adapter zum lokalen Interface und stellt damit eine über ein Netzwerk verlängerte Dateizugriffsmethode dar. Er ist ebenso beliebig im Connector kombinierbar.

Multi-Tier-Connector

Umsetzung des Schnittstellenbusses

Hier ist ein Container geschaffen, welcher Schnittstellen der gegebenen Spezifikation aufnehmen kann und beliebig kaskadierbar ist. Er setzt die Anforderungen des Mehr-Klassen-Konzepts um. Die durch den Connector geschaffene „Connectivity“ ermöglicht die variable Kombination von Schnittstellen, weshalb in diesem Zusammenhang auch von einem Schnittstellenbus gesprochen werden kann. Dabei wird es übergeordneten Schichten ermöglicht, beliebige unterhalb des Softwarebusses befindliche Schnittstellenadapter zu nutzen und zu kombinieren. Der Server kann so z.B. ein lokales Dateisystem für entfernte Aufrufe zugänglich machen. Er kann aber auch nur als Weiterleitung zu einem weiteren entfernten Dateisystem dienen oder gar in Form eines Clients mit einem Connector weitere Dateisysteme verbinden.

Client-Logik

Abstraktere Kommandoumsetzung im Client

Die Client-Logik ist auf die neue, zugrunde liegende Schichtung angepasst und dadurch wesentlich abstrakter in der Handhabung. Sie bildet für jedes mögliche Kommando des Clients eine Methode nach, welche sie intern über die Funktionalität des Connectors und der dort beinhalteten Schnittstellen realisiert. Dabei betreffen Anfragen und Befehle jeweils nur einen der in den Connector dynamisch eingehängten Interfaceadapter. Die Datentransfers hingegen verbinden je zwei Adapter miteinander und werden komplett innerhalb des Connectors realisiert.

Flexibles Konzept mit zahlreichen Freiheitsgraden

Durch diese relativ grundlegenden Veränderungen, ergibt sich ein Softwaredesign, welches sich durch große Flexibilität auszeichnet. Betriebssysteme werden dem Anwendungszweck entsprechend abstrahiert und Schnittstellen gleichen Charakters sind verwandt und kombiniert nutzbar. Das Konzept erlaubt es damit sogar, andere Übertragungsmechanismen neben CORBA in Form eines zusätzlich abgeleiteten Interfaces anzusteuern, wodurch eine Kombination zur herkömmlichen Datenübertragung möglich wird.

4.3.2 Der Einsatz von Nutzersichten („User-Views“)

In der beschriebenen Architektur ist in der Schicht zur temporären Administration im Dienstestack die Umsetzung der weiteren Ideen zur Verbesserung der Dienstgüte vorgesehen. Dazu zählt vor allem die Implementierung des Modells zur Konkurrenzkontrolle.

Verwaltungsstruktur zur Handhabung paralleler, temporärer Dateiversionen nötig

Es soll, wie erläutert, auf einem Konzept aus verschiedenen Sichten (Views) auf die Dateien aufbauen. Um dies zu ermöglichen, ist eine geschickte Verwaltungsstruktur notwendig (vgl. Abb. 4.4 auf Seite 101). Sie bildet die benötigten Informationen zur parallelen Handhabung der Dateiversionen praktikabel ab. Im Grunde genommen, sind hier aus Performanzgründen keine großen Probleme zu erwarten, da jeder Nutzer zu jeweils einem Zeitpunkt nur eine Datei bearbeiten kann. So verhält sich also der Umfang der Einträge in der Verwaltungsstruktur direkt proportional zur Anzahl der Nutzer mit linearem Verlauf. Aus diesem Grund können zur Vereinfachung auch die STL-Container von C++ mit Schlüsselzugriff (wie z.B. die „Map“) zum Einsatz kommen.

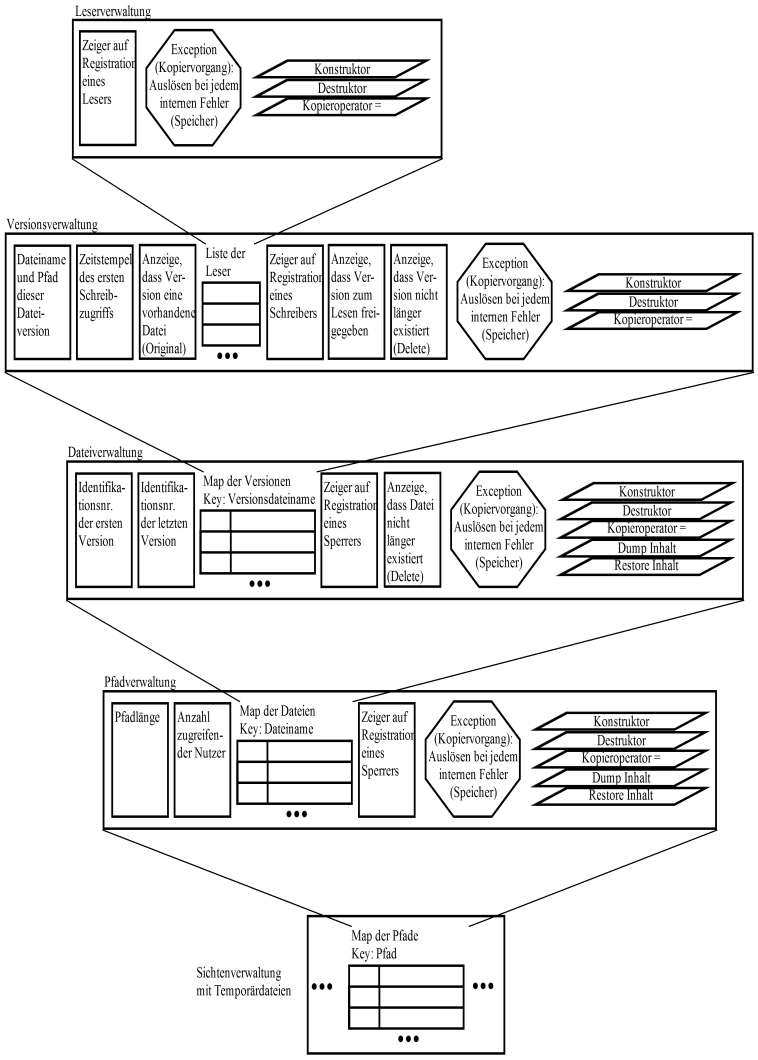


Abbildung 4.4: Die Strukturen zur Verwaltung der Views

Verwaltungsbaum

Bei genauerer Betrachtung der benötigten Zusatzinformationen ist eine grundlegende Aufteilung möglich, welche einen Verwaltungsbaum aufspannt (vgl. Abb. 4.4 auf Seite 101). Seine Wurzel bildet eine Struktur mit allgemeinen Informationen zur Verwaltung von Views (z.B. mit einer Markierung, ob dieses Konzept

überhaupt genutzt werden soll). Alle weiteren Verästelungen sind so gestaltet, dass sie die minimalen Erfordernisse der Container erfüllen (Integration von Kopieroperatoren) und interne Fehler durch Ausnahmebehandlungen preisgeben.

Erste Baumebene bilden Pfadangaben

Die nächste Stufe ergibt sich aus der Notwendigkeit heraus, Zusatzinformationen zu den Pfaden der Dateien zu verwalten. Dabei werden die Pfade nicht als hierarchisch angesehen. Jeder Pfad (Schlüssel in eine Map), auch wenn er ein Teilpfad eines anderen ist, erhält einen eigenen Eintrag. Dies ist wegen des linearen Verwaltungsaufwandes möglich, birgt allerdings eine Bedingung: Generelle Absorptionen (Durchläufe durch die Strukturen mit dem Zweck nicht weiter benötigte Einträge zu entfernen und betroffene Elemente je nach Zustand zu übernehmen oder zu löschen) müssen entsprechend der Pfadhierarchie stattfinden. Das bedeutet, dass die Absorption bei den längsten Pfaden beginnen muss, um dann in den Wurzelverzeichnissen zu enden. Deshalb wird zum einfacheren Sortieren der Pfadlängen in die über den Pfad als Schlüssel zugreifbare Pfadverwaltung auch direkt die Angabe zur Pfadlänge mit integriert.

Eine weitere Kontrollinformation der Pfadverwaltung ist die Anzahl zugreifender Nutzer, so dass evtl. einsetzende Löschaktionen abgeblockt werden können, solange mehr als ein Nutzer zugreift. Eine Vormerkung ist ebenfalls nicht nötig, da verwaltete Einträge auf zugreifende Nutzer deuten und somit von einem Inhalt in dem Verzeichnis ausgegangen werden kann. Außerdem werden rekursive Vorgänge nicht von der Schnittstelle unterstützt, sondern sind von den Anwendungen als zusätzliche Basisbefehle einer höheren Ebene zu realisieren.

Sperrverweise für ganze Verzeichnisbäume sind in dieser Ebene angesiedelt

Um schon hier den nützlichen Sperrmechanismus einzubringen, existiert ein Eintrag, welcher auf einen sperrenden Nutzer verweist (Zeiger auf die Registrierung seiner Sitzung). Dadurch ist es möglich, komplette Pfade und damit Speicherbereiche zu sperren. Sperren wirken dabei autoritär. Alle aktiven Zugriffe ohne Sperren anderer Nutzer werden sofort beendet und begonnene mehrteilige Aktionen verworfen. Bei existierenden Sperren gilt das Vorrecht des Ersten, so dass weitere abgewiesen werden. Ab dann hat nur noch der Halter der Sperre ein exklusives Nutzungsrecht zum Lesen und Schreiben, bis er es freiwillig aufgibt. Sperren dienen somit zur Verhinderung von sog. „Lost Updates“ (unbemerktes Verschwinden von Veränderungen, sobald mindestens zwei Autoren gleichzeitig an derselben Datei arbeiten und diese zu verschiedenen Zeitpunkten freigeben). Zur Vermeidung von Deadlocks wird regelmäßig die Existenz des Inhabers einer Sperre dadurch überprüft, dass sein letzter Zugriff nicht länger zurückliegen darf, als eine vordefinierte Zeitspanne. Wird sie überschritten, wird sein exklusives Recht zwangsweise zurückgezogen, und alle bis dato unvollständigen Abläufe werden verworfen.

Eine weitere Baumebene sind die Dateiangaben

Einem Pfadelement ist weiter eine Liste für die Menge der gerade in Bearbeitung befindlichen Dateien (Originale, Replikate, Versionen) eingegliedert. Ein Element der Liste beinhaltet die eindeutige Identifikationsnummer der ersten Version (Zeitstempel erweitert durch eine fortlaufende Nummer und eine Zufallszahl) und der letzten. Auch hier kann wieder ein exklusives Nutzungsrecht eingetragen werden. Allerdings sperrt der Inhaber nur die jeweilige Datei und ihre Versionen mit ähnlichen Konsequenzen wie bei Pfadsperren. Soll eine Datei gelöscht werden, während noch Versionen geschrieben werden, so existiert ein Eintrag zur Vormerkung des Löschvorgangs. Da dieser in der logischen Ordnung nach allen bisherigen Startzeiten von Schreibaktionen stattfindet, tritt er nach Beendigung des letz-

ten Schreibvorgangs in Kraft. Dieses Vorgehen wurde umgesetzt, um eine logische Reihenfolge von Aktionen zu erhalten. Denkbar wäre stattdessen auch, Löschaktionen während schreibender Zugriffe generell zu verbieten. Inhaltsauflistungen eines Verzeichnisses zeigen in der vorliegenden Umsetzung markierte Dateien bereits vom Zeitpunkt des Löschvermerks an nicht mehr. Hier ist generell anzumerken, dass die temporären Versionen nicht aufgelistet werden. Nur ein ausgezeichneter Nutzer, der Superuser, erhält die komplette Liste und darf auch direkt Aktionen auf temporären Dateien ausführen (mit allen evtl. Folgen). Wird nach dem Anbringen des Löschvermerks kein, erneuter Schreibvorgang mehr angestoßen, wird die letzte Versionen nach Beendigung der Zugriffe gelöscht.

Die folgende Baumebene beschreibt die temporär existierenden Versionen

Der Dateiverwaltung ist eine Liste mit den Versionen eingegliedert. Schlüssel sind hier die Dateinamen der jeweiligen Versionen. Ein Eintrag der Liste enthält den kompletten Pfad zur Version, da dies Zugriffe erleichtert. Der Zeitstempel des ersten schreibenden Zugriffs bildet die logische Ordnung der Versionen. Diese bestimmt jedoch nur zum Teil die Freigabe. Sie wird hauptsächlich vom Zeitpunkt der Fertigstellung des Schreibens definiert. Hier ist auch der Verweis auf den schreibenden Nutzer (Zeiger auf Registration einer Sitzung eines Nutzers), falls er noch aktiv ist. Falls nicht, ist eine Markierung gesetzt, die anzeigt, dass diese Version bereit zum Lesen ist. Gelesen wird immer die Datei, welche in der zeitlichen Reihenfolge aktueller ist, d.h., deren Schreibvorgang bereits vollständig abgeschlossen ist und spätest möglich begonnen hat. Eine weitere Markierung zur Vormerkung des Löschens wird entweder gesetzt, wenn das Original gelöscht wurde oder wenn der schreibende Zugriff vorzeitig endet und die Version verworfen werden muss. Diese Vormerkung erleichtert die Absorption der Versionen, welche immer von der ältesten ausgeht und bis zur momentan aktuellsten verläuft. Somit werden die nicht benötigten Zwischenversionen nicht einfach gelöscht, was die logische Ordnung über den gesamten Zeitraum beliebiger Zugriffe auf eine Datei erhält.

Letzte Ebene beschreibt lesende Zugriffe auf die Versionen

In die Versionsverwaltung ist schließlich noch eine Liste für lesende Zugriffe integriert. Eine Version kann beliebig viele Leser haben, jedoch nur je einen Schreiber. Da alle Leser gleich behandelt werden, reicht eine einfache Liste ohne Schlüsselzugriffe aus. Ein Eintrag darin beinhaltet einen Verweis zur Registration eines lesenden Nutzers. Welche Version ein Leser erhält entscheidet sich zum Startzeitpunkt des Lesevorgangs, an dem er immer die zu diesem Zeitpunkt nach oben beschriebenem Muster aktuellste Version erhält. Nur in diesem Moment haben nicht deterministische Zeitverzögerungen des Netzes einen Einfluss darauf, welche Version gelesen wird. Ab dann bleibt für den Leser die Sicht auf seine zu lesende Version solange erhalten, bis er die Aktion abschließt. Dabei ist es durchaus denkbar, dass dem Leser anhand von Rückgabecodes mitgeteilt wird, ob zum Lesezeitpunkt noch aktuellere, jedoch noch unvollständige Versionen existieren.

Gestalt eines Verwaltungsablaufs

Zusammengefasst bildet sich somit die erwähnte Verwaltungshierarchie, welche durch die linearen Zusammenhänge zwischen Nutzerzahl und Verwaltungsaufwand in der beschriebenen Form hinreichend schnelle Zugriffe ermöglicht. Aus welchen notwendigen Teilaktionen sich so ein Verwaltungsablauf nun grundsätzlich zusammensetzt, soll nachfolgend am Beispiel eines schreibenden Zugriffs aufgezeigt werden. Lesende und sperrende Zugriffe gestalten sich auf ähnliche Weise. Im Wesentlichen können zwei Methoden aufgeführt werden:

1. **Einen Schreiber hinzufügen (AddWriter)**
   Das Hinzufügen eines schreibenden Zugriffs lässt sich im Allgemeinen in wenige Schritte aufspalten:

   (a) Verhinderung von Folgeschäden nach einem vorhergegangenen Alarm (Breakdown-Indikation): Tritt während der Verarbeitung ein Fehlzustand auf, so wird unmittelbar anschließend ein Kohärenzcheck ausgeführt (z.B. stimmen die Nutzer mit denen in der Verwaltung überein, existieren die Dateiversionen noch, usw.), der zudem bestimmte Aufräumaktionen einleitet. Führen diese nicht zum Erfolg, ist ein kritischer Zustand erreicht, der nicht mehr vom System selbsttätig behoben werden kann. Es wird eine übergeordnete Markierung eines Alarms gesetzt. Alle weiteren Aktionen sind von da ab zur Vermeidung von weiteren Schädigungen zu blockieren, solange das Problem nicht manuell behoben wurde. Die Überprüfung der Alarmmarkierung und die Sperre nachfolgender Aktionen wird in diesem ersten Schritt überprüft.

   (b) Zur Sicherstellung von erlaubten Handlungen erfolgt eine explizite Prüfung der Nutzerrechte (im einfachsten Fall ein Zugriffsversuch auf die betroffene Datei).

   (c) Falls die Datei bereits existiert und noch nicht als Original in der Verwaltung eingetragen ist, wird sie ergänzt (sie wird auch als erste Version eingetragen).

   (d) Die eindeutige Versionsbezeichnung wird ermittelt (besteht aus einer Kopfkennung, dem Dateinamen, der Versionsidentifikationsnummer und einer Folgekennung). Existierende Versionen in den Dateinamen eines Verzeichnisses und in den Verwaltungsstrukturen werden berücksichtigt.

   (e) Evtl. gesetzte Sperren werden kontrolliert.

   (f) Die neue Version wird in der Liste angehängt. Die Container für Versionen ist zeitlich geordnet und als `First In First Out Buffer (FIFO)` implementiert. Neue Versionen werden am Ende eingefügt, komplette Versionen vom Anfang her absorbiert.

2. **Einen Schreiber entfernen (DeleteWriter)**

   (a) Auch hier wird als erstes der interne Zustand überprüft (Breakdown-Indikation)

   (b) Anschließend startet der Übernahmevorgang. Dazu werden erst Kohärenzbedingungen (Existenz der Version, Rechte des Nutzers, etc.) geprüft. Es folgt die Entfernung des Verweises auf einen Schreiber und die Freigabe der Version (einschließlich der Dekrementierung der Anzahl zugreifender Nutzer auf einen Pfad), falls diese nicht vorher als nicht verwendbar (Löschvermerk) markiert wurde.

   (c) Nun kann die Absorption beginnen. Sie beginnt von den Blattknoten des Verwaltungsbaums (Versionen). Der `FIFO` wird vom Anfang her aufgeräumt, bis die erste in Bearbeitung befindliche Version erreicht wird. Jede komplett vorliegende Version wird entweder akzeptiert oder zurückgewiesen. Die Übernahme nach dem Akzeptieren erfolgt in drei Schritten: Umbenennen des bisherigen Originals, Umbenennen der neuen Version in das Original und Löschen der zwischengespeicherten, ehemaligen Originalversion (oder bei Aktivierung einer

Versionierung, beibehalten der ehemaligen Version, welche dann nur noch durch den Superuser gelöscht werden kann). Sind alle Versionen absorbiert, wird die Freigabe der Verwaltungseinheiten schrittweise bis zu den Pfaden weitergeführt. Hier wird die Absorption entsprechend der Pfadlängen solange weiter betrieben, bis alle Teilpfade behandelt wurden.

Interne Kontrolle des Bauminhalts durch Kohärenzprüfung

Diese Abläufe sorgen in sich abgeschlossen für die Korrektheit der Verwaltung. Allerdings können externe Gegebenheiten auftreten, welche die Abwicklung störend beeinflussen. Um diese zu kontrollieren, wird an bestimmten Stellen eine Kohärenzprüfung ausgelöst (z.B. bei der Freigabe einer Registrierung eines Nutzers). Sie überprüft im Wesentlichen die Inhalte der Verwaltungsstrukturen und löst gezielt Absorptionsläufe aus. Dabei ist sie fehlertolerant und erfasst geänderte Bedingungen automatisch, solange garantiert kein nicht behebbarer Schaden (z.B. Datenverlust) daraus resultiert (z.B. nicht mehr existierende Dateien werden akzeptiert und die Verwaltung entsprechend angepasst). Insgesamt zeichnet sich das Verhalten jedoch durch ein pessimistisches Grundkonzept aus, welches lieber voreilig einen Alarm auslöst (z.B. bei fehlerhaften, inkonsistenten Verwaltungseinträgen), als unbemerkt in eine unwiderrufliche, kritische Situation zu laufen. Ein Alarm wird durch ein Signal repräsentiert, nach dem keine weiteren Aktionen mehr erlaubt sind, was wiederum bei den Klienten Klärungsphasen auslöst und meist zu einem manuellen Eingriff führen wird (Breakdown-Situation).

In diesem groben Ablaufschema wird bereits deutlich, welcher Verwaltungsaufwand notwendig ist, um konkurrierende Zugriffe zu verwalten. Dies ist wahrscheinlich auch der Grund, weshalb gängige Dateizugangssysteme dies nicht implementieren. Darauf aufsetzend kann nun ein geeignetes Fehlermanagement aufgebaut werden.

4.3.3 Das Fehler- und Ausfallmanagement

Grundprinzip: Selbstverantwortung

Das eigentliche Fehlermanagement basiert auf simplen Mechanismen. Jeder Knoten (Server oder Client) ist selbst dafür verantwortlich, seinen stabilen Zustand zu erhalten. Nur ein Client kann dabei Kontakt zu Servern aufnehmen und evtl. aktiv in die Fehlerhandhabung anderer Knoten (Server) eingreifen. Die Entscheidungsgewalt eines Servers ist damit lokal auf sich begrenzt, die eines Clients betrifft ihn und die angeschlossenen Server.

Grundlage sind verfeinerte Fehlercodes

Grundlage ist ein erweiterter Fehlercode auf der Basis abgefangener und analysierter Ausnahmebehandlungen (Exceptions). TAO bietet hierzu Verfeinerungen durch die sog. Minor-Codes. Sie enthalten weitergehende Informationen über Ausnahmesituationen in TAO in Form eines 32 Bit Wertes (u.a. ORB -Hersteller, Fehlerort und Fehlernummer)[13]. Durch diese gewonnenen Erkenntnisse kann z.B. der Fortschritt eines Kommunikationsflusses bis zum Fehlerfall eingegrenzt werden. Letztendlich ist in den ersten Prototypen bei knotenübergreifenden Problematiken der Mensch die agierende Instanz, indem er Folgeentscheidungen trifft. Die Software ist jedoch schon so ausgelegt, dass dies teilweise automatisiert werden kann.

[13] vgl. dazu [OCI00] a.a.O. S. 128f.

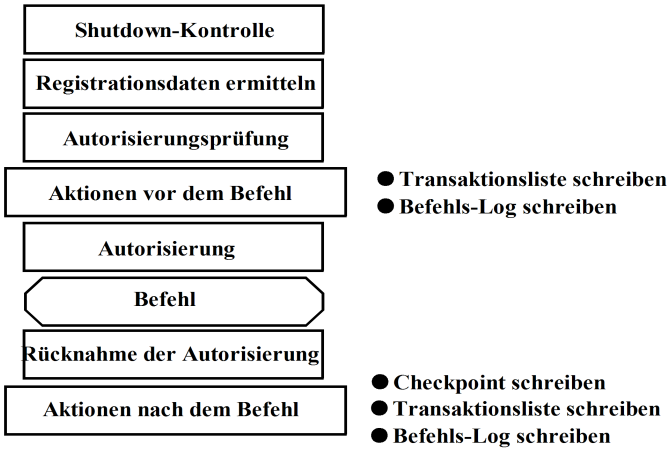


Abbildung 4.5: Aufbau eines Befehls mit Fehlermanagement

Weiterhin besteht die Möglichkeit, mittels der Methoden einer im Fehlerfall ausgelösten Exception-Klasse den Status der Vervollständigung auf die Werte „YES“ (Aktion war komplett vor Fehlersituation), „NO“ (Aktion war noch nicht begonnen) und „MAYBE“ (Aktion war in Bearbeitung) einzugrenzen[14]. Dadurch können einige Klärungsphasen (siehe später) vermieden werden. Da aber noch ein Zustand mit ungewissem Ausgang („MAYBE“) existiert, muss ein zusätzliches zustandsbasiertes Fehlermanagement zumindest je Knoten integriert werden.

Integration eines Zustandsmanagements

Damit jeder Knoten eines Systems selbstständig stabile Zustände erhalten kann, wurde ein Zustandsmanagement integriert. Es dient dazu, ehemalige, stabile Zustände zu lokalisieren und diese im Fehlerfall erneut einzunehmen. Damit können im laufenden Betrieb fehlerhafte Aktionen zurückgenommen werden. Im Falle von Crash-Situationen dient es dem Ausfallmanagement zum Wiederanlauf in einen stabilen Ausgangszustand. Dabei greift es nur für lokal administrierte Anteile. Ein Client mit entfernten Ressourcen tritt bei einem Ausfall dieser somit generell in eine Klärungsphase ein, in der er Verbindungen re-initiiert und den Zustand einer Aktion erfragt.

Kombination von Transaktionen, Logs und Checkpoints

Das Fehlermanagement als solches ist eine Kombination aus Transaktionen, Logs und Checkpoints. Ein Ablauf für verändernde Befehle (eine Zustandsänderung wird herbeigeführt, dadurch dass physikalische Zusammenhänge geändert werden) sieht damit wie in Abb. 4.5 auf Seite 106 aus. Eine Darstellung einer kompletten, serverseitigen Beispielsequenz mit den entsprechend zugehörigen Methodenaufrufen ist im Anhang G auf Seite 247 zu finden. Vor dem Befehl werden die Logs geschrieben. Das betrifft die Ablage der Transaktionsnummer (extern als `Transaction Number (TAN)` und intern als `Transaction Identifier (TID)` bezeichnete Kombination aus Zeitstempel, fortlaufender Zahl und Zufallszahl) einer aktuellen Transaktion und ihres momentanen Status (NOK = not ok) getrennt je Nutzer. Zudem werden der Befehl und die Parameter abgelegt, so dass er nach einem Absturz noch ausgeführt werden kann. Durch ihn sind auch seine Teilkomponenten bekannt, welche beim Wiederanlauf nacheinander abgeprüft

[14] vgl. dazu [OCI00] a.a.O. S. 104

und vervollständigt werden. Jeder Befehl ist deshalb entweder atomar oder in solche eindeutigen Teilschritte zerlegbar, welche auf erfolgte Abarbeitung überprüft werden können oder idempotent sind.

Beispiel 4.2 für „Annahme einer neuen Version“
Die Annahme besteht aus den Teilen „Umbenennen der alten Version in Archivdatei“, „Umbenennen der neuen Version in die gültige“ und evtl. „Löschen der Archivdatei“. Jeder Zustand kann dabei separat geprüft werden. Existiert noch die alte Version? Existiert noch die temporäre Version? usw. Jede Überprüfung kann dabei auch auszulösenden Aktionen zugeordnet werden (alte Version in Archiv umbenennen oder neue Version als neues Original umbenennen).

Nach einem Befehl wird ein Checkpoint geschrieben, auf den nachfolgende Aktionen aufsetzen. Der neue Status einer Transaktion wird übernommen (OK) und der Befehl aus dem Logbuch entfernt. Abstürze innerhalb dieser Sequenz haben keine Auswirkungen auf die korrekte Arbeitsweise, da bei Wiederanlauf die Überprüfung der Einträge konsistente Zustände erzeugt.

Wiederanlaufszenario

Bei einem Wiederanlauf wird das System in den Zustand des letzten, gültigen Checkpoints hochgefahren, worauf alle unvollständigen Aktionen überprüft und vervollständigt werden. Dies führt zu einem stabilen Zustand. Nach einem Wiederanlauf sind alle Registrierungen gelöscht. Ein Client muss sich deshalb wieder neu anmelden und entsprechend seiner Aktion eine Klärungsphase starten, in der er die abgelegten Transaktionsnummern auf ihre Zustände prüft. Kann ein Wiederanlauf nicht erfolgen, wird ein Alarm ausgelöst und der Knoten in diesem Zustand eingefroren. Kein Zugang wird mehr erlaubt, bis ein manueller Eingriff erfolgt ist.

Bei dem geschilderten Vorgehen handelt es sich um einen relativ unkomplizierten Algorithmus. Er hat sich in der Praxis als praktikabel herausgestellt und funktionierte in den Testaufbauten tadellos.

4.3.4 Besonderheiten von ECFT

ECFT wurde entwickelt, um Funktionalität für ein zukünftiges Produktionssystem zu stellen. Deshalb beinhaltet es zusätzliche Erweiterungen, welche die Dienstqualität und die komfortable Handhabung erleichtern. Der Client ist jedoch von seiner Struktur vorerst noch auf zwei Interfaces begrenzt worden, wobei eines mit der Remote-Verbindung und das andere mit dem lokalen Dateisystem belegt ist. Diese Einschränkung stammt aus der Überlegung, FTP nachzubilden.

Der restliche Ausbau wurde stufenweise dem Prototyp hinzugefügt. Da er eine wichtige Basis für ein stabiles System darstellt, werden nachfolgend die wesentlichsten Besonderheiten beschrieben.

Authentizität und Autorisierung

Übernahme der Nutzerrechte aus dem Betriebssystem durch Authentifizierung und Autorisierung

Ein großer Mangel in CFT ist, dass keine exklusive Authentifizierung auf dem Serverrechner möglich ist. Das Einbinden von SSL lockert diese Beschränkung zwar etwas, die Nutzer haben jedoch weiterhin dieselben Rechte, welche von den Zuordnungen zum Serverprozess abhängig sind. Deshalb sind in ECFT ein geeigne-

tes Anmeldeverfahren mit Authentifizierung und die Nutzung der Nutzerrechte aus dem Betriebsystem mit entsprechender Autorisierung integriert. Natürlich bleiben zusätzlich die Möglichkeiten des während der Laufzeit aktivierbaren SSL (sog. „plugable“ SSL, vgl. Abschnitt 3.3.4 auf Seite 71) erhalten.

Generell sind zwei Aufgaben zu erfüllen:

1. Die Authentifizierung überprüft mit Hilfe des Nutzernamens und des Passwortes, ob dem Nutzer die Anmeldung am Server und damit die Nutzung erlaubt wird.
2. Die Autorisierung sorgt dafür, dass jeder Nutzer bei Aktionen nur die Rechte hat, welche ihm vom Betriebssystem erlaubt wurden.

Eingliederung in den Dienstestack des Betriebssystemadapters

Da diese Aufgaben jeweils betriebssystemspezifisch gehandhabt werden müssen, wird dazu der Dienstestack verwendet, welcher die Eigenheiten der Betriebssysteme verdeckt. Die Authentifizierung ist somit eine Methode, welche intern die Systemaufrufe zur Überprüfung eines Nutzers bereitstellt. Sie liefert über den Fehlercode, ob der Nutzer die Erlaubnis zur Anmeldung hat und übergibt in den Parametern seine Nutzeridentifikation und das Heimatverzeichnis.

Während in Windows die Authentifizierung mit der Funktionalität „LogonUser“ ausgeführt wird, ist in Linux/Unix ein Vergleich des eingegebenen und anschließend verschlüsselten Passworts mit dem in der Shadow-Passwortdatei eingetragenen, kryptischen Betriebssystempasswort des Nutzers nötig. Ist der Nutzer authentifiziert, wird eine Registrierung eingetragen und er kann Folgeaktionen ausführen. Im anderen Fall wird er abgewiesen. Beim Abmelden werden die Registrierungen entfernt und der Nutzer wird je nach Betriebsystem explizit abgemeldet.

Autorisierung durch Impersonifikation

Die Überwachung der Rechte eines authentifizierten Nutzers wird durch die Autorisierung gewährleistet. Dabei wird ein Trick angewandt. Zu Beginn einer Aktion schlüpft der Prozess in die Rolle des jeweiligen Nutzers, er „impersonifiziert“ sich also mit den jeweiligen Rechten. Beim Verlassen gibt er diese wieder zurück.

Server läuft als Betriebssystemprozess mit Superuser-Rechten

Damit dies alles möglich wird, benötigt der ausführende Serverprozess Superuser-Rechte. Er muss also als ein Betriebssystemprozess laufen (in Linux/Unix als root-Prozess). Ob dies der Fall ist, wird deshalb auch bei der Authentifizierung überprüft. In Windows muss zusätzlich die Systemvariable „SE_TCB_NAME“ gesetzt sein. Dies wird erreicht, wenn die Nutzergruppe, unter welcher der Server laufen soll, in den lokalen Sicherheitsrichtlinien als Teil des Betriebssystems eingesetzt wird (in der Verwaltung der Systemsteuerung).

Sicherheit durch dedizerte Nutzerbereiche

Die dadurch eingehandelten Sicherheitslücken sind im Regelfall nicht weiter gefährdend, weil jeder Nutzer nicht einfach beliebige Kommandos ausführen kann, sondern nur die in der Schnittstellenbeschreibung spezifizierten Befehle nutzen darf. Es ist nur dem sog. Superuser (ein mit Administrationmöglichkeiten ausgezeichneter Nutzer) erlaubt, eine komplette Sicht auf einen Server zu erhalten und damit jedes beliebige Kommando auszuführen. Für alle anderen Anwender kann sogar der Pfadabstieg in Wurzelverzeichnisse außerhalb des Nutzerbereichs verboten werden („Homelock“). Ein zusätzlicher Schutz z.B. im WAN ist zudem durch den Einsatz von SSL geboten.

Mögliche Erweiterung um verteilte Authentifizierung

Durch diese Maßnahmen werden die Mängel behoben, und es wird ein komfortables Verwaltungssystem für die Nutzerrechte über das Betriebssystem bereitgestellt. Für zukünftige Versionen ist es auch denkbar, extrem sichere, verteilte Authentifizierungsmechanismen, wie z.B. das im `Request For Comment (RFC)` 1510 als Internet-Standard vorgesehene Kerberos einzusetzen. Mittels netzwerkweiter, eindeutiger Identitäten (Principals) müßte sich ein Anwender damit nur noch einmal im Netzwerk ausweisen, um dann transparent Zugriff auf verschiedene Ressourcen zu erhalten. Die integrierte Verschlüsselung und die sicheren Mechanismen böten zusätzliche Sicherheit[15]. Allerdings ist der Aufwand einer Integration von Kerberos in der vorliegenden Arbeit nicht vorgesehen.

Das Logbuch

Laufzeitprotokollierung

Zur besseren Kontrolle von Gegebenheiten während der Laufzeit ist es nützlich, eine Laufzeitprotokollierung mit in das Programm aufzunehmen (Laufzeit-Logbücher). Ein entsprechend gestaltetes Log-File liefert weit mehr Informationen über den Ist-Zustand eines Systems und dessen Probleme als das herkömmliche Debugging.

Protokoll ist logisch geordnet, liefert aber keine harten Echtzeitinformationen

Durch Compiler-Direktiven kann die Erstellung von Protokollen generell aktiviert oder deaktiviert werden. Es ist dabei zu beachten, dass die Ausgabe von Protokollen einen zusätzlichen Zeitaufwand kostet und das Ergebnis mit Zeitangaben mehr oder weniger verfälscht.

Die Ausgabe beinhaltet Angaben in folgender Reihenfolge:

1. **Zeitstempel:** Zeitangabe mit Datum und lokaler Uhrzeit (msec); Beispiel: 2003.04.17 15:27:25:959
2. **Rechner/Port:** Angabe des Rechnernamens bzw. der IP-Adresse und der Portnummer; Beispiel: neidht:12456
3. **Programm/PID:** Name des Programms und evtl. der Prozess-ID; Beispiel: WDAServer.exe 23750
4. **User/UID:** Nutzername mit Nutzer-ID; Beispiel: Administrator 41
5. **Meldung:** Ausgabe des Meldungstextes mit eindeutiger Kennzeichnung der Meldungsart; Beispiel: [EVENT] Proceed context order: OrderID = 123 Parameter = [T1050586045-0-12537T] ls

Meldungen staffeln sich nach Prioritätsstufen

Die Meldungen selbst werden an festgelegten Stellen (z.B. beim Methodeneintritt und -austritt oder bei Fehlerzuständen) an die Standardfehlerausgabe geschrieben. Sie werden in Prioritätsgruppen eingeteilt. Eine niedrige Zahl bedeutet dabei hohe Priorität. Die Ausgabe kann über eine obere Prioritätsgrenze (Ausgabe-Level) eingeschränkt werden. Alle Meldungen mit höherer Priorität als der Ausgabe-Level, (d.h. mit kleinerem Zahlenwert) werden geschrieben. Festgelegt wurden die Prioritätsstufen und damit Meldungskennzeichnungen:

[15] vgl. dazu [WAE04]

1. **LOG_OFF (= 0):** Keine Protokollierung durchführen
2. **LOG_ALERT (= 1):** Ausgabe von kritischen Systemzuständen, die zu Absturz führen oder einen manuellen Eingriff erfordern (z.B. wichtige Systemdatei fehlt plötzlich). In diesem Zustand wird kein Auftrag mehr angenommen.
3. **LOG_ERROR (= 2, einschl. 1):** Ausgabe von Fehlerzuständen, welche erkannt wurden, aber keine unmittelbare Bedrohung darstellen (z.B. Nutzer hat falsches Passwort angegeben). Fehlermanagement hat wirkungsvoll eingegriffen.
4. **LOG_EVENT (= 3, einschl. 1, 2):** Ausgabe bei Erreichen eines markanten Programmpunkts (= Ereignis ; z.B. Übertragung initiiert). Hier werden übergeordnete Zustandsfolgen sichtbar, wie sie z.B. in den Zustandsdiagrammen in UML definiert sind.
5. **LOG_TRACE (= 4, einschl. 1, 2, 3):** Ausgabe einer Spur mit jeweiligen Ein- und Aussprungpunkten in Funktionen mit Einrückung je nach Aktivierungstiefe (z.B. BEG iRead, END iRead). Diese Ausgabe dient zur Erzeugung von sog. Gantt-Diagrammen (siehe weiter unten).
6. **LOG_DEB (= 5, einschl. 1, 2, 3, 4):** Ausgabe von zusätzlichen Debugging-Informationen (z.B. Parameterinhalte, Varibalenwerte, funktionsinterne Zustände und Entscheidungen). Hier wird das Bild des momentanen Zustands nahezu komplettiert, da momentane Gegebenheiten beliebig granular ausgegeben werden können.

Gantt-Diagramm

Das erwähnte Gantt-Diagramm (vgl. Abb. 4.6 auf Seite 111) wurde von H. L. Gantt 1919 entworfen, um Leistungsaussagen über die Organisation industrieller Prozesse zu treffen. Diese Diagramme visualisieren detaillierte Leistungsaussagen in den Dimensionen Ort (Recheneinheit, Prozess, Programm) und Zeit. Dadurch, dass Gantt-Diagramme die zeitlichen Wechsel verschiedener Zustände über einer gemeinsamen Zeitachse darstellen, werden sie auch als Zeit-Zustands-Diagramme bezeichnet[16].

Gantt-Diagramme als Laufzeitzuordnung der Sequenzdiagramme

In mit UML entworfenen Softwarestrukturen kann gerade diesen eigentlich UML - fremden Diagrammen eine besondere Bedeutung zukommen. Sie visualisieren nämlich eine zur Laufzeit erfasste, zeitliche Zuordnung einer Sequenz von Ereignissen. Durch eindeutige Abbildung der Ereigniskennungen auf die Methoden der Klassen (jeder Methode wird eine eindeutige Kennung zugewiesen) kann so ein Zusammenhang zu den im Design erstellten Sequenzdiagrammen und damit eine praktische Verifizierung der theoretischen Vorgaben zur Laufzeit erfolgen. Sequenzdiagramme (als Beispiel vgl. Anhang G auf Seite 247) beschreiben einzelne Abläufe anhand der Reihenfolge darin auftretender Nachrichten entlang der Zeit. Nachrichten sind hierbei Methodenaufrufe zwischen den Klassenobjekten. Ein Sequenzdiagramm ist damit ein Pfad durch einen Entscheidungsbaum[17]. Durch die während der Laufzeit ermittelten Gantt-Diagramme können somit diese Pfad unter realen Bedingungen verifiziert werden. Zur Realisierung wird die Logbuchausgabe

[16] vgl. [KLAR95] a.a.O. S. 135 f.
[17] vgl. [LEX04] unter . . . /Sequenzdiagramm.html (06.01.2005)

„LOG_TRACE“ verwendet, da dadurch gewährleistet ist, dass nach einem Rücksprung aus einer Methode auch wieder der Zustand vor der Aktivierung im Gantt-diagramm eingenommen wird.

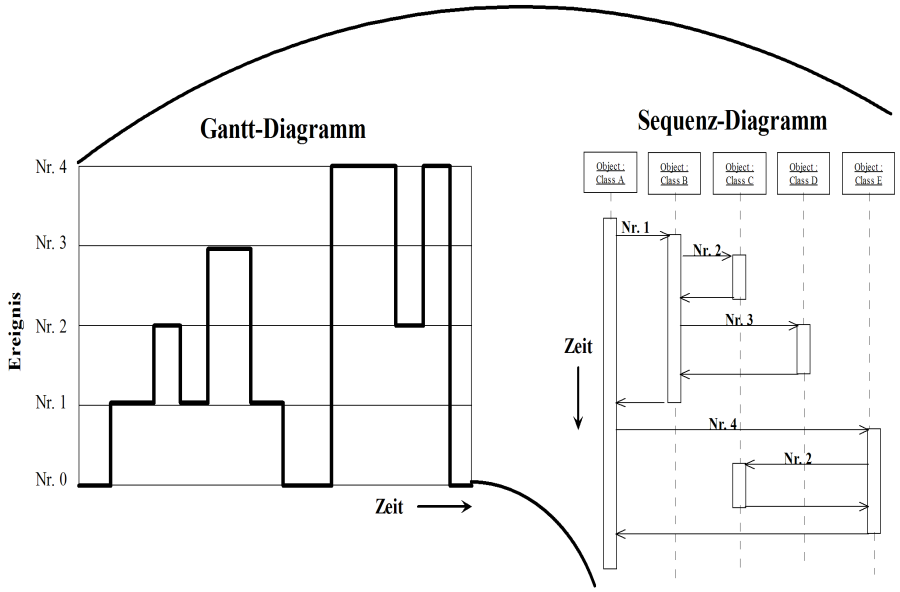


Abbildung 4.6: Schematische Entsprechung eines Gantt-Diagramms mit einem Sequenzdiagramm

Laufzeitbewertung soll in Zukunft generell in Entwicklungen der Fundamentalstation integriert werden

Durch den Einsatz dieser Logbücher wird eine effektive und umfangreiche Möglichkeit gegeben, bestehenden Code unter Laufzeitbedingungen zu bewerten. Allerdings sei nochmal erwähnt, dass die Zeitangaben ein um die Zeit des Eintragens des Logbuchtextes verfälschtes Bild darstellen. Auch sind noch keine Maßnahmen enthalten, Messreihen über Verteilungsgrenzen eines verteilten Systems hinweg zu erstellen. Allerdings kann bereits hier erwähnt werden, dass momentan die im Rahmen dieser Arbeit entwickelte Grundform eines Logbuchs für den allgemeinen Einsatz in der Fundamentalstation Wettzell im Zusammenhang mit Entwicklungen in C++ ausgebaut wird und in zukünftigen Entwicklungen generell zum Einastz kommen soll.

Der erweiterte Funktions- und Befehlsumfang

Erweiterter Befehlssatz mit Skriptsteuerung

Zu allen bisher beschriebenen Erweiterungen kommt noch ein ausgeweiteter Befehlssatz mit entsprechender Funktionalität hinzu. In diesem Zusammenhang ist auf Clientseite vor allem nochmal die ermöglichte Nutzung einer Skriptsprachendatei zu nennen. Sie beinhaltet eine lineare Folge der Kommandozeilenbefehle, welche schrittweise eingelesen und abgearbeitet werden. Tritt ein Fehler auf, stoppt die Verarbeitung. Zusätzlich wurde die Möglichkeit der Angaben einer Round-Trip-Delay geschaffen, welche eine Deadline für den kompletten Nachrichtenumlauf setzt und über eine Richtlinie für Zeitüberschreitungen (Timeout-Policy) in TAO gesetzt wird[18].

[18] vgl. [OCI00] a.a.O. S. 162 f.

Die zu CFT zusätzlichen Befehle (vgl. auch Abschnitt 3.3.4 auf Seite 69) sind:

1. Eine Möglichkeit zur Erkennung des Dateityps (binär oder textbasiert als ANSI/ASCII). Dazu wird der Inhalt der Datei durchlaufen und dahingehend geprüft, ob Zeichen außerhalb der ANSI-Spezifikation enthalten sind (z.B. Steuerzeichen), was auf eine Binärdatei hindeutet. Dieser Durchlauf kostet zwar etwas Verarbeitungszeit, funktioniert jedoch fehlerfrei. Eine explizite Prüfung kann über den Befehl „type“ erfolgen. Sollen grundsätzlich vor den Übertragungen solche Kontrollen stattfinden, so kann mittels „autotype“ eine entsprechende Einstellung gesetzt werden.

2. Über „set“ können variable Einstellungen definiert werden. Dazu zählen momentan die Größe der Übertragungsblöcke („set block“) in Bytes und die Grenzgröße in Bytes, ab der eine Dateiübertragung als akzeptiert und brauchbar gilt („set accept“). Letzteres erlaubt es, bei schlechten Datenleitungen zumindest Fragmente der Datei zu retten, auf die später evtl. wieder aufgesetzt werden kann (die Fortführung einer Dateiübertragung ist jedoch noch nicht implementiert).

3. Über den Befehl „tans“ kann der Gebrauch von Transaktionen für alle verändernden Zugriffe erzwungen werden. Nach einem Ausfall einer Teilkomponente, kann der Client über „tanstatus“ eine Klärungsphase einleiten. Dabei wird als Parameter die Transaktionsnummer der fehlerhaften Aktion übergeben und beim Server nachgefragt, welchen Zustand sie eingenommen hat (fertig oder nicht).

4. Eine weitere Ergänzung betrifft die Handhabung des lokalen Interfaces. Dafür wurden Pendants zu den Remotebefehlen geschaffen, die sog. „l“-Kommandos (für lokal). Dazu zählen „lcd“, „ldir“, „lls“, „lpwd“, „ltanstat“ und „ltype“. Ihre Funktionalität entspricht denen der Befehle für das entfernte Dateisystem.

5. Zur Vereinfachung des Laufzeitdebuggings wurde im Client der Befehl „verbose“ direkt mit der Online-Ausgabe des Logbuchs auf oberster Stufe „LOG_DEBUG“ nach Standard-Out verknüpft.

6. Verfügt der Nutzer beim Server über voreingestellte Superuser-Rechte, so kann er über „shutdown“ den Server regulär herunterfahren. Diese Möglichkeit ist dafür gedacht, evtl. eine automatische Instandhaltungsmaßnahme (Maintenance-Task) zur Kontrolle des Servers zu implementieren. Diese Task soll zur Vereinfachung dieselbe Struktur wie ein Client haben.

7. Entsprechend der neuen Gegebenheiten wurde auch „help“ und „status“ ausgeweitet.

Zusammengefasst ergibt sich somit eine Ansammlung nützlicher Befehle, welche die aus FTP übernommenen Kommandos praktikabel erweitert.

4.4 Eine weiterführende Idee: Der „High Speed Data Channel“

Paralleler, weniger überwachter Kanal zur schnellen Übertragung der Daten

Aufgrund der regelmäßig durchgeführten Transfermessungen und dadurch erkennbarer Einbussen im Vergleich zu anderen Übertragungsmechanismen (vgl. dazu Abschnitt 3.5.3 auf Seite 77) wurde die Idee geweckt, parallel zum geschaffenen CORBA -Transfer einen zweiten Datenkanal auf TCP /IP zu nutzen. Über diesen Weg könnten Daten ohne den doch relativ zeitaufwändigen CORBA -Abarbeitungsweg übertragen werden, weshalb ein erheblicher Geschwindigkeitsgewinn zu verzeichnen sein sollte (d.h., ein „High Speed Data Channel“ wäre ermöglicht), der u.a. bei Windows an die Übertragungsleistungen des herkömmlichen FTP heranreichen sollte .

Nutzung des Audio/Video Streaming Service von TAO

Die Suche nach existierenden Möglichkeiten führte zum Audio/Video Streaming Service von TAO , welcher zum Übertragen von großen Datenmengen unter gegebenen Echtzeitbedingungen entwickelt wurde. Die Idee zum Einsatz wurde in Form einer Diplomarbeit vergebe. Es sollte nachgewiesen werden, dass sich der Service in das bestehende Konzept einbinden lässt bzw. dass sich die in der Literatur angegebenen Transferraten bestätigten[19].

Wunschvorstellung bestätigte sich nicht, wobei offizielle Angaben hierzu gegenteiliges zeigen

Das Ergebnis der Arbeit bestätigte dies allerdings nicht und führte zu einem erheblich komplexeren Design [20]. Insbesondere die Integration der Multimediageräte mit ihren Datenstromendpunkten, welche als Hauptbestandteile des Audio/Video Streaming Services in der vorliegenden Anwendung für allgemeine Daten zweckentfremdet wurden, führte zu zahlreichen weiteren Klassenkonstrukten. Die auf sog. „Callback“-Methoden (Funktionen, welche beim Eintreffen von Daten aktiviert werden) basierenden Datenendpunkte mit den dafür nötigen Datenpuffern und -kopiervorgängen verkomplizieren die Anwendung und können zudem auch zu Leistungseinbussen führen[21]. Trotzdem sind die erhaltenen Ergebnisse mit Vorsicht zu bewerten, weil offizielle Quellen davon weitgehend abweichen. Zudem wies die während der Diplomarbeit entstandene Testsofware noch zahlreiche Laufzeitfehler auf, so dass Tests nur bedingt bewertet werden konnten. Aus der Überlegung heraus sollte sich eigentlich eine Verbesserung ergeben, weil weniger Speicheranforderungen existieren, das mehrfache Kopieren von Daten im CORBA -Stack vermieden wird und kein Marshalling/Demarshalling stattfindet. In Auswertungen sollte deshalb nahezu die Latenzzeit des TCP /IP -Protokolls erkennbar sein[22].

Grund ist wohl im Design und der Implementierung der Streaming-Integration zu suchen

Die Ursache für die fehlende Übereinstimmung dazu liegt wohl in der umgesetzten Erweiterung des bisherigen Designs durch die zusätzlichen „Audio/Video Streaming“-Konstrukte. Die Nutzung von Callback-Funktionen widerspricht im Grunde dem rein sequentiellen Vorgehen des bisherigen ECFT , so dass an den Kopplungsstellen wahrscheinlich die Leistungseinbussen hinzunehmen sind. Da offizielle Quellen die Grundidee bzgl. der Übertragungsleistungen eigentlich bestätigen, sollte es evtl. bei einer geschickteren, qualitativ hochwertigeren Integration durchaus zu den vermuteten Geschwindigkeitszunahmen kommen. Eine Feststel-

[19] vgl. [ANS02] a.a.O. S. 29 f.
[20] vgl. [ANS02] a.a.O. S. 74
[21] vgl. [ANS02] a.a.O. S. 44 f.
[22] vgl. [MUNG04] a.a.O. S. 11 f.

lung ist jedoch nicht von der Hand zu weisen: Bei Nutzung des Services bindet man sich stärker an die CORBA -Implementierung TAO [23].

4.5 Ergebnisse

4.5.1 Erhaltene Systemunabhängigkeit des Codes

Anzahl der Codezeilen je Modul erneut Maßzahl für Transparenz

An dieser Stelle soll noch einmal die Anzahl der Codezeilen im Verhältnis zueinander betrachtet werden (vgl. Abb. 4.7 auf Seite 114). Dies ist vor allem deshalb sinnvoll, da der eigene, geschriebene Code stark zugenommen hat (nimmt ca. 2/3 des Gesamtcodes von rund 46.000 Zeilen ein) und enorme Umstrukturierungen stattgefunden haben.

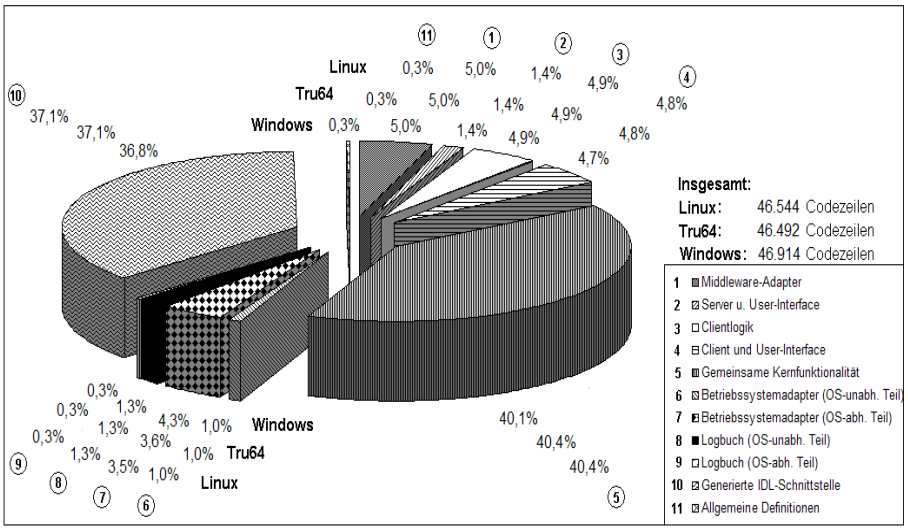


Abbildung 4.7: Verteilung der Codezeilen auf ECFT-Module

Die selbe Kernfunktionalitäten für Client und Server

Es ist ersichtlich, dass die Berührungspunkte zum Betriebssystem immer noch gering sind, auch wenn der Funktionsumfang im Betriebssystemadapter enorm angewachsen ist. Ersichtlich werden das Konzept der Bündelung von Funktionalität mit bestimmten Kompetenzen und seine Wirkungsweise. Die Kernfunktionalität (ca. 40 %) ist deshalb sowohl für Client als auch für Server gleich und unabhängig von Hardwaregegebenheiten. Dies hat extreme Vorteile bei der Weiterentwicklung, da nur an einer Stelle geändert werden muss. Tests der Hauptfunktionalität können mit einem der Kommunikationspartner ausgeführt werden. Im Kern befinden sich die beschriebenen Sicherungsmechanismen der Dienstgüte. Die zweite Säule bildet weiterhin die IDL-Schnittstelle und deren Code (ebenfalls ca. 40 %).

Individuelle Codeanteile gering

Auch der individuelle Code sowohl des Clients als auch des Servers wurde erweitert und umstrukturiert, so dass sie eine höherwertige Dienstleistung bilden. Trotzdem nehmen sie nur einen geringen Anteil ein. Die Zugangspunkte zum Betriebssystem sind relativ schmal gehalten (z.B. 4% beim Betriebssystemadapter)

[23] vgl. [ANS02] a.a.O. S. 74

und werden durch die Schichtung in der Kernfunktionalität ausgebaut. Zur leichten Wiederverwendung des Logbuchs wurde aber ein unabhängiger Zugang belassen.

Durch den Gesamtumfang von ECFT im Vergleich zu CFT (Zunahme um den Faktor 2,4) wird ersichtlich, wie viel Aufwand notwendig ist, um Systeme mit der nötigen Dienstqualität auszustatten. Technologietransparenz wurde hierbei durch die Bündelung und Homogenisierung der Funktionalität erreicht, so dass jeder Knoten im Kern aus den selben Grundmodulen aufgebaut ist und nur geringe, unbedingt erforderliche Individualeigenschaften aufweist (weit über 90% des Codes sind unabhängig von Systemeigenschaften).

4.5.2 Beispielanwendung mit internationalen GPS-Permanentstationen

Test unter realen Bedingungen

Wie schon bei CFT sollte der Prototyp in einer realen Umgebung getestet werden. Durch die weltweite Verteilung betreuter GPS-Permanentstation bot sich dieser Einsatzzweck erneut an. Die Stationen werden komplett von Wettzell aus betrieben und gewartet, so dass ein Zugriff und eine Installation keine Auswirkungen auf externe Partner haben. Zudem können somit reale Bedingungen getestet werden, wie sie im WAN vorherrschen. Ergebnis sollte sein, dass der Prototyp funktioniert und neben herkömmlichen Systemen bzgl. seiner Leistung bestehen kann, dabei aber die gewünschte Dienstqualität liefert. Diese wurde vor der Inbetriebnahme der Teststrecken lokal getestet, indem Übertragungen mit Fehlerfällen simuliert wurden. Speicherverbrauch und Programmierfehler etc. konnten beobachtet und behoben werden.

Abbildung 4.8: Karte der Teststationen im klassischen GPS-Netz

Zusätzlicher Aufwand aufgrund individueller Charakterisktiken der GPS-Stationen

Die ersten Versuche zeigten jedoch, dass zusätzliche Elemente berücksichtigt werden müssen. Da Programmfehler nie ganz auszuschließen sind, muss ein Knoten vor Ort dafür sorgen, dass dadurch zumindest kein Deadlock entsteht. Eine

überlagerte Ausnahmebehandlung über den gesamten Code sorgt deshalb dafür, dass sich der fehlerbehaftete Client beendet. Ansonsten mussten die richtigen Stellwerte z.B. für die Blockgröße ermittelt werden. Zu große Blöcke und zu kleine Blöcke verursachen nämlich lange Transferzeiten bei schlechten Langstreckenverbindungen (genutzt werden momentan Blöcke im 100KByte-Bereich). Die vor Ort vorliegenden Runtime-Bibliotheken der Messrechner mit Windows NT 4.0 mussten mit neueren Versionen überladen werden. Alles in allem war dadurch ein nicht in dieser Form erwarteter Zusatzaufwand nötig.

Testszenario

Bei der eigentlichen Übertragung werden die Stundendatensätze in Form von Dateien jeweils zur vollen Stunde nach Wettzell sowohl im Rahmen des herkömmlichen Produktionssystems als auch zum Testsammler geschickt. Das Versenden wird über ein ECFT -Skript gesteuert. Ein Perl-Programm stößt die Übertragung an und erstellt zuvor das besagte Skript. Es zeigte sich (wie schon bei den lokalen Versuchen), dass die Integration der neuen Software hier kein weiteres Problem darstellte. Während im Produktionssystem die Daten weiterverarbeitet werden, werden sie im Testsammler nur archiviert.

Für die Tests wurden die folgenden Stationen (vgl. Abb. 4.8 auf Seite 115) ausgewählt:

- **Lhasa/Tibet:** Da dort die GPS-Station (vgl. Anhang H auf Seite 249) während der Projektzeit der vorliegenden Arbeit erneuert wurde und währenddessen erste Tests bereits positiv verliefen.
- **Reykjavik/Island:** Die Station lief seit einiger Zeit sehr stabil. Allerdings kam es genau während der Testzeit zu einem Ausfall des GPS -Empfängers.
- **Concepción/Chile:** Dort wird eine weitere Fundamentalstation in Form eines Transportablen Integrierten Geodätischen Observatoriums (TIGO) betrieben, so dass bei Problemen fachkundiges Personal vor Ort ist.
- **Helgoland/Deutschland:** Um eine kleinere Entfernung innerhalb des deutschen Weitverkehrsnetzes zu Testen, wurde diese Station mit in den Test aufgenommen.

	Lhasa	**Reykjavik**	**Helgoland**	**Concepción**
GPS -Steuerungs-Software	GPS Base	GPS Base	GPS Base	GPS Base
Aktionsscheduler	Launchpad	Launchpad	crons	Launchpad
Fernwartung	PolyPM (Telefonleitung)	PolyPM Internet	PolyPM Internet	PolyPM Internet
FTP -Client	ncftp	Perl-Modul	ncftp	Perl-Modul
Anbindung	ADSL*	Uni-Netz (Glasfaser)	ISDN**/TDSL***	Uni-Netz (Kupfer)
Betriebssystem	Windows NT 4.0	Windows NT 4.0	Windows NT 4.0	Windows NT 4.0

*Asymmetric Digital Subscriber Line (ADSL)
**Integrated Services Digital Network (ISDN)
***Telekom Digital Subscriber Line (TDSL)

Tabelle 4.1: Konfigurationen auf den GPS-Messrechnern

Die dort gegebenen Umgebungsbedingungen sind in Tabelle 4.1 auf Seite 116 dargestellt. Insgesamt lief der Server über 7 Monate ohne Unterbrechung kontinuierlich durch. Die Ergebnisse der Messungen stammen aus den ersten beiden Monaten des Tests und sind aufschlussreich und teilweise überraschend.

Über 7 Monate ausfallfreier Serverbetrieb

4.5.3 Die Transferleistung anhand vergleichender Messungen in einer Produktionsumgebung

Bisher wurden die Prototypen zwar unter realen Bedingungen getestet, für die Leistungsmessungen galten im Allgemeinen aber speziell dafür ausgelegte Szenarien, welche zudem in einer eng begrenzten, lokalen Umgebung (LAN , vgl. Abschnitt 3.5.3 auf Seite 77) stattfanden. An dieser Stelle sollen nun die aus der Einbindung in das reale, globale Produktionssystem des klassischen GPS -Netzes ermittelten Messungen zur Bewertung genutzt werden. Sie lassen weitere Rückschlüsse zu, wie sich ein entsprechend dimensionierter Prototypaufbau in die bestehende Umgebung einfügt und welche Leistungsmerkmale er darin aufweist. Für die Messungen konnten keine experimentellen Versuche mit unterschiedlichen Übertragungsprogrammen durchgeführt werden, da dies den Ablauf im Produktionsbetrieb gestört hätte. Stattdessen wurde eine einheitliche Variante der ECFT -Übertragung mit der jeweils vor Ort eingesetzten FTP -Übertragung verglichen.

Verhalten im realen Produktionsbetrieb

Die Versuchsreihen (vgl. dazu auch Anhang J auf Seite 253) im WAN waren von einigen Widrigkeiten geprägt, welche jedoch charakteristisch für die weltweit verteilten Permanentstationen im GPS -Verbund und ihre individuellen Infrastrukturen sind. Ausfälle z.B. werden zwar bemerkt, können jedoch aufgrund der Unzugänglichkeit einiger Stationen nicht immer sogleich behoben werden. Vor allem Hardwarestörungen stellen bei manchen Stationen ohne Betreuung vor Ort Probleme dar. So war zum Beispiel der erste Abschnitt der Messungen in Reykjavik davon geprägt, dass der GPS -Empfänger defekt war und nur abschnittsweise Daten lieferte. Von ähnlichen Schwierigkeiten waren auch die Messungen in Lhasa/Tibet betroffen. Die extremen Entfernungen machen es zudem schwierig, Wartungsarbeiten mit Fernzugriffen über Telefon durchzuführen und auch das meist wenig qualifizierte Personal vor Ort kann oft nur teilweise behilflich sein.

Messungen von Hardware-Problemen überschattet

Aus einer ersten Überblicksbewertung können bereits allgemeine Erkenntnisse bzgl. der Implementierung und darin noch enthaltener Fehlstellen abgeleitet werden.

Generelle Bewertung der Implementierung anhand der Messungen

Durch die Testinstallationen konnten in der neuen Software ECFT Problemstellen und Anfangsmängel festgestellt werden. Zum einen gibt es noch kleinere Programmfehler. Dazu zählt vor allem, dass es trotz vorhergehender, erfolgreicher, lokaler Dauertests hauptsächlich in Concepción/Chile zu absturzbedingten Ausfällen des neuen Clients kam. Eine fehlerhafte Speichernutzung ist die Ursache für diesen sporadischen Fehlerfall, der zwar vom Gesamtsystem abgefangen wird, jedoch anfangs zu Blockaden vor Ort führte. Der Sammelserver lief jedoch während des Bewertungszeitraums über ca. zwei Monate ohne Fehlerfall durchgehend.

Erkennen kleinerer Programmfehler

Erkannte lineare Zunahme der Übertragungszeiten aufgrund der Umsetzung des Sichtenkonzepts war behebbar

Zum anderen gibt es einige umsetzungsbedingte Folgeerscheinungen aus dem Design. Es konnte eine Drift in den Übertragungszeiten festgestellt werden. Diese lineare Verschlechterung kommt dadurch zu Stande, dass z.B. Überprüfungen der Existenz von Dateien (wie für die Versionsüberwachung genutzt) auf die elementaren Methoden des Betriebsystemadapters abgebildet werden. Dazu wird eine Methode zum Lesen des Verzeichnisinhalts genutzt, deren Rückgabe nach den gesuchten Namen durchforstet wird. Mit zunehmender Anzahl von Dateien eines Verzeichnisses, verbraucht diese Abfrage kontinuierlich mehr Zeit, wodurch der lineare Verlauf entsteht. Nachgewiesen konnte dies werden, indem nach einer gewissen Zeit am 22.07.2004 zwischen 16:00 und 17:00 Uhr (`Universal Time Coordinated (UTC)`) alle Messdateien in ein anderes Verzeichnis kopiert wurden. In der Messreihe ergab sich dadurch bei allen Übertragungen um 17:00 Uhr (`UTC`) ein klar sichtbarer Sprung in den Übertragungszeiten hin zum ursprünglichen Verhalten, von dem aus der lineare Anstieg wieder begann. Dieses Problem konnte letztendlich bereits durch abgewandelte Methoden im Betriebssystemadapter behoben werden, welche z.B. anhand des Erfolgs oder des Scheiterns einer Abfrage der Datei-/Verzeichnisattribute die Existenz einer Ressource überprüfen.

Vorteil durch Einheitlichkeit

Die Probleme sind damit als nicht kritisch einzustufen und durch kleinere Ausbesserungsarbeiten behebbar. Es ist jedoch bereits jetzt klar erkennbar, dass mit Hilfe von `ECFT` die heterogene Vielzahl von unterschiedlichen, jeweils vor Ort genutzten `FTP` -Clients verhindert und durch eine, für alle Stationen einheitliche Umsetzung ersetzt werden kann. Allerdings ist noch die Übertragungsleistung im Vergleich zu den bisherigen Verfahren zu betrachten.

Bewertung der Transferleistungen durch Vergleich

Korrektur der Messungen um die linearen Abweichungen und anschließender Vergleich

Neben der Bewertung des umgesetzten Codes kann auch die Transferleistung des neuen Systems selbst im Vergleich zum jeweiligen, vor Ort eingesetzten Übertragungsmechanismus betrachtet werden. Da oben beschriebene Laufzeiteffekte (wie z.B. die linearen Driften) des Prototyps erst in echten Umgebungsszenarien sichtbar werden und im späteren System nicht mehr enthalten sind, mußten die Messwerte um dieses lineare Fehlverhalten korrigiert werden. Eine genauere Betrachtung der Messungen zeigt, dass sich in den Übertragungscharakteristika in etwa ein Verlauf ergibt, welcher sich täglich wiederholt. Deshalb wurden zum Beginn und zum Ende eines linearen Anstiegs jeweils eine passable Sequenz (ohne große Ausreißer) über 24 Stunden zur Mittelwertbildung genutzt. Die so erhaltenen Punkte bilden eine Ausgleichsgerade für den betrachteten Abschnitt. Nach der Ausgleichung bleiben die zu erwartenden Übertragungszeiten übrig. Sie dienen als Basis für die statistischen Auswertungen und die Vergleiche der Übertragungsverfahren.

Messszenario

Für die Messungen selbst wurden die zu jeder vollen Stunde stattfindenden Übertragungen der in Stundendateien zusammengefassten Messwerte genutzt. Dabei handelt es sich um gepackte `ASCII` -Dateien. Die Übertragungen wurden nacheinander sowohl mit `FTP` im normalen Produktionssystem und mit dem neuen `ECFT` zu einem zusätzlichen Sammler durchgeführt. Gemessen wurde jeweils die Zeit vom Start des Clientprogramms bis zu seinem Beenden, was als effektive Latenzzeit einer Übertragung angesehen werden kann. Enthalten sind darin die Anmeldung, die `IOR` -Abfrage, die Registrierung und sämtliche Voreinstellungen

im Script (z.B. Verzeichniswechsel). Bei ECFT wurden zusätzlich die Zeitspannen eines PUT-Befehls mitprotokolliert, was mit geringem Overhead der reinen Übertragung über den Protokollstack entspricht. Bei der Auswertung wurden in längeren Lücken ohne Werte (z.B. bei deaktiviertem Client) pro Stunde äquidistante Stützstellen mit Nullwerten eingefügt.

Aus den Messungen können damit folgende Feststellungen gemacht werden: Erkenntnisse:

1. Die Versuche mit der Permanentstation auf der Insel Helgoland in der Nordsee lieferten die besten Ergebnisse. Es zeigte sich der erwartete Verlauf, welcher von den Messungen im LAN (vgl. dazu Abb. 3.15 in Abschnitt 3.5.3 auf Seite 80) abgeleitet werden kann. Der genutzte ncftp-Client zeigt beim „PUT“ auch hier, wie schon im LAN , ein bedeutend schlechteres Verhalten als die ECFT -Variante.

   Helgoland erfüllt Erwartungen ideal

2. Es ist generell in allen Messungen zu erkennen, dass sich die Gesamtlatenz mit 2/3 auf den PUT-Befehl und mit 1/3 auf den umgebenden Overhead aufteilt. Diese Aufteilung ist bei allen Stationen in etwa gegeben. In den Stationen, in denen ein anderer FTP -Client (z.B. ein Perlmodul) genutzt wurde, ist die effektive Gesamtlatenz von ECFT wesentlich schlechter. Die reine Übertragung ist (wie schon in den lokalen Messungen) vergleichbar. Diese langen Laufzeiten in der Summe aller Clientaktivitäten können in zukünftigen Weiterentwicklungen evtl. dadurch etwas gemildert werden, dass statt dynamischer Bibliotheken statische genutzt werden, zum Kontaktieren des Servers fest vorgegebene IOR zum Einsatz kommen und der Client von zusätzlicher Funktionalität abgespeckt wird. Letzteres ist möglich, da er hauptsächlich PUTs durchführt und damit nicht die gesamten administrativen Anteile eines Servers benötigt. Allerdings darf die Umstellung nicht auf Kosten des Designs gehen. Es soll weiterhin die Kernfunktionalität von Client und Server aus den selben Komponenten aufgebaut bleiben.

   Aufteilung der Gesamlatenz mit 2:1 auf eigentliche Übertragung und Overhead

3. Die Station Lhasa zeigt eine seltsame Inversion ihres Verhaltens (vgl. folgenden Logbuchausschnitt). Lokale Aufrufe (z.B. „lcd“) dauern extrem lange, während remote betriebene Aufrufe (z.B. „cd“) und die eigentlichen Übertragungen relativ dazu mit akzeptabelem Zeitverbrauch laufen. Dadurch entstehen extrem lange lokale Aktivitäten, welche die effektiven Übertragungszeiten stark beeinträchtigen. Allerdings ist anzumerken, dass bei den Messversuchen während des Reparaturaufenthalts in Lhasa (vgl. dazu Anhang I auf Seite 251) die erhaltenen Werte für die Übertragung auch wesentlich besser waren, wobei dazu auch ein eigener, externer Rechner genutzt wurde. Dies lässt darauf schließen, dass die Ursachen für die Inversion und die Verschlechterungen aus evtl. existierenden Nebeneffekten durch lokal im Messrechner genutzte, andere Programme oder zugehörige Bibliotheken stammen. Dass IIOP generell ein Problem darstellt, ist unwahrscheinlich, da es in anderen Teststationen keinerlei Anzeichen dafür gibt. Die wirkliche Ursache für das Verhalten konnte bisher nicht geklärt werden, da ein direkter Zugriff auf den Messrechner in Lhasa aufgrund der Entfernungen sehr langwierig ist und es auch regelmäßig zu längerfristigen Ausfällen kommt.

   Unerklärliches Inversionsverhalten der Station in Lhasa

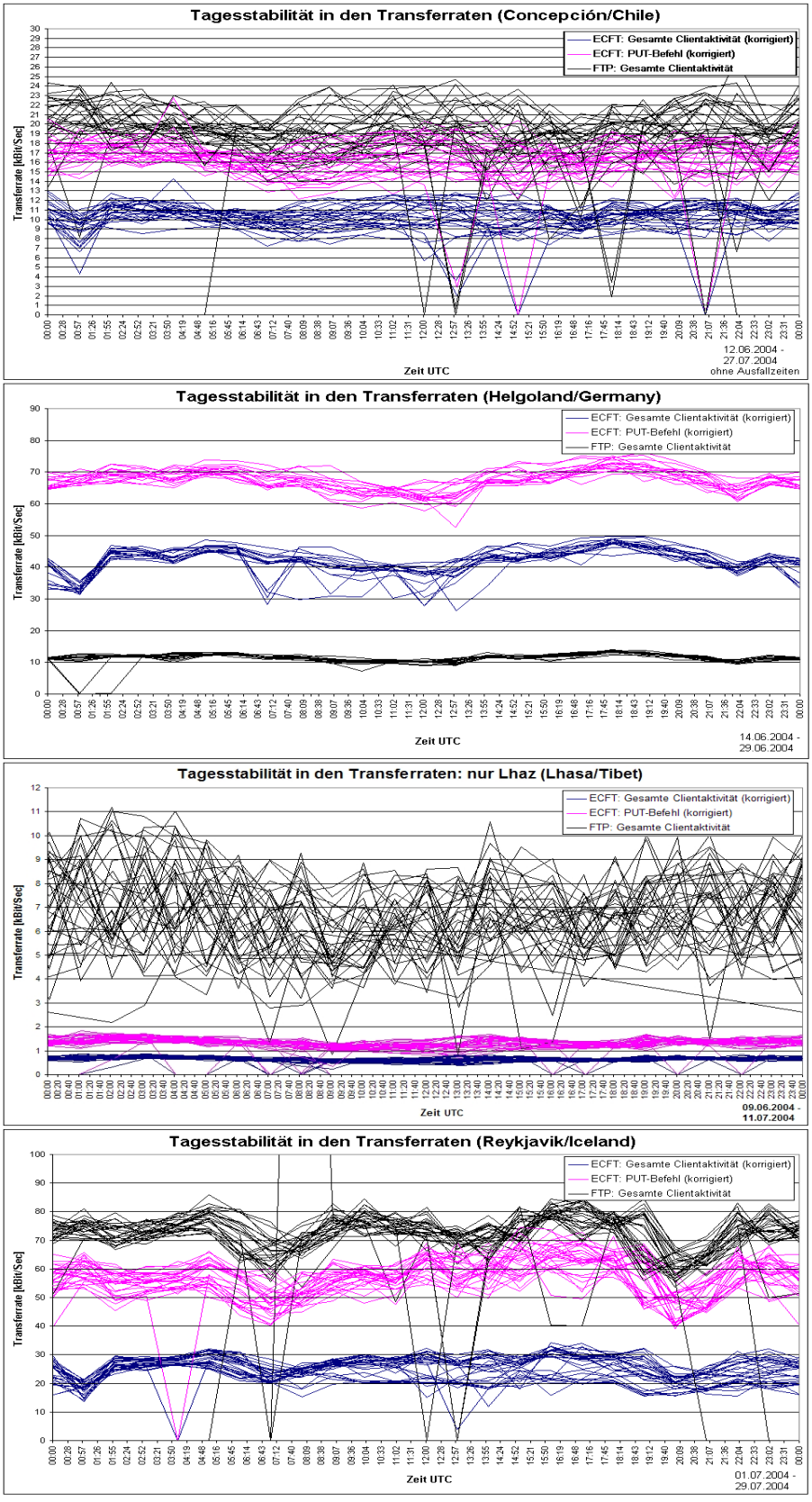


Abbildung 4.9: Transferstabilität je Tag für Concepción/Chile, Helgoland/Germany, Lhasa/Tibet, Reykjavik/Iceland

```
163454084 2004.04.26 13:06:04:182 [BEG] WDA nach Wettzell
*********************************
* CORBA File Transfer (Client)   *
* Version 1.0 (Beta)             *
*********************************

Set type to ASCII/ANSI
User Administrator logged in

CFT> autotype
Autotyping mode off
[Action took 30 millisecond(s)]
CFT> set block 5
Blocksize set to 5 Bytes!
[Action took < 1 millisecond]
CFT> binary
Set type to BINARY
[Action took < 1 millisecond]
CFT> cd /e:/dav/LhasHourly
Changed into /e:/dav/LhasHourly!
[Action took 480 millisecond(s)]
             ------------------- entfernte Aktion "cd" dauert 480 msec
CFT> lcd /d:/hourly/rinex30
Working directory changed into /d:/hourly/rinex30!
[Action took 36203 millisecond(s)]
             ------------------- lokale Aktion "lcd" dauert 36203 msec
                                 (Inversionsverhalten!)
CFT> put LHAZ117M.04d.z
Current TAN (on remote interface) is [T1082985520-1-10227T]
LHAZ117M.04d.z is put!
[Action took 2118005 millisecond(s)]
CFT> bye
555 Goodbye!
165723417 2004.04.26 13:43:53:515 [END] WDA nach Wettzell
```

4. Im Fehlerfall zeigt sich der Vorteil des neuen ECFT -Systems. Zum einen werden Fehler durch ihre Kennungen direkt erkenn- und zuordenbar. Zum anderen wird ein restriktives Verhalten verfolgt. Tritt ein Übertragungsproblem auf, so wird davon ausgegangen, dass es sich auch nicht innerhalb einer nachfolgenden, mittelfristigen Wartezeit beheben lässt. Die Übertragung wird abgebrochen und muss später erneut gestartet werden (automatische Wiederholungen sind momentan noch nicht implementiert, jedoch vorgesehen). Durch die Fehlererkennung während der Kommunikation werden somit nur selten „Round Trip Delay“-Begrenzungen überschritten und damit Zeitüberschreitungen in Form von Timeouts erzeugt. In Verbindung mit einem stabileren Zeitverhalten ergibt sich damit, dass die statistischen, mittleren Fehler bei ECFT -Übertragungen wesentlich günstiger ausfallen, als bei FTP .

   Besseres Verhalten bzgl. des RMS der Transferraten im Vergleich zu FTP

5. Die Stabilität bzgl. der Transferraten ist besonders in einer Überlagerung der Tageskurven ersichtlich. Es ergeben sich lastabhängige Kurven, welche im Mittel einen charakteristischen Verlauf haben. Je stabiler das Übertragungssystem ist, desto klarer ist der Verlauf ersichtlich, da dieser als Mittelwert einen Polygonzug bildet, um den sich die Messwerte mit geringen mittleren Fehlern reihen. In den Überlagerungen für die Messreihen ist ersichtlich (vgl. Abb. 4.9 auf Seite 120), dass der charakteristische Tagesverlauf in der Regel bei ECFT gut erkennbar ist (reproduzierbares Tagesverhalten mit Erwartungswerten als Qualitätsmerkmal), während dies bei FTP nicht immer gegeben ist (siehe z.B. in Concepción). Auch kann gut zwischen Lastwerten aus dem Netzwerk (Kurve für den PUT-Befehl) und vom lokalen Prozessor (Kurve der gesamten Clientaktivität ist in der Regel die um eine Translation der Transferrate verschobene Kurve des PUT-Befehls mit Ausnahme einiger Abweichungen) unterschieden werden. Eine periodische, lokale Abweichung durch die Prozessorauslastung ist z.B. um ca. 1:00 Uhr UTC gegeben, weil dann lokale Aufräumaktionen und Löschvorgänge stattfinden.

   Erkennbare, reproduzierbare Tagesstabilität

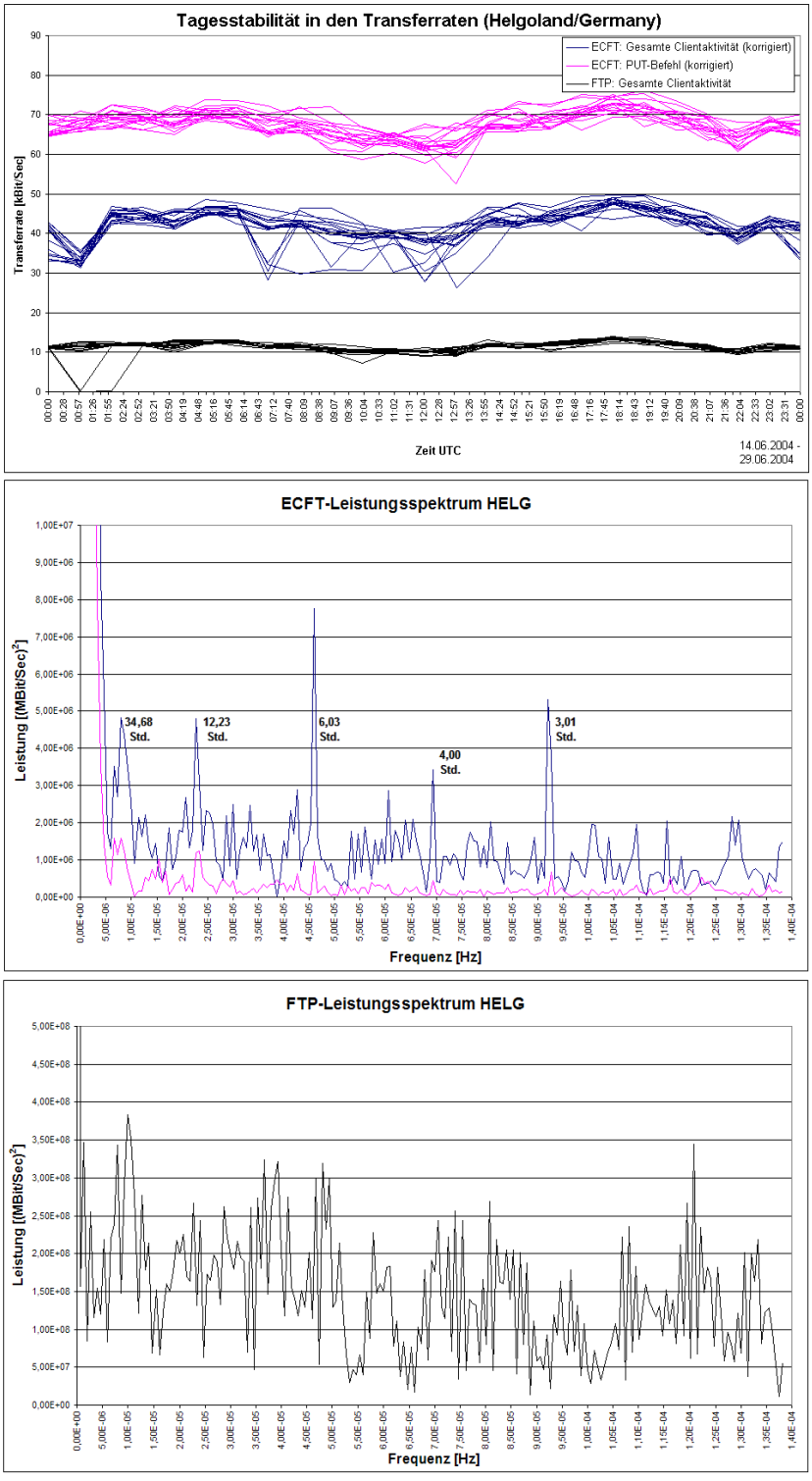


Abbildung 4.10: Leistungsspektren zu Helgoland

Fingerprint der Übertragungsleistung eines Stationsszenarios

Aufgrund der aufgezeigten Tagesstabilität kann eine weitere Idee verfolgt werden. Errechnet man über die Messwerte ein Leistungsspektrum (Peaks sind durch die quadratische Abszisse stärker ausgeprägt) für die Frequenzverteilung mit Hilfe einer Fast Fourier Transformation (FFT) -basierten Methode, so ergibt sich eine Art „Fingerabruck“ der Übertragungsleistung einer Station.

Versuch zur Rechnung eines Leistungsspektrums

Durch die Hilfe von Dr. Ulrich Schreiber, der zur Auswertung der Messreihen vom Laserkreisel in Wettzell ähnliche Verfahren nutzt, konnten solche Spektren für Helgoland mit Hilfe eines LabView-Programms berechnet werden. Dabei stellt sich die erwartete Charakteristik ein, dass sich bei ECFT bei einigen Frequenzen klar sichtbare Peaks ergeben, während diese bei FTP im Rauschen untergehen (vgl. Abb. 4.10 auf Seite 122 und Anhang J auf Seite 253). Zu beachten ist hierbei, dass aufgrund des doch relativ kantigen Eingangssignals zahlreiche harmonische Frequenzen bei Vielfachen der Hauptfrequenzen im höheren Bereich auftreten. Aus Zeitgründen konnten weitere Untersuchungen für die anderen Stationen nicht mehr durchgeführt werden, so dass noch keine allgemein gültige Aussage getroffen werden kann. Die dargestellten Tagesverläufe sind diesbezüglich jedoch sehr vielversprechend.

Variation der Spektren im zeitlichen Verlauf

In zukünftigen Überlegungen könnte weiterführend mit Hilfe der Spektren eine optische Darstellungsmöglichkeit der Übertragungsstabilität einer Station geschaffen werden. Dabei wären regelmäßig die Fourier Transformationen für definierte Zeitabschnitte (z.B. ein Tag, eine Woche) zu rechnen. Der Vergleich so erhaltener Fingerabrücke liefert ein Qualitätsmerkmal, da bei stabilen Verhältnissen diese gleich bleiben müßten. Es wäre sogar überlegenswert, die Leistungen in den Spektren durch Farben entsprechend ihrer Stärke zu kodieren und bzgl. der Erstellungszeit sortiert anzuordnen. So sollte sich ein optisches Muster für die Langzeitstabilität einer Messstation ergeben. Relative Veränderungen in Regelmäßigkeiten werden in Verschiebungen der Frequenzverteilung sichtbar. Schwankungen in den Übertragungszeiten können teilweise aus den Amplituden ersehen werden. Ein ähnliches Verfahren wird z.B. in der Seismik zur Aufzeichnung und Analyse von Erbebenwellen genutzt. Bisher sind allerdings noch keine Diagramme dieser Art verfügbar.

Weiterentwicklungen sind notwendig aber auch lohnend

Zusammengefasst ist damit festzustellen, dass am Prototyp noch Verbesserungen anzubringen sind, dass er sich aber ohne Probleme in bestehende Strukturen integrieren lässt und dabei die vorherrschende Heterogenität beseitigen kann. Dabei ist das gezeigte Leistungsverhalten momentan meist zwar noch schlechter als bei der FTP -Übertragung, zeigt jedoch ein reproduzierbares Verhalten bzgl. der Tagesstabilität.

4.5.4 Die Firewall-Problematik

Das Überwinden von Firewalls als hartnäckiges Problem

Während der Entwicklungsarbeiten stellte sich ein Problem als sehr hartnäckig heraus: die Kommunikation durch eine Firewall hindurch. Da dieses Gebiet ein komplettes Thema für sich wäre, können hier gemachte Erfahrungen und in der Literatur gefundene Lösungen nur kurz angeschnitten werden.

Begriffsdefinition DEF4.8 „Firewall“ frei nach [LEX04][24]:
Firewalls sind Rechner, welche ein Netz mit bestimmten Sicherheitsvorgaben vor unbefugtem Zugriff aus einem anderen Netz (z.B. Internet) dadurch schützen, dass sie als einziger Verbindungspunkt der Netze den Netzwerkverkehr kontrollieren und überwachen.

Ein- und ausgehende Transferwege

Der Schutz kann sich dabei auf den eingehenden (inbound) und ausgehenden (outbound) Netzverkehr erstrecken. Im ersten Fall wird die Erlaubnis der Kommunikation von externen Clients mit innenliegenden, geschützten Servern geregelt (z.B. will eine GPS -Permanentstation Daten auf einen inneren Server ablegen). Der zweite betrifft den umgekehrten Weg (z.B. will ein Anwender von intern extern abgelegte Daten herunterladen)[25].

„Schutzschichten“

Eine Kontrolle kann dabei in verschiedenen Schichten des Protokollstapels angesetzt werden. Einfache Paketfilter agieren auf der Netzwerkschicht (z.B. IP). Transport-Proxies (Server-Stellvertreter) regeln den Verkehr in der Transport-Schicht (z.B. TCP). Und auf Applikationsebene können höherwertige Schutzdienste auf oberster Protokollebene stattfinden (z.B. FTP , GIOP)[26].

IIOP meist nicht durch die Firewalls unterstützt

Das Hauptproblem[27] bei CORBA liegt nun darin, dass das für die entfernte Aktivierung von Prozeduraufrufen am meisten eingesetzte IIOP von den gängigen Firewalls nicht unterstützt wird. Das übliche Freigeben von dedizierten Ports oder das Anlegen von Tunnelkanälen (z.B. HTTP -Tunnel) in der Firewall ist dabei wenig hilfreich, da im IOR , also der Netzreferenz z.B. zum Server, ein kompletter Satz an Adressierungsdaten für eine direkte Punkt-zu-Punkt-Verbindung enthalten ist. Ein Server antwortet damit mit seiner realen Adresse, auf die der Client schließlich zuzugreifen versucht. Da die Adresse aber von außen nicht sichtbar (z.B. bei Network Address Translation (NAT) , bei der nicht öffentliche Adressen des internen Netzes in erlaubte, externe umgesetzt werden) oder aufgrund der Sicherheitsregeln nicht direkt zugänglich ist, schlägt dies unweigerlich fehl.

Transparenzverlust

Auch weitere Konzepte in CORBA bleiben ungenutzt. Callback-Funktionen zum Aktivieren von Clientaktivitäten durch den Server (z.B. bei asynchroner Kommunikation) sind nicht möglich. Des Weiteren geht ein Großteil an Transparenz verloren, da die Informationen zu den Firewalls direkt bei den Kommunikationspartnern bekannt sein müssen, was die Interoperabilität durch dynamische Objektzuordnungen enorm einschränkt. Noch schwieriger wird es, wenn ganze Kaskaden aus mehreren Schutz-Enklaven zu überwinden sind.

Erweiterung des Standards durch die OMG

Aufgrund dieser Erkenntnisse brachte die OMG eine Standardisierung zum Überwinden von Firewalls für CORBA heraus[28]. Darin werden Mechanismen vorgeschlagen, um herkömmliche Firewalls zu überwinden. Zudem sind Regelungen beschrieben, welche die Firewalls nebst der Applikationen umsetzen müssen, um die volle CORBA -Funktionalität zu ermöglichen und trotzdem einen Schutz auf höherer Protokollebene zu unterstützen.

[24] vgl. dazu [LEX04] unter . . . /Firewall.html (31.07.2004)
[25] vgl. dazu [ABI02] a.a.O. S. 3
[26] vgl. dazu [ABI02] a.a.O. S. 6 f.
[27] vgl. zum folgenden Abschnitt [ABI02] a.a.O. S. 4 ff.
[28] vgl. dazu [OMGF04]

Im Wesentlichen werden zwei Typen vorgeschlagen:

1. **TCP -Proxies:** Im einfachsten Fall wird ein herkömmlicher Proxy auf der Transportschicht eingesetzt (ausgezeichneter, „well-known“ Port: 683)[29]. Der IOR kann dabei manuell so angepaßt werden (die sog. „Proxifikation“), dass z.B. der Client nicht mehr die direkte Verbindung zum Server aufnimmt, sondern mit der Firewall kommuniziert[30].
   Nachteil: Entweder diese Möglichkeit wird von der CORBA -Implementierung gestellt oder es muss in diese eingegriffen werden, was aber der Festlegung von Middleware als Black Box in Abschnitt 3.3.1 auf Seite 56 widerspricht.

2. **GIOP -Applikations-Proxies:** Diese Art von Proxy kann GIOP -Nachrichten und damit auch IIOP verstehen und umsetzen und ermöglicht somit sogar einen Schutz auf Objektzugriffsebene. Als eine der wenigen IIOP -Firewalls ist hier „I-DBC“ von der Xtradyne Technologies AG zu nennen, welcher transparent in CORBA -Umgebungen eingesetzt werden kann[31].
   Nachteil: Hier muss die Firewall dafür ausgelegt sein, was nicht alle sind. Bei den unterschiedlichen Bedingungen, wie sie z.B. beim klassischen GPS -Netz vorliegen, ist dies eine enorme Einschränkung, da es immer Verbindungen geben wird, welche diese Proxies nicht unterstützen.

Zusätzlich werden in der Spezifikation Vorschläge gemacht, um u.a. Callbacks zu nutzen, wie der „Traversal Algorithmus“ zum Überwinden von Firewalls auszusehen hat oder wie ein „TAG_FIREWALL_PATH“ im IOR zu nutzen ist[32]. Allerdings kann es noch einige Zeit dauern, bis diese Ideen umgesetzt sind, auch wenn es schon Firewallsysteme gibt, wie z.B. Wonderwall“ von IONA, welche dies ermöglichen. Allerdings ist dies vorerst keine Lösung für bestehende Umgebungen.

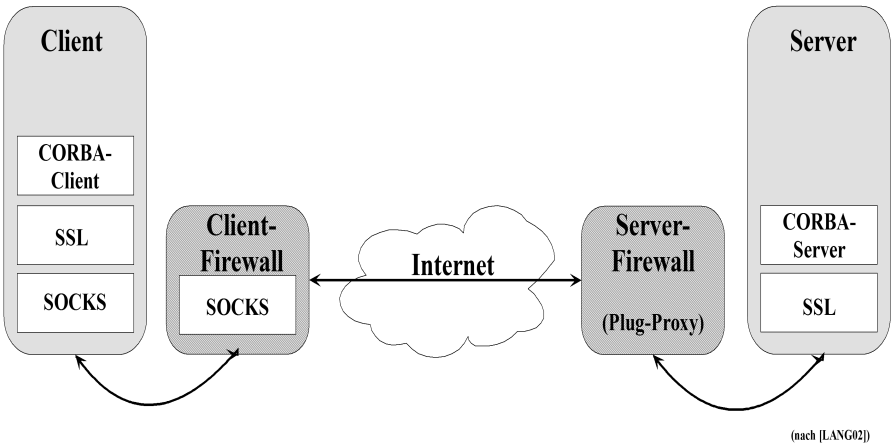


Abbildung 4.11: Überquerung von Firewallgrenzen mit SOCKS

[29] vgl. dazu [OMGF04] a.a.O. S. 1-4 f.
[30] vgl. dazu [SCHR199]
[31] vgl. dazu [XTRA04]
[32] vgl. dazu [OMGF04] a.a.O. S. 1-6 ff.

Weitere Möglichkeiten

In der Literatur findet man dazu andere Möglichkeiten, welche dazu genutzt werden können.

1. **SOCKS-Proxies:** Sockets Secure (SOCKS) ist ein Standard der Internet Engineering Task Force (IETF) , welcher hauptsächlich die clientseitige Firewallüberwindung auf Transportebene ermöglicht. Ein Client kann dabei bestimmen, mit welchem Server hinter der Firewall er verbunden werden will, so dass eine Art dynamischer TCP -Proxy entsteht. Benötigt wird dazu nur, dass der Firewall einen SOCKS -Server stellt und der Client statt der Standardbibliothek zur Kommunikation die modifizierte SOCKS -Bibliothek mit korrespondierenden Befehlen nutzt (dies wird als „Socksifikation“ bezeichnet)[33]. Über eine Konfigurationsdatei kann nun der SOCKS -Server angegeben werden[34]. Eine mögliche Konfiguration sowohl mit clientseitiger als auch mit serverseitiger Firewall unter zusätzlicher Nutzung von SSL ist in Abb. 4.11 auf Seite 125 dargestellt[35].
   Nachteil: Auch hier muss in die Middleware eingegriffen werden. Der IOR verliert an Bedeutung und damit auch seine Dynamik. Callbacks o.ä. werden nicht unterstützt.

2. **Tunnel z.B. über HTTP :** Bei dieser Methode werden die Pakete des IIOP -Datentransfers in HTTP eingebettet. Da herkömmliche clientseitige Firewalls HTTP -Proxies nutzen, um dieses Protokoll für das Browsen im Netz offenzuhalten, kann so die Firewall quasi durchtunnelt werden[36].
   Nachteil: Standardkomponenten dazu nutzen nichts, weil wiederum die IOR manipuliert werden müsste. Getestete, frei erhältliche Standardkomponenten wie HTTHost 1.7.1 oder Gnu HTTP Tunnel 3.3 zeigen dieses Verhalten. So ist wiederum ein Eingriff in die Middleware notwendig.

3. **Ersatzprotokolle (SOAP):** Eine weitaus zukunftsorientiertere Methode ist die Nutzung von anderen Protokollen. Interworking Spezifikationen z.B. zur standardisierten Umsetzung von CORBA nach Web-Service Description Language (WSDL) und damit nach SOAP erlauben die Nutzung des auf HTTP basierten Internetprotokolls für Web Services[37]. Somit wird zur Kommunikation nicht mehr IIOP genutzt, sondern die für die neuen Dienste entwickelten Protokollumgebungen.
   Nachteil: IIOP als eigentliches Protokoll wird nicht mehr verwendet. Die Hersteller der CORBA -Implementierung müssen die Erweiterungen einbringen.

Bisher keine sofort vefügbare, zufriedenstellende Lösung greifbar

Zusammengefasst bestätigten die durchgeführten Tests die Problematik, Firewalls zu überwinden. Bisher konnte kein zufriedenstellender Weg gefunden werden, der sofort zur Verfügung stehen würde. Erst im Rahmen zukünftiger Entwicklungen bei den CORBA -Implementierungen kann auf eine Verbesserung dieses Missstands gehofft werden. Allerdings bleibt dabei weiterhin offen, welches Risiko es beinhaltet, entfernte Prozeduraufrufe offen zugänglich zu machen. Bisher kann niemand abschätzen, wie anfällig ein CORBA -Server für direkte Attacken ist. Schließlich wurden diese nicht von Sicherheitsexperten programmiert, was sich immer wieder durch zufällige Feststellungen zeigt (z.B. durch Vorspielen eines

[33] vgl. dazu [ABI02] a.a.O. S. 8 f.
[34] vgl. dazu [SCHR299]
[35] vgl. dazu auch [LANG02]
[36] vgl. dazu [ABI02] a.a.O. S. 13
[37] vgl. dazu [OMGSOAP04]

IIOP -Clients mittels eines Telnet-Clients)[38]. Immer wieder laufende Virenattacken gegen offene RPC -Ports von Microsoft-Betriebssystemen lassen hier nichts Gutes erahnen, so dass diese Problematik wohl nur durch ein geeignetes Design der Hardwarekomponenten und ihrer Lokalitäten (vgl. dazu den verbesserten Datenzugang in Abschnitt 6 auf Seite 155) sicher in den Griff zu bekommen ist.

4.6 Zusammenfassung

ECFT ist eine konkurrenzfähige Alternative zur Datenübertragung

Zusammenfassend kann zu diesem Kapitel festgestellt werden, dass die entwickelten Methoden und Strukturen als eine Art Kernfunktionalität eine konkurrenzfähige Alternative zu herkömmlichen Dateiübertragungsmechanismen, wie z.B. Systeme mit FTP oder auch File Transfer , Access and Management (FTAM) (ISO 8571), darstellen. Dabei basiert ECFT auf einer weitverbreiteten, allgemein anerkannten Middleware und ist von der Handhabung und seinem Aufbau so einfach gehalten wie Abläufe mit FTP . Der enorme Entwicklungsvorsprung und die daraus resultierende Bekanntheit und Programmstabilität bei FTP muss dabei anerkannt werden.

Umfangreiche Mechanismen zur Schaffung von Dienstgüte

Trotzdem vereinigt ECFT basierend auf den elementaren Schnittstellen von CFT Lösungen von bisher im Bereich der Dateiübertragung oft zu wenig beachteten Anteilen. Zum einen wird durch den Multi-Tier-Connector ein flexibles Design- und Architekturkonzept für eine busbasierte Schnittstellenprogrammierung geschaffen. Zum anderen wird auf höherer Ebene ein Konzept der Nutzersichten eingeführt, welches den konkurrierenden Zugriff geordnet und harmonisch löst und jedem Nutzer eine Umgebung bietet, als wäre er alleine im Dialog mit dem System. Komplettiert wird das Ganze durch ein relativ einfaches und doch effektives Ausfall- und Fehlermanagement, welches stabile Zustände erzeugt und selbstständig unvorhergesehene Wiederanläufe verkraftet.

FTAM/FTP Interworking Specification

Da auch die OMG die Heterogenität des Dateizugangs beheben will, wurde im März 2002 eine CORBA-FTAM/FTP Interworking Specification[39] verabschiedet, welche eine allgemeine Schnittstelle definiert, um verschiedene Dateizugangsmechanismen für CORBA zu vereinigen. Berücksichtigt sind FTP , FTAM , lokale Dateisysteme und sonstige dateibasierende Netzwerkelement.

Vorteile durch die Verallgemeinerung jedoch nicht uneingeschränkt auf Laufzeitsystem erweiterbar

Eine solche Verallgemeinerung bringt zahlreiche Vorteile, da es für einen Nutzer unerheblich wird, welchen Zugangsmechanismus er einsetzt. Allerdings birgt es auch Gefahren. Wie bei den Messungen im LAN gezeigt, kann man nicht einfach davon ausgehen, dass jeder Zugangspunkt dasselbe Verhalten zeigt, nur weil man mit einer standardisierten Schnittstelle darauf zugreift. Auch bei der Verallgemeinerung des Zugangs zu Dateien werden individuelle (teilweise vorteilhafte) Anteile verdeckt und auf Gemeinsamkeiten zurückgeführt. Damit wird zwar der Zugriff einfacher, zu der aber bereits aufgezeigten Leistungsabhängigkeit vom Betriebssystem (vgl. Abschnitt 3.5.3 auf Seite 77)) kommen noch weitere, schwer abschätzbare Faktoren aus den Unterschieden in den Übertragungsverfahren und -protokollen hinzu. Trotz allem ist aber eine Umsetzung der Spezifikation wünschenswert. Durch die bei der Entwicklung von ECFT beabsichtigte Anlehnung an

[38] vgl. dazu [SCHR199]
[39] vgl. dazu [OMGFT04] a.a.O. S. 1-1 f.

FTP sollte eine Angliederung an Implementierungen davon möglich sein. Bislang sind jedoch noch keine brauchbaren Umsetzungen bekannt.

Abschließend hat sich damit gezeigt, dass die Umsetzung noch mit einigen, kleineren Programmierproblemen zu kämpfen hat, dass jedoch der eingeschlagene Weg vielversprechend ist. Aus Sicht der Transparenz (vgl. Abschnitt 2.4 auf Seite 31) wurden damit insbesondere die Kriterien bzgl. konkurrierendem Verhaltens und bzgl. Ausfällen eingehalten.

Kapitel 5

Verbesserung der Datendarstellung

Schwerpunkt des Kapitels:

Das nachfolgende Kapitel beschäftigt sich mit der Repräsentation von Daten. Es versucht den Informationsgehalt dieser allgemein zugänglich zu machen. Bereits in den ersten Designphasen der Studie wurde deshalb die Extensible Markup Language (XML) favorisiert. Im Laufe der Entwicklungen gaben parallel laufende Arbeiten den Entscheidungen Recht. Es entwickelten sich zahlreiche Formate und Metadatenstrukturen, welche genutzt werden können und nicht in Eigenleistung erstellt werden müssen. Alle gemeinsam weisen aber einen Mangel auf: Bestehende, nicht XML -konforme Daten müssen mühevoll mit zusätzlichen Programmen erst nach XML konvertiert werden. Deshalb soll hier eine XML-konforme Sprache zur Integration von bestehenden Datenbeständen geschaffen werden. Sie trägt die Bezeichnung Anything to XML (A2X).

5.1 Ausgangsüberlegung

In den bisherigen Kapiteln waren die Daten, also die transportierten Inhalte, unwesentlich. Es wurde versucht, ein stabiles Grundsystem zum Datenaustausch zu erschaffen. Dabei wurden weder Unzulänglichkeiten durch unterschiedliche Datenformate berücksichtigt, noch Möglichkeiten vorbereitet, Datenbestände wandelbar und zugänglich zu machen. Ziel war es Dienstgüte für das Transportproblem zu gewährleisten.

Abstrakte Betrachtung der Daten ohne inhaltliche Details

Nachdem diese nun hinreichend angenommen werden kann, wird sich das folgende Kapitel mit den Daten an sich beschäftigen (entsprechend der zweiten Säule zur Abstraktion in Abschnitt 2.4 auf Seite 34). Genauer gesagt, werden die verschiedenartigen Daten einer abstrakten Betrachtung unterzogen, ohne sie inhaltlich zu beleuchten. Im Allgemeinen bilden kombinierte Einheiten von Daten Dokumente, welche erst dann zur Informationsgewinnung dienlich sind. Bislang werden Dokumente in verschiedenen Formaten abgespeichert, welche individuell und proprietär von entsprechenden Programmen verstanden werden. Moderne Techniken mit Hilfe von Markup-Sprachen erlauben es jedoch, allgemeinverständliche, abstrakte eigene Formatsprachen zu definieren. Somit war die Extensible Markup Language (XML) in dieser Arbeit von Anfang an Mittel zum Zweck.

Einsatz von XML im Rahmen von Datenzentren ohne einheitliche Definition der Konvertierung in die neue Beschreibung

Aktuelle Entwicklungen im Bereich von Datenzentren und verteilter Datenhaltung nicht nur im Bereich der Geodäsie (hier z.B. IERS) sondern auch generell z.B. bei der Entwicklung von Web Services bestätigen die ersten Überlegungen. Allerdings setzen alle auf ein wohlgeformtes Datenkonzept auf, welches davon ausgeht, dass alle Daten bereits in XML vorliegen. Diese Annahme ist aber keineswegs auf die Realität zu übertragen, auch wenn mehr und mehr Dokumente diese Techniken nutzen und sie evtl. irgendwann als Standarddarstellungsmethode eingesetzt werden.

Spezielle Programme als häufigste Lösung

Um dieses Problem zu umgehen, nutzen gängige Entwicklungen eigene Programme in beliebigen Programmiersprachen (zumeist aus Einfachheit Java), welche erst ein bestehendes Dokument nach XML konvertieren, um schließlich daraus mit den wohldefinierten XML -Tools die weitere Verarbeitung zu ermöglichen. Das hat z.B. zur Folge, dass aus der XML -Repräsentation wieder in einem Extraschritt das Original gewonnen werden muss, falls dieses benötigt wird.

Einbeziehung bestehender Dokumente entscheidend

Da sich die Entwicklungen der vorliegenden Arbeit in ein bestehendes System aus Arbeitsabläufen, Programmen und Formaten einfügen soll, ist aber gerade dieser Aspekt der Nutzung bestehender Quellen von großer Bedeutung. Die zur Verarbeitung dieser Quellen umständliche Einbeziehung weiterer Programmiersprachen ist dabei nicht gerade zufriedenstellend, da ein Datenverantwortlicher nicht nur XML beherrschen, sondern zudem entsprechende Fähigkeiten in der Programmierung aufweisen muss.

Idee einer einheitlichen, automatisierten Umwandlung mittels Standardtransformator

Aus diesem Grund entstand die Idee, es einem Standardtool für XML zu ermöglichen, beliebige Daten zu lesen und diese automatisiert in eine XML-Struktur umzuwandeln. Die syntaktischen Zusammenhänge sollen dabei im wohlgeformten Stil einer XML -Beschreibung zur Verfügung gestellt werden, so dass sich ein ähn-

liches Verarbeitungsschema abzeichnet, wie bei der Transformation von XML in beliebige andere Dokumente. Der verantwortliche Datenverwalter bleibt damit in seiner XML -Welt.

„Anything 2 XML“

Da es mit Hilfe einer solchen Sprache möglich wird, beliebige, elektronisch erfassbare Datenquellen in konforme XML -Bestände umzuwandeln, trägt die neue Sprache die Bezeichnung „Anything to XML“ (kurz A2X ; vgl. dazu die Beschreibung im Abschnitt 5.3 auf Seite 139). Dabei ist der Strukturierungsgrad des resultierenden XML -Dokuments bei gleichgehaltenem Informationsgehalt geringer oder äquivalent zu dem des Ausgangsdokuments. Einzelne Informationen lassen sich im Resultatdokument nicht mittels XML - Strukturen identifizieren, wenn sie nicht auch im Originaldokument interpretierbar sind. Zudem ist die Umwandlung mit einem entsprechenden Aufwand verbunden. Es soll hier nicht der Eindruck erweckt werden, dass ein beliebiges Dokument ohne weiteres Zutun in seiner Allgemeinheit direkt eingelesen und transformiert werden kann. Jede Umwandlung bedarf einer exakten Beschreibung, wie welche Ausgangsanteile in welche XML - Resultate umzusetzen sind.

Transparenzkriterien

Zur Eingliederung in die Transparenzkriterien (vgl. Abschnitt 2.4 auf Seite 31) kann hier auf die Transparenz bzgl. der Daten hingewiesen werden, da damit jede Form von Daten allgemeingültig verständlich wird. Auch werden nochmals Aspekte der Zugriffstransparenz angeschnitten, weil der Zugriff auf Daten auf eine einheitliche Art und Weise stattfinden kann.

Bedeutung für die FS Wettzell: Vorbereitung auf neue Trends in der Datenhaltung

Lösungen auf diesem Gebiet bereiten die Datenverwaltung in der Fundamentalstation Wettzell auf zukünftige Trends in der Datenhaltung vor. Die Vereinheitlichung ermöglicht einen flexibleren Umgang mit verschiedenartigsten Daten. Es können so auch individuelle eigene Formate in offizielle Strukturen eingebunden werden, ohne dass großflächige Umbaumaßnahmen an der Software der datenerzeugenden Systeme durchgeführt werden müssen.

Bedeutung für die Geodäsie: Vereinheitlichung und Vereinfachung der Konvertierungen in neuen Datenzentren

Im Bereich der allgemeinen Geodäsie können moderne Datenzentren, wie sie zurzeit entstehen, mit ähnlichen Verallgemeinerungen wesentlich vereinfacht werden, da individuelle Softwarelösungen vermieden werden und einheitliche Beschreibungskonstrukte an ihre Stelle treten.

Bedeutung für die Informatik: Lösung genereller Probleme der Datenkonvertierung

Ähnliches gilt für die Datenhaltung im Allgemeinen. Egal in welchem Bereich der Informatik man sich befindet, neue Systeme und ihre Daten müssen sich mit bestehenden arrangieren und sich in sie einfügen. Warum bisher keine Entwicklungen in diese Richtung laufen, liegt wohl an der Tatsache, dass dies Entwicklungskosten verursachen würde, welche durch die modulare Bauweise bei der Programmentwicklung nicht anfallen.

Da bereits einige Begriffe bzgl. der Markup-Sprache XML gefallen sind, wird nachfolgend nun erst der theoretische Grundstock für die späteren Entwicklungen bzgl. A2X gelegt.

5.2 Theoretische Grundlagen

Geeignete Repräsentation mit Vorteilen für die Präsentation

Wie bereits in der Analyse im Kapitel 2 ab Seite 19 festgestellt, wird der Umgang mit Daten in die Repräsentation und die Präsentation eingeteilt. Eine geeignete Repräsentation ist Grundlage für eine flexible Datenhaltung und Datentransparenz.

Wenn diese Darstellung zudem auch noch Vorteile für die Präsentation der Daten für unterschiedliche Zwecke liefert, so wäre damit auch noch die Zugriffstransparenz wesentlich verbessert.

Lösung: Auszeichnungssprachen

Moderne Auszeichnungssprachen stellen sich diesen Herausforderungen und lösen Probleme früherer Datenhaltungssysteme. Eine dieser Sprachen ist die Extensible Markup Language (XML), welche mehr und mehr auch im Rahmen der Web Services Einsatz findet und durchaus als Lingua franca für die Datenrepräsentation angesehen werden kann.

5.2.1 Die Nutzung der Auszeichnungssprache XML

Markups als Basis der Auszeichnungssprachen

Für die Handhabung von Daten beliebiger Art benötigt man zum einen eine Möglichkeit zur Verwaltung und Beschreibung (Repräsentation) und zum anderen Verarbeitungsvorgaben zur Umwandlung und Darstellung (Präsentation). Ersteres wird in modernen Formatierungssprachen mittels Markups erreicht[1].

Begriffsdefinition DEF5.1 „Markup“ nach [RAY01][2]:
Markups sind Zusatzinformationen zu den eigentlichen Informationen eines Dokuments, mit denen eine weitere Bedeutung bzgl. Kennzeichnungen und untereinander bestehender Beziehungen ergänzt wird.

XML zur Beschreibung von Sinn und Struktur eines Dokuments

Während ursprüngliche Markup-Sprachen (wie z.B. TeX) dafür geschaffen wurden, eine Eingliederung des Inhalts in feste Strukturen und eine Beschreibung des Aussehens von Dokumenten u.a. für den Druck zu geben, gehen moderne Sprachkonstrukte dazu über, neben Inhaltlichem auch variabel den Sinn und die Struktur eines Dokuments zu beschreiben[3]. Dazu entstand im Rahmen des World Wide Web Consortium (W3C) aus der Einfachheit von Hypertext Markup Language (HTML) und der Flexibilität von Standard Generalized Markup Language (SGML) Mitte der 90er Jahre als Beschreibungssprache die Extensible Markup Language (XML). Sie definiert keine Auszeichnungssprache (Markup Language) an sich, sondern eine Möglichkeit selbst solche Sprachen zu entwerfen und ist damit ein Werkzeug zur Erstellung eigener, strukturierter Formate[4].

Wohlgeformtes XML

Gerade die Struktur in Form eines Baums ist wesentlich. Ein Dokument muss eindeutige Namen und eine vordefinierte Reihenfolge und Hierarchie der Bestandteile aufweisen. Dem ist hauptsächlich „wohlgeformtes“ XML zu nutze, weil hier zur Dokumentenmodellierung z.B. über eine Document Type Definition (DTD) Regeln und Deklarationen eingefügt werden, welche die Strukturen definieren und beschränken. Damit ist auch eine Validierung bereits im Parser möglich[5]. Dies bietet ein Maximum an Fehlerüberprüfung bei doch relativ einfacher Strukturierung[6]. Auch wenn man eingestehen muss, dass umfangreiche Dokumente ohne entsprechende Werkzeuge nur noch schwer handhabbar sind.

[1] vgl. [RAY01] a.a.O. S. 2
[2] vgl. [RAY01] a.a.O. S. 2
[3] vgl. [RAY01] a.a.O. S. 10
[4] vgl. [RAY01] a.a.O. S. 12
[5] vgl. [RAY01] a.a.O. S. 6
[6] vgl. [RAY01] a.a.O. S. 14

Begriffsdefinition DEF5.2 „Parser“ nach [RAY01][7]:
Ein Parser ist ein Prozessor (Verarbeitungseinheit), der einen Strom aus Zeichen (z.B. aus einer Datei) in sinnvolle Informationseinheiten zerlegt (Token), welche entweder zur Steuerung von Programmabläufen dienen oder auf temporäre Strukturen abbildet, die von einem Programm genutzt werden können.

Parser sind Hauptbestandteile der XML-Verarbeitung

Parser sind die Hauptverarbeitungseinheiten für XML -Dokumente. Jedes Programm, welches XML nutzen will, benötigt einen solchen Prozessor[8]. Der wichtigste Anwendungsfall ist hier wohl die Präsentation von Inhalten in unterschiedlichen Arten und Weisen. Dies ist vor allem dadurch möglich, weil der Inhalt von der Darstellung völlig unabhängig und getrennt ist. Formatierungsinformationen werden aus dem Dokument fern gehalten. Je nach gewünschtem Resultat kann so aus dem strukturierten XML-Dokument über Stylesheet-Vorgaben nahezu jede beliebige andere Dokumentenstruktur erzeugt werden[9].

Transformatoren zur Umwandlung von XML in andere Darstellungen

Gewährleistet wird eine solche Umwandlung durch entsprechende Transformatoren, welche aus einem XML -Dokument und einer ebenfalls in der Art von XML definierten Stilangabendatei in der Sprache Extensible Stylesheet Language for Transformations (XSLT) auf standardisierte Weise eine neue Präsentation erzeugen[10]. XSLT gehört zur Sprachenfamilie Extensible Stylesheet Language (XSL) , welche Mechanismen zur Definition von XML -Layouts, z.B. zur Darstellung der Inhalte auf Papier oder dem Monitor, zur Verfügung stellt. Weitere Bestandteile von XSL sind Extensible Stylesheet Language - Formatting Objects (XSL-FO) als eigentliche Formatierungssprache und XML Path Language (XPath) zur Adressierung der Baumelemente einer XML -Repräsentation eines Dokuments[11].

XML besteht aus einer Kernsyntax und darauf aufbauenden Erweiterungen

Bei allen Standards im Umfeld von XML handelt es sich um kontinuierlich weiterentwickelte Versionen. Zur Kern-Syntax kamen so weitere Gebiete hinzu, welche z.B. Anwendungen betreffen, verbesserte Modellierungen durch die XML Schema Definition (XSD) erlauben und lokale und entfernte Datenadressierungen und -abfragen transparent ermöglichen[12].

Baumstruktur als XML-Anatomie

Die Anatomie eines XML -Dokuments ist direkt in ein Baum-Diagramm abbildbar. Der Prolog bildet sich aus der XML -Deklaration (Angaben zu XML), die Dokumenttyp-Deklaration mit der Verknüpfung zur DTD und einem Satz aus Deklarationen. Dem Prolog folgt der Wurzelknoten des eigentlichen Dokuments.

Baumelemente sind Container

Das eigentliche Dokument besteht aus Container-Elementen, welche ein Gemisch aus Text und weiteren Container-Elementen beinhalten können. Jedes Element ist durch ein Start-Tag und ein End-Tag gekapselt, wobei im Start-Tag Attribute integriert sein können. Es ist möglich Namensräume auszubilden oder Platzhalter- Entitäten zu schaffen, welche durch den vordefinierten Inhalt ersetzt wer-

[7] vgl. [RAY01] a.a.O. S. 8
[8] vgl. [RAY01] a.a.O. S. 8 f.
[9] vgl. [RAY01] a.a.O. S. 13
[10] vgl. [RAY01] a.a.O. S. 28
[11] vgl. [HERP02] a.a.O. S. 21 ff.
[12] vgl. [RAY01] a.a.O. S. 17

den. Über solche Entities ist es auch möglich ungeparste Blöcke (z.B. Fremddokumente und Bilder) einzubinden. Steueranweisungen (für spezielle Prozessoren) und Kommentare runden die Flexibilität ab[13].

Beispiel 5.1 für „Hello World“ (nach [TID02][14]

```
<?xml version="1.0"?>
<gruss>
   Hallo, Welt!
</gruss>
```

Formatbeschreibung wird als Metainformation ins Dokument integriert

Es wird damit möglich, die Formatbeschreibung als Zusatzinformation (Metainformation) mit in die Struktur des Dokuments aufzunehmen. Somit ist mit XML eine Art Kompromiss zwischen den beiden Extremas der Strukturierung (Datenbanken mit ihrer starken Strukturierung und beliebige Dokumente mit unstrukturierten Texten) möglich [15]. Diese lassen sich damit auch ideal auf XML abbilden.

Begriffsdefinition DEF5.3 „Semistrukturierte Daten“ nach [KAZ02][16]***:***
Semistrukturierte Daten weisen zwar eine Struktur auf, jedoch ist diese variabel, teilweise implizit und nur a posteriori erkennbar, indikativ nicht einschränkend und oft nur partiell wirkend.

Zusätzliche Transparenz durch Navigation innerhalb und zwischen Dokumenten

Zusätzlich gibt es zahlreiche Erweiterungen zum Bewegen innerhalb der Baumstruktur (mit XPointer) oder über Dokumentgrenzen hinweg (mit XLink)[17]. Gerade diese Möglichkeiten führen zu weiterer Transparenz bzgl. des Zugriffs, da so z.B. auf weiterführende oder erklärende Dokumente verwiesen werden kann.

Zur Bearbeitung werden im Rahmen dieser Arbeit die Open Source Werkzeuge der Apache Software Foundation genutzt. Dazu gehören Programme in C++ und Java zum Parsen von XML -Dokumenten (z.B. Xerces C++ Parser) und zum Transformieren dieser (z.B. Xalan C++)[18].

Standardisierte XML-APIs

Dabei werden die standardisierten Application Programming Interface (API) zur Verarbeitung von XML angeboten. Dazu gehört zum einen das Simple API for XML (SAX) , welches Dokumente ereignisorientiert verarbeitet und bei erkannten Token eine Callback-Funktion auslöst. Es gibt keine Abbildung des Dokuments im Speicher in Form eines Baums. Dadurch wird ein erheblicher Geschwindigkeitsgewinn und wenig Speicherverbrauch erzielt[19].

Die weitere, vom W3C empfohlene Schnittstelle ist das Document Object Model (DOM). Dabei handelt es sich um eine baumstrukturierte, objektorientierte Programmierschnittstelle, welche das Dokument im Speicher als Baum abbildet. Im Grunde ist DOM das komfortabelste, strukturierteste und am meisten anpaßbare Parsing-Werkzeug für XML [20].

[13] vgl. bisher [RAY01] a.a.O. S. 31 ff.
[14] vgl. [TID02] a.a.O. S. 24
[15] vgl. dazu [KAZ02] a.a.O. S. 26
[16] vgl. [KAZ02] a.a.O. S. 28
[17] vgl. [RAY01] a.a.O. S. 91 bzw. a.a.O. S. 80
[18] vgl. [APACH04]
[19] vgl. [RAY01] a.a.O. S. 311 f.
[20] vgl. [RAY01] a.a.O. S. 328 f.

Diverse Anwendungsweisen werden bereits eingesetzt

Im Allgemeinen gibt es auf dem heutigen Markt verschiedene Annäherungen an die Eingliederung von XML . Einerseits kann eine XML -Middleware aufgesetzt werden, welche eine Zwischenschicht bildet und eine bestehende Persistenzschicht zur Datenhaltung nutzt (in der Art auch bei den Web Services umgesetzt). Andererseits können Datenbanken dazu erweitert werden, XML zu unterstützen und damit z.B. auch Anfragen mit Hilfe der Anfragesprache XQuery o.ä. zu bearbeiten. Der durch einige weitere Entwicklungen propagierte Weg ist der Einsatz von sog. nativen XML -Datenbanken. Diese verbinden eine komplexe Zwischenschicht mit einer eigenen Persistenzschicht (Datenablage), um so alle an eine Datenbank gestellten Anforderungen zu erfüllen[21]. Eines dieser Konzepte ist z.B. die Transaktionsarchitektur zum Management von Internet-Objekten (Tamino) der Software AG, welche als Integrationsserver nicht nur eine native Datenbank stellt, sondern auch bestehende Datenbanksysteme integriert und XML -fähig macht[22]. Gerade auch die Integration solcher neuen Technologien sollte für weitere Entwicklungen im Auge behalten werden, sofern sie auch auf Serverseite unterschiedliche Betriebssysteme unterstützen.

Im gegebenen Umfeld der Arbeit soll aber auf vertraute Techniken aufgebaut werden, weshalb der grundsätzliche Einsatz von XML vorbereitet werden soll. Weitere Bestandteile der XML -Familie unterstützen dies und verbessern vor allem die Überprüfung der Daten auf korrekte Syntax und Semantik.

5.2.2 Modellierung und Validierung durch Schemas

XSD statt DTD zur Dokumentenvalidierung

Die bereits erwähnte DTD ist nicht im Stil von XML aufgebaut, sondern folgt zur Modellierung von Dokumenten eigenen Regeln. Zudem bietet sie mit ihren eingeschränkten Mechanismen (Festlegung von Element- und Attributstrukturen, Vorgabe von Standardwerten, Entities und Notationen) nur ungenügende Validierungsmöglichkeiten. Gerade aber die Validierung ist ein äußerst wichtiger Schritt zur Fehlervermeidung in Datendokumenten[23]. Aus diesen Gründen wird zur Datenmodellierung statt der DTD ein Schema-Umfeld mit der XML Schema Definition (XSD) eingesetzt.

XSD agiert im lexikalischen Dartsellungsraum und im Werteraum

Die DTD wirkt im Grunde im lexikalischen Sinne. Auch XML Schema kann den lexikalischen Darstellungsraum validieren und entsprechend Whitespaces (für die momentane Darstellung unwichtige Zeichen) ignorieren, normalisieren oder zusammenfassen. Zusätzlich ist es mit XSD möglich, Restriktionen für den Werteraum zu definieren und damit eine semantische Überprüfung der Datentypen durchzuführen[24].

Werterestriktionen

Im Grunde genommen wird dies möglich durch die Festlegung von regulären Ausdrücken zur Spezifikation einfacher Datentypen. Diese Muster wirken nicht lexikalisch, sondern in einem wesentlich strengeren Sinne als Wertrestriktionen[25].

[21] vgl. dazu [KAZ02] a.a.O. S. 287 f.
[22] vgl. dazu [KAZ02] a.a.O. S. 312 f.
[23] vgl. [VLI03] a.a.O. S. 1 f.
[24] vgl. dazu auch [VLI03] a.a.O. S. 23 f.
[25] vgl. dazu auch [VLI03] a.a.O. S. 76 f.

Aus den einfachen Datentypen (sind unabhängig von anderen Knoten) lassen sich komplexere ableiten. Dabei handelt es sich um Strukturtypen, welche die Auszeichnungsstruktur eines Datums festlegen[26].

Die Erweiterung um Schlüssel und Verweise darauf vervollständigen die Möglichkeiten, da hier, statt wie bei den Eindeutigkeits-Pendants ID und IDREF, keine Beschränkungen der Datentypen vorliegen und eine Wertevalidierung durchgeführt wird[27].

Implizite Dokumentation

Mit Hilfe von XSD lassen sich somit diffizile Strukturen beschreiben, was gerade aber auch eine zusätzlich gewonnene Dokumentation der Dokumentstrukturen und Dateninhalten bildet. So nutzen z.B. auch die Entwicklungswerkzeuge diese Technik, um graphisch mögliche, erlaubte Alternativen bei der Erstellung von XML - Dokumenten zu liefern[28].

XSD im Einsatz als eine Art Dokument-Firewall

Der wichtigste Vorteil von XSD sei aber nochmals erwähnt: Es wird sowohl die Strukturvalidierung (Struktur erfüllt bestimmte Bedingungen) als auch die Datenvalidierung (Daten genügen gegebenen Regeln für Informationen eines Dokuments) automatisch durch die XML -Prozessoren mitunterstützt. Somit kann dieser Mechanismus als eine Art Firewall für weniger vertrauensvolle Datenlieferanten eingesetzt werden[29]. Deshalb wird für alle weiteren Entwicklungen zur Datenrepräsentation im Zusammenhang mit dieser Arbeit zur Modellierung XSD eingesetzt.

Lösungen zur Integration von XML-fremden Formaten notwendig

Zusammengefasst ist damit festzustellen, dass die XML -Welt gut durchorganisiert und trotzdem flexibel handhabbar ist, solange man sich nicht daraus entfernt. Dokumente außerhalb von XML können nur über Umwege integriert werden. Sie müssen mit expliziten Programmen in XML -Strukturen umgewandelt werden. Vorteilhaft wäre, wenn eine solche Wandlung auch verallgemeinert über Möglichkeiten aus der XML -Welt ermöglicht würden.

Um solche Transformationen ähnlich zu den XSLT zu ermöglichen, bietet es sich an, dieses Konzept näher zu beleuchten.

5.2.3 Die Idee hinter einer neuen Transformationssprache

XSLT zur Beschreibung von Transformationen

Innerhalb der XML -Welt gibt es zahlreiche Möglichkeiten, Dokumenten einen anderen Charakter zu geben[30]. Die älteste Möglichkeit sind Cascading Style Sheets (CSS) . Sie erlauben aber keine Reihenfolgenänderungen, interne Berechnungen oder Kombinationen von mehreren Dokumenten. Deshalb wurde Extensible Stylesheet Language for Transformations (XSLT) als leistungsfähige, flexible Sprache zum Transformieren entwickelt.

Grundlage sind Muster und Ersetzungsregeln

Solche XSLT -Dokumente sind wiederum in XML gehalten, so dass Entwickler die gewohnte Umgebung nicht verlassen müssen. Grundlage ist die Erkennung von Mustern und Zuordnung von Ersetzungsregeln, welche ausgeführt werden, wenn

[26] vgl. dazu auch [VLI03] a.a.O. S. 95
[27] vgl. dazu auch [VLI03] a.a.O. S. 162
[28] vgl. [VLI03] a.a.O. S. 3 f.
[29] vgl. [VLI03] a.a.O. S. 1 f.
[30] vgl. zum folgenden Abschnitt [TID02] a.a.O. S. 2 ff.

das Muster identifiziert wurde. Angelegte Variablen sind nicht mehr modifizierbar, was aus Anlehnung an funktionale Programmiersprachen, wie z.B. Lisp, eingeführt wurde. Allerdings gibt es in XSLT keine Schleifen, wie in üblichen Programmiersprachen.

Iterationen und Rekursionen statt Schleifen

An ihre Stelle treten Iteration und Rekursion. Iterationen besagen, dass für alle gefundenen, gleichartigen Elemente eine bestimmte Regel iterativ (in Folge) ausgeführt werden sollen. Rekursionen hingegen besagen, dass Regeln sich selbst immer wieder aktivieren. Dadurch wird eine Baumstruktur komplett durchlaufen. Allerdings ist diese Vorgehensweise zwar intuitiv verständlich, jedoch nicht unbedingt gleich praktisch umsetzbar.

Vorteile der Transformationsbeschreibung

Die Vorteile von XSLT liegen damit klar auf der Hand. Ihre Mächtigkeit erlaubt nahezu alle Wandlungen von Dokumenten auf höchst flexible Art und Weise. Die innere Anatomie der Sprache ist dabei von XPath-Ausdrücken geprägt[31], da anhand dieser durch das Dokument navigiert wird. Anhand der Knotentypen und ihrer Beziehungen zueinander werden Elemente relativ oder absolut lokalisiert. Wildcards (Ersatzzeichen, zur Quantifizierung beliebiger anderer Zeichen) erlauben den Zugriff auf ähnlich strukturierte Elemente. XPath definiert Verwandschafts-Achsen (z.B. child-Achse für alle Kinder eines Kontextknotens), welche in ausführlicher und abgekürzter Syntax genutzt werden können.

Es steht außer Frage, dass an dieser Stelle eine genügend ausführliche Beschreibung der überaus mächtigen Sprache XSLT möglich wäre. Zumindest soll hier aber ein kurzer Abriss gegeben werden, der in einem Beispiel verdeutlicht ist.

Verzweigungs- und Steuerregeln agieren auf Umschreibungen für Elemente

Hauptbestandteile eines Stylesheets sind Verzweigungs- und Steuerelemente[32]. Zur Verzweigung dienen ähnliche Bedingungskonstrukte wie in anderen Programmiersprachen (z.B. <xsl:if>). Die Suche nach Elementknoten wird mittels Templates beschrieben. Das sind Umschreibungen von Elementen, welche in Form eines Polymorphismus über das gesamte Dokument hinweg bearbeitet werden. Jeder Treffer (Match) führt die zugeordneten Anweisungen aus. Inhalte von Elementknoten und/oder ihre Parameter können ausgegeben werden. Definierte Variablen dienen zur Zwischenspeicherung von Ergebnissen. Zusätzlich können auch interne Querverweise erzeugt werden. Hauptmittel zum Zweck ist für viele Aufgaben die Rekursion (z.B. zur Nachbildung von Schleifenabläufen).

Beispiel einer Umwandlung

Nachfolgendes Beispiel zeigt das Stylesheet zur Transformation des im vorhergehenden Abschnitt 5.2.1 auf Seite 134 angegebenen XML -Beispiels nach HTML . Nachdem sowohl die XML -Quelldatei als auch die Transformationsdatei geparst und als Baum im Transformator liegen, beginnt im Beispiel die Umwandlung bei der Dokumenten Wurzel (match="/"). Dort sucht er den Elementknoten <gruss> und schreibt bei Fund den Inhalt von <xsl:template> in die Ausgabe. Dabei wird der Wert, d.h. der Inhalt, des Elementknotens („Hallo, Welt!“) eingefügt. Dies würde für alle <gruss>-Elemente geschehen[33].

[31] vgl. zum folgenden Abschnitt [TID02] a.a.O. S. 43 ff.
[32] vgl. zum folgenden Abschnitt [TID02] a.a.O. S. 67 ff.
[33] vgl. [TID02] a.a.O. S. 27 f.

Beispiel 5.2 für „Hello World-Transformation“ (nach [TID02][34]

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.0">
   <xsl:output method="html"/>

   <xsl:template match="/">
     <xsl:apply-templates select="gruss"/>
   </xsl:template>

   <xsl:template match="gruss">
     <html>
       <body>
         <h1>
           <xsl:value-of select="."/>
         </h1>
       </body>
     </html>
   </xsl:template>
</xsl:stylesheet>
```

Mächtigkeit einer solchen Umwandlungsbeschreibung zeigt sich

Die kurze Beschreibung zeigt, dass sich das Paket um XML als äußerst praktikabel herausstellt. Grund für die Vielschichtigkeit ist, weil „[...] man vom durchschnittlichen Web-Benutzer nicht erwarten kann, Java-, Visual Basic-, Perl- oder Python-Code zu programmieren, um mit Dokumenten zu arbeiten.“[35] Er kann somit innerhalb seiner Sprachwelt bleiben, die er beherrschen muss. Diese Fürsorge beschränkt sich allerdings nur darauf, wenn der besagte Web-Nutzer nichts anderes als XML als Ausgangspunkt nutzen will.

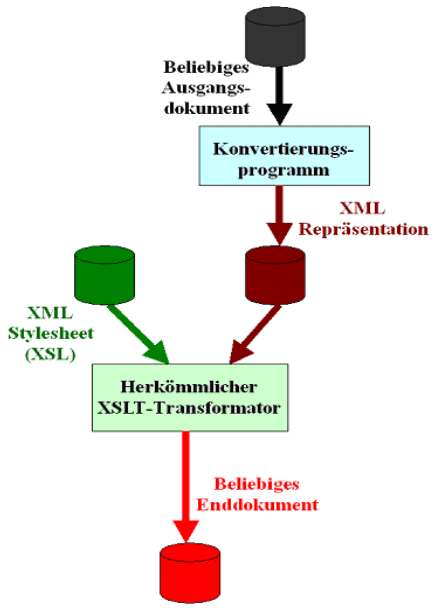


Abbildung 5.1: Der herkömmliche Ablauf zur Verarbeitung beliebiger Formate

Heterogenität der Formate erfordert letztendlich Umwandlungen mittels eigener Transformationsprogramme

Die reale Informationswelt sieht hier allerdings erheblich anders aus. Die Heterogenität setzt sich auch auf dem Niveau der Datendarstellungen fort. Und die „wohlgeformte“ XML -Welt bietet keinerlei Berührungspunkte zur Integration dieser Nicht-XML -Welt. So muss der Web-Nutzer schließlich doch wieder auf weitere Programmiersprachen zurückgreifen. Mit Hilfe von externen Programmen werden

[34] vgl. [TID02] a.a.O. S. 24
[35] [TID02] a.a.O. S. 1

damit aus herkömmlichen Datenbeständen XML -konforme. Ein Verarbeitungsablauf sieht wie in Abb. 5.1 auf Seite 138 dargestellt aus. Dies ist mehr als unzufriedestellend.

Idee zu einem Prä-Prozessor für die Transformation allgemeiner Formate

Aus diesem Mangel heraus und der unabstreitbaren Mächtigkeit der Transformation in XML entstand die Überlegung, eine Art Prä-Prozessor vor der eigentlichen Umwandlung zu entwicklen. Dieser nutzt eine zusätzliche Datei mit Strukturregeln, welche individuell auf beliebige Formate angepasst werden können und den Sprachkontext von XML nicht verlassen. So entstand eine eigene Sprachdefinition mit der Bezeichnung A2X .

5.3 Sprachdesign: Anything to XML (A2X)

Die neue Sprache ist eine XML -konforme Erweiterung und dient als Bindeglied zwischen beliebigen Dokumenten und XML .

5.3.1 Festlegungen

Nutzung eines Entwicklungswerkzeugs für XML

Während den Entwicklungsarbeiten hat sich herausgestellt, dass dies ohne geeignete Werkzeuge extrem schwierig ist. Deshalb wurde das XSD -Design zur neuen Sprache mit Hilfe der Entwicklungsumgebung XMLSpy von Altova erstellt. Dabei handelt es sich um eine Art graphische Programmierung für XML , wodurch übersichtlich XSD geschaffen und zugehörige XML -Dateien erstellt werden können. Zusätzlich unterstützt es die Definition von Transformationsvorgaben etc. Für alle weiteren Aufgaben werden die bereits erwähnten, freien Programme von Apache eingesetzt.

Neue Sprache ähnlich zu herkömmlichen Programmiersprachen handhabbar

Die neue Sprache selbst orientiert sich hauptsächlich an herkömmlichen Programmiersprachen. Zudem wird die Rekursivität nach Möglichkeit vermieden, da dies wohl einfacher vom Nutzer umgesetzt werden kann. Dies bedeutet keine wesentliche Einschränkung, da zumindest die Klasse der μ-rekursiven Funktionen exakt mit der Klasse von WHILE-, GOTO- bzw. turing-berechenbaren Funktionen übereinstimmt[36], also durch herkömmliche Programmkonstrukte nachgebildet werden kann.

Optimal für eine Programmierung mit graphischen Werkzeugen geeignet

Die Sprache beschränkt sich auf die wesentlichen Elemente einer Programmierung. Die Entwicklung von A2X -konformen Umwandlungsstrukturen ist von Grund auf für graphische Editoren gedacht, so dass ein übersichtliches Transformationsdokument erstellt werden kann.

5.3.2 Der neue Verarbeitungsablauf für Transformationen

Gewohnte Vorgehensweise wie bei XSLT bleibt erhalten

Grundidee hinter der neuen Verarbeitung ist, dem Web-Nutzer die Verarbeitung von herkömmlichen Dokumenten mit gewohnten XML -Methoden zu ermöglichen. Dabei soll der Lernaufwand für die Nutzer möglichst gering sein und es soll die Mächtigkeit der XSLT -Transformationen weiterbestehen.

[36] vgl. [SCHOE97] a.a.O. S. 115

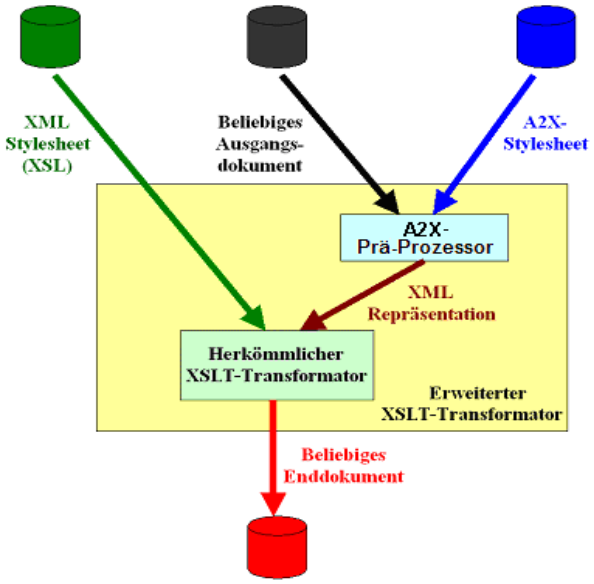


Abbildung 5.2: Der erweiterte Ablauf für A2X zur Verarbeitung beliebiger Formate

Prä-Prozessor-Konzept

Um dies zu ermöglichen, ist die neue Sprache intuitiv verständlich als Graphenbeschreibung umgesetzt und als eine Art Prä-Prozessor vor den eigentlichen Transformator gesetzt. Sie ist somit quasi eine Erweiterung zu XSLT . Es wird eine beliebige Datei mit Regeln beschrieben, welche zum Parsen des Inhalts genutzt werden. Der A2X -Parser erkennt die beschriebenen Token und setzt sie entsprechend der Angaben in ein relativ flaches XML -Design um, welches eine lineare Abfolge der nun mit Auszeichnungen (Tags) versehenen, ursprünglichen Token ist. An dieser Stelle kann damit die flexible und mächtige Transformationssprache XSLT zur weiteren Abbildung eingesetzt werden.

Zusatz zu herkömmlichen Transformatoren

Aus dem Ablauf in Abb. 5.2 auf Seite 140 wird ersichtlich, dass es sich bei den Entwicklungen um einen Zusatz zu den herkömmlichen Transformatoren handelt. Gerade aber diese kleine Erweiterung ermöglicht eine Konvertierung von einer beliebigen Darstellung eines Dokuments in eine beliebige andere.

Wie bereits in den Ausgangsüberlegungen erwähnt, kann keine allgemeingültige Umwandlung für alle beliebigen Dokumente in ihrer Allgemeinheit vorgegeben werden. Jede Beschreibung einer Umwandlung ist in ihre Abläufe individuell eingebettet. Der Einsatz lohnt sich damit nicht für Einzeldokumente sondern nur, wenn wiederholt Dokumente gleicher Charakteristik umgewandelt werden sollen. Diese kann auf simplen Bytezuordnungen bis hin zu übergeordneten Strukturierungen aufsetzen. Der Strukturierungsgrad des resultierenden XML -Dokuments ist jedoch bei gleichgehaltenem Informationsgehalt unmittelbar mit dem des Ausgangsdokuments gekoppelt und damit entweder geringer oder gleichwertig dazu. Damit lassen sich keine beliebigen Strukturierungen im XML -Resultat erzielen, welche nicht auch schon im Ursprungsdokument ersichtlich sind. Das Enddokument ist nur so gut, wie es sein Ursprung erlaubt. Es sind damit zwar beliebige Dokumente umsetzbar, wobei aber gilt: Je strukturierter und interpretierbarer das Ausgangsdokument ist, umso vielfältiger sind die Möglichkeiten, einzelne Informationseinheiten zu identifizieren und durch eigene XML -Konstrukte kenntlich zu

machen. In letzter Konzequenz bedeutet dies, dass komplett unstrukturierte, freie Dokumente somit nur in eine Kopie des Originalinhalts mit einer XML -Hülle gewandelt werden können und damit wenig sinnvoll zu nutzen sind.

Der Arbeitsablauf zur Vorbereitung einer Umwandlung lässt sich dabei in die folgenden Arbeitsschritte zerlegen:

1. Analyse des Ausgangsdokuments mit dem Ziel, daraus benötigte Informationseinheiten, ihre Identifizierbarkeit, ihre Abfolge und evtl. Beziehungen untereinander im Dokument festzustellen. Daraus ergibt sich im Prinzip die DTD des flachen XML -Zwischenformats als erste XML -Repräsentation
2. Definition der Struktur des resultierenden XML -Enddokuments mittels DTD oder XSD und Festlegung des Aufbaus eines Resultatdokuments beliebigen Formats
3. Definition der Umwandlung des Zwischendokuments in das Enddokument mittels XSLT
4. Definition der A2X -Parserregeln zum Einlesen des Originaldokuments und zum Umwandeln in das Zwischenformat

Nachfolgend wird nun die entworfene Sprache beschrieben. Es soll bereits hier erwähnt werden, dass es noch keine Umsetzung eines erweiterten Transformators gibt, da dies im zeitlichen Umfang der Arbeiten nicht mehr möglich gewesen wäre. Allerdings liegt in diesem Ansatz viel Potential zur Anbindung von Dokumenten außerhalb der XML -Umgebung.

5.3.3 Die neue Sprache A2X

A2X als erster Vorschlag eines Entwurfs

Eine automatisch vom Bearbeitungswerkzeug XMLSpy der Firma Altova generierte, komplette Dokumentation zur neuen Sprache A2X ist im Anhang K auf Seite 277 abgedruckt. Die komplette Sprachbeschreibung in XSD ist im Anhang L auf Seite 305 zu finden. An dieser Stelle nun soll der Aufbau von A2X erläutert werden. Dabei handelt es sich um einen ersten Vorschlag, welcher sukzessive noch weiterentwickelt werden muss.

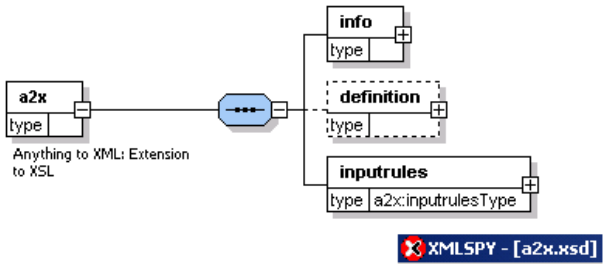


Abbildung 5.3: Die oberste Hierarchiestufe von A2X

Die oberste Hierarchiestufe

Die oberste Hierarchiestufe (vgl. Abb. 5.3 auf Seite 141) einer A2X -Beschreibung zeigt drei Hauptbestandteile, von denen zwei Pflicht sind: einem Informationsabschnitt (<info> ist Pflicht), dem Definitionsteil (<definition> ist optional) und den Eingaberegeln (<inputrules> ist ebenfalls Pflicht).

Der Infoteil über A2X

Im Infoteil (vgl. Abb. 5.4 auf Seite 142) befinden sich einerseits Angaben zu A2X (<a2xspecification>), welche die genutzte Definitionsstufe als Revisionsdaten (Versionsnummer, Datum der Freigabe und Internetadresse der verantwortlichen Autorisierungsstelle) und zugehörige, optionale Kommentare offenlegen. Da das gesamte A2X -Dokument eine Formatkonvertierung beschreibt, werden im Infoteil an dieser Stelle allgemeine Angaben dazu angegeben(<convspecification>). Dazu sind Revisionsbezeichnung, Versionsnummer, Erstellungsdatum, Angaben zur verantwortlichen Institution und die Namen der Entwickler eingetragen. Wichtig ist hier auch die Angabe eines Wurzelelements für das später automatisch generierte XML -Dokument. Auch hier kann wieder optional ein Kommentaren ergänzt werden.

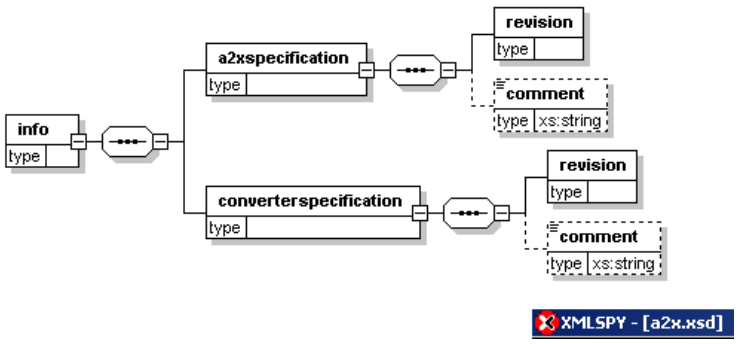


Abbildung 5.4: Der Infoteil

Der Definitionsteil für Variablen

Als nächstes folgt der optionale Definitionsteil (vgl. Abb. 5.5 auf Seite 143). Er dient hauptsächlich zur Deklaration und Definition von Variablen (<declare>). Sie sind generell global und allgemeingültig, was keine Einschränkungen bedeutet, jedoch einen verantwortungsvollen Umgang erfordert, da sie von überall aus benutzt werden können. Eine Variable <variable> legt über die Attributangaben ihren Kontext fest. Sie hat einen Namen als Bezeichner, welcher als Restriktion mit der Buchstabenkombination „VAR“ beginnen muss. Er ist vom Typ ID und damit ein im späteren Dokument referenzierbarer, eindeutiger Bezugspunkt. Ferner kann der Variable ein XML -Tag zugewiesen werden, falls sie in die XML -Hierarchie aufgenommen werden soll. Jede Ausgabe einer mit Tag versehenen Variablen erzeugt im Speicher ein äquivalentes Abbild, als wäre „<TAG> Variableninhalt </TAG>“ gelesen worden. Nur so kann eine folgende XSD -Beschreibung darauf weitere Transformationen ausüben. Alle anderen Variablen dienen der internen Verarbeitung während einer Transformation und können nicht als Ausgabe genutzt werden.

Variablen sind von einem bestimmten Basistyp und können mit Initialwerten belegt werden. Es gibt keine komplexeren, zusätzlich definierten Typen. Die Basistypen sind in Tabelle 5.1 auf Seite 143 angegeben. Zusätzlich kann jeder Variable ein beschreibender Kommentar als Text in einem eingegliederten Element hinzugefügt werden.

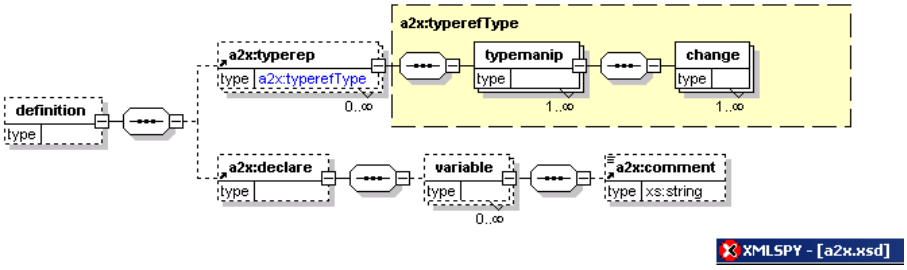


Abbildung 5.5: Der Definitionsteil

Zusätzliche Angabe einer Typrepräsentation

Ein weiterer, nützlicher Bestandteil des Definitionsteils ist die Angabe von eigenen Typrepräsentationen (<typerep>). Da mit A2X beliebige Dokumente eingelesen werden können, so z.B. auch binäre Darstellungen, welche sich von Plattform zu Plattform unterscheiden können, ist hier eine rudimentäre Möglichkeit geschaffen, Bit- und Bytemanipulationen durchzuführen und direkt einen plattformspezifischen Kontext (Anzahl der Bytes für bestimmte Typen etc.) zu wählen.

A2X-Datentype	Äquivalenter C/C++-Datentyp
ushort	unsigned short (kurze Ganzzahl, vorzeichenlos)
uint	unsigned int (allgemeine Ganzzahl, vorzeichenlos)
ulong	unsigned long (lange Ganzzahl, vorzeichenlos)
short	short (kurze Ganzzahl, vorzeichenbehaftet)
int	int (allgemeine Ganzzahl, vorzeichenbehaftet)
long	long (lange Ganzzahl, vorzeichenbehaftet)
float	float (kurze Gleitpunktzahl)
double	double (lange Gleitpunktzahl)
char	char (ein Zeichen)
string	std::string Container für Zeichenketten

Tabelle 5.1: Basisdatentypen in A2X

Jede Repräsentation ist in den Parameterangaben eindeutig bezeichnet (Restriktion: Name beginnt mit „REP“) und kann auf eine Internetadresse mit einem kompletten Satz an Repräsentationsinformationen verweisen. In einer solchen Repräsentation wird für beliebige Basistypen die genutzte Darstellung mit entsprechenden Manipulationen (<typemanip>) beschrieben. Die Attribute legen dabei generelle Kontextanforderungen fest (Byteanzahl und Reihenfolge des Quelltyps und des Zieltyps). Werden sie nicht angegeben, wird der Darstellungstyp der wandelnden Plattform genutzt. Eine Manipulation enthält nur noch eine Reihe von Umtauschanweisungen (<change/>), welche ein angegebenes Quellbit eines bestimmten Quellbytes in ein Zielbit eines Zielbytes verschiebt. Dabei handelt es sich nicht um eine Folge von aufeinander aufbauenden Befehlen (d.h., ein Bit könnte mehrfach verschoben werden, bis es wieder am Ausgangspunkt landet).

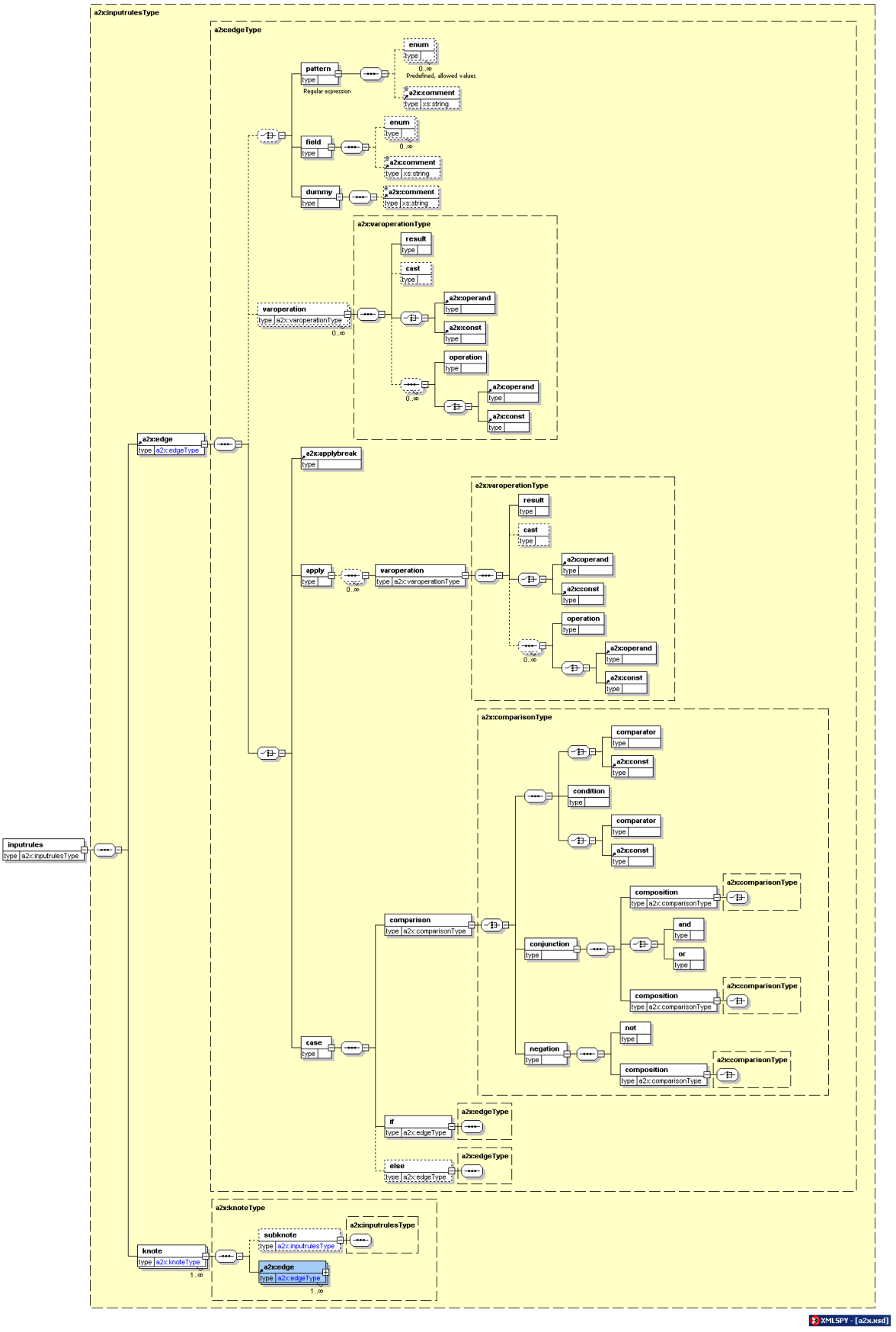


Abbildung 5.6: Der Hauptteil mit den Beschreibungsregeln

Vielmehr wird eine abbildende Kopie einer Variable angelegt, in welche dann die Verschiebungen aus dem Original überlagert werden. Jedes Bit kann also nur einmal verschoben werden.

Mangel im XSD-Standard: Inhalte von Attributen lassen sich nicht in gegenseitige, semantische Abhängigkeit bringen (sog. Business-Rules)

Bereits hier zeigt sich ein Mangel in der momentanen Standardisierung von `XSD` . Es ist nicht möglich Inhalte von Attributen gegeneinander in semantische Beziehung zu setzen, um damit bereits beim Parsen fehlerhafte Einträge oder wie hier Mehrfachnennungen zu vermeiden. Bei `A2X` wurde jedoch versucht, möglichst alle benötigten syntaktischen und semantischen Überprüfungen anhand der `XSD` -Beschreibung auf den `XML` -Parser abzuwälzen. Der genannte Mangel führt aber zwangsläufig dazu, dass in einem erweiterten Parser die Überwachung der benötigten, semantischen Beziehungen explizit als Ergänzung programmiert werden muss, was sehr unbefriedigend ist.

Der Hauptteil von A2X: die Beschreibungsregeln

Sind die genannten Blöcke (<info>, <definition>) ausreichend mit Inhalt gefüllt, folgt der eigentliche Hauptteil einer `A2X` -Beschreibung, die Eingaberegeln (<inputrules>; vgl. Abb. 5.6 auf Seite 144). Sie beschreiben den syntaktischen und semantischen Aufbau eines Quelldokuments in Form eines Graphen. Er besteht aus Knoten (<knote>) und Kanten (<edge>) und verfügt zudem über Speicher (die definierten Variablen), anhand dessen Inhalt auch eine Entscheidung für einen möglichen Übergang stattfinden kann. Ferner kann sich ein Graph iterativ in mehrere, kleinere Subgraphen (<subknote>) zerlegen, welche denselben Eigenschaften folgen wie der übergeordnete Hauptgraph und damit wieder wie die allgemeinen Eingaberegeln aufgebaut sind. Diese logische Aufsplittung ermöglicht quasi eine Quotierung und damit Blockbildung, wie sie vergleichbar auch in der herkömmlichen Programmierung bei der Unterteilung eines Programms in mehrere Prozeduren eingesetzt wird.

Startkante als Einstiegspunkt in einen Teilgraph

Jeder Block (so auch der Hauptblock) beginnt mit einer Startkante, welche den Startzustand referenziert. Aus diesem Grund wird auch mit der Betrachtung der Kantenbeschreibung (<edge>) begonnen. Sie bildet die lexikalische Komponente. Kanten sind in erster Linie Zustandsübergänge mit lexikalischen Bedingungen. Sie können mit generellen Operationen verknüpft sein, welche bei jedem Übergang ausgeführt werden. Dabei stellen Kanten eine Referenz zu möglichen, folgenden Zuständen her, welche durch die Knoten repräsentiert werden. Eine Besonderheit ist, dass Zustandswechsel an Logikbedingungen geknüpft werden können, welche explizite Operationen mit auslöst.

Kanten bilden die Basisbausteine der lexikalische Struktur eines Dokuments

Kantenübergänge bilden durch ihre lexikalischen Bedingungen den Mechanismus, ein Dokument entsprechend der definierten Beschreibung zu validieren. Kanten können dabei Variablen mit Inhalt füllen und ein entsprechendes Eingliedern in Tags erlauben (Attribut TaggingOn). Die lexikalische Prüfung kann dabei mittels Pattern (<pattern>; reguläre Ausdrücke), vordefinierter Felder fester Breite (<field>) oder ohne Einschränkungen immer (<dummy>; automatischer Übergang ohne lexikalische Zuordnung) stattfinden, sofern der Inhalt bis zur nächsten lexikalischen Angabe der folgenden Kante nicht näher analysiert werden braucht. Solche <dummy>-Übergänge dienen auch zur Prüfung von externen Ereignissen (Attribut event), wobei momentan nur das Erkennen des Dateiendes (`End Of File (EOF)`) berücksichtigt ist. Reguläre Ausdrücke werden in den Pattern entsprechend allgemeiner Definitionen in der Art von Perl gehandhabt. Die lexikalischen Entsprechungen aus dem Dokument können an Variablen geknüpft werden,

in welche die erkannten Abschnitte abgelegt werden. Über beliebig viele eingegliederte Element zur Enumeration (<enum/>), können für nicht-automatische Kanten erlaubte Wertevorgaben zur Restriktion angegeben werden. Auch bei den lexikalischen Bedingungen kann wieder ein Kommentar eingefügt sein.

Kanten erlauben Operationen auf definierten Variablen

Weitere Bedeutung kommt den Kantendefinitionen durch die Operationen auf definierten Variablen zu. Nur an dieser Stelle sind Operationen erlaubt. Ob sie nun an bestimmte logische Bedingungen geknüpft sind oder generell ausgeführt werden (z.B. bei der Inkrementation eines Zählers), ihnen allen ist eine Operationsstruktur gemein. Sie bilden im Prinzip die gewohnte Vorgehensweise in Programmiersprachen nach. Ein Operand (Inhalt einer Variable) oder eine Konstante wird über eine Operation mit einem weiteren Operanden oder einer Konstante auf ein Ergebnis (Resultat wird in eine Variable abgelegt) abgebildet. Dabei kann eine beliebige Wiederholung von Operationen und Operanden/Konstanten zusammenwirken. Eine Quotierung ist nicht vorgesehen, was bedeutet, dass eine Klammerrechnung in eigene Operationseinheiten zerlegt werden muss. Allerdings kann jeder Operand und jede Konstatnte über eine Attributangabe bzgl. des Typs mit einem „cast“ (explizite Typwandlung, es gibt keine implizite) versehen werden. Auch das Gesamtergebnis kann vor der Resultatübergabe mit einem <cast/>-Element explizit gewandelt werden. Die erlaubten Operationen sind in Tabelle 5.2 auf Seite 146 angegeben.

Mangel im XSD-Standard macht sich im Rahmen der semantischen Prüfung einer Operation erheblich bemerkbar

Auch bei den Operationen führt die fehlende Validierung von Inhalten der Attribute untereinander zu einem erheblichen Mangel. Variablen und Konstanten einer Operation müssen vom gleichen Typ oder mit „Casting“-Informationen versehen sein. Eine solche Überprüfung ist allerdings nicht mittels der Sprachdefinition von XSD möglich, welche sich auf Struktur- und Datenvalidierung beschränkt, sondern muss in einem Zusatz zum Parser selbst als prozeduraler Code implementiert werden. Die Flexibilität und Abgeschlossenheit der XML -Umgebung leidet durch dieses Fehlen einer Validierung von sog. „Business-Regeln“ bzgl. Beziehungen zwischen Informationen und auf höher angelegte Sinnhaftigkeit erheblich[37].

A2X-Operation	Beschreibung
+	mathematische Addition
-	mathematische Subtraktion
*	mathematische Multiplikation
/	mathematische Division
%	mathematische Modulo-Opertion
cat	lexikalische Konkatenation

Tabelle 5.2: Operationen in A2X

Eigentliche Aufgabe der Kante: Aufspannen eines Graphen

Die eigentliche Aufgabe einer Kante ist die Referenzierung eines Nachfolgeknotens. Diese wird über den Verweis zum Anwenden (<apply>) der Kante festgelegt. Dort befindet sich im Attribut „next“ ein Verweis zum nächsten Knoten. Mehrere anzuwendende Kantenübergänge als Folge werden in der Reihe ihres Auftretens bearbeitet. Der erste anwendbare Übergang wird ausgeführt. Zusätzlich kann als Element eines <apply>-Elements eine Variablenoperation nach oben beschriebenem Muster eingefügt werden, welche nur bei diesem Übergang ausgeführt wird.

[37] vgl. dazu auch [VLI03] a.a.O. S. 2

Markierung eines Endzustands

Eine Sonderform einer Kante ist die Markierung eines Endzustands (<applybreak/>) für den Block, in dem diese Kante angewandt wird. Wird diese Kante genutzt, so wird an den übergeordneten Block zurückübergeben, bzw. dort der Durchlauf beendet.

Bedingte Kantenübergänge

Um eine erhöhte Flexibilität zu geben und damit auch Bedingungen aus den Kontextzusammenhängen zu ermöglichen, wurden für eine Kante auch Elemente zur Aussagenlogik in der Art einer herkömmlichen IF-Anweisung integriert. Sie bestehen in A2X aus den Teilen <comparison> (logische Vergleich), <if> (wird ausgeführt, wenn der Vergleich wahr ist) und <else> (wird ausgeführt, wenn der Vergleich falsch ist). Diese Teile sind im Element <case> vereinigt. Der IF- und der optionale ELSE-Teil setzen sich wieder aus einer Kantenbeschreibung zusammen, so wie sie vorweg beschrieben wurde. Besonderes Augenmerk sollte man darauf werfen, dass sich die Verzweigung durch die Klammerung anhand der Auszeichnungen als eindeutig erweist und somit das Problem des „Dangling Else“ nicht auftreten kann. Dabei handelt es sich bei herkömmlichen Sprachen um die Problematik, dass ein optionaler ELSE-Zweig bei geschachtelten Konstruktionen nicht mehr eindeutig zugeordnet werden kann[38]. Durch die in XML vorgeschriebenen Endemarken in der Auszeichnung ist aber die Zuordnung eines folgenden ELSE-Zweiges eindeutig zu einem vorausgehenden IF-Zweig gegeben.

Bedingungen in Form von Aussagenlogik

Bei der Bedingung selbst wird eine Variable (<comparator/>) oder eine Konstante (<const/>) in vergleichende Beziehung zu einer anderen Variable oder Konstante gesetzt (atomare Formel der Aussagelogik). Die möglichen Vergleichsoperationen sind in Tabelle 5.3 auf Seite 147 dargestellt. Es ist auch möglich, mehrere Verknüpfungen von Bedingungen (Kompositionen <composition> mit <and/> oder <or/> verknüpfen) in eine Art Klammerkonstrukt zu setzen (<conjunction>) und so geschachtelte Abfragen zu ermöglichen. Weiter kann eine Komposition auch negiert werden. Aus der Verbindung all dieser Elemente wird die komplette Syntax der Aussagenlogik erbracht, welche als induktiver Prozess aus atomaren Formeln und darauf wirkenden Disjunktionen (<or/>), Konjunktionen (<and/>) und Negationen (<not/>) entsteht[39].

A2X-Vergleichsoperation	Beschreibung
lt	kleiner als („<“, „lower than“)
eq	gleich („==“, „equal“)
gt	größer als („>“, „greater than“)
!lt	nicht kleiner als entspricht größer oder gleich („>=“)
!eq	nicht gleich entspricht kleiner oder größer („!=“)
!gt	nicht größer als entspricht kleiner oder gleich („<=“)

Tabelle 5.3: Vergleichende Operationen in A2X

Zusammengefasst sind damit Kanten im Prinzip die aktiven Bindeglieder zwischen den statischen Knoten. Die Knoten selbst sind deshalb relativ simpel gehalten.

[38] vgl. dazu [JOB92] a.a.O. S. 23 f.
[39] vgl. [SCHOE95] a.a.O. S. 24

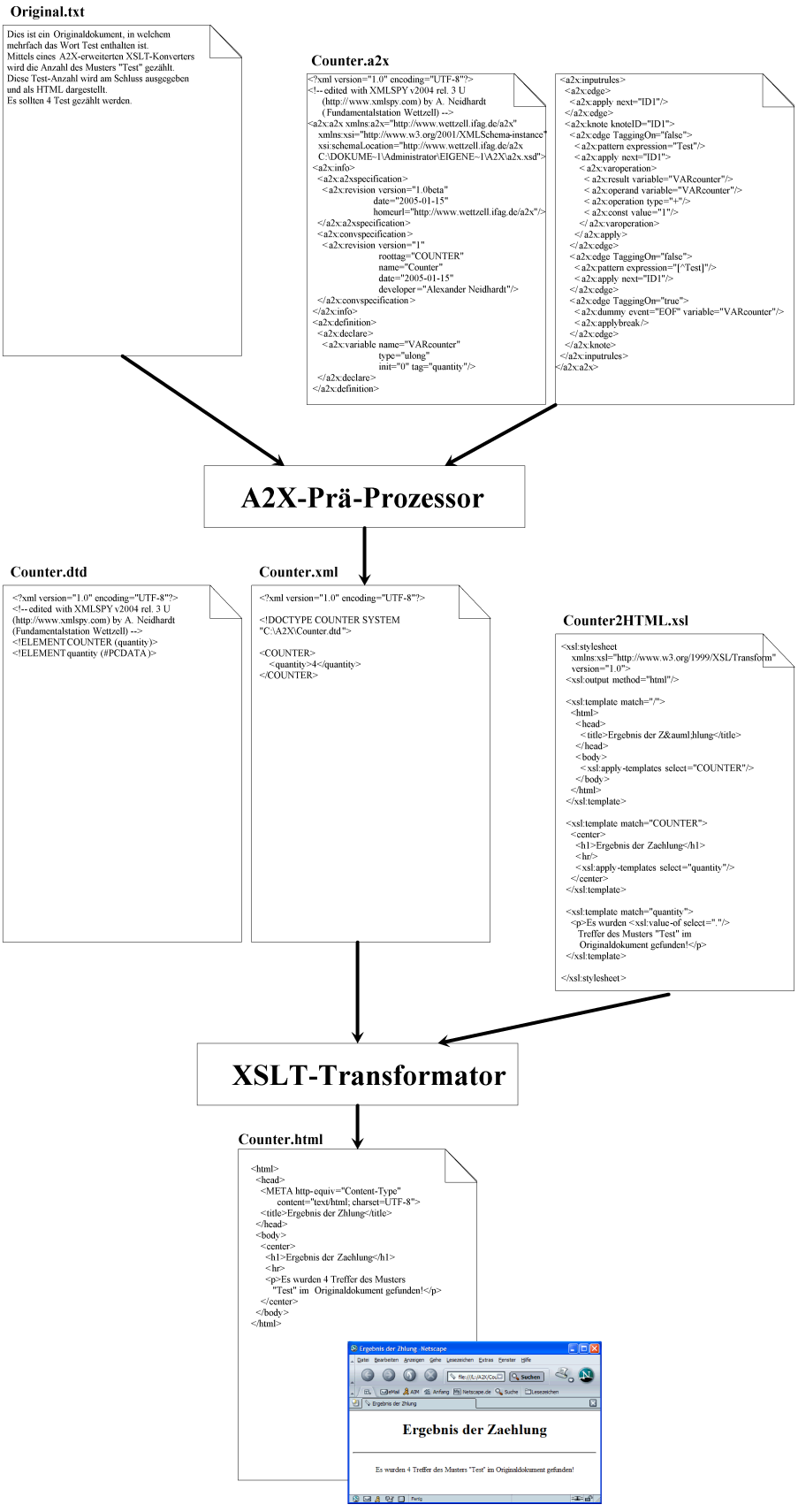


Abbildung 5.7: A2X-Anwendungsbeispiel: Realisierung eines Zählers für das Suchmuster „Test“

Die statischen Knotenelemente

Das Element zur Knotenbeschreibung kann nur aus den weiteren Anteilen zur Teilgraphenbildung (<subknote>) oder einer beliebigen Anzahl von Kanten nach oben beschriebenem Muster bestehen. Die Kantenfolge wird sequentiell auf in Frage kommende Übergänge hin überprüft. Wird kein passender Übergang gefunden und existiert auch keine „Default“-Kante ohne jegliche Bedingungsangaben, so wird die Verarbeitung des damit als fehlerhaft erkannten Dokuments an dieser Stelle angehalten. Der Knoten selbst wird in den Attributen über eine Kurzbezeichnung identifiziert, welche als Restriktion mit den Buchstaben „ID“ beginnen muss. Darauf verweisen auch die „Apply“-Vermerke in den Kanten. Zusätzlich kann ein Knoten noch einen ausführlicheren Namen und einen umfangreicheren, eindeutigen Identifizierungsnamen tragen.

Möglichkeit zur Aufteilung einer komplexeren Aufgabe in Teilgraphen

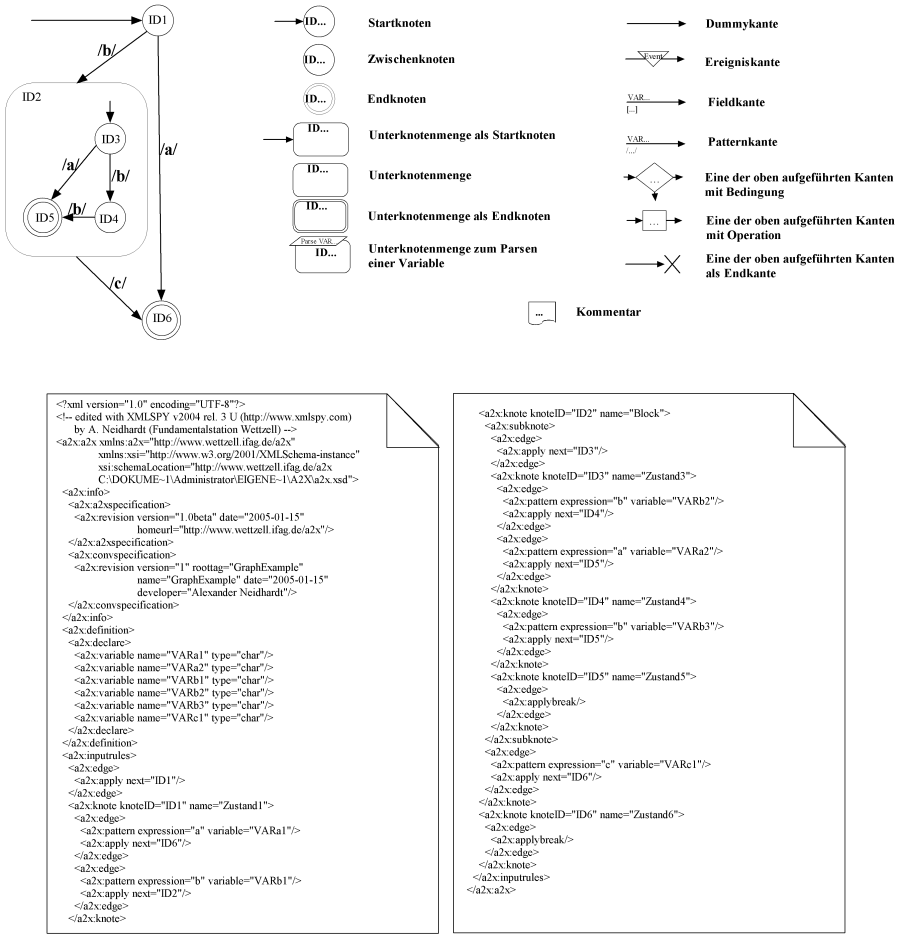
Der mögliche Teilgraph <subknote> selbst ist ein identisches Abbild der Struktur der Eingaberegeln. Er kann mit extra Auszeichnungselementen („tagblock“-Attribut) versehen werden, so dass durch ihn hierarchische Strukturen im XML -Zwischenformat entstehen. Zusätzlich kann seine Eingabe auf eine Variable umgelenkt werden („ReadFromVariable“-Attribut), so dass er einen bereits bestehenden Variableninhalt erneut entsprechend gegebener Strukturmerkmale parsen und zerlegen kann. Dadurch wird eine stufenweise verfeinerte Abarbeitung ermöglicht. Teilgraphen bilden damit eine Möglichkeit, welche vergleichbar ist zur in herkömmlichen Sprachen üblichen Zerlegung von Problemen in mehrere Prozeduren. Wird ein Subgraph über eine Kante erreicht, so wird anschließend als nächstes erst sein interner Ablauf abgearbeitet, bis ein lokaler Endknoten erreicht ist (<applybreak>-Kante). An diesem erfolgt der Rücksprung in den weiteren, externen Ablauf. Ist ein Subgraph z.B. selbst ein Endknoten, bedeutet dies, dass die Verarbeitung nach Ausführung seiner internen Struktur erlaubt enden darf.

A2X-Anwendungsbeispiel eines Zählers für ein festes Muster

Das gesamte Dokument bildet sich im einfachsten Fall aus einer Reihung von Knotendefinitionen, welche intern beliebige Verweise zueinander in Form der Kantenbeschreibungen aufweisen. Die Ablauflogik selbst steckt dann in den Kantenübergängen. So lassen sich mehr oder weniger komplexe Umsetzungen definieren. In Abb. 5.7 auf Seite 148 wird ein kompletter Ablauf einer Nutzung von A2X anhand eines Beispiels verdeutlicht, welches einen Zähler für das in einem beliebigen Ausgangstext enthaltene Muster „Test“ realisiert. Dieser Zähler erzeugt zunächst ein XML -Zwischenformat mit dem Resultat der Zählung, welches mittels XSLT -Transformation in Form einer HTML -Darstellung präsentiert werden kann. Da bisher kein Prä-Prozessor für A2X realisiert ist, handelt es sich jedoch bei dem Beispiel um einen rein theoretischen Anwendungsfall.

Flexibles Design zur Beschreibung von Strukturen und Grammatiken

Aufgrund des internen Aufbaus von A2X ist somit ein flexibles Design möglich, welches die Beschreibung von mehr oder weniger komplexen Strukturen und Grammatiken, also eine Beschreibung erlaubter syntaktischer und semantischer Zusammenhänge in einem Dokument zulässt. In Abb. 5.8 auf Seite 150 sind als weiteres Anwendungsbeispiel die „Inputrules“ angegeben, welche den Graphen eines endlichen Automaten zur regulären Grammatik für die einfache Sprache mit den Buchstabenfolgen „a“, „bbbc“ und „bac“ beschreiben, ohne dass eine Umwandlung durchgeführt wird. Ein Konverter mit solchen Eingaberegeln könnte somit nur Ausgangsdokumente lesen und validieren, welche eine der drei Buchstabenfolgen beinhalten. Bei allen weiteren würde ein Fehler gemeldet.




```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com)
     by A. Neidhardt (Fundamentalstation Wettzell) -->
<a2x:a2x xmlns:a2x="http://www.wettzell.ifag.de/a2x"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.wettzell.ifag.de/a2x
         C:\DOKUME~1\Administrator\EIGENE~1\A2X\a2x.xsd">
 <a2x:info>
  <a2:a2xspecification>
   <a2x:revision version="1.0beta" date="2005-01-15"
                 homeurl="http://www.wettzell.ifag.de/a2x"/>
  </a2:a2xspecification>
  <a2x:convspecification>
   <a2x:revision version="1" roottag="GraphExample"
                 name="GraphExample" date="2005-01-15"
                 developer="Alexander Neidhardt"/>
  </a2x:convspecification>
 </a2x:info>
 <a2x:definition>
  <a2x:declare>
   <a2x:variable name="VARa1" type="char"/>
   <a2x:variable name="VARa2" type="char"/>
   <a2x:variable name="VARb1" type="char"/>
   <a2x:variable name="VARb2" type="char"/>
   <a2x:variable name="VARb3" type="char"/>
   <a2x:variable name="VARc1" type="char"/>
  </a2x:declare>
 </a2x:definition>
 <a2x:inputrules>
  <a2x:edge>
   <a2x:apply next="ID1"/>
  </a2x:edge>
  <a2x:knote knoteID="ID1" name="Zustand1">
   <a2x:edge>
    <a2x:pattern expression="a" variable="VARa1"/>
    <a2x:apply next="ID6"/>
   </a2x:edge>
   <a2x:edge>
    <a2x:pattern expression="b" variable="VARb1"/>
    <a2x:apply next="ID2"/>
   </a2x:edge>
  </a2x:knote>
```

```
  <a2x:knote knoteID="ID2" name="Block">
   <a2x:subknote>
    <a2x:edge>
     <a2x:apply next="ID3"/>
    </a2x:edge>
    <a2x:knote knoteID="ID3" name="Zustand3">
     <a2x:edge>
      <a2x:pattern expression="b" variable="VARb2"/>
      <a2x:apply next="ID4"/>
     </a2x:edge>
     <a2x:edge>
      <a2x:pattern expression="a" variable="VARa2"/>
      <a2x:apply next="ID5"/>
     </a2x:edge>
    </a2x:knote>
    <a2x:knote knoteID="ID4" name="Zustand4">
     <a2x:edge>
      <a2x:pattern expression="b" variable="VARb3"/>
      <a2x:apply next="ID5"/>
     </a2x:edge>
    </a2x:knote>
    <a2x:knote knoteID="ID5" name="Zustand5">
     <a2x:edge>
      <a2x:applybreak/>
     </a2x:edge>
    </a2x:knote>
   </a2x:subknote>
   <a2x:edge>
    <a2x:pattern expression="c" variable="VARc1"/>
    <a2x:apply next="ID6"/>
   </a2x:edge>
  </a2x:knote>
  <a2x:knote knoteID="ID6" name="Zustand6">
   <a2x:edge>
    <a2x:applybreak/>
   </a2x:edge>
  </a2x:knote>
 </a2x:inputrules>
</a2x:a2x>
```

Abbildung 5.8: A2X-Beispiel zur Beschreibung eines endlichen Automaten

Aus dem Beispielgraphen und den darin verwendeten Symbole wird deutlich, dass ein graphisches Entwicklungswerkzeug für die Erstellung von A2X -Beschreibungen eine direkte Abbildung einer graphischen Beschreibung in eine XML -Repräsentation davon ermöglichen würde. Somit könnte die doch eher mühsame Erstellung mittels Texteditor oder die zwar komfortablere, aber dennoch teilweise aufwändige Gestaltung mittels graphikunterstützter XML -Tools für einen möglichen Nutzer umgangen werden. Dieser wäre dann in der Lage anhand der entsprechenden Graphenlogik direkt zu programmieren. Als mögliche Symbole könnten dabei bereits die in der Abb. 5.8 auf Seite 150 verwendeten Zeichen genutzt werden. Bislang wurden auf diesem Gebiet jedoch noch keine Entwicklungen durchgeführt.

Ein weiteres Beispiel einer graphischen Darstellung einer A2X -Beschreibung ist im Anhang M auf Seite 313 zu finden. Dort ist ein Anwendungsbeispiel für die Umwandlung des Normalpunkteformats, welches vom International Laser Ranging Service (ILRS) standardisiert ist, angegeben (vgl. auch 5.4.1 auf Seite 151).

A2X füllt damit die Lücke bzgl. der Umwandlung von beliebigen Formatdarstellung und Inhalten außerhalb von XML in eine Darstellung mittels der standardisierten und umfangreichen XML -Sprachenfamilie und bleibt dabei innerhalb der XML -Sprachenfamilie.

5.4 Ergebnisse

Bisher keine Realisierung von erweiterten Parsern

Da der Umfang einer Erweiterung des Prototyps um eine Implementierung eines A2X -Parsers den zeitlichen Rahmen dieser Arbeit überzogen hätte, wurde nur die vorgenannte Definition der neuen Sprache vorgenommen. Allerdings wurden mit den „Open Source“-Produkten von Apache (SAX - und DOM -Parser) erste Versuche unternommen, welche generell gezeigt haben, dass die geplante Integration des neuen Verarbeitungsablaufs für Nicht-XML -Dokumente möglich ist.

Bisher keine äquivalenten Beschreibungen existierender Formate in XML

Diese Arbeit bleibt es auch schuldig, für existierende Formate äquivalente Formatdefinitionen in XML zu liefern, weil dies von verschiedenen Seiten bereits betrieben wird. Redundante Entwicklungen sollen so vermieden werden. Damit bleibt A2X vorerst eine rein theoretische Überlegung, welche als einer der Vorschläge aus der Projektstudie hervorgeht.

Allerdings wurde eine Beispielbeschreibung in A2X für das Normalpunkteformat aus der Laserentfernungsmessung exemplarisch zu Demonstrationszwecken durchgeführt.

5.4.1 A2X-Beschreibung des Normalpunkteformat

Die Messergebnisse der Laserentfernungsmessung werden nach der lokalen Auswertung in Wettzell als Normalpunkte (statistisch charakterisierende Auswertungspunkte einer Satellitenpassage) in einer Datei abgespeichert. Diese Datei ist in einem standardisierten Format zu erstellen, welches vom ILRS vorgegeben ist. Dabei handelt es sich um eine Art ASCII -Tabelle mit festgelegten Spaltenbreiten. Diese lässt sich ideal auch mittels A2X abbilden und beschreiben, da es sich um eine Blockstruktur handelt, welche aus Textspalten mit jeweils festen Längen aufgebaut ist. Exemplarisch wurde deshalb in Ansätzen eine Beispielbeschreibung erstellt, welche im Prinzip zur Validierung des standardisierten Normalpunkteformats dienen könnte und zur Konvertierung nach XML eingesetzt werden kann (vgl. dazu Anhang M auf Seite 313).

Jedes Datum der Tabelle wird dabei gelesen und in das definierte Tagging eingegliedert. Beim Lesen werden Typüberprüfungen ausgeführt. Die syntaktische und semantische Überprüfung ist jedoch nur eingeschränkt möglich, da durch das blockweise Abbilden mittels „field“-Übergängen keine strenge lexikalische Überprüfung ausgeführt werden kann. Erst die Zuordnung zu einer Variablen erlaubt zumindest in gewisser Weise eine Typüberprüfung des zugeordneten Textfeldes. Fehler in der Spaltenanzahl lassen sich durch Abprüfen des Zeilenendes erkennen.

Bei dem Beispiel handelt es sich um eine rudimentäre, theoretische Umsetzung. Reale Anwendungen werden sich erst ergeben, wenn nutzbare Transformatoren für den Einsatz von A2X vorliegen. Die Vorteile einer solchen Einle-

semöglichkeit von existierenden Formaten und damit die Nutzbarkeit innerhalb der XML -Welt sind jedoch vielseitig. Als Beispiel könnte direkt aus einer Datei im ILRS -Format mittels eines A2X -erweiterten XSLT -Transformators, der im Anhang gegebenen A2X -Beschreibung und einer entsprechenden, zusätzlichen XSLT -Transformationsbeschreibung eine graphische Ausgabe (z.B. als Scalable Vector Graphics (SVG) -Datei[40]) erzeugt werden, welche die Normalpunkte in ein Koordinatensystem plottet.

5.5 Zusammenfassung

A2X als wesentlich allgemeinerer Ansatz als individuelle Metadatenumsetzungen in XML

Aus dem Ansatz der Entwicklung einer neuen Konvertierungssprache wird ersichtlich, dass hier ein wesentlich allgemeinerer Weg eingeschlagen wird, als nur feste Zusatzinformationen in Form von Metadaten in neuen, individuellen XML -Formaten zu definieren. Die Vielzahl der verschiedenen Anwendungen und die etablierten Formate würden dies nur erschweren.

Vorteile von A2X: Standardisierte Kopplung von individuellen Datenrepräsentationen

Durch die Möglichkeit einer Beschreibungssprache wird es jedoch möglich, dass jeder Entwickler und jeder Betreiber eines Messsystems seine eigenen Metainformationen definieren und veröffentlichen kann. Diese werden mit den Daten in Form eines A2X -Stylsheets mitgeliefert und enthalten beschreibende Kommentare, so dass Entwicklungsfremde schnell Einblick in die Formatstrukturen und semantischen Bedeutungen erhalten können. Die gemeinsame Schnittmenge der Informationsinhalte wird dabei durch die offiziellen Standardisierungen und Vorgaben der weltweiten, geodätischen Dienste definiert, welche bereits jetzt für die aktuellen Formate zuständig sind und in welche die Messsysteme eingegliedert sind. Weitere Zusatzinformationen können individuell gestaltet und mittels A2X auf definierte Weise beschrieben werden.

Graphische, abstrakte Nutzungsweise

Die Nutzung von XML und insbesondere auch von A2X fördert eine graphische, mehr abstrakte Umgangsweise mit Strukturen, Daten und Metadaten. Konvertierungsmechanismen zwischen Formaten beliebiger Art (z.B von beliebigen Ausgangsformaten in beliebige Folgedarstellungen) sind durch allgemein standardisierte Transformatoren einer Familie möglich. Aber auch das direkte Nutzen fremder Daten wird erleichtert, da die Formatbeschreibung im Stylesheet enthalten ist und allgemeine Parser angeboten werden können.

A2X steigert die Transparenz der Datennutzung

A2X selbst ist eine Art Präprozessorsprache, welche vor der eigentlichen Konvertierung nötige Vereinheitlichungen vornimmt. Trotzdem handelt es sich um eine komplette Programmiersprache, mit der graphenbasiert auch komplexere Probleme gelöst werden können. Sie ist die textuelle Vorstufe einer graphischen Umgebung zur Erstellung von allgemeinen Konvertern. Somit wurde das Kriterium zur Transparenz der Daten (vgl. Abschnitt 2.4 auf Seite 31) sehr gut erfüllt.

Zahlreiche praktische Anwendungen

Praktische Anwendungen von A2X müssen sich nicht nur auf die reine Konvertierung beschränken. Auch zur Dokumentation und Definition von Formaten kann eine Beschreibungssprache, wie A2X , eingesetzt werden. Bislang werden Formatvorgaben anhand von Textdokumentationen erläutert. Mit Hilfe von A2X ist es möglich, in einer formalen Art und Weise eindeutig syntaktische und seman-

[40] vgl. dazu [TID02] a.a.O. S. 34 f.

tische Regeln für ein Format zu definieren, welche unmittelbar von einem Parser verstanden werden können. Der Anwender bleibt dabei von der Beschreibung bis hin zur Umwandlung in einem Sprachraum, nämlich dem von XML , so dass keine zusätzlichen Sprachkonstrukte und Fremdprogramme mehr nötig sind.

Einsatz in Verbindung mit ECFT

Eine mehr oder weniger automatische Anwendung mit Formatvalidierung könnte dabei im Zusammenhang mit ECFT so aussehen, dass innerhalb einer bestimmten Datenumgebung (z.B. ein Verzeichnis) eine A2X -Vorgabe erfüllt werden muss. Ein Server z.B. schreibt dann die erhaltenen Dateien nicht mehr direkt auf die Festplatte, sondern lässt sie erst über einen mit A2X erweiterten Parser für XML laufen. So kann sogar blockweise, quasi während des Schreibvorgangs („On-the-fly“) eine automatische Validierung der Daten vorgenommen werden (vgl. dazu auch die Beschreibungen des internen Aufbaus eines Wettzell Data Access Point im Abschnitt 6.3.2 auf Seite 163). Dies kann als eine Art einfache Firewall für beliebige semi-strukturierte und strukturierte, vordefinierte Datenformate angesehen werden.

Alternative zu den existierenden Datenkonvertierungen

Bereits diese kurze Zusammenfassung zeigt, dass die Einsatzgebiete von A2X sehr weit gefächert sind. Ob nun A2X als Vorschlag einer Beschreibungssprache für allgemeine Daten als Sprache der Wahl anzusehen ist oder nicht, die XML -Gemeinschaft wird über kurz oder lang nicht um eine komfortable, intuitiv verständliche und einheitliche Möglichkeit zur Angliederung fremder Dokumente in beliebigen Formaten an XML herumkommen. A2X ist hierzu eine erste, neuartige Alternative zu den existierenden Datenkonvertierungen in die XML -Welt. Letztendlich werden jedoch erst Anwendungen der Sprache A2X ihre Vorzüge und ihre Mängel greifbar und nachvollziehbar aufzeigen.

Kapitel 6

Die Verbesserung des Datenzugangs

Schwerpunkt des Kapitels:

Das nachfolgende Kapitel beschäftigt sich mit der Verallgemeinerung des Zugangs zum neuen System und der Integration bestehender Strukturen. Es wird ein Konzept eines Datenservers vorgestellt, welcher auf die in den vorherigen Kapiteln beschriebenen Kernfunktionalitäten aufbaut und verschiedenartige Zugangsmöglichkeiten (z.B. `FTP` , `HTTP` , etc.) bietet. Deshalb wird aus der Schnittstelle eine dynamische Bibliothek erzeugt (vorerst nur eine `Dynamic Link Library (DLL)` für Windows), welche als Dateizugangspunkt in beliebige Anwendungen integriert werden kann. Als weitere Hauptelemente des Systems werden `WebDAV` und der freie `HTTP` -Server von Apache angedacht. Dadurch wird die Veröffentlichung von Inhalten im `World Wide Web (WWW)` vereinfacht. Teile der nötigen Abänderungen im Apache-Quellcode werden aufgezeigt, womit die funktionelle Vorgehensweise bestätigt ist.

6.1 Ausgangsüberlegung

Die Systemstrukturierung von Zugangspunkten und Verarbeitungsbeziehungen

Die im Abschnitt 2.4 auf Seite 34 zu Beginn der Arbeit erkannte Dreiteilung zum Vorgehen bei der Abstraktion für ein Datenmanagementsystem kann an dieser Stelle nun durch Lösungsvorschläge für die dritte Säule ergänzt werden. Bisher wurde bereits die technische Heterogenität durch eine geeignete Middleware homogenisiert (I.). Im vorhergehenden Abschnitt kam dann der Einsatz einer Beschreibungssprache zum Tragen, welcher ähnliches auf Ebene der Daten bewirkt. Somit sind in den nachfolgenden Kapiteln zur Systemstrukturierung Datenzugangspunkte und ihre Verarbeitungsbeziehungen zueinander zu betrachten. Ein erster Einstieg dazu ist die Schaffung eines funktionalen Datenknotens mit einer breiten Zugangsmöglichkeit für unterschiedliche Nutzerbedürfnisse.

Ausgangspunkte

Auf den vorliegenden Fall angewandt sind als Ausgangspunkt dafür folgende Überlegungen zu nennen:

1. Das bisher existierende Pendant zu FTP soll zu einem echten Netzwerkdateisystem ausgeweitet werden, so dass Fremdanwender Verzeichnisse direkt einbinden („mounten“) können. Sie sind damit in der Lage, im Stil von lokal vorliegenden Dateibäumen auf Dateninhalte zuzugreifen.

2. Das spezielle IIOP verursacht zahlreiche Reibungsstellen im herkömmlichen Netzverbund (z.B. Firewall-Problematik), was gerade die allgemeine Nutzbarkeit angeht.

3. Bisherige Systeme und Nutzer sollen im Sinne eines „Smart Updates“ (ohne dass eine Veränderung ersichtlich wäre) angeschlossen werden, was Schnittstellen der herkömmlichen Verfahren zum neuen System erfordert.

4. Es soll eine selbsterklärende Verwaltung möglich werden, welche auch ohne zusätzliche Komponenten Auskunft über Zusatzinformationen liefert. Eine Standardabfrage aus dem WWW sollte in der Lage sein, Metainformationen zumindest auf niedrigster Ebene (z.B. Lesen von Textdateien) zu erhalten. Von dieser Stufe an können zusätzliche, komplexere Informationssysteme ergänzt werden.

Programmbibliothek zur komfortablen Integration von ECFT in Fremdanwendungen

Es liegt auf der Hand, als inneren Kern das flexible und selbststabile ECFT zu nutzen, es jedoch nach außen durch zahlreiche weitere und herkömmlich nutzbare Zugangsmechanismen zu ergänzen. Dazu ist es nötig, verschiedene Datenzugangsserver ECFT -fähig zu machen, weshalb sie um die entsprechende Schnittstelle erweitert werden müssen. Die Server fungieren damit als eine Art Proxy für herkömmliche Zugangsarten auf ECFT . Damit bietet sich die Erstellung einer Bibliothek mit der entsprechenden ECFT -Schnittstelle an, welche komfortabel in Fremdanwendungen integriert werden kann.

WebDAV zur Überwindung von Firewallgrenzen mittels HTTP

Zur Verbesserung der Firewall-Problematik bietet es sich zudem an, Protokolle oder Erweiterungen einzusetzen, welche auf HTTP basieren, da diese bisher noch ohne größere Schwierigkeiten von den Firewalls gehandhabt werden. Da es für Web-based Distributed Authoring and Versioning (WebDAV)[1] zahlreiche Möglichkeiten gibt, sowohl per Kommandozeilenanwendung als auch in Form eines direkten Mountings Netzressourcen einzubinden, bietet sich diese Erweiterung zu HTTP

[1] vgl. dazu [STE04]

an. Eine weitere Chance besteht darin, dass die zu ECFT ähnliche Vorgehensweisen (z.B. Sperrmechanismen, Dokumenteigenschaften, etc.) evtl. auf ECFT abgebildet werden können. Dies erfordert aber einen weiteren Eingriff in eine existierende Fremdimplementierung der WebDAV -Spezifikation.

Datenmanagementserver für Dateien

In erster Näherung bietet es sich damit an, einen geeigneten Fremdserver (z.B. den HTTP -Dämonen von Apache) anzupassen, so dass in erster Linie die Funktionalität für ein Netzdateisystem entsteht, welches auch über Standard-HTTP genutzt werden kann. Der nachfolgende Schritt gliedert sich damit in die Erstellung einer Bibliotheksdatei, ersten Tests zur Integration in den Apache-Server und einer geeigneten Definition einer elementaren Verwaltungsumgebung auf niedrigster Stufe. Zusammengefasst entsteht damit ein Datenmanagementserver für Dateien, welcher als Wettzell Data Access Point (WDAP) bezeichnet wird. Bezogen auf die Transparenzkriterien, werden hier in abgewandelter Weise die Transparenz bzgl. des Zugriffs (jeder Nutzer kann seine gewohnte Art des Zugriffs beibehalten) und in Teilen der Technologie (verschiedene Applikationstechnologien werden gebündelt) berücksichtigt.

Theoretische Überlegungen nur teilweise realisiert

Es sollte an dieser Stelle noch erwähnt werden, dass von diesem Kapitel beginnend hauptsächlich theoretische Überlegungen dargestellt werden, welche nur in Teilen experimentel umgesetzt und praktisch nachgeprüft sind. Da der Umbau erhebliche Auswirkungen auf weite Teile der bisherigen Informationslandschaft in der Fundamentalstation Wettzell hat, wird das Konzept erst ausgearbeitet. Damit kann es für die Station als ein Vorschlag für zukünftige Entwicklungen und Strukturierungen angesehen werden. Ähnliche Strukturen lassen sich auch auf die Netzwerke geodätischer Verbünde ausweiten. Im Bereich der allgemeinen Informationstechnologie stellen sie eine mögliche Lösungsweise für einen Datenverbund dar.

6.2 Theoretische Grundlagen

Planung eines realen Dateisevers

Eine logisch folgende Weiterentwicklung zum bisherigen Entwicklungsstand ist der Ausbau zu einem kompletten Server. Darunter ist zu verstehen, dass ein realer Rechner geplant wird, welcher das Dateisystem für die Speicherung und Verwaltung von Daten trägt und über verschiedene Schnittstellen angesprochen werden kann.

Zusätzliche Administrationsschnittstelle zu einem ECFT-Koordinator

Dafür wird im Hintergrund hauptsächlich das entwickelte Übertragungssystem ECFT eingesetzt. Es dient damit quasi als zentraler Koordinator. Um jedoch an dieser Stelle die Situation eines „Bottlenecks“ (Flaschenhals an dem sich die Bearbeitung gegenseitig behindert) zu vermeiden und einen Mehrfachdurchlauf des Protokollstacks beim Datenzugang zu verhindern, wird zur bisher beschriebenen Schnittstelle eine zusätzliche Erweiterung für den administrativen Zugang ermöglicht. Dieser fungiert wie ein Dummy und prüft jede Anfrage auf Korrektheit und liefert entsprechend benötigte Versionsinformationen (z.B. Dateinamen). Er führt die eigentlichen Datenzugriffe nicht selbst aus, sondern überlässt es den Fremdanwendungen, wie z.B. einem HTTP - oder FTP -Server, die Dateien zu manipulieren. Schließen sie ihre Arbeit ab, muss das wiederum an den ECFT -Koordinator gemeldet werden, da dieser sonst die Version nicht übernimmt.

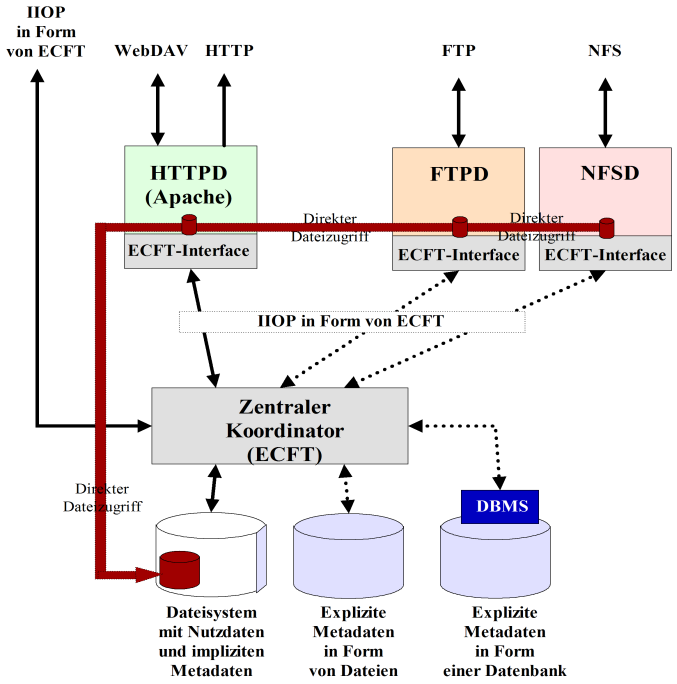


Abbildung 6.1: Idee eines zentralen Koordinators

Zusatzinformationen auf unterschiedliche Art und Weise eingliederbar

Notwendige Zusatzinformationen (Metainformationen für das System) können dabei direkt über Betriebssystemeinträge zum Dateisystem, über Zusatzdateien zu den einzelnen Nutzdaten oder mit Hilfe eines DBMS verwaltet und strukturiert werden. Letzteres ist damit nicht zwingend notwendig, kann aber die Suche und damit das Navigieren durch eine große Datenbasis wesentlich erleichtern. Zudem werden aufwändige Suchalgorithmen auf hochgradig effektive Implementierungen im DBMS abgewälzt. Als Grundlage für die Metainformationen sind existierende Standards zu nutzen, welche auf XML basieren bzw. umgesetzt werden können. Zu nennen sind hier z.B. die Spezifikation von OpenGIS(tm) Metadaten (ISO/TC 211 DIS 19115) für GIS [2] des Open GIS Consortium oder für allgemeinere, elementarere Aufgaben das Dublin Core Metadata Element Set[3] nebst weiteren Spezifikationen des Dublin Core Metadata Initiatie (DCMI) (ein Standard der National Information Standards Organisation (NISO) (ANSI)), welche in dieser Arbeit nicht näher ausgeführt werden.

Einbringen von Gütemerkmalen in Fremdanwendungen

Dieses Koordinatorprinzip (vgl. dazu Abb. 6.1 auf Seite 158) vereint die Vorteile eines verteilten Zugriffs über heterogene Mechanismen mit der homogenen Verwaltung eines zentralistischen Servers. Alle übergeordneten Kontroll- und Verwaltungsarbeiten zur Qualitätssicherung werden dabei vom ECFT -Koordinator übernommen. Damit können alle bisher erbrachten Gütemerkmale auch in Fremdanwendungen einfließen.

Kombination von Fremdanwendungen auf Koordination des Dateizugriffs beschränkt

Nachteil ist, dass nur die allen Fremdanwendungen gemeinsame Funktionalität, d.h. der Zugriff auf das Dateisystem, homogen genutzt werden kann. In den Anwendungen zusätzlich enthaltene Umsetzungen (z.B. Metadaten etc.) sind nur dort nutzbar und sollten deshalb vermieden werden, bzw. müssen aufwändig auf

[2] vgl. dazu [OGIS04]
[3] vgl. dazu [ANSI01]

ECFT abgebildet werden. Deshalb muss auch dort eine elementare Verwaltungsmöglichkeit von Zusatzinformationen geschaffen werde.

Erste Zielumgebung ist der HTTP-Server von Apache

Wie erwähnt bietet sich der Einsatz von HTTP und WebDAV an, um ein webfähiges Netzwerkdateisystem zu schaffen. Eine geeignete, freie Implementierung eines HTTP -Servers mit integriertem WebDAV ist von Apache realisiert. Seine weite Verbreitung und stabile Arbeitsweise unterstützt die Entscheidung. Deshalb werden erste Umsetzungsversuch für den Server von Apache unternommen.

Vorerst soll aber noch ein Einblick in WebDAV , das Web-based Distributed Authoring and Versioning, gegeben werden.

6.2.1 Die Nutzung von WebDAV

Ähnliche Prinzipien und Methoden der Umsetzung wie bei ECFT

Bei der Betrachtung von WebDAV fällt auf, dass dort ähnliche Prinzipien und Methoden umgesetzt sind, wie sie auch bei ECFT existieren[4]. Da es auf HTTP basiert, dient es auf effiziente Weise dem Verknüpfen von heterogenen Datensystemen. HTTP selbst basiert auf einer Client/Server-Architektur und stellt einen verlässlichen, verbindungsorientierten, jedoch ohne Sitzungsverwaltung agierenden Transportdienst als ein Request/Response-Protokoll dar. Der Client öffnet die Verbindung und stellt seine Anfrage (Request), welche der Server bedient (Response), worauf der Client die Verbindung meist wieder abbaut.

Entwicklungsgeschichte des WebDAV-Standards

Die Entwicklung für WebDAV selbst begann 1996 mit einem Zusammenschluss verschiedener Firmen und Einrichtungen (u.a. Microsoft, Netscape, W3C , etc.) zur WebDAV Working Group. Da es dem Internet als enormes Dateisystem noch an Mechanismen zur gemeinsamen Arbeit an Dokumenten fehlte, wurden im Rahmen der Entwicklungen zu WebDAV Methoden zu HTTP ergänzt. Die so entstandenen Funktionen zum verteilten Schreiben, Verändern und Verwalten von Daten- und Dokumentstrukturen und deren Inhalten fördern die teamorientierte Handhabung von verteilten Daten. WebDAV wird weiter dadurch begünstigt, dass aufgrund der Kombination mit HTTP reine HTTP -Clients weiterhin Daten abrufen können, während für erweiterte Klienten der volle Funktionsumfang nutzbar ist. Zudem ist die Überwindung von Firewalls ohne zusätzliche Portöffnungen möglich. Mittlerweile ist WebDAV ein IETF -Standard (RFC 2518).

Erweiterung von HTTP

Der WebDAV -Standard erweitert HTTP um folgende, neue Konzepte:

1. **Sperrmechanismen zum Überschreibschutz:** Der umgesetzte Locking- Mechanismus mit persistenten Sperren dient zur Verhinderung von „Lost Updates“. Sperren wirken nur auf Schreibzugriffe, da Lesen immer erlaubt bleibt. Es gibt „Shared Locks“ (bei einer Sperre wird ein Nutzer darauf aufmerksam gemacht und kann entscheiden, ob er ebenfalls schreibend zugreifen will) und „Exclusive Locks“ (nur genau ein Nutzer hat exklusives Zugriffsrecht). Die Verwaltung funktioniert mittels sog., eindeutiger „Locktoken“.

   Vergleich zu ECFT

   Im Vergleich dazu nutzt ECFT teilpersistente Sperren, welche nach einem Fehler nur bis zu einer bestimmten Zeit weiterexistieren, um ungewollte Dauersperren zu verhindern. ECFT -Sperren sind immer exklusiv und wirken auf alle Zugriffsarten. Die Identifiaktion erfolgt über die Registrationsinformation des jeweiligen Nutzers.

[4] vgl. zum nachfolgenden Abschnitt [KOW03]

2. **Namensraumverwaltung:** Es ist möglich, sich eigene, vom Dateisystem unabhängige, virtuelle Dateiverbünde zu schaffen, welche als Namensraum bezeichnet werden.

   Vergleich zu ECFT

   Zum Vergleich gibt es in ECFT nur die physikalischen Namensräume der Verzeichnisstrukturen.

3. **Metadaten:** Die Metainformationen werden als Eigenschaften, sog. Properties, einem Dokument zugeordnet. Sie bestehen aus einem Bezeichner und einem Wert. Letzterer kann aus einem beliebigen XML -Ausdruck aufgebaut sein. Es gibt einige vordefinierte, allgemeine Eigenschaften (z.B. die in der Spezifikation von HTTP festgelegten Angaben zu Content-Type oder Content-Language[5]).

   Vergleich zu ECFT

   ECFT verfügt bislang nur über die vom Dateisystem gegebenen Informationen zur Datei.

4. **DeltaV - die Versionskontrolle:** Seit 2002 gibt es weitere Methoden, welche eine Versionskontrolle erlauben, welche in der Funktionalität vergleichbar zu CVS ist. Allerdings gibt es erst wenige Implementierungen, welche nur teilweise Open-Source sind[6].

   Vergleich zu ECFT

   Im Vergleich dazu verfügt ECFT über eine einfache Versionsverwaltung, indem frühere Versionen entsprechend ihres Gültigkeitsdatums abgelegt werden können.

5. **Weitere Planungen:** In Planung befinden sich die Unterstützung zur Suche von Ressourcen und zur Zugriffskontrolle.

   Vergleich zu ECFT

   Im Vergleich dazu findet eine generelle Ressourcensuche in ECFT über den CORBA Naming Service statt. Die Zugriffskontrolle ist auf das Betriebssystem verlagert.

Zahlreiche freie und lizenzierte Anwendungen für WebDAV

Mittlerweile gibt es zahlreiche Client- und Serveranwendungen für WebDAV . Zu erwähnen sind hier die freien Softwareprojekte „cadaver“ (ein Kommandozeilenclient) und „DAVfs“ (ein Client zum Mounten von WebDAV). Beides ist für Linux geschrieben. Unter Windows verfügt Windows 2000 bereits mit Hilfe des Internet Explorers ab Version 5 über die Möglichkeit, zu Web-Ordnern zu verbinden. Diese können aber nur über den Datei-Explorer genutzt werden. Eine weitere kommerzielle Software ist „WebDrive“ von South River Technologies, welche die Web-Ordner als eigene Laufwerke einbindet und sie damit in unter Windows gewohnter Weise nutzbar macht.

Nachfolgende Entwicklungen und Überlegungen nutzen nur einen geringen Teil der Fähigkeiten von WebDAV . Sie dienen hauptsächlich dazu, das bisher entwickelte System an bestehende Gegebenheiten anzubinden und den Nutzern gewohnten Komfort zu bieten.

[5] vgl. dazu [NWG04]

[6] vgl. dazu [MOE04] a.a.O. Folie 44

6.3 Umsetzungsdesign: Wettzell Data Access Point (WDAP)

Repräsentation von Datenlieferanten durch einen Datenzugangspunkt

Die Schaffung eines einzelnen Datenservers in Form eines Datenzugangspunktes ist ein weiterer, wichtiger Schritt, um später verschiedene Datenlieferanten (z.B. die Messsysteme) zu einer Einheit zu verknüpfen. Jeder Lieferant kann so durch einen einzelnen Datenzugangspunkt repräsentiert werden, welcher zahlreiche Möglichkeiten bietet, an die angebotenen Daten zu gelangen. Jeder dieser Knoten erhält für sich einen stabilen Zustand und hat eine intuitiv verständliche innere Strukturierung.

6.3.1 Festlegungen

Zwiebelringstruktur

Die Planungen und Umsetzungen orientieren sich am Aufbau in Form von Ringen („Zwiebelringstruktur“). Alle Zugangsdienste setzen auf `ECFT` auf, welcher als zentraler Koordinator fungiert. Somit wird im Allgemeinen die Grundfunktionalität unterstützt. Alle spezifischen Fähigkeiten der folgenden Zugangsschichten stehen nur dort zur Verfügung (vgl. dazu auch Abb. 6.2 auf Seite 161). So kann z.B. von einem angepassten `FTP` -Server und von einem erweiterten `WebDAV` -Server gemeinsam auf die von `ECFT` verwalteten Dateien zugegriffen werden. Der `FTP` -Server hat jedoch keine Möglichkeit, die in `WebDAV` verwalteten Properties zu nutzen, da beide Erweiterungen auf unterschiedlichen Ästen (Branches) des Datenzugangspunkts sind. Sie sitzen auf gleichem Niveau in der Ringstruktur. Deshalb sollten auch nur die von `ECFT` vorgesehenen Mechanismen genutzt werden, auf die evtl. höherwertige Dienste abgebildet werden können.

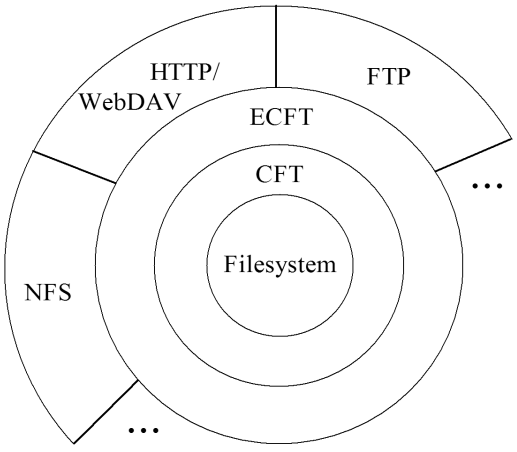


Abbildung 6.2: Die Zwiebelringstruktur

Nachteil durch Einschränkung, Vorteil durch Kombinierbarkeit

Dieser Aufbau hat den Nachteil, dass sich alle Anwendungen gleicher Ebene auf die Gegebenheiten niedrigerer Schichten beschränken müssen. Ein erheblicher Vorteil jeoch ist, dass dadurch verschiedene Dienste miteinander gekoppelt werden können. `ECFT` fungiert somit als eine Art Middleware für höhere Dienste.

Drei Klassen von Metadaten

Ein wesentlicher Punkt hierbei ist, dass zu den herkömmlichen Daten zusätzliche Metadaten eingebracht werden können, welche die Verwaltung unterstützen. Die Metadaten werden in drei Klassen unterteilt:

1. Metadaten des Systems („system“): Sie bestehen aus dem durch ECFT vereinheitlichten Textfeld der Dateiattribute des Betriebssystems.
2. Metadaten zur Ablageverwaltung („storage“): Sie erleichtern die Verwaltung durch Vorgabe von definierten Regeln und weiterführenden Verwaltungsinformationen (z.B. Name des Erstveröffentlichers, Verantwortliche Stelle, Validierungsregeln, etc.) zu den eigentlichen Daten.
3. Metadaten zum Inhalt („content“): Sie sind individuell von den Datenerzeugern zu definieren und dienen zur genaueren Beschreibung inhaltlicher Elemente.

Metainformationen vom Betriebssystem

Systemmetadaten können nicht beeinflusst werden und müssen als Gegebenheit hingenommen werden. Allerdings wird die über die Betriebssysteme hinweg gebildete Schnittmenge über die systemgegebenen Zusatzinformationen verwendet, um eine Vereinheitlichung zu erzielen. Für Verzeichnisse und Dateien sind dies:

- Rechteangaben im UNIX-Stil
  - Art der Ressource: 1 Zeichen mit „d“ = Directory, „a“ = Archive, „l“ = Link, „-“ = normale Datei (beliebig erweiterbar)
  - Rechte des Nutzers: 3 Zeichen jeweils mit „r“ = Read, „w“ = Write, „x“ = Execute; „-“ = Recht nicht wahrgenommen
  - Rechte der Gruppe des Nutzers: 3 Zeichen jeweils mit „r“ = Read, „w“ = Write, „x“ = Execute; „-“ = Recht nicht wahrgenommen
  - Rechte für alle restlichen Nutzer: 3 Zeichen jeweils mit „r“ = Read, „w“ = Write, „x“ = Execute; „-“ = Recht nicht wahrgenommen
- Angabe des Nutzernamens des Erstellers
- Angabe des Gruppennamens des Erstellers
- Angabe der Dateigröße in Bytes
- Angaben zum Erstellungsdatum im UNIX-Stil
- Angabe des Dateinamens

XML-Metainformationen

Alle weiteren Metainformationen sind in XML abgefasst und können während der Verarbeitung gesetzt werden. Die Zusatzinformationen zum Inhalt sind beliebig vom Anwender zu wählen und brauchen deshalb an dieser Stelle nur kurz betrachtet werden. Zu ihnen zählt in erster Linie das Tagging, d.h., das Klammern von realer Information in bezeichnende Umklammerungen, wie es unter XML nötig ist und u.a. mittels A2X erzeugt werden kann. Diese mehr implizite Form von Metainformation durch logische Strukturierung kann individuell durch weitere Informationen je Einsatzgebiet erweitert werden.

Beschränkung auf Ablageverwaltung

Nachfolgendes Kapitel beschäftigt sich hauptsächlich mit den Zusatzinformationen zur Ablageverwaltung. Diese werden möglichst knapp gehalten, da sich in realen Systemen immer wieder zeigt, dass ein großer Umfang an auszufüllenden Zusatzdaten vom Anwender nicht gerne genutzt wird. Es gilt damit der Grundsatz: so viel Verwaltungsoverhead wie nötig, jedoch so wenig wie möglich. Dies zeigt sich auch in der nachfolgenden Strukturierung.

6.3.2 Planung der inneren Struktur

Strukturierung des Dateisystems

Die innere Struktur orientiert sich am Aufbau von Dateisystemen. Diese sind in Verzeichnisse oder Ordner unterteilt und beinhalten die Dokumente und Daten als Dateien. Die Verwaltung dieser Dateien und ihrer Rechte übernimmt bereits ECFT . An dieser Stelle sollen nun Vorgaben zur Strukturierung und Ordnung geschaffen werden.

Eingliederung einer Beschreibungsdatei je Ordner

Zur Klassifikation der Charakteristika eines Ordners beinhaltet jedes Verzeichnis eine ECFT -spezifische Beschreibungsdatei in XML . Dort wird in erster Instanz festgelegt, welche Dateitypen in einem Verzeichnis aufgenommen werden können und wie sie zu behandeln sind. Die Beschreibung gilt nur für den aktuellen Ordner, so dass jeder Unterordner ebenfalls mit einer eigenen Beschreibungsdatei ausgestattet sein muss, welche aber auf übergeordnete verweisen kann. Existiert sie nicht, so werden die Dateien wie in einem gewöhnlichen Dateisystem verwendet (nur die Metadaten „system“ stehen zur Verfügung). Die Metadatendatei ist jeweils in zwei Blöcke aufgeteilt. Im ersten Teil befinden sich die Informationen zum „storage“ im zweiten die zum „content“.

Einteilung in homogene Sets und heterogene Container

Jeder Ordner kann entweder einen homogenen (alle Dateien sind vom selben Typ mit demselben Format) oder einen inhomogenen (unterschiedliche Dateitypen sind abgelegt) Datenbestand beinhalten. Für Ordner gibt es damit eine grundsätzliche Aufteilung in „Set“ (homogene Menge) und „Container“ (heterogene Ansammlung). In Containern können jegliche Formen von Dateien eingelagert werden, die bei der Ablage auch keiner Überprüfung auf Korrektheit zu unterziehen sind. Jede Datei steht damit für sich genommen und kann eigene Regeln definieren. Diese Form eines Ordners unterstützt damit hauptsächlich Bürodokumente, welche im Allgemeinen semi-strukturiert oder unstrukturiert sind.

Set mit restriktiven Beschreibungsvorgaben

Ein Set erlegt hingegen die meisten Restriktionen auf. Jede dort abgelegt Datei hat exakt einer vorgegebenen Beschreibung zu gehorchen. Diese wird für XML -Dateien in Form einer DTD oder einer Schemadefinition und für allgemeine Dateien in Form der Sprache A2X angegeben. Wird eine neue Datei innerhalb eines Sets angelegt oder eine bestehende verändert, so wird sie erst mittels eines Parserlaufs z.B. durch einen Transformator, evaluiert. Ist diese Überprüfung zufriedenstellend, wird die Änderung übernommen. Ansonsten wird sie zurückgewiesen.

Set als Spezialsierung der Metabeschreibungen eines Containers

Generell verfügen sowohl Sets als auch Container über die Zusatzinformationen, welche vom Dateisystem stammen (z.B. ein identifizierender Name, Nutzungsrechte, Erstellungsdatum, etc.). Bei den zusätzlichen Informationen in den Beschreibungsdateien ist die logische Struktur eines Sets eine Spezialisierung eines Containers. Ein Set erweitert die Zusatzinformationen eines Containers.

Als Vorschlag für Metadaten zu einem Ordner wird in Tabelle 6.1 auf Seite 164 gegeben.

Container	**Update**	**Restriktion**
SUBJECT: Schlagworte zum Inhalt	Manuell	Pflicht
CONTRIBUTOR: Verantwortlicher Vertreiber (Firma etc.)	Manuell	Optional
HISTORY: Protokoll/Liste der Veränderungen im Ordner bestehend aus: PUBLISHER: Herausgebender Nutzer DATE: Änderungsdatum und Zeit CHANGES: Angabe der Veränderung	Automatisch	Aktivierbar
RIGHTS: Zusätzliche Festlegungen bzgl. der Rechte, (wirken nur im Rahmen der Festlegungen im Dateisystem): DENY/ALLOW: Verbot oder Erlaubnis ACCESS: Angabe entsprechend dem Modus beim Dateizugriff (z.B. „w“ oder „a“) USER: Nutzeridentifikation des Betriebssystems oder „ALL“)	Manuell	Optional
DESCRIPTION: Erklärende Beschreibung	Manuell	Optional
Set (als Zusatz zum Container)		
TYPELINK: Verweis auf eine Formatbeschreibung in DTD , XSD oder A2X	Manuell	Pflicht
SRCLINK: Verweise auf Stylesheets zur Transformation bestehend aus einer Liste von: TRANS: Verweis auf Transformator SHEET: Zu nutzendes Stylesheet	Manuell	Optional

Tabelle 6.1: Metadaten zu einem Ordner

Erweiterbare Vorgaben von Grundelementen für Zusatzinfos

Diese Grundelemente können bei Bedarf erweitert werden und sind als eine flache XML -Struktur abgespeichert. Jeder manuell setzbare Eintrag kann über einen Parameter auf eine Metadatendatei für einen Ordner verweisen, von welcher die Einträge übernommen werden. Pflichteinträge müssen existieren. Optionale Einträge können bei Bedarf genutzt werden. Und aktivierbare Metadaten brauchen nicht genutzt werden, solange sie nicht aktiviert sind (über einen Parametereintrag zum Metadatum). Ab dann müssen sie laufend gehalten werden. Die History-Einträge enthalten sowohl Änderungen des verwalteten Inhalts, als auch Veränderungen an der Metadatendatei selbst.

Metadatendateien sind ECFT -Systemdateien

In der vorliegenden Arbeit gibt es keine weiteren Zusatzinformationen für Ordner. Werden noch weitere benötigt, so kann jeder Nutzer individuelle Einträge als „content“-Teil ergänzen. Die Dateien mit Metadaten zu Ordnern werden im Dateinamen als ECFT -Systemdateien kenntlich gemacht (z.B. „WDAPSET<Ordnername> WDAPSET.tmp“ für Sets und „WDAPCON<Ordnername>WDAPCON.tmp“ für Container) und bei einer Abfrage des Ordnerinhalts nur dem Superuser angezeigt.

Dateienspezifische Metadatenangaben

Zusätzlich zu den mehr oder weniger global wirkenden Vorgaben, kann für jede Datei egal ob im Container oder im Set noch eine weitere, dateispezifische Metadatendatei ergänzt werden. Sie ist ebenfalls als Systemdatei markiert (z.B. als „WDAPMD<Dateiname>WDAPMD.tmp“) und wird nur dem Superuser angezeigt. In ihr werden keine History-Einträge verwaltet, da sie einzigartig und immer der Datei zugeordnet bleibt, für die sie angelegt wurde. Die Versionsverwaltung der Dateien übernimmt bereits ECFT durch wahlweises Anlegen von Sicherungskopien.

Element	Beschreibung
Title	Der Name der Datei
Creator	Der Erzeuger der Datei
Subject	Schlagworte zum Inhalt
Description	Beschreibung des Inhalts
Publisher	Der Herausgeber der Datei
Contributor	Zur Erzeugung beitragende Verantwortliche
Date	Datum eines Ereignisses im Lebenszyklus der Datei
Type	Eingliederung der Datei zu einem bestimmten Genre
Format	Angaben zum genutzen Format (evtl. Verweis auf Definition mittels DTD , XSD oder A2X)
Identifier	Eine eindeutig zuordenbare Kennung
Source	Angaben zu einer Quelle, von der die vorliegende Datei abgeleitet wurde
Language	Genutzte, reale Sprache für den Inhalt
Relation	Verweise zu verwandten Dateien
Coverage	Angaben zum weiteren Umfeld der Datei
Rights	Angaben zu den Lizenz- und Nutzungsrechten

nach [ANSI01]

Tabelle 6.2: Metadaten zu einer Datei

Orientierung am Dublin Core Metadaten Elemente Sets

Die Zusatzinformationen zu den einzelnen Dateien orientieren sich an der Standardisierung des Metadaten Elemente Sets im Dublin Core[7]. Es werden dort die Einträge definiert, welche in Tabelle 6.2 auf Seite 165 beschrieben sind. Dazu gibt es von der Dublin Core Metadata Initiatie (DCMI) auch bereits vorgegebene Schemadefinitionen. In der beschriebenen Umgebung bilden sie die Metadaten zur Ablageverwaltung (storage) von Dateien. Eine Ergänzung zu den standardisierten Informationen ist ein Eintrag vergleichbar zu „SRCLINK“ in den Ordnerbeschreibungen. Jede Datei kann hier bereits über Informationen verfügen, welche Folgeprodukte daraus generiert werden können. Der Aufbau dieses Eintrags orientiert sich an dem für Ordner. Zusätzlich können die Verwaltungsinformationen wieder durch nutzerspezifische „content“-Erweiterungen ergänzt werden.

Vereinfachung der Datenverwaltung

Die so gegebenen Strukturierungen und Verwaltungseinheiten bilden den Grundstock für gezielte Anfragen. Die Zusatzangaben können zur schnelleren Suche auch in ein angegliedertes DBMS abgespeichert werden. Da sie generell im strukturierten XML -Stil vorliegen, ist eine Abbildung in eine Tabellenstruktur möglich. Allgemein ist jedoch zu beachten, dass nach jedem manuellen Eingriff, welcher nicht vom System überwacht werden kann (z.B. ein lokaler Dateizugriff), ein Update der Zusatzinformationen für die betroffenen Dateien und Ordner eingeleitet werden muss.

Virtuelle Karteikarten

Die Metainformationen selbst bieten die Grundlage für spätere Suchfunktionen und Verwaltungsaufgaben. In erster Näherung aber können sie als eine Form von Karteikarte dargestellt und eingesehen werden. Dabei können sie durch die XML -Anbindung für die Darstellung auf beliebige Weise umgeformt werden, so dass die Einsichtnahme durch jede angegliederte Anwendung stattfinden kann. Die Nutzung erfolgt mittels erweiterter CFTContext- und CFTInfo-Befehle in der ECFT -Schnittstelle.

Neue Kontextbefehle

Der Kontextbefehl zum Verändern von Einträgen lautet:

- „setmeta [<TAN>] <Datei> <spezieller Eintrag>=[<wert>](, <spezieller Eintrag>=[<wert>])*“ zum Setzen der Einträge für die Datei <Datei>. Ist kein

[7] vgl. dazu [ANSI01]

<wert> angegeben, so wird der jeweilige Metadateneintrag gelöscht. Sämtliche Metadaten können durch Reihung auf einmal gesetzt werden. Von den beschriebenen, festgelegten Zusatzinformationen abweichende Einträge, werden automatisch als „content“-Teil übernommen.

Die Abfragebefehle lauten:

- „getmeta <Datei> [<spezieller Eintrag>]“ zum Abfragen der Einträge. Wird kein spezieller Wert angegeben, so wird die gesamte Registerkarte mit Metainformationen ausgegeben.
- „ls meta <spezieller Eintrag>=[<Suchausdruck>](AND|OR [NOT] <spezieller Eintrag>=[<Suchausdruck>])*“ zum Suchen nach speziellen Eigenschaften in Form eines erweiterten „ls“-Befehls. Der Suchausdruck ist ebenfalls dem gewohnten „ls“-Befehl nachempfunden.

Die Funktionalität von ECFT selbst wird über eine Bibliothek in die Fremdamwendungen integriert.

6.3.3 Eingliederung in Fremdanwendungen

Bibliothek zur Eingliederung in Fremdanwendungen

Bezüglich der Anbindung von Fremdanwendungen zeichnet sich der in der Programmierung gewohnte Aufbau der ECFT -Schnittstelle zur Dateihandhabung aus. Bestehende Funktionen zur Dateimanipulation aus den Standardbibliotheken der Programmiersprachen können dadurch nahezu direkt ersetzt werden. Deshalb liegt es nahe, eine eigene Bibliothek zu entwickeln, welche in eine bestehende Anwendung eingebunden werden kann und die volle Funktionalität von ECFT zugänglich macht, indem herkömmliche Dateizugriffe über die neuen Bibliotheksfunktionen stattfinden.

Schnittstelle zur Administration

Erste Tests haben gezeigt, dass der mehrmalige Durchlauf durch den Protokollstack bei der Manipulation kleiner Datenmengen in den Dateien nicht praktikabel ist. Damit ist der direkte Einsatz der neuen Bibliothek nicht rentabel. Um dieses Problem zu umgehen, sind die Transfermethoden der elementaren Schnittstelle in ECFT zu erweitern, bzw. in einer abgeleiteten Methodensammlung zu überladen. Die neuen Transfermethoden können entweder wie bisher genutzt werden oder als eine Art Dummy alle administrativen Aktionen eines Aufrufs erledigen, die eigentliche Aktivität jedoch nicht selbst durchführen. Damit kann jede Fremdanwendung quasi Dateioperationen anmelden und dann selbst direkt auf den Dateien über herkömmliche Methoden agieren. Voraussetzung ist, dass der Koordinator und der Server der Fremdanwendung auf demselben physikalischen Serverknoten sitzen. Stabile Zustände werden hierbei vom Koordinator erhalten. Anfallende Unstimmigkeiten zwischen Fremdanwendungssicht und Koordinatorsicht müssen vom Server des Fremddienstes abgefangen werden.

Dummy- oder XML-Schnittstelle

Das zusätzlich benötigte Methodenattribut lautet in IDL -Schreibweise „in unsigned int uiDummy“ und erlaubt die Nutzung des Dummy-Modus. Eine komplette Schnittstelle ist im Anhang O auf Seite 339 gegeben. In einem weiteren Entwicklungsschritt kann eine zusätzliche Erweiterung dahingehend integriert werden, die

komplette Interpretation von Befehlen auf den Server zu verlagern. Dazu wird eine Methode für alle Aufgaben eingesetzt. Die Befehle können in Form von Beschreibungen in XML dargestellt werden. Die Methode verfügt zudem über einen Parameter zur Rückgabe von möglichen Antworten. Dies wird vorerst aber nicht benötigt, da es die gewohnte Handhabung im Programmierumfeld durchbricht und jede Aktion erst vorbereitet werden muss (Aufbau der Darstellung in XML).

Nutzung der neuen Bibliothek

Zur Nutzung einer vorübersetzten Bibliothek benötigt man zum einen die Headerdateien (jeweils speziell für C und C++, da bei gemischter Nutzung Kennzeichnungsregeln für den Linker notwendig sind) mit den Methodendefinitionen und zum andern die lauffähige Bibliotheksdatei (unter Windows die DLL und unter Unix/Linux eine „so“-Datei). In den Quellcodedateien, welche die neue Bibliothek nutzen wollen, muss nur die Headerdatei eingebunden werden, so dass die Funktionsköpfe validiert werden können. Beim Linken wird anschließend die Bibliothek entweder statisch oder dynamisch dazugebunden. Die Anwendung verfügt damit über einen Zugang zu ECFT .

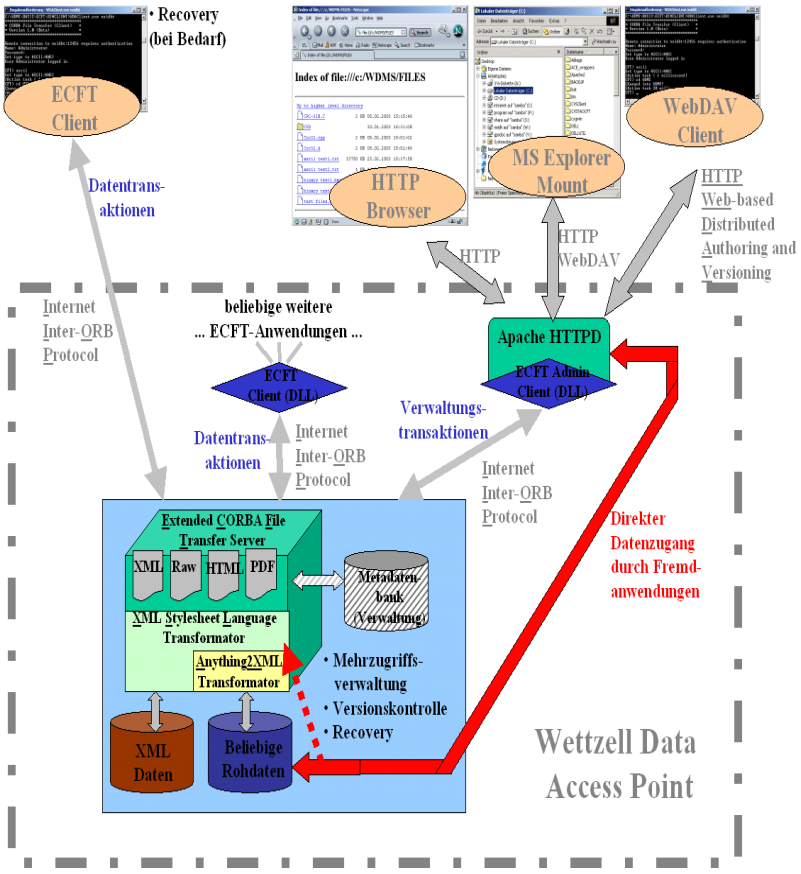


Abbildung 6.3: Der Wettzell Data Access Point (WDAP)

Vorhalten paralleler Darstellungen

Die Dateiverwaltung und Qualitätssicherung bleibt bei dieser Vorgehensweise Aufgabe von ECFT . Dieser muss dahingehend ausgebaut werden, mögliche erzeugbare Formate anzubieten und zu erzeugen. Die effektivste Möglichkeit ist das Vorhalten der unterschiedlichen Darstellungen neben der ursprünglichen Datei. Damit werden Zugriffe wesentlich beschleunigt. Es ist nur dafür zu sorgen,

Möglichkeit zur Hin- und Rücktransformation von Formatdarstellungen

dass solche abhängigen Folgedateien entsprechend neu erzeugt werden, falls das Hauptdokument geändert wird. Das Vorhalten von Folgedarstellungen kann auch dazu genutzt werden, mittels A2X Rücktransformationen in das ursprüngliche Dateiformat zu definieren. Voraussetzung ist die eindeutige Zuordenbarkeit einzelner Auszeichnungselemente in beide Transformationsrichtungen. Damit kann mit A2X ein bidirektionaler Transformationsweg realisiert werden, was in bisherigen Datensystemen ein generelles Problem darstellt.

Transformation „on demand“

Die zweite Möglichkeit besteht darin, Folgedateien „on demand“ (auf Abruf) neu zu erzeugen. Dies hat den Vorteil, dass sich die Datenspeicherung nicht unnötig ausdehnt. Allerdings werden dadurch auch erhebliche Wartezeiten verursacht, was für hauptsächlich synchron arbeitende Übertragungsmechanismen (z.B. auch durch IIOP gegeben) für erhebliche Störeinflüsse sorgen kann.

Generell sollten deshalb Konvertierungen immer als eigene Prozesse im Hintergrund ablaufen. Diese Prozesse müssen aber zumindest für die Dateiverwaltung immer auf der Schnittstelle zu ECFT aufsetzen.

Der resultierende Zugangspunkt kann dann wie in Abbildung 6.3 auf Seite 167 dargestellt werden. Er ermöglich die Nutzung als Fileserver genauso, wie den Zugang über herkömmliche Zugangsarten und bildet so einen vielseitig nutzbaren Dienst.

6.4 Erste Erfahrungen

Eingliederung nach Abschätzung von Nebeneffekten möglich

Dieser Teil der Arbeit gehört bereits zur Integration und Etablierung der neuen Software. Im vorliegenden Fall wurden im Zusammenhang mit der Einrichtung eines Testservers während der Tibet-Reise erste Erfahrungen gesammelt. Es stellte sich heraus, dass die Eingliederung in bestehende Software zwar relativ unproblematisch möglich ist, die nicht abschätzbaren Nebeneffekte aber eine gezielte Planung und Überprüfung der Integration erfordern.

Deshalb wurde in einem ersten Schritt erstmal nur die DLL für Windows-Integrationen erstellt, welche im Apache-Server für einige Versuche erfolgreich genutzt werden konnte.

6.4.1 Erzeugung einer DLL-Schnittstelle

Genereller Aufbau

In den globalen Zugangsfunktionen zur Bibliothek „ECFT_Lib“ werden allgemein benötigte Funktionalitäten auf die im Inneren liegende, objektorientierte Methodenschnittstelle abgebildet. Die DLL verfügt über eine Art spezielles Hauptprogramm, welches bei bestimmten Aktionen (z.B. erster Zugriff auf die DLL durch einen Prozess) aktiviert wird. In diesem „DllMain“ wird die benötigte Umgebung erzeugt und auch wieder vernichtet, so dass die Bibliothek als einer Art Bibliotheksklasse angesehen werden kann, obwohl der Zugangspunkt in C abgefasst ist.

Technische Realisierung

Die DLL exportiert über ein vor die Methode geschaltetes „__declspec (dllexport)“ seine Methoden, welche in der eigentlichen Anwendung mittels „__declspec (dllimport)“ importiert werden. Um dies komfortabler und von herkömmli-

cher Headerdateinutzung nicht unterscheidbar zu machen, wird über Definitionen die Auswahl automatisch getroffen. Um sowohl die Einbindung in C als auch in C++ zu gewährleisten, muss hier ebenfalls unterschieden werden. Da der Zugangspunkt in C programmiert ist, muss bei einer Einbindung in C++-Quellen vor jede Funktion die Erweiterung „extern "C"“ ergänzt werden. Um dem Anwendungsprogrammierer diese Auswahl zu ersparen, gibt es zwei Headerdateien „ECFT_RI.h“ für C++ und „ECFT_RIC.h“ für C. Darin werden entsprechende Zusätze automatisch angefügt.

Erweiterte Registrationsstruktur

Alle für die Nutzung der Schnittstellenbibliothek von ECFT wichtigen Informationen werden zwar in der Bibliothek verwaltet jedoch nicht dort gespeichert. Sie werden in Form einer erweiterten Registerstruktur „STRUCT_CFTFILE“ an die Anwendung übergeben (ein in Anlehnung an C zu handhabender Dateizeiger). Der Aufbau der Struktur ist in Tabelle 6.3 auf Seite 169 abgedruckt. Die Headerdateien selbst sind im Anhang P auf Seite 345 abgedruckt.

Element	Typ	Beschreibung
UserID	unsigned short	Nutzeridentifikation
StreamID	unsigned short	Eindeutige Kennung des Datenstroms
TimeStamp	unsigned short	Zeitmarke der letzten Aktivierung
Interface	void *	Zeiger zur Schnittstelle
CallingProg	char *	Name des aufrufenden Programms
InetAddr	char *	IP-Adresse, mit der eine Verbindung besteht
Port	char *	Port, zu dem eine Verbindung besteht
RemoteName	char *	CORBA-Applikationsname
RemoteKind	char *	CORBA-Applikationsart
RoundTripDelay	unsigned long	Deadline für einen kompletten Kommunikationslauf
Username	char *	Benutzername zur Registrierung
Password	char *	Passwort zur Registrierung

Tabelle 6.3: Erweiterte Verwaltungsstruktur „STRUCT_CFTFILE“ für die DLL

6.4.2 Erste Versuche mit dem Apache HTTP-Server

Ersatz der „file_io“

Erste Arbeiten mit der neuen Bibliothek wurden beim Nutzungsversuch des Apache-Servers für HTTP und WebDAV unternommen. Dort sind alle Dateimanipulationen in Modulen enthalten, welche im Ordner „. . . /httpd-2.0.47/srclib/apr/file_io“ zusammengefasst sind. Alle in den dortigen Funktionen enthaltenen Zugriffsmethoden auf das Dateisystem sind zu ersetzen oder zu erweitern.

Problem im Zeitverhalten beim zeichenweisen Lesen

Es stellte sich relativ schnell heraus, dass der Apache-Server zwar mit den umgebauten Funktionen problemlos arbeitet, jedoch in seinem Zeitverhalten stark verschlechtert wird. Dies liegt daran, dass z.B. beim Lesen von Konfigurationsdateien zeichenweise vorgegangen wird. Der Overhead durch den Protokollstack kommt daher extrem zum Tragen. Das bereits in Apache in ähnlicher Form programmierte Nutzen von Zwischenpuffern, in die blockweise eingelesen wird um von dort zeichenweise wieder auszulesen, bringt nur teilweise Verbesserungen.

Generelle Nutzung der Dummy-Schnittstelle zur Administration

Aus diesem Grund ist es damit generell ratsam, die Schnittstellenmethoden in Form der Dummy-Version als Verwaltungsaufrufe zu aktivieren. Die eigentlichen Zugriffe können mittels herkömmlicher Standardfunktionen wesentlich effektiver umgesetzt werden.

Die Arbeiten mussten vorerst auf diesem Stand belassen werden, da es für die Tests von Lhasa/Tibet nach Wettzell notwendig war, eine stabile Umgebung zu haben. Trotzdem ist nachgewiesen, dass die Bibliothek funktioniert und auf die beschriebene Weise nutzbar ist.

6.5 Zusammenfassung

Erste Vorschläge für Folgeprodukte

Mit diesem Kapitel begann der mehr aus Überlegungen aufgebaute Teil der Arbeit. Die Umsetzungen beziehen immer stärker bestehendes mit ein. Damit können die Planungen als erste Vorschläge für Folgeprodukte und langsam durchzuführende Integrationen angesehen werden. Umgesetzt wurde bisher die Erstellung einer `DLL` für Windows und eine teilweise Integration in die Module zur Dateimanipulation in den Apache-Quellen. Die Metadaten werden bislang noch nicht eingesetzt.

Systemüberblick

Beginnend von der Überlegung unterschiedliche Zugangsmechanismen zu vereinen, entstand der Gedanke einer eigenen Bibliothek für den Zugang zum Dateisystem oder zumindest für die Handhabung von Verwaltungsaufgaben. Eine innere Struktur ordnet den physikalischen Namensräumen, den Ordnern oder Verzeichnissen, eine Unterteilung in Set und Container zu, welche zusätzlich zu den systemgegebenen Attributen weitere Speicherungsmetainformationen besitzen. Damit ergab sich eine Aufteilung der Metadaten in systembezogene, speicherbezogene und zusätzlich inhaltbezogenen Informationen. Auch jede Datei kann zudem über weitere Zusatzinformationen verfügen, welche sich im Bereich der systembezogenen Daten an der Dublin Core Definition orientieren.

Vorteil von Metadatensammlungen

Ein Hauptgrund für den Einsatz von solchen Metadatensammlungen ist, dass dort Verweise zu maschinenlesbaren Formatbeschreibungen (z.B. in `DTD` , `XSD` oder `A2X`) eingegliedert werden können. Damit ist eine automatische Validierung schon bei der Speicherung möglich. Desweiteren sind dadurch automatische Transformationen möglich, welche entsprechend aufgesetzt werden können.

Angliederung von Fremdsoftware

Die geschaffene Bibliothek kann als Ersatz oder im Verbund mit den Standardbibliotheken zur Dateimanipulation der Programmiersprachen genutzt werden. In diesem Zusammenhang haben erste Erfahrungen die Nutzbarkeit bestätigt. Die Angliederung des Apache-Servers für `HTTP` und `WebDAV` bietet sich dabei besonders an, da damit die Möglichkeit zur Nutzung als eigenes Netzdateisystem gegeben wird.

WebDAV bereits weit verbreitet

Die Wahl von `WebDAV` stellt sich dahingehend als geschickt heraus, dass zwar der große Trend bereits etwas nachgelassen hat, `WebDAV` aber von zahlreichen Anwendungen in unterschiedlichen Betriebsystemen und mit unterschiedlichen Aufgaben unterstützt wird. So basiert z.B. der Nachfolger von `CVS` , die Versionsverwaltung „Subversion“, ebenfalls auf diesem Standard. Die zahlreichen, in den Betriebssystemen standardmäsig vorhandenen Einbindungsmöglichkeiten lassen `WebDAV` zu einer echten Alternative zu den herkömmlichen Netzwerkdateisystemen werden[8].

[8] vgl. dazu [FIN04] a.a.O. S. 122

Ziel ist die Schaffung neuartiger Serverkomponenten

Alle Planungen münden somit in der Ausarbeitung von neuen Serverkomponenten und der entsprechenden Anpassung von Fremdquellen. Besonders im Bereich eines GPS -Sammelservers kann hier ein leistungsfähiges und umfassendes Konzept entwickelt werden. Die Nutzung von HTTP -basierten Netzdateisystemen ist für die Überwindung von Firewallgrenzen ein großer Vorteil, weil damit ein gemounteter Web-Ordner ohne Umwege direkt vor der Firewall liegende Server ansprechen kann. Gepaart mit der Selbstabilität und Güteerhaltung des im Hintergrund agierenden ECFT ist so ein qualitativ hochwertiger und stabiler Server möglich.

Kapitel 7

Strukturelle Verbesserung der Informationsverbünde

Schwerpunkt des Kapitels:

Das nachfolgende Kapitel beschäftigt sich mit den Gedanken und Ideen zur Strukturierung der einzelnen Einrichtungen der Informationstechnologie in der Fundamentalstation Wettzell. Grundüberlegung ist die Schaffung von Sicherheitsenklaven und Kompetenzbereichen, welche zusammengefasst das Kompetenzzentrum Fundamentalstation Wettzell ergeben. Aufbauend auf diese Unterteilung müssen entsprechende Überlegungen zur Informationspräsentation und -verteilung geschaffen werden. Dazu sollen die gewohnten hierarchischen Strukturen anhand natürlicher Beziehungen zueinander ausgebaut und umstrukturiert werden. Daraus ergibt sich die Vision zu einem `Wettzell Data Management System (WDMS)`.

7.1 Ausgangsüberlegung

IT-Umstrukturierung

Im Zuge neuer Entwicklungen, wie z.B. dem neuen, automatischen `Satellite Observing System Wettzell (SOS-W)` zur Laserentfernungsmessung (Fertigstellung voraussichtlich im Jahr 2007), kommt es zu Erweiterungen und Umstrukturierungen der gegebenen `Information Technologie (IT)`. Besondere Fragestellungen gelten hierbei der Gewährleistung von Sicherheit innerhalb der Informationstechnologie in der Fundamentalstation Wettzell. Deshalb wird auch darüber nachgedacht, einzelne Messsysteme in eigene Sicherheitszonen einzuteilen, wo sie vom restlichen Datenverkehr abgeschirmt sind.

Anwendung der bisher erstellten Elemente

Gerade diese Untergliederung der gesamten `IT` in kleinere, handhabbare Zonen fordert geradezu, auch eine verwaltbare Unterteilung der Daten und ihrer zugehörigen Arbeitsumfelder durchzuführen. Während im bisherigen Verlauf der Arbeit im Wesentlichen das elementare Grundhandwerkszeug aufgebaut wurde, um Datenzugangspunkte zu schaffen und anzusprechen, werden diese nun auf die Gegebenheiten in der Fundamentalstation Wettzell übertragen und in Beziehung gesetzt.

Handhabbare Verwaltungsbereiche

Jeder Verwaltungsbereich bildet für sich genommen eine logische Einheit, welche für externe Bereiche Daten über entsprechend vordefinierte Schnittstellen (z.B. `ECFT` oder `WebDAV`) bereitstellt. Mit diesem Grundgedanken der beherrschbaren Aufteilung waren die ersten Strukturierungsmaßnahmen entstanden.

Hierarchisches Pipelining

Weiter muss nun eine Möglichkeit zur Datenverteilung und zum Zugang von verteilten Stellen aus geschaffen werden. Das bedeutet, dass die Arbeitsabläufe strukturiert werden müssen, um die Informationspräsentation entsprechend auszulegen. Hier hilft die bereits in den meisten Fällen existierende Arbeitsabfolge in Form eines Fließbands. Ein Arbeitsschritt folgt dem nächsten. Dabei kann dieses Pipelining auch über mehrere physikalische Ressourcen verteilt werden, wobei dann meist eine hierarchische Anordnung besteht. Dieses Prinzip der hierarchischen Fließbandverarbeitung muss deshalb entsprechend funktional ausgebaut werden und stufenweise in sich geschlossene Prozesse formen. Damit würde sich auch hier als logische Struktur des Ablaufs das bereits aus dem vorherigen Kapitel bekannte Schichtenmodell oder Zwiebelringmuster ausprägen. Entsprechend elegant ergeben sich so die Abbildungen der bestehenden, natürlichen Strukturen. Verschiedene Verwaltungsbereiche (z.B. die Messsysteme) zusammengefasst, ergeben einen weiteren, umfangreicheren Bereich (z.B. die Fundamentalstation).

Bedeutung für die FS Wettzell: Generelle Erhaltung der Transparenzkriterien

Bedeutung für die Geodäsie: Beispielaufbau mit an der Realität orientierten Verwaltungsstrukturen

Bedeutung für die Informatik: Beispiel einer Strukturierung

Die Lösungen in diesem Umfeld gehören noch zur Realisierung der dritten Säule des Abstraktionsprozesses (vgl. dazu Abb. 2.5 auf Seite 34) und erhalten mit Hilfe erklärter Verfahren und einer Strukturierung dieser die Transparenzkriterien für die komplette Fundamentalstation Wettzell. Für die Geodäsie kann dies damit als Beispielaufbau angesehen werden. Dieser Aufbau existiert nur in Form theoretischer Überlegungen und muss erst Schritt für Schritt praktisch überprüft werden. Aus Sicht der Informatik bedeutet es, in verschiedenen Bereichen bekannte Mechanismen (Prozedurbildung, Vererbungsmechanismen, etc.) als strukturierende Maßnahmen einzusetzen. Somit ist gerade dieses Zusammenspiel eine exemplarische Überlegung zum Strukturieren.

7.2 Theoretische Grundlagen

Zu Beginn der Planungen sollen benötigte Begriffe eingeführt und erklärt werden, welche im folgenden Abschnitt Verwendung finden.

Sicherheitsenklaven

Bereits mehrfach wurde die Einteilung der Stations-IT in kleinere Sicherheitszonen genannt. Diese werden fortan als Enklaven bezeichnet.

Begriffsbestimmung BEG7.1 „Enklave“:
Eine Enklave ist ein in sich abgeschlossener IT -Bereich, welcher komplett von einer für ihn nicht weiter wichtigen und damit fremden IT -Landschaft umgeben ist, welche teilweise zudem durch feindliche Einflussnahme auf die für den Bereich eigenen Abläufe zur Erbringung eines Dienstes eine Gefahr darstellt.

Selbstverwaltete Zellen mit Schutzmechanismen

Enklaven bieten damit selbstverwaltete Zellen in einem größeren, übergeordneten System. Sie können mittels geeigneter Schutzmechanismen (Firewall, etc.) auf Hardware- und/oder Softwarebasis von der übrigen Umgebung abgetrennt und geschützt werden. Ein wirksamer Schutz z.B. vor einem „Broadcaststorm“ (Ausschüttung von übermäßig vielen Nachrichten an alle Mitglieder eines Netzverbundes), wie er z.B. bei fehlerhafter Netzwerkhardware manchmal auftritt und ein Netz zum Stillstand bringen kann, ist das simple Abtrennen einzelner Teilnetze durch einen Router.

Autarke Kompetenzbereiche

Diese Unterteilung orientiert sich am Zusammenspiel der einzelnen Komponenten und damit an der Art ihrer Aufgaben. Es werden dadurch Bereiche ausgeprägt, welche für sich genommen eine gewisse Art von Autarkie aufweisen. Zur Erbringung dieser müssen sie deshalb über bestimmte Fähigkeiten verfügen (entsprechende Hardware- und Softwarelösungen für innere Abläufe), welche ihre Kompetenzen darstellen. Somit bilden Enklaven eine sehr strikte Form von Kompetenzbereichen aus.

Begriffsbestimmung BEG7.2 „Kompetenzbereich“:
Ein Kompetenzbereich ist eine für sich genommen mit all den inneren Mechanismen und Fähigkeiten ausgestattete Zone, welche zur Erbringung einer speziellen Dienstleistung benötigt werden.

Gekapselte Kommunikation

Der Hauptteil der Kommunikation findet somit innerhalb dieser logischen Zone statt. Dieser Datenaustausch kann beliebiger Art und Weise sein, da er keinerlei andere, fremde Bereiche betrifft. Für Kommunikationsvorgänge, welche zur umgebenden Informationstechnologie stattfinden, müssen bestimmte, allgemein standardisierte Regeln eingehalten werden. Deshalb muss jeder dieser Bereiche mindestens über eine allgemein bekannte Schnittstelle verfügen (z.B. WDAP).

Logische Verwaltungseinheiten

Kompetenzbereiche sind allgemein betrachtet logische Verwaltungseinheiten. Sie fassen Aufgaben eines zusammengehörenden, in sich geschlossenen Tätigkeitsbereichs zusammen. Diese können dabei entweder auf einer eigenen Hardware ablaufen oder gemeinsame Ressourcen nutzen und definieren keine speziellen Anforderungen bzgl. Sicherheit. Damit bilden Kompetenzbereiche eine weniger strikte, logische Verwaltungsumgebung als z.B. Enklaven.

Zusammenfassen zu einem Kompetenzzentrum

Während Kompetenzbereiche in sich nur der Erfüllung eines Dienstes genügen, können mehrere solcher Bereiche auch zu einem übergeordneten Gebilde zusammengefasst werden. Diese Anordnung erfüllt damit über ihre eingegliederten Diensterbringer verschiedene Aufgabenbereiche und stellt diese einer größeren Mehrheit zur Verfügung. Eine solcher Verbund soll im Folgenden als Kompetenzzentrum benannt werden.

Begriffsbestimmung BEG7.3 „Kompetenzzentrum“:
Ein Kompetenzzentrum ist ein übergeordneter Kompetenzbereich, welcher mehrere unterschiedliche (heterogene) Dienste zusammenfasst und diese einer weiter gefassten Umgebung anbietet.

Die Fundamentalstation als Kompetenzzentrum

Die Fundamentalstation Wettzell wäre damit ein Kompetenzzentrum, da es die erbrachten Daten der verschiedenen Messsysteme zusammengefasst nach außen liefert. Dabei kann intern eine Datendoppelung stattfinden, so dass diese redundant vorliegen und nach außen direkt von einer übergeordneten Ressource des Kompetenzzentrums stammen. Des Weiteren können auch Zusatzdienste angeboten werden, welche das Zentrum selbst erbringt. Damit ist u.a. die Weiterverarbeitung von Daten zu neuen, evtl. verdichteten Informationen gemeint.

Selbstähnliche Zugangsschnittstellen

Von außen gesehen besitzt ein Kompetenzzentrum dieselbe Zugangsschnittstelle, wie ein einfacher Kompetenzbereich. Diese kann einerseits ein Spiegel der innen liegenden Schnittstelle eines Kompetenzbereichs sein, wodurch ein Zugriff direkt auf den inneren Dienst möglich ist. Andererseits kann es sich, wie gesagt, um eine komplett duplizierte Version eines inneren Dienstes handeln. Zusammengefasst ergeben sich hierarchische Zugangsstrukturen, bei denen jede Stufe selbstähnlich zu ihren untergeordneten Ebenen aufgebaut ist.

Objektorientierte Strukturierung einer virtuell nachgebildeten Realität

Bei genauerer Betrachtung wird ersichtlich, dass die Einteilung in Verwaltungsbereiche reale Gegebenheiten in die virtuelle Systemstruktur abbildet. Die verschiedenen Kompetenzzonen bilden damit Objekte, mit denen eine objektorientierte Strukturierung eines komplexeren Bereichs in greifbare, kleinere durchgeführt werden kann. Es sind nun noch die Beziehungen der Objekte untereinander zu defnieren.

Schnittstellenvererbung

Wird nur die Schnittstelle gespiegelt, so soll im weiteren Verlauf von einer Vererbung gesprochen werden, da quasi die äußerlich sichtbaren Eigenschaften eines Kompetenzbereichs an das überlagerte Kompetenzzentrum vererbt werden.

Begriffsbestimmung BEG7.4 „Derivation“:
Unter Derivation (Vererbung) wird im Folgenden die Übernahme von Wesenseigenschaften (Charakter) der Schnittstelle eines untergeordneten Bereichs verstanden. Ein übergeordneter Dienst verfügt dann über die selben Zugangsmechanismen, welche aber auf einen Diensterbringer eines untergeordneten Bereichs verweisen.

Ableitung von komplexen Mehrwertdiensten

Ein Kompetenzbereich kann von einem anderen erben, solange die vererbenden Dienste derselben logischen Aufgabe zugeordnet sind (z.B. könnten GPS - Permanentstationen einen direkten Zugriff auf ihre Daten an einen übergeordneten GPS-Sammler vererben, so dass eine Abfrage auf den Sammler direkt weitergeleitet wird an die Station). Andernfalls bilden sich Kompetenzzentren aus. Je-

der diesen Zentren übergeordneter Bereich kann ebenfalls entweder die komplette Schnittstelle des Zentrums oder nur Teile daraus erben (im letzteren Fall, können z.B. wieder übergeordnete Kompetenzbereiche mit nur einem einzigen Typus an Dienstleistung abgeleitet werden). Wird eine solche Vererbung durchgeführt, so kann man auch von einer Ableitung sprechen, weil sich damit höhere Dienste von niedrigeren ableiten lassen.

Der Proxymechanismus als eine virtuelle Vererbung

Bisher wurde eine Vererbung der Schnittstellen angesprochen, welche in Form eines Namensdienstes auf die eigentlichen Diensterbringer verweist, mit denen der Dienstnehmer dann direkt kommunizieren muss. Es ist jedoch auch möglich, dass ein Klient nur mit dem erbenden Bereich kommuniziert und dieser die Kommunikation an den eigentlichen Diensterbringer weiterleitet. Es wird damit zwar die gleiche Schnittstelle vererbt, jedoch verweist sie auf einen lokalen Proxy als Vermittler. Dieser wirkt für andere wie ein eigenständiger Blattknoten des Ableitungsbaums. Diese Form der Vererbung soll deshalb als virtuelle Vererbung bezeichnet werden.

Begriffsbestimmung BEG7.5 „Virtuelle Derivation“:
Bei der virtuellen Derivation erbt ein abgeleitetes Element die selbe Schnittstelle, wie bei der bisherigen Vererbung, sie verweist jedoch auf einen lokalen Vermittlungsdienst (Proxy), welcher die Anfragen an den Diensterbringer weiterleitet. Ein Klient kommuniziert damit nicht mehr direkt mit dem Anbieter, sondern mit einem Vermittler, weshalb dieser als neuer, virtueller Blattknoten wirkt.

Weitergabe von Informationen

Die bisher eingeführte Terminologie umfasst im Prinzip nur die Verbreitung von Verweisen auf Zugangsschnittstellen zu Informationen, welche von einem bestimmten Diensterbringer angeboten werden. Ein Klient kann so die Informationen abfragen. Oft ist es jedoch auch nötig, dass auf definierte Weise Information und damit Wissen an andere Verarbeitungsknoten automatisch übertragen wird, welche dieses entweder ebenfalls anbieten oder zu neuem Wissen weiterverarbeiten. Im ersten Fall kann so eine gewisse Lastverteilung zwischen Anbietern mit redundantem Datenangebot erreicht werden. Durch Verarbeitung hingegen entsteht ein neues, individuelles Angebot an Informationen.

„Inokulation“ als entliehener Begriff aus der Medizin

In beiden Fällen kann dies in Form hierarchischer Beziehungen umgesetzt werden, so dass ein untergeordneter Dienstanbieter seine Daten an einen übergeordneten weitergibt. Er muss dieses dabei quasi in eine ihm fremde, externe Umgebung und Verarbeitungszelle einbringen. Gerade dieser Vorgang des Einbringens von Material in eine Zelle wird in der Medizin als „Inokulation“ bezeichnet[1], weshalb dieser Begriff auch hier in diesem Zusammenhang dafür entliehen werden soll. Dabei handelt es sich nicht um eine Art von Caching-Mechanismus, da die Informationen zumeist nicht nur temporär abgelegt, sondern auf unterschiedlichen Knoten gespeichert und häufig zusätzlich auch in einer weiterverarbeiteten Form genutzt werden.

[1] vgl. dazu [LEO05] unter dem Suchbegriff „Inokulation“

Begriffsbestimmung BEG7.6 „Inokulation“:
Unter Inokulation in diesem Zusammenhang versteht man das Einbringen von Informationen (Wissen) in eine fremde Umgebung (meist eine übergeordnete Ressource). Dabei kann zwischen aktiver (der Datenempfänger muss sich die Informationen selbständig holen) und passiver Inokulation (der Datenempfänger bekommt die Daten automatisch zugesandt) unterschieden werden.

Pipelines mit aktiver und passiver Datenweitergabe

Eine Reihung solcher Datenweitergaben bildet Inokulationsketten oder Pipelines aus, welche die Datenweitergabe von einem Knoten zum nächst höheren widerspiegeln. Die Datenweitergabe kann dabei aktiv oder passiv sein. Aktiv ist sie dann, wenn ein Knoten (Client) von sich aus aktiv die von ihm benötigten Daten holt. Bei der passiven Einbringung von Daten, agiert der Empfänger in der Art eines Servers, der von einem Client mit den Daten versorgt wird. Da damit Daten redundant vorliegen, ist es bei einer Änderung an einer Stelle nötig, dass alle weiteren Speicherorte ein Update erfahren. Diese Auffrischung kann im Stil einer herkömmlichen Dateneinbringung ablaufen, so dass man von einer aktiven und von einer passiven Auffrischung sprechen kann. Ein spezieller Typus von Daten lässt sich dabei immer einem bestimmten Zustand des Datenflusses zuordnen, so dass veränderte Daten an den für ihre Art geeigneten Stellen in die Verarbeitung eingebracht werden müssen. Die Auffrischung erfolgt dann immer in Richtung des Datenflusses, so dass die Verteilung intuitiv geregelt ist (vgl. dazu auch den Abschnitt 7.3.2 auf Seite 184).

Arbeitsflüsse

Über die Datenflüsse werden zugleich auch die Arbeitsabläufe geregelt. Diese verlaufen entlang einer dieser Pipelines, so dass stufenweise höherwertige Dienste aufgebaut werden. Diese Workflows lassen sich ideal über standardisierte Beschreibungen, wie z.B. `XML Process Definition Language (XPDL)` und `Workflow XML (Wf-XML)` 2.0 von der `Workflow Management Coalition (WfMC)` [2], definieren. Darin ist z.B. festgelegt, welcher Ausgangszustand welche Aktion anstößt und damit zu welchem Endzustand führt. Da die Wahl der Beschreibungssprache keinerlei Einfluss auf die weiteren Umsetzungen hat, wird in dieser Arbeit nicht weiter darauf eingegangen.

Nachfolgend werden nun einige Ideen theoretisch auf die Fundamentalstation Wettzell und ihre Gegebenheiten angewandt. Natürlich lassen sich die in den folgenden Abschnitten genannten Prinzipien darüber hinaus auch auf die Folgeabläufe der Arbeitsflüsse für geodätische Daten ausweiten.

7.3 Die Vision für ein Wettzell Data Management System (WDMS)

Während bislang nur mehr oder weniger Einzelteile entwickelt wurden, welche unabhängig von den Belangen der Fundamentalstation Wettzell ebenfalls genutzt werden können, soll im weiteren Verlauf nun ein Konzeptvorschlag zur Anwendung der entwickelten Elemente in der Station gegeben werden.

[2] vgl. dazu [WFMC04]

Drei Stufen der Umsetzung

Dieser Vorschlag gliedert sich grundsätzlich in drei Teile. Als erstes wird eine Einteilung der Stations-IT in handhabbare, selbstverwaltete oder zumindest logisch zusammenhängende Zonen vorgenommen. Diese Zonen müssen weiter miteinander in Beziehung treten. Zudem muss die Struktur von Verarbeitungsketten vorbereitet werden. Schließlich kann in einem letzten Punkt noch ein spezieller Lösungsversuch unternommen werden, um Firewallgrenzen zu überschreiten.

7.3.1 Die Einteilung in Kompetenzbereiche

Sinnvolle Bildung von Sicherheitsbereichen

Aufgrund langjähriger Erfahrungen auf der Station kristallisierte sich heraus, dass es durchaus sinnvoll ist, die einzelnen Messsysteme vom übrigen Stationsnetz physisch zu trennen und in einzelne Enklaven aufzuteilen. Gerade im Hinblick auf Störeinflüsse durch zahlreiche Broadcast-Nachrichten der vielen Bürorechner oder durch fehlerhafte Hardwarekomponenten bietet dies schon bei geringem Aufwand erheblichen Schutz. Zudem kann mittels eigener, kleiner Gateway-Rechner ein wesentlich sicherer Zugangspunkt zu und von den Messsystemen zu anderen Komponenten geschaffen werden. Da z.B. momentan von beliebiger Stelle aus HTML-Seiten für den Messablauf des WLRS aufgerufen werden können, kann dies ein gewisses Maß an Unsicherheit bilden, was bei der Aufteilung in Enklaven unterbunden ist.

Lokale Verwaltungen

Wenn nun diese Enklaven von Natur aus eine Art Selbstverwaltungsbereich darstellen, so bietet es sich in zunehmendem Maße an, auch die Abläufe und zugehörigen Daten lokal in dieser Zone zu verwalten. Dies beschränkt die Kommunikation zu umliegenden Einrichtungen während eines Messprozesses auf das Wesentliche. Eine schnellere, in sich konsistente und lokal kontrollierbare Verwaltung wird möglich, was zu einer Erhöhung der lokalen Stabilität innerhalb der Verwaltungsbereiche führt.

Individuelle, selbstgestaltete Auslegung der Verarbeitungseinheiten in den Enklaven

Für Enklaven bedeutet dies, dass sie über die nötigen Ressourcen an Rechen- und Speicherkapazität verfügen müssen, um den Hauptteil ihrer Aufgaben erfüllen zu können. Das heißt aber wiederum, dass man zentrale Einrichtungen der IT weniger leistungsstark auslegen kann, weil sich die Aufgaben damit auf mehrere Leistungsträger verteilen. Diese können der Situation entsprechend ausgelegt sein, was dem jeweiligen Verwaltungsbereich vorbehalten bleibt. Die Beschaffungskosten sollten sich hierbei nicht erheblich erhöhen, da insgesamt auf weniger leistungsstarke Recheneinheiten zurückgegriffen werden kann.

Kostengünstig durch VLAN

Eine kostengünstige Alternative zu physikalisch getrennten Netzen mit eigenen Netzwerkkomponenten kann hierbei die Errichtung von sog. Virtual Local Area Network (VLAN) bilden. Entsprechende VLAN-Switches verwalten verschiedene Zonen in einem LAN und erlauben die Definition von Regeln für Kommunikationsverbindungen zwischen den im Netz agierenden Kommunikationspartnern. Die dadurch mögliche, virtuelle Aufteilung eines flachen LAN in mehrere private Bereiche kann so den globalen Netzwerkverkehr begrenzen und vor sicherheitsgefährdenden Netzattacken schützen, wobei die Verwaltung an zentralen Switch-Komponenten stattfindet. VLAN-fähige Betriebssysteme erlauben es darüber hinaus, dass bestehende Kabelverbindungen zumeist ohne viel Umbauaufwand für virtuelle Netzbereiche genutzt werden können. Linux-Systeme unterstüt-

zen VLAN weitgehend (teilweise ist jedoch der Kernel anzupassen). Unter Windows sind die Netzwerktreiber dafür verantwortlich[3].

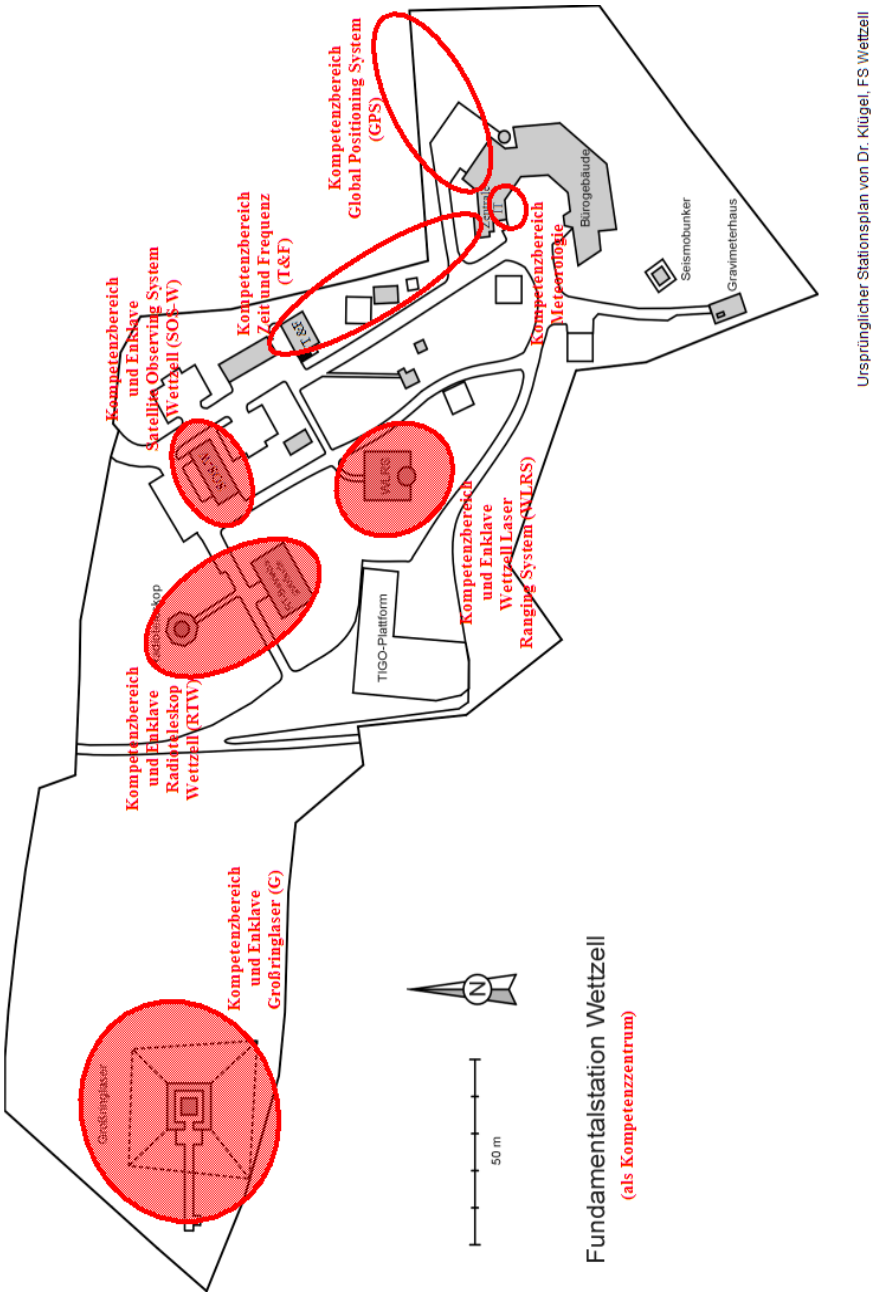


Abbildung 7.1: Die wichtigsten Kompetenzbereiche der Fundamentalstation

„Black Boxes“ hoher Abstraktionsstufe

Durch die Strukturierung, egal ob mit physikalisch getrennten Netzen oder mit VLAN -Bereichen, bildet sich das bereits mehrfach erwähnte Konzept der „Black Boxes“ aus, welches diesmal auf relativ hoher Abstraktionsebene angesiedelt ist. Wie welcher Anteil innerhalb einer solchen Box realisiert ist, bleibt den dort zuständigen Verantwortlichen überlassen, welche die Umsetzungen entsprechend ihrer Vorkenntnisse und Ausbildung durchführen können und nicht gezwungenermaßen den Tendenzen der zentralen IT folgen müssen. Damit ergibt sich auch für

[3] vgl. dazu [BEN05] a.a.O. S. 92 ff.

die Nutzer an den Messsystemen ein höheres Maß an Unabhängigkeit und Selbstverantwortung, was den gestiegenen administrativen Aufwand und die Verwaltung der Informationstechnik auf mehrere Schultern verteilt. Beide Seiten können so entsprechend entlastet und mehr oder weniger entkoppelt werden.

Schnittstellen zwischen den Enklaven gebildet durch WDAP

Will jedoch eine dieser Enklaven mit anderen in Kontakt treten, so muss sie den Vorgaben der Schnittstellendefinition folgen. Im Idealfall sind dies dann Realisierungen des WDAP . Sowohl Daten, welche von der Enklave angeboten werden, als auch solche die sie konsumiert, sind über diese einheitlichen Schnittstellen zugänglich, welche intern die Gütekriterien zur selbststabilen Erhaltung von Systemzuständen erfüllen.

Enklaven der FS Wettzell

Für die Station Wettzell können hier nachfolgende Enklaven definiert werden (vgl. dazu auch Abb. 7.1 auf Seite 180). In der Aufteilung sind nur Datentransfers über Netzwerkkomponenten berücksichtigt und nicht die zusätzliche Verteilungshardware mit ihren übergreifenden Beziehungen zwischen den Systemen, wie sie z.B. für die Verteilung von Zeitinformationen eingesetzt wird.

Die Enklaven der Fundamentalstation Wettzell:

- **WLRS :** Die momentane Realisierung der Wettzeller Laserbeobachtung hat nur wenige Datenverbindungen zu anderen Diensten innerhalb der Station (z.B. Meteorologie). Die restliche Kommunikation findet hauptsächlich mit externen Partnern statt, welche im Laserranging-Verbund arbeiten (Messstationen oder Datenzentren).
- **SOS-W :** Das bis zum Jahr 2007 neu zu entwickelnde, automatische Laserbeobachtungssystem kann vgl. zum WLRS ebenfalls als ein in sich abgeschlossenes System angesehen werden.
- **RTW :** Das Radioteleskop ist bzgl. der Datenmenge ein Novum. Die Terrabyte umfassenden Speicherungen werden physisch an die Korrelatoren versandt. Deshalb gibt es auch hier nur geringe Fremdwirkungen zu anderen Stationsdiensten (z.B. zur Meteorologie).
- **Großringlaser Wettzell (G):** Der Großringlaser erwirtschaftet sich den Großteil der benötigten Daten durch eigene Hardware und ist somit ebenfalls ein abgeschlossenes System. Die Errichtung einer Enklave würde hier sogar einen zusätzlichen Schutz vor Fremdeinflüssen über ein Netzwerk bieten.

Die ersten Umsetzungen dieser Strukturen laufen bereits. Die Enklaven WLRS , SOS-W und RTW sollen physikalisch eingerichtet werden. Dies ist der erste Schritt hin zur Realisierung dieser Vision.

Weitere Kompetenzbereiche der FS Wettzell

Zusätzlich zu den physischen Aufteilungen können auch noch weitere logische erkannt werden. Zum Beispiel ist die Datenerfassung der Meteorologie als ein in sich abgeschlossener Vorgang anzusehen. Allerdings läuft dieser auf zentralen Einrichtungen der IT . Damit können zu den bisherigen Kompetenzenklaven weitere logische Kompetenzbereiche hinzugefügt werden (vgl. dazu auch Abb. 7.1 auf Seite 180). Die Zugangspunkte zu den daraus resultierenden Informationen werden ebenfalls von zentralen Stellen ermöglicht, weshalb somit der zentrale WDAP

um diese Schnittstellen mit eigenen Datenbereichen und Zugangsrechten erweitert werden muss.

Die weiterhin erkannten logischen Kompetenzbereiche der Fundamentalstation Wettzell sind (alle weiteren Einrichtungen werden meist fremdbetreut):

- **GPS :** Der GPS -Sammler und seine Permanentstationen bilden einen weiteren logischen Bereich, der in die gegebene IT eingebettet ist. Die Aktionen laufen zwar auf eigener Hardware, diese ist jedoch nicht von der übrigen Umgebung abgeschirmt, sondern nur mittels der allgemeinen Firewall gesichert.
- **Zeit und Frequenz:** Auch die Überwachung der Zeitstabilität, bzw. die Verteilung von Zeitinformationen über das Netzwerk kann als solcher Dienst angesehen werden. Da jedoch zahlreiche Klienten darauf zugreifen müssen, ist es sinnvoll, ihn nicht vom übrigen Netz zu trennen, sondern ihm nur einen eigenen Bereich einzurichten.
- **Meteorologie:** Gerade auch die Wetterdatenbank mit ihren Zulieferprozessen und Datendienstleistungen für beliebige andere Anwendungsgebiete kann als solcher zentraler Prozess identifiziert werden.

Diese Einteilung bringt eine wesentliche Abstraktion, welche die einzelnen Einheiten als selbstverwaltete Objekte sieht. Damit werden im Grunde die real existierenden Strukturen, welche sich sogar in der Aufgliederung zu Betriebsgruppen wiederspiegeln, auch auf die IT -Struktur abgebildet. Die zentrale IT wird entlastet, wobei sich die einzelnen Verwaltungseinheiten bzgl. der Informationsverarbeitung nur in dem Maße spezialisieren müssen, wie es für die Erbringung ihrer Aufgaben nötig ist.

Dieser ganzen Strukturierung muss nun nur noch ein geeignetes Verarbeitungskonzept für die Datenprozesse überlagert werden.

7.3.2 Die Überlagerung hierarchischer Relationen

Je ein Namensdienst pro Kompetenzbereich

Ein angebotener Dienst eines Kompetenzbereichs wird zur Bewahrung der Ortstransparenz über einen Eintrag im Naming Service unter CORBA gefunden. Dieser liefert die entsprechende, momentan gültige IOR . Nun könnte man dafür sorgen, dass sich alle Dienste bei einem zentralen Namensdienst registrieren müssen. Dies bietet aber eine Angriffstelle für Ausfälle, da bei einer Störung dieser zentralen Einheit die Ausfalltransparenz nicht mehr gegeben wäre. Deshalb unterhält jeder Kompetenzbereich für sich einen Namensdienst, in den sich alle angebotenen Dienste des logischen Verbundes einzutragen haben.

Aktive Registrierung beim Namensdienst

Ein Server trägt sich immer aktiv beim Namensdienst ein. Er kann dies selbständig tun oder dazu angestoßen werden. In jedem Fall weiß er, bei welchen Diensten er registriert ist und muss sich nach einem Ausfall dort auch wieder anmelden, sofern keine persistenten IOR zum Einsatz kommen. Auch der Namensdienst muss seine Einträge sichern, so dass ein Aktivierungsclient nach einem Ausfall des Naming Daemons alle eingetragenen Dienste erneut zum Eintragen anregen kann.

Derivation von Charaktereigenschaften

Dieses Vorgehen ist jedoch nur bei kleinen Umgebungen praktikabel. Bei grösseren Systemverbänden müssten die einzelnen Einheiten zu viele Informationen über ihre Fremdbeziehungen verwalten. Hier kann aber ein Ableitungs- oder Vererbungsmodell zum Einsatz kommen. In diesem Modell registriert sich ein Dienst nur bei seinem eigenen Namensdienst. Dort sammeln sich somit die `IOR` der lokalen Dienste eines Kompetenzbereichs. Man kann dies damit als eine Sammlung der Wesenseigenschaften (Charakter) eines solchen Bereichs ansehen. Ein weiterer Namensdienst und damit übergeordneter Kompetenzbereich kann diese Einträge nun ererben (Derivation), indem er sie ausliest und sich die gewünschten Sätze auswählt. Danach kann er die erhaltenen `IOR` entweder direkt eintragen oder über sie die Dienste zum Eintragen auffordern. Es ergibt sich damit eine Vererbungsstruktur, welche den Speicheraufwand möglichst klein hält. Ein Namensdienst muss nur noch die Zugänge zu seinen nächsten Verwandten kennen, um von ihnen nach einem Ausfall seine benötigten Referenzen erneut zu erben (vgl. dazu als Beispiel die Vererbungspfeile in Abb. 7.2 auf Seite 184). Zusammengefasst bildet sich also eine Art Vererbungsbaum.

Zentrale Sammeldienste zur Informationspräsentation

Die bisher beschriebene Hierarchie betrifft nur die Namensauflösung und damit das Finden von Datenzugangspunkten. Sie beschreibt also die logischen Zusammenhänge der als Objekte eines Datensystems definierten Kompetenzzonen mit Hilfe selbstähnlicher Zugangsschnittstellen. Die Kommunikation mit diesen Zugangspunkten, also der Datentransfer selbst, findet dabei noch direkt mit den ursprünglichen Dienstanbietern statt. Dies kann jedoch zu Problemen führen. Insbesondere wenn zahlreiche Abfragen stattfinden, kann dadurch ein Kompetenzbereich übermäßig in der Performanz gestört werden, was gerade im Betrieb geodätischer Messungen störend ist. Somit ist es durchaus anzuraten, zentrale Sammeldienste einzurichten, welche letztendlich die Informationspräsentation nach außen darstellen.

Umsetzung der passiven Inokulation

Dabei handelt es sich meist um eigene, überlagerte Datendienste, welche als Aufgabe das Anbieten von Wissen haben. Dieses ist entweder eine Kopie bestehender Daten untergeordneter Bereiche oder eine Weiterverarbeitung davon. Das Weitergeben solchen Wissens wird hier als Inokulation bezeichnet, weil ein fremder Bereich mit den für ihn wichtigen Informationen komplett als Redundanz des Originals ausgestattet wird. Es gibt nun zwei Möglichkeiten, Wissen zu erwerben. Ein Empfänger kann ohne Zutun mit den notwendigen Daten versorgt werden. Diese Versorgung kann er zwar anstoßen, was gerade bei einer Auffrischung der Daten nach einem Ausfall nützlich ist, er spielt jedoch für die eigentliche Datenversorgung nicht den aktiven Part. Ein externer Client führt nämlich den Abgleich aus. In diesem Fall spricht man von einer passiven Inokulation. In Abb. 7.2 auf Seite 184 trifft dies vor allem auf die herkömmlichen GPS-Permanentstationen zu. Sie legen in regelmäßigen Zeitabständen ihre Daten bei einem Sammelserver ab.

Umsetzung der aktiven Inokulation

Die zweite Möglichkeit des Erwerbs ist die aktive Inokulation. Dabei greift ein übergeordneter Dienst gezielt auf Datenzugangspunkte von untergeordneten Diensterbringern zu und holt sich aktiv die gewünschten Datensätze. Diese Methode wird hauptsächlich dazu eingesetzt, Daten von verschiedenen Messsystemen in zentralen Einrichtungen zu sammeln und in Form eines Backups abzulegen. Die Kombination beider Möglichkeiten ergibt die Hierarchie für den Datenfluss (vgl. dazu erneut Abb. 7.2 auf Seite 184).

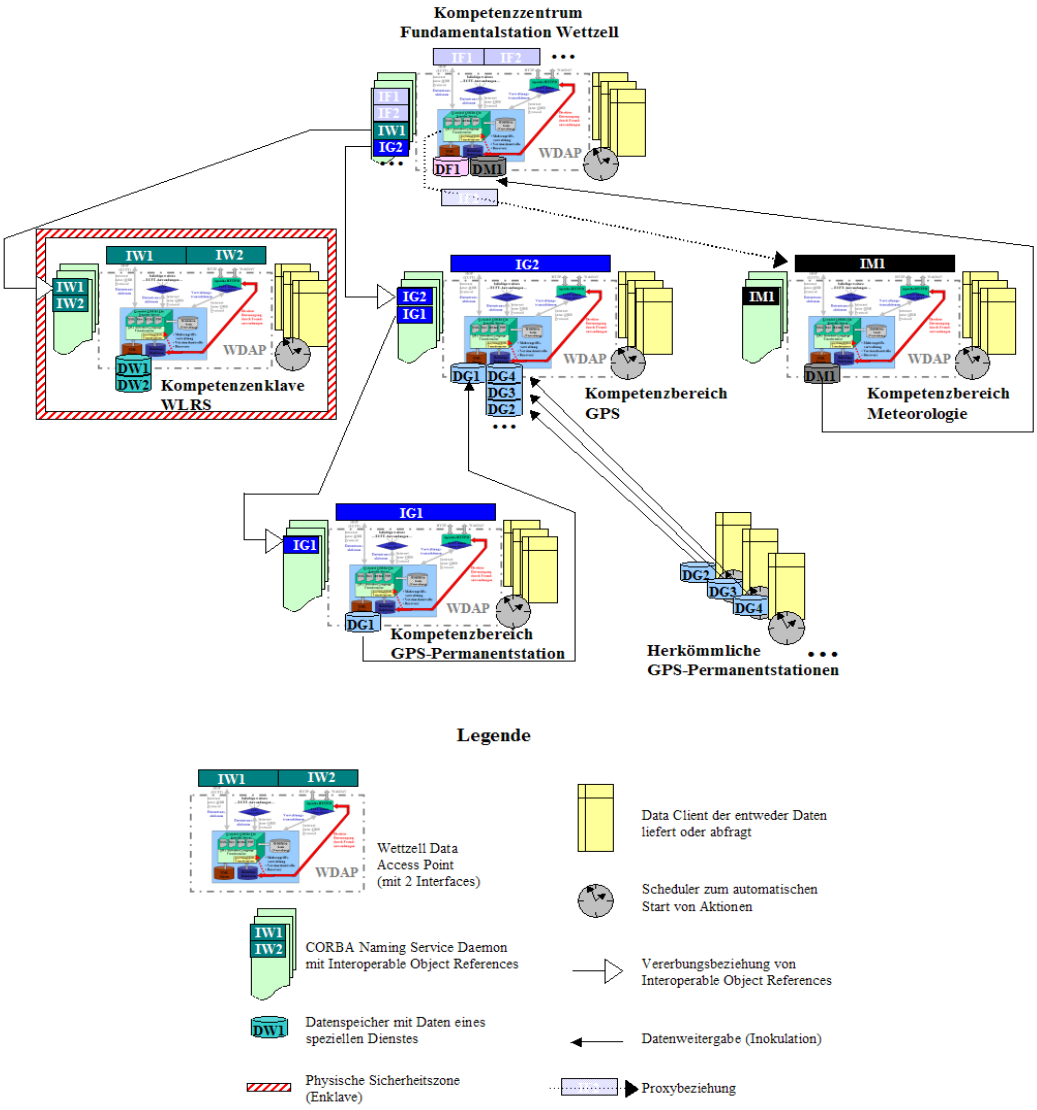


Abbildung 7.2: Das Grundprinzip der überlagerten Hierarchien

Umsetzung des Pipelinings

Dieser Datenfluss verläuft generell von untergeordneten Kompetenzbereichen zu übergeordneten. Insgesamt gesehen bildet sich eine weitere Baumstruktur aus. Jede Kantenlinie von einem Blatt hin zu einem inneren Knoten bildet ein eigenes Verarbeitungsfließband. Die einzelnen Verarbeitungsschritte einer solchen Pipeline werden immer in zeitlicher Abfolge vom Blatt zum Zentralknoten ausgeführt. Jeder Knoten (Arbeitsschritt) hat mindestens einen Eingang (welcher passiv oder aktiv an die benötigten Daten kommt) und kann mehrere Ausgänge als Datenzugangspunkte anbieten. Interne Abläufe folgen dabei genauso der zeitlichen Ordnung, wie die gesamte Pipeline. Resultatdaten stellen somit einen anderen Datenzugangspunkt dar, als zum Ausgang gespiegelte Eingangsdaten.

Vorgehen bei Änderung von Dateien

Eine gelesene und veränderte Datei kann somit nicht einfach auf den selben Ort zurückgeschrieben, sondern muss an der für sie geltenden Stelle in den Verarbeitungspfad wieder eingebracht werden. Dieser Vorgang kann für den Anwender unbemerkt von den beteiligten Knoten automatisch erfolgen. Das bedeutet, dass ein

verändertes Datum solange entgegengesetzt zu seinem Verarbeitungspfad läuft, bis es an dem dafür vorgesehenen Knotenpunkt in den regulären Verarbeitungsablauf eingebracht wird. Um eine komplexe Vermaschung zu verhindern, ist darauf zu achten, dass die Datenströme jeweils Baumpfade aufspannen (Vermeidung von Datenweitergabe auf einer Baumebene). So können parallel mehrere Verarbeitungsbäume entstehen, welche in sich eine klare Struktur und Verarbeitungsreihenfolge in Form mehrerer Pipelines aufzeigen.

Abbildung mittels XML-Beschreibungen

Die beschriebenen Produktionslinien lassen sich ideal mittels Pfadbeschreibungen durch den Baum definieren und können somit auch ideal über XML -Dateien angegeben werden. Sie definieren zu den logischen Zusammenhängen Arbeitsflüsse und damit Verarbeitungspipelines. Somit können beliebige, eigene oder standardisierte XML -Schemas (z.B. XPDL) eingesetzt werden. Es entsteht eine relativ klare Strukturierung. Die Beschreibungen können sich dabei auf den kompletten Baum oder nur einzelne Teilpfade beziehen und einem Konten entweder als Gesamtheit oder in für ihn wichtigen Abschnitten bekannt sein.

Verteiltes Datenmanagement zur Abbildung realer Gegebenheiten

Gerade das Zusammenspiel dieser beiden Baumstrukturen zur Vererbung von Zugangseigenschaften und zur Weitergabe von Wissen bietet eine ideale Basis zum Aufbau eines verteilten Datenmanagements, welches die Vorteile der Verteilung (z.B. Ausfälle von Teilen beeinflussen andere nicht) mit den Vorteilen zentraler Einheiten (z.B. gibt es aufgabenspezifische Zentren, welche an zentraler Stelle gegen Ausfälle abgesichert werden können) vereinigt. Es ist ersichtlich, dass sich die Strukturen und ihre Beschreibungen den hierarchischen Gegebenheiten auf ideale Weise anpassen lassen. Sie bilden natürliche Baumstrukturen aus und zeigen übersichtlich Relationen und Zusammenhänge eines komplexen Zusammenspiels einzelner Kooperationspartner in einem System. Allerdings schränken sie den Datenverkehr insofern ein, dass jede Verarbeitungsfolge geeignet in die Baumstruktur eingegliedert werden muss und z.B. einen weiteren Verarbeitungsast aufspannt. Fällt ein Dienst weg, so wird an geeigneter Stelle der zugehörige Verarbeitungspfad abgetrennt, wodurch dann auch alle Folgeprodukte entkoppelt sind.

Nutzung von systemeigenen Clients

Trotzdem bleibt in dem gesamten Gefüge weiterhin der Grundsatz der Client-Server-Technik bestehen. Das bedeutet, dass Server nur rein bedienende Elemente des Ablaufs sind, während die Aktivitäten von den Clients ausgehen. Somit ist es auch klar, weshalb jeder Server über mehrere Klienten mit unterschiedlichen Aufgaben verfügen muss. Sie werden mittels eines Schedulers, der als Koordinator und Aktivierungseinheit fungiert, angestoßen und überwacht. Zur weiteren Überwachung können zudem auch „Watchdogs“ eingesetzt werden. Diese spezialisierten Clients überprüfen in regelmäßigen Abständen entsprechende Gegebenheiten, indem sie z.B. bestimmte Anfragen an einen Server stellen. Auch sind weitere Server denkbar, welche regelmäßig Statusinformationen („Heartbeat“) von einzelnen Einheiten erhalten, die sozusagen als Lebenszeichen dieser gelten. Dazu können Server in vordefinierten Zeiteinheiten angetriggert werden, sofern sie keine Aufträge bearbeiten.

Umsetzung von Proxymechanismen

Eine Sonderform in diesem Gefüge stellt eine weitere Art von Verbindung zwischen zwei Diensten dar. Aufgrund des internen Aufbaus von ECFT ist es möglich, an den Multi-Tier-Connector oder direkt an das virtuelle Skeleton eine weitere, entfernte Schnittstelle anzuhängen. Somit kann auf überschaubare Weise eine Art von Proxymechanismus realisiert werden. Ein Server bedient die Anfragen nicht selbst,

sondern leitet sie über sein Interface an einen damit verbundenen, weiteren Server weiter. Bei der Umsetzung können sogar Überprüfungen auf Korrektheit und gegen feindliche Attacken durchgeführt werden, so dass der ursprüngliche Daten- und Befehlstrom nicht unmittelbar weitergeleitet wird.

Dieses Proxykonzept ist eine echte Weitervermittlung, was bedeutet, dass ein Client nur die angesprochene Schnittstelle sieht, jedoch intern darüber von einer anderen Ressource bedient wird. Der Protokollstack wird pro Vermittlungsstufe einmal durchlaufen. Damit ergeben sich zusammengefasst die folgenden Möglichkeiten an die gewünschten Daten zu gelangen, worauf auch Mechanismen zur Datensuche aufbauen sollten:

- Ein Dienst kann direkt ohne Umwege angesprochen werden, sofern die aktuelle `IOR` bekannt ist.
- Ein Klient kann sich über Namensdienste, welche hierarchisch einen Vererbungsbaum aufspannen, die Referenz eines Dienstes holen. Anfragen, welche nicht direkt beantwortet werden können, müssen entsprechend im Baum weitergeleitet („geroutet“) werden. Der Datenaustausch findet dann mit dem Urheber direkt statt.
- Ein übergeordneter Bereich kann sich mit Daten untergeordneter Dienste inokulieren lassen, so dass ein Client nicht mehr direkt mit dem Urheber kommunizieren muss. Ein entsprechendes Update-Management zur Auffrischung veränderlicher Daten ist dazu notwendig.
- Ein übergeordneter Dienst kann als Proxy fungieren. Ein Client interagiert weiterhin mit der übergeordneten Instanz, jedoch werden die Anfragen an den Urheber weitergeleitet, welcher über den Umweg des Proxies seine Datenlieferungen ausführt.

Nutzer navigieren durch Baumstrukturen

Ein einzelner Endnutzer kann sich damit entweder direkt an betreffende Datenlieferanten wenden, sofern er die Baumstruktur kennt, oder er kann sich durch den Baum navigieren. Zu solchen Recherchen sollte zur Vereinfachung jeder Knoten nicht nur seine direkten Vorgänger und Nachfolger kennen, sondern auch von der Generation davor und danach wissen. Dadurch wirken sich Ausfälle einzelner Knoten nicht unmittelbar aus. Die Navigation kann dann in Form von Pfadabstiegen entweder dienstespezifisch oder allgemein stattfinden. Dadurch werden Parallelen zum herkömmlichen Surfen im `WWW` deutlich, wo Server durch hierarchische Kennungen als eindeutige Adressen angesprochen und Seiten in Form hierarchischer Dateibäume genutzt werden können.

Zeitliche Ordnung verteilter Ereignisse

In allen Überlegungen blieb bisher die zeitlich logische Einordnung von Ereignissen unberücksichtigt. Diese ist jedoch im Hinblick auf die Vergleichbarkeit und Synchronisierbarkeit von Zeitangaben in den verteilten Rechensystemen wichtig. Im momentan geplanten System für die Fundamentalstation Wettzell wird dafür das `Network Time Protocol (NTP)` vorgesehen, welches eine Genauigkeit im Nanosekundenbereich vorsieht[4]. Nach eigenen Erfahrungswerten liegt die

[4] vgl. dazu [MEI04]

Genauigkeit je nach Umsetzung in einem heterogenen LAN im Millisekundenbereich. Da im vorliegenden Kommunikationssystem mit ECFT jede entfernt ausgeführte Aktion in der Regel mehr als 10 Millisekunden benötigt, kann damit zumindest eine Grobeinschätzung der Ordnung von synchronisierenden Transfers vorgenommen werden. Nach dem Shannon'schen Abtast-Theorem[5] sollte jedoch darauf geachtet werden, dass die Genauigkeit der Zeitsynchronisation mindestens doppelt so hoch ist, wie die minimal aufzulösenden Zeitabschnitte der Ereignisanordnung. Im vorliegenden, Anwendungsfall reicht die Genauigkeit für das Management der Zeitangaben in Dateiattributen im Sekundenbereich sowohl im LAN als auch im WAN ohne Probleme aus. Im lokalen, heterogenen Netzwerk kann zudem eine zeitliche Ordnung von Ereignissen mit einem Zeitverbrauch größer als 10 Millisekunden ohne Schwierigkeiten durchgeführt werden. Sollen ähnliche Ordnungen auch im weltumspannenden Netz durchgeführt werden, so ist evtl. ein erheblicher Aufwand zu betreiben, um eine globale, zeitliche Ordnung zu gewährleisten[6]. Im vorliegenden System reichen die Mechanismen mit NTP vollkommen aus, so dass keine weiteren Mechanismen integriert werden müssen.

Damit sind nun alle wichtigen Bestandteile der neuen Überlegungen beleuchtet worden. Da in der bisher aufgezeigten Umsetzung innerhalb der Fundamentalstation Wettzell Sicherheitszonen in Form von mehr oder weniger gesicherten Enklaven entstehen, ist ein Konzept zum Überwinden dieser Grenzen von erheblicher Bedeutung. Zentrale Bedeutung hat hierbei die Bildung von Proxies.

7.3.3 Das Überqueren von Sicherheitszonen

Geeignete Überwindung von Sicherheitszonen

Das neue Konzept sieht die Aufteilung in Sicherheitszonen vor. Aus diesem Grund sind geeignete Maßnahmen zu schaffen, um die Grenzen dieser Zonen problemlos und ohne Umgehung der Sicherheit zu überqueren. Dazu sollen die in Abschnitt 4.5.4 auf Seite 123 beschriebenen Methoden zum Einsatz kommen. Es soll jedoch möglichst unabhängig von weiteren Entwicklungen der CORBA -Standardisierung sein.

Unabhängiger Proxy vor einer Firewall

Die Grundlage bildet das im vorhergehenden Abschnitt beschriebene Proxy-Konzept. Vor jede Sicherheitsgrenze (z.B. mit Firewall) wird ein unabhängiger Proxy-Server angesiedelt. Er besteht aus einem WDAP und dient als Schnittstellenduplikation für interne Dienstanbieter und führt eine echte Weitervermittlung zum eigentlichen Diensterbringer durch. Dabei muss eine Verbindung durch die Firewall geschaffen werden, welche die Sicherheit nur mäßig einschränkt. Es wird dazu zwischen eingehenden und ausgehenden Anfragen differenziert.

Ausgehende Verbindungen

Ausgehende Anfragen stammen von Dienstnehmern innerhalb der Sicherheitszone. Dazu können sie bereits existierende Kanäle durch die Firewall nutzen, wie z.B. für HTTP , worüber auch WebDAV verwendet werden kann. Auf diesem Weg ist es damit möglich, einen externen Netzordner vom Proxy direkt in interne Rechner einzubinden. Zusätzlich ist es bei einigen Firewalls möglich, die in Abschnitt 4.5.4 auf Seite 123 beschriebene Socksification des Clients zu nutzen und über SOCKS zu agieren.

[5] vgl. dazu [KOP97] a.a.O. S. 204

[6] vgl. dazu die Beschreibungen in [SING94] ab Seite 97

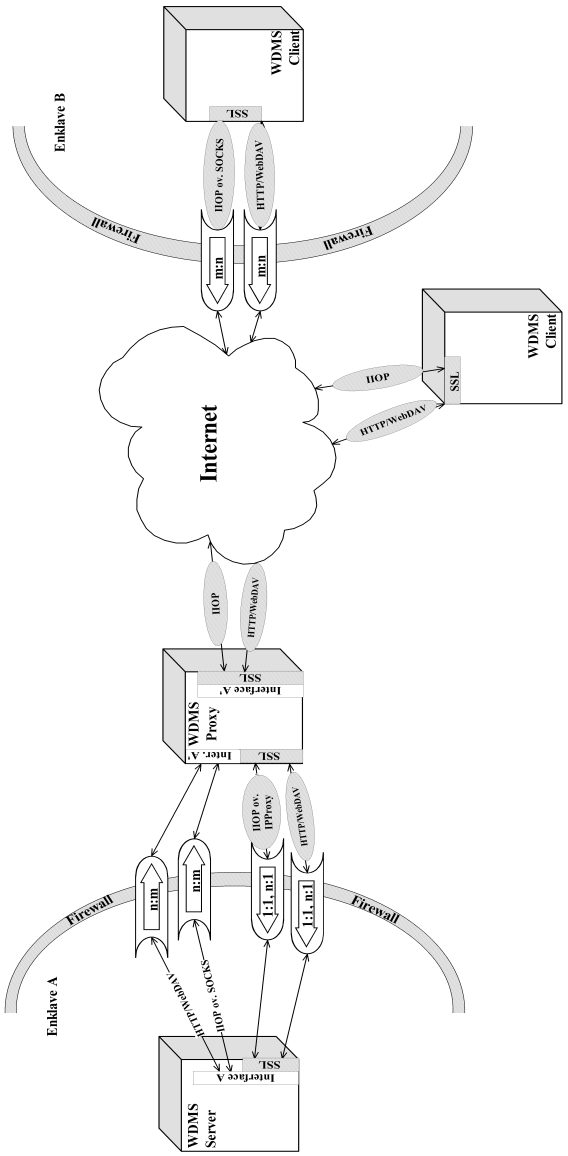


Abbildung 7.3: Überquerung von Sicherheitszonen

n:m-Beziehung

Bei diesen ausgehenden Verbindungen gibt es keine Beschränkungen für die Anzahl kommunizierender Partner. Beliebig viele interne Klienten können mit beliebig vielen externen Proxys oder ungeschützten Servern Daten austauschen (n:m-Verbindung). Ein interner Server, der vom Proxy gespiegelt wird, bzw. zu dem der Proxy verbindet, verhält sich nach diesem Clientmuster, falls er eine Datenverbindung nach außen initiieren muss.

Eingehende Verbindungen mit 1:1- oder 1:n-Beziehung des Proxys

Eingehende Anfragen, also Verbindungen, welche der Proxy initiiert, unterliegen strengeren Richtlinien. Nur der Proxy allein darf einen der geschaffenen Zugangskanäle nutzen, sonst niemand. Dies wird durch hartes Setzen des `IP` -Routings im Router der Firewall gewährleistet. Damit gibt es entweder 1:1- oder 1:n-Verbindungen vom Proxy in die Sicherheitszone. Zusätzlich ist eine `SSL` -Verschlüsselung zu empfehlen.

Externe Zugangskanäle

Zur Realisierung sind erneut zwei Kanäle denkbar. Einerseits kann ein in Abschnitt 4.5.4 auf Seite 123 beschriebener IP -Proxy in der Firewall genutzt werden, was eine Anpassung der IOR erfordert. Andererseits kann auch ein Kanal für HTTP und damit für WebDAV errichtet werden. Letzteres verursacht keine größeren Sicherheitslücken, da nur ein einziger Rechner (der Proxy) dafür erlaubt wird.

Nutzung aktiver und passiver Dateneinbringung

Durch den Aufbau ist es damit auch möglich, sowohl die aktive als auch die passive Inokulation über Firewallgrenzen hinweg zu nutzen. Der Proxy kann z.B. mit dem Aufruf eines CGI -Skripts über HTTP eine passive Dateneinbringung in seinen Bestand anstoßen. Es gibt zahlreiche weitere Wege und Kombinationen zum Datenaustausch. Auch das Vererbungskonzept kann dadurch beibehalten werden. Ein Proxy setzt dabei die virtuelle Derivation um und wirkt damit für andere wie ein neuer Blattknoten des Ableitungsbaums.

Proxies sind virtuelle Derivationen

Eine Proxyverbindung kann dabei ebenso wie die herkömmliche Vererbung während der Laufzeit stattfinden. Während bei der herkömmlichen Ableitung die Liste eines Naming Service ausgelesen wird, um jeden eingetragenen Server zu einem Registrieren beim erbenden Naming Service anzustoßen, wird bei der virtuellen Derivation in den Multi-Tier-Connector ein weiterer Remote-Interface-Adapter eingehängt. Dieser verbindet sich zuvor mit dem Dienstanbieter aus der geerbten Schnittstelle.

Höherwertige Firewalldienste durch den Proxy

Alle Anfragen von externen Klienten laufen damit über den Proxy. Dieser hat als zentraler Zugangspunkt auch die Möglichkeit, Anfragen auf Korrektheit oder Angriffe zu untersuchen. Klienten, welche selbst hinter einer Firewall sitzen, kommunizieren nach demselben, oben beschriebenen Muster für die ausgehende Kommunikation. Clients ohne weitere Sicherheitsrichtlinien können direkt mit dem Proxy kommunizieren. Es wird jeweils empfohlen, alle Verbindungen über eine SSL -Verschlüsselung laufen zu lassen.

Erweiterungen sind notwendig

Zur uneingeschränkten Nutzung der genannten Möglichkeiten muss entweder die Middleware für SOCKS ausgebaut oder ECFT um einen Adapter für die HTTP - und WebDAV -Nutzung erweitert werden. Auch ist es durchaus denkbar, dass zukünftige Firewalls bereits für CORBA ausgestattet sind, so wie sie z.B. von der Xtradyne Technologies AG bereits angeboten werden[7].

Zusammengefasst ist diese Vision deshalb nur über mittel- und langfristige Planungen zu realisieren. Doch auch ohne diese umfassenden Erweiterungen können bereits zahlreiche Zugangspunkte geschaffen werden, welche in einer etwas reduzierten Form des Datenmanagements ablaufen (z.B. nur HTTP als Kommunikationswege über Firewallgrenzen hinweg).

7.4 Erste Versuche

Ideen zur Strukturierung entstanden durch Analyse

Die bisher beschriebenen Ideen entstanden bereits aus den Analysen heraus, da sich eine Einteilung der Messsysteme in eigene Objektbereiche mit abgegrenzten Kompetenzen anbietet. Um eine Strukturierung der Arbeitsabläufe und Schnitt-

[7] vgl. dazu [XTRA04]

stellenverteilung zu schaffen, eignete sich ideal das Gedankenmodell abgeleiteter Schnittstelleneigenschaften.

Um dies zu verifizieren wurden anhand von kleinen Testprogrammen einige Szenarien durchgespielt. Dazu gehörte auch ein Beispiel zur herkömmlichen Vererbung von Einträgen eines Naming Service.

7.4.1 Beispiel einer Vererbung von Naming Service Einträgen

Bereits zu Beginn der Arbeit entstand ein Testprogramm (vgl. dazu auch Anhang Q auf Seite 349), mit welchem nachgewiesen werden konnte, dass eine Vererbung von Schnittstellen, welche in einem Naming Service verwaltet werden, möglich ist. Dazu wurde ein Client und ein Server geschrieben, den man mit kleinen Abänderungen unter verschiedenen Kennungen ansprechen konnte.

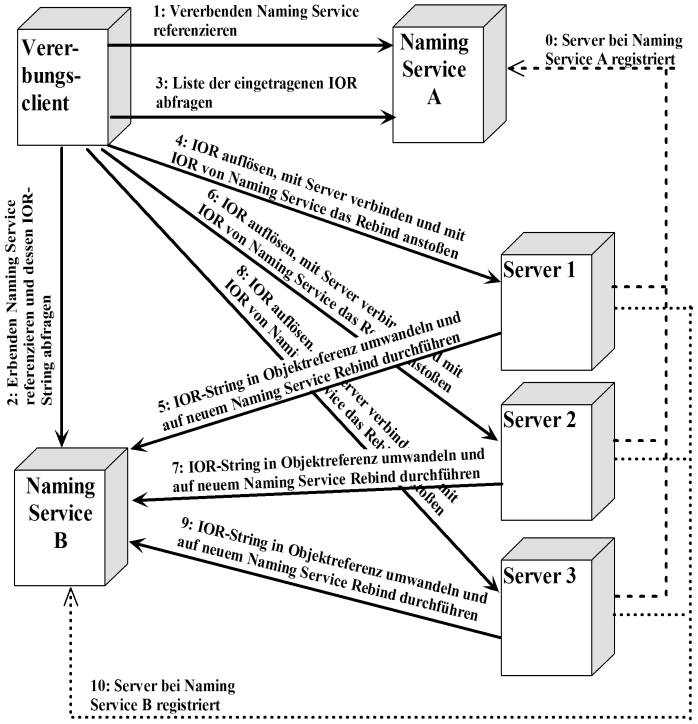


Abbildung 7.4: Aktionenfolge bei einer herkömmlichen Vererbung

Methode zum „Rebind“

Besonderheit am Server ist dabei eine Methode „usRebind“, welche aktiv von einem Initiatorclient der Vererbung angestoßen wird. Dieser Funktion wird eine Textversion des `IOR` eines weiteren, erbenden Naming Service übergeben. Diese wird innerhalb der Funktion in eine reale Objektreferenz umgesetzt. Anschließend führt der Server damit einen herkömmlichen „Rebind“ seiner Referenz in die Einträge des neuen Naming Service aus und überprüft anhand des Rückgabewerts den korrekten Ablauf des Einbindevorgangs.

Aktionenfolge bei Vererbungen

Der Initiator der Vererbung ist ein herkömmlicher Client, welcher die in Abbildung 7.4 auf Seite 190 schematisch dargestellte Aktionsfolge durchläuft. Bei

der Umsetzung wurde darauf geachtet, dass das Client-Server-Modell streng eingehalten wird und keine von der OMG definierte und in den Implementierungen des Standards umgesetzte Komponente verändert werden muss. Deshalb propagieren Naming Service Dämonen Registrationsinformationen nicht selbstständig.

Im umgesetzten Ablauf verbindet sich der Initiatorclient als erstes mit den beiden Namensdiensten, wobei er vom erbenden nur seine IOR erfragt, um sie dann während des Vererbungsvorgangs an die verschiedenen Server weiterzugeben. Vom vererbenden Namesdienst holt der Client anschließend die Liste der Einträge und aktiviert nacheinander die „Rebind“-Methode der referenzierten Server. Diese tragen sich beim übergebenen, durch die IOR angegebenen Naming Service ein. Nach dem Durchlauf besitzt der neue Auskunftdienst dieselben Eintragungen wie der bisherige, was einer Vererbung der Zugangseigenschaften gleichkommt.

Beliebig erweiterbar

Diese elementare Methode kann beliebig komplex ausgeweitet werden. Wird zum Namensdienst ein HTTP -Interface umgesetzt, welches z.B. HTML -Seiten ausgibt oder via CGI Vererbungsaktionen anstößt, so kann die Sichtweise auf alle möglichen Zugangsarten ausgeweitet werden.

7.4.2 Ausweitung der Inokulation

Inokulationsszenarien für das klassische GPS-Netz

Im Rahmen des GPS -Dienstes können Inokulationsabläufe getestet werden, da hier bereits in den vorangegangenen Kapiteln ein solches Verhalten durchgeführt wird. Als Ausweitung der bisherigen Versuche werden bedingte Übertragungswiederholungen im Skript integriert, welche entweder sofort oder zeitversetzt stattfinden können.

Zeitgetriggerte Steuerung

Die zeitlichen Abfolgen werden, wie bereits bekannt, vorerst über fremde Schedulingprogramme angesteuert. Es ist durchaus denkbar, die dort verwendeten Skripteinträge (z.B. beim „cronjob“, welchen es sowohl für Unix-Derivate als auch für Windows gibt, die „crontab“) bei Fehlersituationen aktiv zu manipulieren, um zeitgetriggerte Fremdstarts auszuführen. Es ist dabei nur dafür zu sorgen, dass es keine Mehrfachaktivierungen gibt oder dass der Client entsprechend ausgelegt ist.

Insgesamt zeigen die Versuche, dass die passive Inokulation weitgehend problemlos von statten geht.

7.5 Zusammenfassung

Harmonische Vereinigung aller Ideen

Im vorangegangenen Kapitel wurde das bisher stufenweise entwickelte Konzept zu einem harmonischen Ganzen vereinigt und auf das Umfeld der Fundamentalstation Wettzell angewandt.

Erster Schritt: Kompetenzbereiche

Erste Stufe dieser Anpassung an reale Gegebenheiten ist das Bilden von logisch zusammenhängenden Bereichen, welche als Kompetenzbereiche deklariert werden. Sie vereinigen alle für einen IT -Bereich (z.B. die jeweiligen Messsysteme) nötige Einrichtungen und bilden damit die Objekte einer Abbildung in ein Managementsystem.

Zweiter Schritt: Logische Zusammenhänge und Beziehungen

Nächster Schritt ist das Erkennen von logischen Zusammenhängen zwischen den Bereichen und damit ihre Abhängigkeiten. Es gibt hierzu zwei Möglichkeiten. Zum einen können sie über den Namensdienst einen logischen Zugangspunkt zu einem echten Dienstanbieter liefern. Zum anderen können sie in Form eines Proxies das Interface vom eigentlichen Diensterbringer spiegeln und alle Anfragen als Zwischenstation weiterleiten. In beiden Fällen bieten sie auch die Schnittstelle an, welche der echte Dienstanbieter besitzt. Sie erben sozusagen das Interface, wobei die Proxy-Variante als eine Art virtueller Zugang gesehen werden kann, weil der Dienstnehmer den Eindruck vermittelt bekommt, als würde der Zwischenverteiler als eigentlicher Diensterbringer fungieren.

Eine weitere Form der Beziehung zwischen den Objekten besteht durch die Verarbeitungsketten. Objekte können in Form eines Fließbands aneinander gereiht sein und gemeinsam stufenweise Mehrwertdienste schaffen, d.h., die Daten zu höherwertigen Informationen verarbeiten. Der einfachste Fall ergibt sich daraus, wenn z.B. ein Messsystem Zusatzinformationen von der Meteorologie benötigt. Etwas komplexere Strukturen ergeben sich aus mehrstufigen Verarbeitungslinien, in denen jeder Abschnitt ein neues Produkt generiert. Diese lassen sich dann am besten mittels Workflow-Beschreibungen allgemein gültig beschreiben. Alles in allem ist dabei aber der elementare Hintergrund das Einbringen von Informationen in einen abhängigen Arbeitsbereich (Inokulation).

Dritter Schritt: Überwinden von Sicherheitsgrenzen

Eine weitere Betrachtung ist darüber hinaus nötig, um Firewall- und damit sichere Objektgrenzen zu überwinden. Hierzu werden die bekannten Mittel der Proxynutzung etc. verwendet.

Einzelne Tests haben gezeigt, dass die Grundgedanken dieser Strukturierung realisierbar sind. Allerdings greift diese nachhaltig in bestehende Gegebenheiten ein, so dass eine komplette Realisierung für die vorliegende Arbeit zu umfangreich gewesen wäre. Eine Umsetzung muss deshalb über einen entsprechend langen Übergangszeitraum geplant werden. In jedem Fall lohnt sich zumindest eine teilweise Realisierung, da dadurch die realen Gegebenheiten besser im virtuellen Abbild durch die IT widergegeben werden können.

Kapitel 8

Überleitung

Schwerpunkt des Kapitels:

Mit diesem letzten Kapitel sollen noch einmal die gesamten Entwicklungen und die dafür getroffenen Entscheidungen rückblickend zusammengefasst werden. Es ist eine abschließende Bewertung, in der auch zeitliche Projektstrukturen einfliessen können. Der Ausblick stellt letztendlich die weiteren Vorgehensweisen nach der Arbeit und damit die Bedeutung der Arbeit als Bewertung für zukünftige Entwicklungen dar. Dieses Kapitel ist somit der Abschluss des Softwareprozesses für das vorliegende Projekt.

8.1 Zusammenfassung

Neue Gedanken zur Datenhaltung auf der FS Wettzell

Die Arbeit hat sich zum Ziel gesetzt, neue Gedanken und Möglichkeiten für die Datenhaltung auf der Fundamentalstation Wettzell zu entwickeln. Ursache dafür waren immer wieder auftretende Problematiken mit heterogenen, technischen Lösungen, welche mit immer mehr steigenden Datenmengen und Datenarten fertig werden müssen.

Geeignete, intuitiv verständliche Vorgehensweisen und Mechanismen

Da sich die Arbeit in die Geodäsie, hauptsächlich aber in die Informatik eingliedert, wurde während der gesamten Laufzeit versucht, verständlich informationstechnische Abläufe und Vorgehensweisen darzulegen. Erste dieser Massnahmen war die Einführung des Softwareprozesses als allgemeine Gliederung der Entwicklungen. Zur besseren Dokumentation und Übersichtlichkeit wurden alle markanten Stellen mittels UML beschrieben. Des Weiteren basieren die Entwicklungen auf der Objektorientierten Programmierung, welche mittels Designregeln als eine Art Hausstandard vordefiniert werden. Alle entstandenen Codesequenzen werden zudem mittels Versionskontrolle verwaltet.

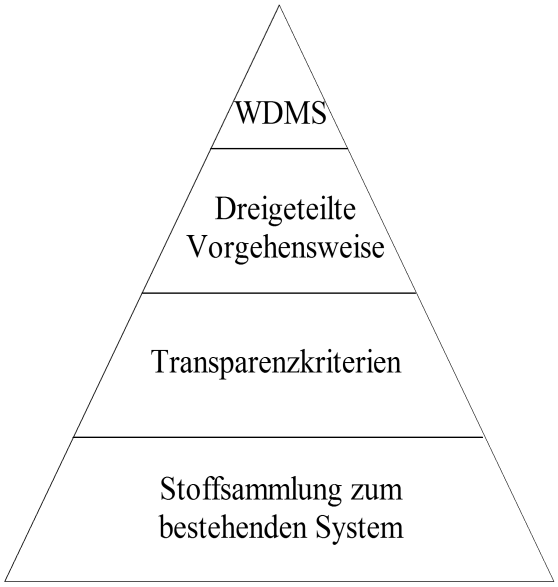


Abbildung 8.1: Abstraktionspyramide der Analyse

Abstraktionspyramide

Die Vorgehensweise während der Analyse kann entsprechend der in Abb. 8.1 auf Seite 194 dargestellten Abstraktionspyramide untergliedert werden. Ausgangspunkt war die Festlegung eines geeigneten begrifflichen Umfelds und die Stoffsammlung zu den realen Gegebenheiten vor Ort in der Fundamentalstation Wettzell. Die Abstraktion aller Informationen mündete in der Erkenntnis, dass ein System erstellt werden muss, welches die Kriterien zur Einhaltung der Transparenz erfüllt. Dieses konnte über drei Stufen erreicht werden, welche die Säulen der Abstraktion bilden.

Erste Säule: Abstraktion der Technik

Die erste Stufe beschäftigte sich mit der Abstraktion der technischen Gegebenheiten. Dazu wurde über zwei Schritte ein Kommunikationsverfahren auf der Basis der Middleware CORBA geschaffen, welches auf Dateisystemen aufsetzt. Diese

Middleware bot sich aufgrund ihrer Flexibilität und Offenheit gegenüber den anderen an.

CFT

In der ersten Annäherung wurde so ein passendes Schnittstellenkonzept in IDL erstellt, welches die bekannten Gegebenheiten unter FTP nachbildet. Die Realisierung wurde mit Hilfe der CORBA -Implementierung TAO umgesetzt und ergibt die mehrschichtige Architektur CFT . Darin ist ein Großteil der in einem FTP -Client bekannten Funktionalität nachgebildet. Ein Versuch zur Erhöhung der Sicherheit konnte über das „Plugable SSL “ eingebracht werden. Zudem wurde auch ein Austausch von Middleware-Implementierungen erfolgreich vollzogen. In einigen ersten Versuchen wurde die Funktionalität des von System- und Technologiespezifika unabhängigen Codes mittels eines Langzeitversuchs nachgewiesen. Testmessungen der Übertragungsleistung ergaben jedoch, dass trotz der logischen Abstrahierung der Heterogenität bei der Laufzeit noch Unterschiede und damit technologische Eigenheiten erkennbar sind.

ECFT

Trotz alledem bietet diese Abstraktion genügend Flexibilität während der Entwicklung, dass in einem zweiten Schritt eine stabile, erweiterte Version des Kommunikationssystems entwickelt werden konnte: das ECFT . Hierbei wurde versucht, eine höhere Dienstgüte als mit anderen Verfahren zu erzielen. Dazu gehört ein verbessertes Softwaredesign, die Handhabung von konkurrierenden Zugriffen, sowie ein geeignetes Ausfall- und Fehlermanagement.

ECFT bietet eine erweiterte Architektur

Die erweiterte Architektur basiert dabei auf einem Bus aus Kommunikationsadaptern, welche sich an gemeinsame Richtlinien halten müssen, jedoch beliebige Übertragungsmechanismen unterstützen können. Der Zugang zum Betriebssystem wird weiter untergliedert und ebenfalls in Form eines Verarbeitungsstacks aufgebaut (ähnlich zum Netzwerkzugang). Höherwertige Dienste bauen auf niederwertigeren auf und abstrahieren das Betriebsystem stufenweise durch Mehrwertdienste.

ECFT bietet Konkurrenzkontrolle

Die Konkurrenzkontrolle sieht ein Sichtenkonzept vor, bei dem ein Nutzer solange eine gewisse Sichtweise auf seine behandelten Daten erhält, solange er einen aktiven Transfer durchführt. Zudem kann er zur Vermeidung von „Lost Updates“ Sperren für Dateien bzw. für ganze Verzeichnisbäume auferlegen. Geschriebene Versionen werden nach Wunsch automatisch archiviert.

ECFT bietet ein Fehler- und Ausfallmanagement

Das Fehler- und Ausfallmanagement sieht ausgefeilte Fehlercodes vor. Zudem wird ein Checkpoint- und Logging-Mechanismus eingebracht, welcher dafür sorgt, dass nach einem Fehler stabile Zustände wieder eingenommen werden. Ein geeignetes Recoveryverfahren mit den nötigen Aufräumaktionen behebt Schäden des schlimmsten aller Fehler, dem Systemcrash. Die Verwaltungsstrukturen werden zudem intern regelmäßigen Validierungen unterzogen, so dass auch Abweichungen im Fehlermanagement erkennbar sind.

ECFT bietet zahlreiche zusätzliche Erweiterungen

Die eingebrachte Autorisierung und Authentifizierung, das aktivierbare Logbuch und der erweiterte Befehlsumfang steigern den Komfort der Software. Versuche zur schnelleren Übertragung im Rahmen einer während der Arbeit stattgefundenen Diplomarbeit brachten leider nicht den gewünschten Erfolg. Trotzdem können sich die Ergebnisse sehen lassen. Der Code ist weiter strukturiert und nur noch zu geringen Teilen von umgebenden Bedingungen abhängig. Die Messungen im Rahmen des GPS -Verbundes zeigten erstmalig die Schwankungen in den WAN

-Übertragungen und teilweise existierende Gegebenheiten vor Ort. ECFT ist zwar nicht unbedingt das schnellste Verfahren, jedoch bezogen auf die Tagesstabilität das stabilste Verfahren. Allerdings gibt es vorerst einige Einschränkungen bzgl. der Überquerung von Sicherheitsgrenzen.

Zweite Säule: Abstraktion der Daten

Eine weitere Säule des Abstraktionsmodells beschäftigt sich mit der Abstraktion der Daten. Hierzu sollten Metadatensätze geschaffen werden, welche allgemeingültig die Daten repräsentieren. Allerdings gibt es so zahlreiche Abhängigkeiten zu externen Stellen und auch zahlreiche individuelle Zusatzinformationen, dass dieser Weg wenig praktikabel erschien. Deshalb ging die Arbeit einen anderen Weg. Sie definiert eine geeignete Beschreibungssprache, um Daten aus beliebigen Quellen und in beliebigen Formaten in beliebige andere zu wandeln. Dadurch wird es möglich, Metadatensätze und Daten zu konvertieren und allgemeingültig für Fremdanwender zu beschreiben.

A2X und XML

Hauptbestandteil ist dabei die Nutzung von XML . Die dort existierenden Transformatoren werden mit einem Präprozessor erweitert, so dass sie die neu definierte Sprache A2X behandeln können. Sie ist eine komplette Sprache, welche über Knoten, Kanten und Speicherelemente einen Strukturplan für strukturierte und semistrukturierte Datenquellen ermöglicht. Ein erstes Ergebnis ist die Umsetzung einer Beschreibung für das Normalpunkte-Format der Laserentfernungsmessung. Erste Versuche bzgl. der Erweiterung des freien Apache Transformators schienen erfolgversprechend.

Dritte Säule: Abstraktion der Systemstruktur

Die dritte Säule für die Vorgehensweise wird durch die Abstraktion der Systemstruktur gegeben. Hierzu wurden in der Arbeit ebenfalls zwei Schritte vorgesehen. Der erste definiert einen allgemeinen Datenzugangspunkt, der dann im zweiten für einen Verbund der Datendienste in der Fundamentalstation eingesetzt wird.

WDAP

Der Datenzugangspunkt WDAP versucht verschiedene Verfahren miteinander zu vereinigen. Dadurch können Schwächen der bestehenden Systeme und konkurrierende Techniken vereint werden. Erste Annäherung ist dabei die Integration von WebDAV , welches auf HTTP basiert und dadurch ohne größere Schwierigkeiten über Firewallgrenzen transferiert werden kann. Intern werden zudem mehrstufige Metainformationen zu den Dateien abgelegt. Die erste Stufe (System) besteht aus den vom Dateisystem gelieferten Informationen. Die nächste baut diese Informationen aus und bildet den Grundstamm der XML -Metadaten zur Speicherung (Storage), welche in einer weiteren Stufe, z.B. durch den Dublin Core, individuell mit Zusatzinformationen zum Inhalt (Content) ergänzt werden können.

WDAP bietet eine breite Schnittstellenbasis

Hauptelement des WDAP ist aber die breite Schnittstelle aus verschiedenen Zugangstechniken, welche über einen ECFT -Koordinator vereint werden. Die eigentlichen Zugriffe erfolgen hierbei durch die jeweilige Zugangssoftware, welche einen Teil der Verwaltung an den Koordinator abgibt. Diese Verwaltungsschnittstelle steht vorerst über eine DLL unter Windows zur Verfügung.

Das resultierende WDMS

In einem letzten Schritt werden nun die Datenzugangspunkte in einen logischen Verbund für die Fundamentalstation Wettzell (WDMS) gesetzt. Zuerst werden hierzu handhabbare, der Realität entsprechende Verwaltungsbereiche festgelegt, welche als Kompetenzbereiche bezeichnet werden. Sie sind teilweise hierar-

chisch angeordnet. Die hierarchische Weitergabe von Zugangsschnittstellen wird dabei in einer Form von Vererbung realisiert, welche entweder nur die Namensraumauflösung bereitstellt, bzw. eine direkte Weitervermittlung (Proxy) ist. Die eigentlichen Arbeitsflüsse laufen in Form von linearen Pipelines, welche zusammengefasst einen Verarbeitungsbaum aufspannen. Diese Inokulation von Information zwischen den Baumebenen und damit hierarchisch abgeleiteten Verwaltungsbereichen kann durch den Sender angeregt (aktiver Empfänger) oder von ihm auch ausgeführt werden (passiver Empfänger).

Überwindung von Sicherheitsgrenzen im WDMS

Über das Proxy-Konzept lassen sich auch Sicherheitszonen überwinden. Es muss nur ein eigener Proxy vor der Firewall angebracht sein, der die Anfragen kontrolliert weiterleitet und an den internen Diensterbringer vermittelt. Auch haben erste Versuche zur Vererbung gezeigen, dass das Konzept für von der Realität abgeleitete Verwaltungsbereiche durchaus Sinn macht. Auch werden momentan aktiv Versuche zur passiven Inokulation im Rahmen einer Neuentwicklung eines `GPS` -Sammlers unternommen.

Insgesamt ist somit eine Art Kompendium über einige neue Wege bei der Datenhaltung in der Fundamentalstation Wettzell entstanden, welches im Laufe der nächsten Zeit in Teilen umgesetzt werden kann.

8.2 Bewertung

Abstrakte Vereinheitlichung bei erhaltener Individualität

Generell kann festgestellt werden, dass die vorliegende Arbeit zahlreiche, neue Ideen entwickelt und damit neue Wege beschreitet. Sie unternimmt den Versuch, die Individualität in weiten Teilen zu erhalten und trotzdem ein gemeinsames Ganzes zu schaffen. Ihre Lösungskonzepte betreffen sowohl elementare Anteile, wie z.B. die Übertragung von Daten an sich, als auch überlagerte, strukturierende Verwaltungsbereiche. Der Lösungsweg ist dabei auf nur drei Stufen reduziert, welche eine schrittweise Abstrahierung vorliegender Gegebenheiten bilden.

ECFT: Stabile Basis mit komfortablen Umsetzungen

Der eigene Übertragungsmechanismus `ECFT` (in der Grundform `CFT`) abstrahiert die vorliegende Technik für die Entwicklung (Hardware- und Betriebsystemanteile). Es wurden zahlreiche, komfortable Zusätze integriert, welche in herkömmlichen Verfahren nicht oder nur teilweise realisiert sind (Sperrmechanismen, Sperren für Heimatverzeichnisse, Fehlerkontrollen, Konkurrenzkontrolle, um nur einige zu nennen). Damit bildet das neue Übertragungssystem eine stabile Basis für weitere Entwicklungen.

Problem: Umständliche Passage von Firewalls

Leider ist auch festzustellen, dass die Passage von Firewalls zwar durchaus möglich ist, sie sich jedoch nicht gerade eleganter Tricks bedienen muss (z.B. Anpassung des `IOR` oder Nutzung von `SOCKS`). Schade ist auch, dass sogar unter gleichen Hardwarebedingungen während der Laufzeit unterschiedliche Qualitätsrichtlinien bzgl. der Übertragungszeiten existieren. Das bedeutet, dass die Abstrahierung nur im Rahmen der Entwicklungssicht gegeben ist. Die hin und wieder entlarvten Programmfehler können hingegen als Anfangsschwierigkeiten abgetan werden, da bis dato keine echten Designfehler aufgezeigt werden konnten.

A2X: Mächtige, in sich geschlossene Vorgehensweise zur Transformation von Datenrepräsentationen

Nächster Schritt ist die Abstrahierung der Daten- und Metadatenformate mittels einer eigenen Beschreibungssprache A2X und XML . Diese Kombination ermöglicht eine einheitliche Beschreibung beliebiger Formate, was nicht nur die Dokumentation unterstützt, sondern direkt maschinenlesbar ist. Sie erlaubt die Abbildung von Strukturregeln einer Grammatik für strukturierte und semi-strukturierte Einsatzgebiete. Zudem bietet das Konzept des einheitlichen Transformators zur Konvertierung von beliebigen Formaten über XML in beliebige andere ein komfortables, in sich geschlossenes Vorgehen. Es bleibt damit innerhalb eines Sprachraums und benötigt keine weiteren Sprachen zusätzlich.

Nachteil: Ungewohnte, umfangreiche Beschreibungen

Obwohl die Beschreibung der Formate anhand von Graphen mit unterstützenden Speicherungsmöglichkeiten relativ mächtig ist, ist die Programmierweise ziemlich ungewohnt. Der Programmierer muss quasi den eigenen Pfad durch sein Programm festlegen. Daraus resultiert auch, dass sich Beschreibungen erheblich aufblähen und zu relativ langen Skripten führen können. Diese sind ohne graphische Hilfsmittel und Werkzeuge nahezu nicht mehr zu lesen. Um dem entgegen zu wirken, muss ein Anwender einen Großteil der XML -Welt mit ihren Ausprägungen im Bezug auf Schemas, Stylesheets etc. beherrschen.

WDAP und WDMS: Ideale Unterstützung realer Gegebenheiten

Letzte Stufe der Entwicklungen ist dann die Planung von abstrakten Zugangspunkten (WDAP) und deren Kombination (WDMS). Diese Vorgehensweise fügt sich ideal in die momentanen Trends zur Bildung weiterer Sicherheitszonen ein, welcher aufgrund interner Virenattacken angeregt ist. Die so entstehenden, handhabbaren Einheiten mit in sich geschlossenen Umgebungen bieten eine höhere Sicherheit in den Einheiten selbst, da Querverstrickungen entweder vermieden oder auf fest definierte Weisen abgewickelt werden. Es vereinigt sozusagen die Vorteile eines verteilten Systems (z.B. Ausfallsicherheit, Flexibilität, Transparenz) mit denen eines zentralistischen (jeweilige Zentren können einheitlich verwaltet werden, bieten einheitliche Zugangspunkte, etc.). Das an den Tag gelegte intuitive Vorgehen mit Schaffung handhabbarer, kompakter Objekte bildet so auch eine leicht verständliche Strukturierung. Diese unterstützt die Eigenheiten der jeweiligen, verantwortlichen Gruppen für ein System, welche selbst über Interna entscheiden kann und bietet doch eine Kombinationsmöglichkeit über eine fest vorgegebene Zugangsschnittstelle zu den von außen als Black Box gesehenen Diensten.

Nachteil: Bisher nur ein rein theoretisches Modell

Teile dieser Überlegungen sind bisher nur in theoretischer Form erstellt und nur durch wenige Experimente untermauert, so dass zu diesem Zeitpunkt noch keine echte qualitative Aussage über reale Erfolge und Probleme getroffen werden kann. Die Realisierung, welche durchaus stufenweise ausgeführt werden kann, betrifft alle Bereiche des IT -Gefüges und ist so als mittel- bis langfristiger Prozess zu sehen.

Vorteile sollten aufgrund der Flexibilität überwiegen

Zusammengefasst hat auch diese Arbeit, wie alle neuen Techniken, gewisse Vor- und Nachteile. Bei geschicktem, gefühlvollem Einsatz der neuen Möglichkeiten sollten sich aber die positiven Trends durchsetzen. Die Stärke liegt gerade in der enormen Flexibilität der Lösungen aufgrund ihrer umfassenden Abstraktion der bisher heterogen gegebenen Anteile. Das Konzept erzwingt keine groben Änderungen von Strukturen, sondern fügt sich ideal in das gegebene Umfeld der Realität ein. Die Realisierung kann hierbei sogar soweit gehen, dass ein Großteil der Änderungen in der Benutzung für die letztendlich gegebenen Endanwender der IT nicht einmal sichtbar sind. Die gewonnene Dienstgüte und das erleichterte Ma-

nagement sollten sich dabei aber mittelfristig auszahlen. Gerade die intuitive Vorgehensweise, die Transparenz und Stabilisierung in allen Ebenen und die verwaltungstechnische Nachbildung natürlicher Gegebenheiten bilden zusammengefasst ein mächtiges und umfangreiches Konzept, welches durchaus erfolgversprechend zu sein scheint.

Erste Erfolge: Verwendung der Programmierregeln als Hausstandard in der FS Wettzell

Besonders erwähnt sollte an dieser Stelle auch werden, dass die zu Beginn erstellten Design- und Programmierregeln in einer erweiterten Form als Standard für die hausinterne Programmierung in der Fundamentalstation Wettzell übernommen wurden. Das ist ein ganz besonderer Erfolg, da die neuen Projekte damit bereits auf dieser Ebene eine homogenisierende Sichtweise vorweisen können.

Verwendung des Logbuch-Mechanismus in anderen Programmen

Gleiches gilt für das beschriebene Logbuch mit den unterschiedlichen Prioritätsstufen. Erweiterte und teilweise leicht abgewandelte Umsetzungen davon werden bereits in verschiedenen, von dieser Arbeit unabhängigen Programmen genutzt (z.B. bei einem neu programmierten, extrem stabilen `NTP` -Server oder bei der Entwicklung eines Eventtimers zur Echtzeiterfassung von Pulsechos in der Laserentfernungsmessung). Ein weiteres Einsatzgebiet wird zudem im Rahmen der Entwicklungen des `SOS-W` entstehen.

Bewertung des Entwicklungsaufwands

In eine Bewertung sollte auch einfließen, mit welchem Aufwand welcher Lösungsschritt erreicht wurde. Aus diesem Grund wurde zu jedem Handlungsabschnitt Protokoll geführt. Dies hat den Vorteil, dass Entscheidungen und ihre Folgen sowie daraus resultierende Entwicklungspfade dokumentiert sind. Zudem ist somit eine Kontrollmöglichkeit für das Zeitmanagement im Projekt gegeben.

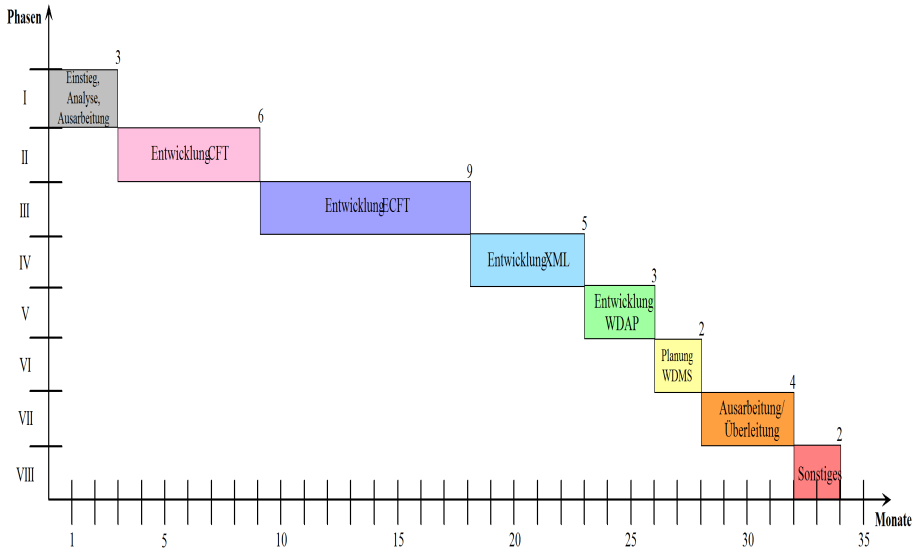


Abbildung 8.2: Verteilung der Entwicklungszeiten auf die Phasen

In der Realität mehr parallele Entwicklungen

Abb. 8.2 auf Seite 199 zeigt den linearisierten Zeitplan der einzelnen Entwicklungsstufen. Der eingesetzte Softwareprozess erlaubt im Grunde eine parallele bzw. feiner granulare Abwicklung der Arbeiten, so dass in der Realität nicht diese Linearität eingehalten wurde. Es soll durch das Diagramm aufgezeigt werden, in

welchen Teilen viel Arbeit steckte. Es ist zu erwähnen, dass Teile der ursprünglichen Planungen in der Ausarbeitungsphase aufgrund geänderter Bedingungen und neuer Kenntnisse nicht ausgeführt, bzw. verkürzt oder anders umgesetzt wurden, wobei trotzdem das Gesamtkonzept beibehalten werden konnte. Dies zeigt die Flexibilität des eingesetzten Softwareprozesses bei trotzdem existierender, zielgerichteter Vorgehensweise.

Problematiken bzgl. der Zeitplanungen u.a. durch Nebeneffekte

Allerdings waren einige Zeitschätzungen als zu gering eingestuft. Die Entwicklung der ersten beiden Abstraktionssäulen war wesentlich umfangreicher. Forderungen blähten sich durch umfangreiche Codesequenzen und Entwicklungsphasen doch erheblich auf. Die Umsetzung wurde zudem auch nur exemplarisch aufgezeigt und noch nicht als ein echtes Produktionssystem realisiert. Die ursprüngliche Zeitplanung wurde teilweise durch unvorhergesehene Arbeiten für projektexterne Aufgaben (Nutzerbetreuung etc.) und administrative Aktivitäten innerhalb der Studie (Erstellen von Präsentationen, etc.) unterbrochen. Trotz alledem wurde der Zeitplan und auch das Gesamtkonzept eingehalten, weshalb das Projekt auch aus dieser Sicht als erfolgreich und „on-time“ angesehen werden kann.

Auswirkungen auf internationale Dienste

Die erbrachten Ideen und Umsetzungen bilden in ihrer Gesamtheit einen Modellversuch innerhalb des begrenzten Umfelds der Fundamentalstation Wettzell. Entsprechend der Verallgemeinerungskette in Abschnitt 1.3 auf Seite 6 lassen sich die Erfahrungen und Lösungen auch auf die weiteren Dienste in der Geodäsie ausweiten, so dass ein einheitliches Gefüge entstehen könnte. Dadurch ließen sich zahlreiche Probleme lösen, welche momentan als generell gegeben eingestuft werden können. Vorerst wird die Umsetzung aber dahingehend stattfinden, dass die internationalen Folgedienste von den Entwicklungen unberührt bleiben. Ein enormer Erfolg wäre es deshalb, wenn in größerem Umfang bemerkbare Qualitätssteigerungen erkennbar wären (z.B. durch eine Verbesserung bzgl. der Fehlzeiten von Datensätzen im klassischen GPS -Netz) und somit die neuen Ideen überzeugen könnten.

8.3 Ausblick

Weiterentwicklung der Programmierregeln und Logbuch-Verfahren

Über diese Arbeit hinaus werden entwickelte und durchdachte Konzepte zumindest in Teilen weiterverfolgt. Hierbei sind, wie bereits im vorhergehenden Abschnitt erwähnt, besonders die entwickelten Programmier- und Designrichtlinien für die Programmierung mit C und C++ zu nennen. Die Nutzung dieser in der Arbeit für die Fundamentalstation erstmalig definierten Regeln bringt ein höheres Maß an Homogenität in den Programmen selbst. In der stufenweise ausgebauten und mit Erfahrungen durch verschiedene Entwickler angereicherten Form wurde das Papier mittlerweile zum akzeptierten, hausinternen Programmierstandard. Dieser wird sich auch in Folgeprojekten weiterentwickeln, verbessern und etablieren. Ähnliches gilt auch für die ebenfalls im vorhergehenden Abschnitt genannten Entwicklungen von Logbuch-Mechanismen.

Erstellung eines GPS-Testsammlers parallel zum Produktionsbetrieb

Ein echtes Folgeprojekt läuft in Zusammenarbeit mit der GPS -Gruppe der Fundamentalstation. Im Rahmen der Umstrukturierung der Permanentstationen zu einem stabilen Linux-System soll auch ein parallel zum bisherigen Produktionssystem arbeitender Testsammler für alle Stationen auf den neuen Möglichkeiten der Übertragung aufbauen. Daraus können dann echte, langfristige Qualitätsmerkmale

erkannt werden. Zu diesem Zweck ist es auch noch erforderlich, den Befehlsumfang in den automatischen Skripten des ECFT -Clients durch bedingte Abläufe und Fehlerreaktionsmöglichkeiten zu erweitern, so dass eine Nachlieferung der Daten automatisch angestoßen werden kann. Zudem kann darüber nachgedacht werden, das Wiederaufsetzen auf eine angefangene Übertragung zu ermöglichen. Diese parallele Entwicklung bietet zum einen eine erhöhte Sicherheit durch Redundanz von klassisch und neu sowie zum anderen einfachere Zugriffs- und Verwaltungsmöglichkeiten für die Betreuer (z.B. die Nutzung von Web-Ordnern mittels WebDAV ohne zusätzliche Zwischenschichten über Firewallgrenzen hinweg)

Erstellung eines Applikationsservers mit Anteilen des beschriebenen WDAP im Rahmen von SOS-W

Diese Realisierung eines Datenservers für GPS ist auch äußerst dienlich für die Erstellung einer Datenhaltung im Rahmen der Entwicklung eines neuen Laserentfernungsmesssystems (SOS-W). In diesem Zusammenhang soll ein eigener Applikationsserver erstellt werden, welcher als Hybridsystem sowohl aus einer Datenbank als auch aus einem Dateisystem besteht. Er soll sämtliche Anfragen sicher bearbeiten und die Anwendungsschicht vom direkten Zugang auf die Daten abstrahieren, so dass zukünftige Umstrukturierungen von Teilbereichen sowohl der Daten, als auch im Bereich der Technik keine Auswirkungen außerhalb dieser festen Applikationsschnittstelle mehr haben. Zudem sollen Verarbeitungsprozesse zustandsbasiert, stabil, wiederanlauffähig und flexibel sein. Somit werden im Inneren dieses Servers Teilaspekte des in dieser Arbeit erstellten Konzepts als Herzstück umgesetzt werden. Das gesamte System läuft dann unter Linux und wird über zahlreiche weitere Zugangsmechanismen (evtl. auch über Query-Sprachen) verfügen.

Stattfindende Strukturierungen im Hinblick auf die Bildung von Enklaven

Eine weitere Folge dieser Arbeit und ergänzender Überlegungen der IT -Gruppe werden fortlaufende Strukturierungsmaßnahmen in der Fundamentalstation Wettzell sein. Das Errichten von Enklaven und in sich abgeschlossenen, nahezu unabhängigen und selbständigen Verwaltungsbereichen wird schon aus Gründen der internen Sicherheit vor allem für die Messsysteme angestrebt. Eine Folge wird somit auch sein, dass diese neuen Objekte einer übergeordneten Verwaltungsstruktur unterzogen werden müssen. Hierzu können Teile der im WDMS durchdachten Lösungsansätze zum Einsatz kommen. Wenn sich diese in der Fundamentalstation bewähren sollten, wäre dies ein wegweisendes Zeichen für alle hierarchischen Verarbeitungssysteme.

8.4 Fazit

Neue Ideen mit kleineren Anfangsschwierigkeiten

Zusammenfassend kann somit festgestellt werden, dass in weiten Teilen neue Ideen für die Datenhaltung der Fundamentalstation Wettzell, aber auch im Hinblick auf beliebige geodätische oder allgemeine Datenumgebungen durchdacht wurden. Die entstanden Prototypen weisen noch einige Anfangsprobleme auf, sind jedoch in ihrem Gesamtkonzept ausbaufähig und flexibel.

Auswirkungen auf die momentane IT-Situation in der FS Wettzell bereits deutlich erkennbar

Die Arbeit hat bereits im momentanen Zustand erste Auswirkungen auf bisherige Gegebenheiten in der Fundamentalstation (z.B. Designregeln, Enklavenbildung). Die angedachten Folgeprojekte bestätigen den Aktualitätsgrad und die in der Arbeit enthaltenen Neuheiten voll und ganz. Ob sich die einen oder anderen Gedankenmodelle wirklich als erfolgreich herausstellen, wird sich in der Anwendung zeigen. Ihre Flexibilität und allgemeine Anpassbarkeit lässt jedenfalls darauf

hoffen. Trotzdem wird eine echte Realisierung zu einem Produktionssystem noch einiges an Zeit und Entwicklungsaufwand verlangen.

Entwicklungsaufgabe der Studie in umfangreicher Form erfüllt

Jedenfalls sind die ersten Schritte hin zu einem verbesserten Datenmanagementsystem getan, so dass die angedachte Aufgabe aus dem Forschungs- und Entwicklungsprogramm der FGS für die Jahren 2001 bis 2005[1] wenn auch nicht in der ursprünglichen Form so in einer noch umfangreicheren und tiefgreifenderen Art und Weise erfüllt wurde. Es liegt nun an den weiteren Entwicklungen in der Fundamentalstation Wettzell, ob sich die Konzepte umsetzen lassen und wie erfolgreich diese Umsetzungen sind.

[1] vgl. [FGS01] a.a.O. S. 59

Kapitel 9

Danksagung

Am Gelingen der vorliegenden Arbeit waren viele Menschen beteiligt, die dafür sorgten, dass meine Tätigkeit in dieser Form stattfinden konnte. Diesen Menschen sei dieser Abschnitt gewidmet, weil ich ihnen zu großem Dank verpflichtet bin.

Hier sei vorweg dem Leiter der Fundamentalstation Wettzell, Herrn Dr.-Ing. Wolfgang Schlüter, Dank ausgesprochen, da ich ohne seine Bemühungen niemals Mitarbeiter der Fundamentalstation Wettzell hätte werden können. Er hat mich darüber hinaus in meinen beruflichen Bestrebungen seit meinen Tätigkeiten als Praktikant der Fundamentalstation Wettzell immer entsprechend gefördert und unterstützt. Durch Diskussionen im Rahmen der Vorbereitung verschiedener Vorträge hat er zudem oft dazu beigetragen, dass ich meinen aus der Informatik gewohnten Sprachschatz noch einmal überdacht und mit über die Informatik hinaus verständlichen Erklärungen angereichert habe.

Ohne die Bereitschaft jedoch, eine Dissertation zu betreuen, wäre die vorliegende Arbeit überhaupt nicht zustande gekommen. Deshalb danke ich vor allem dem ehemaligen Leiter der Forschungseinrichtung Satellitengeodäsie der Technischen Universität München, Herrn Prof. Dr.phil.nat. Markus Rothacher, für seine mir als mein Chef entgegengebrachte Offenheit und Unterstützung. Als er schließlich Ende 2004 zum Geoforschungszentrum (GFZ) Potsdam wechselte, setzte sich Herr Prof. Dr.-Ing. Ulrich Schreiber (damals noch Privatdozent der Forschungseinrichtung) sehr stark dafür ein, die Betreuung zu übernehmen, was mir sehr viel bedeutet, zumal er schon während der schriftlichen Ausarbeitung zusammen mit Herrn Prof. Dr.rer.nat. Manfred Schneider immer wieder beruhigend auf mich eingewirkt hat. Im besonderen Maße ist auch der Lehrstuhlinhaber der Lehr- und Forschungseinheit für Informatik XI „Angewandte Informatik / Kooperative Systeme“, Herr Prof. Dr.rer.nat. Johann Schlichter, zu nennen, der als Dekan der Fakultät für Informatik ohne zu zögern einverstanden war, eine disziplinenübergreifende Dissertation als Zweitbetreuer zu übernehmen, was leider nicht als allgemein üblich vorausgesetzt werden kann. Gerade die Absicherung im Hinblick auf Hochschulbetreuung ist für eine entstehende Arbeit ein enormer Gewinn an Sicherheit und Stabilität. Deshalb danke ich auch Herrn Prof. Dr.-Ing. Reiner Rummel, der als Dekan der Fakultät für Bauingenieur- und Vermessungswesen die nötigen interdisziplinären Kontakte für mich knüpfte.

Besonders erwähnt sei an dieser Stelle vor allen Weiteren auch mein direkter Ansprechpartner vor Ort in der Fundamentalstation, Herr Dr.rer.nat. Reiner Dassing, der als Hauptverantwortlicher die Informationstechnologie der Station be-

treut. Trotz seiner knappen Zeit und der zahlreichen Aufgaben war er stets ohne Einschränkungen bereit, mit mir über meine Ideen zu diskutieren oder mir Tipps und Hilfestellungen zu geben. In diesen Gesprächen wurde der wichtige Bezug zur Praxis geprägt. Gerade diese Gespräche waren oft der nötige Anstoß, um von Zeit zu Zeit auftretende Denkblockaden zu durchbrechen und Problemsituationen zu lösen. Darüber hinaus opferte Dr. Dassing, der mittlerweile ein echter Freund im beruflichen Alltagsstress geworden ist, zusätzlich seine freie Zeit, um als erster Lektor eine erste Durchsicht der schriftlichen Ausarbeitung durchzuführen. Dafür danke ich ihm ganz besonders!

Weiter danke ich natürlich auch allen meinen Kollegen in der Fundamentalstation Wettzell, welche mir immer wohl gesonnen waren und mich in all meinem Tun in jeglicher Form ohne Vorbehalte unterstützt haben. Besonders erwähnt seien hier Herr Dipl.-Ing. Uwe Hessels, welcher mit mir die Testrealisierungen im Rahmen des klassischen GPS-Netzes umgesetzt hat. Des Weiteren sei Herr Dipl.-Ing. (FH) Klaus Röttcher genannt, der die Reinstallation der Permanentstation in Lhasa/Tibet organisiert und zusammen mit mir durchgeführt hat, was zu einem unvergesslichen Erlebnis wurde. Auch die Kollegen Herr Dipl.-Ing. (FH) Karl-Heinz Haufe und Dipl.-Ing. (FH) Günther Herold sind hier zu nennen, welche meinen anfänglichen Vorbehalten gegenüber einer Reise in den Himalaya durch ihre Schilderungen glücklicherweise erfolgreich entgegengewirkt haben. In diesem Zusammenhang ist auch Herr Josef Müller und Herr Dipl.-Ing. (FH) Gerhard Kronschnabl zu erwähnen, die mir vor der Reise bei der Erstellung der seriellen Verbindungskabeln zwischen Messrechner und den GPS-Receivern behilflich waren. Im Zusammenhang mit den Auswertungen der gesammelten Messergebnisse aus den Leistungstests im weltumspannenden Internet waren mir Herr Prof. Schreiber und Herr Dr.rer.nat. Thomas Klügel behilflich, die meine Messwerte mit Hilfe ihrer Programme zur Berechnung von Leistungsspektren prozessierten.

Zuletzt darf ich mich bei den Kollegen bedanken, welche sich regelmäßig beim Mittagessen versammeln und mich dort nicht nur zu ertragen haben, sondern durch ihre angenehme Atmosphäre oft zur Abschwächung von Stresssituationen beitrugen. Dies gilt insbesondere auch für die in den letzten Jahren eingestellten Mitarbeiter, welche sozusagen die „junge, wilde“ Nachwuchsgeneration für die Station bilden. Dazu zählen Herr Dipl.-Ing. (FH) Christian Plötz, Herr Dipl.-Ing. (FH) Matthias Mühlbauer, Herr Dr.-Ing. Pierre Lauber, Herr Michael Wensauer und der mittlerweile nicht mehr in der Fundamentalstation tätige Dipl.-Ing. (FH) Stefan Herbertz. Eine ähnliche, beruhigende Tätigkeit übernahm gerade während der sich hinziehenden Schreibarbeit meine Schwester, Diana Neidhardt, der hiermit auch offiziell Dank ausgesprochen wird. Und zu guter Letzt möchte ich es nicht versäumen, die guten Geister in den Sekretariaten zu nennen, welche für all die nebenbei auftretenden, verwaltungstechnischen Problemchen immer ein offenes Ohr haben. Das ist zum einen in der Fundamentalstation Wettzell, Frau Hannelore Vogl, und zum anderen in der Forschungseinrichtung Satellitengeodäsie, Frau Christiane Horz.

Alle, die bis jetzt noch unerwähnt geblieben sind, wissen selbst am besten, welchen Beitrag sie geleistet haben, so dass meine Dissertation in dieser Form gelingen konnte. Dafür sage ich vielen, herzlichen Dank!

Anhang A

Graphisches Logbuch der Wetterdatenbank

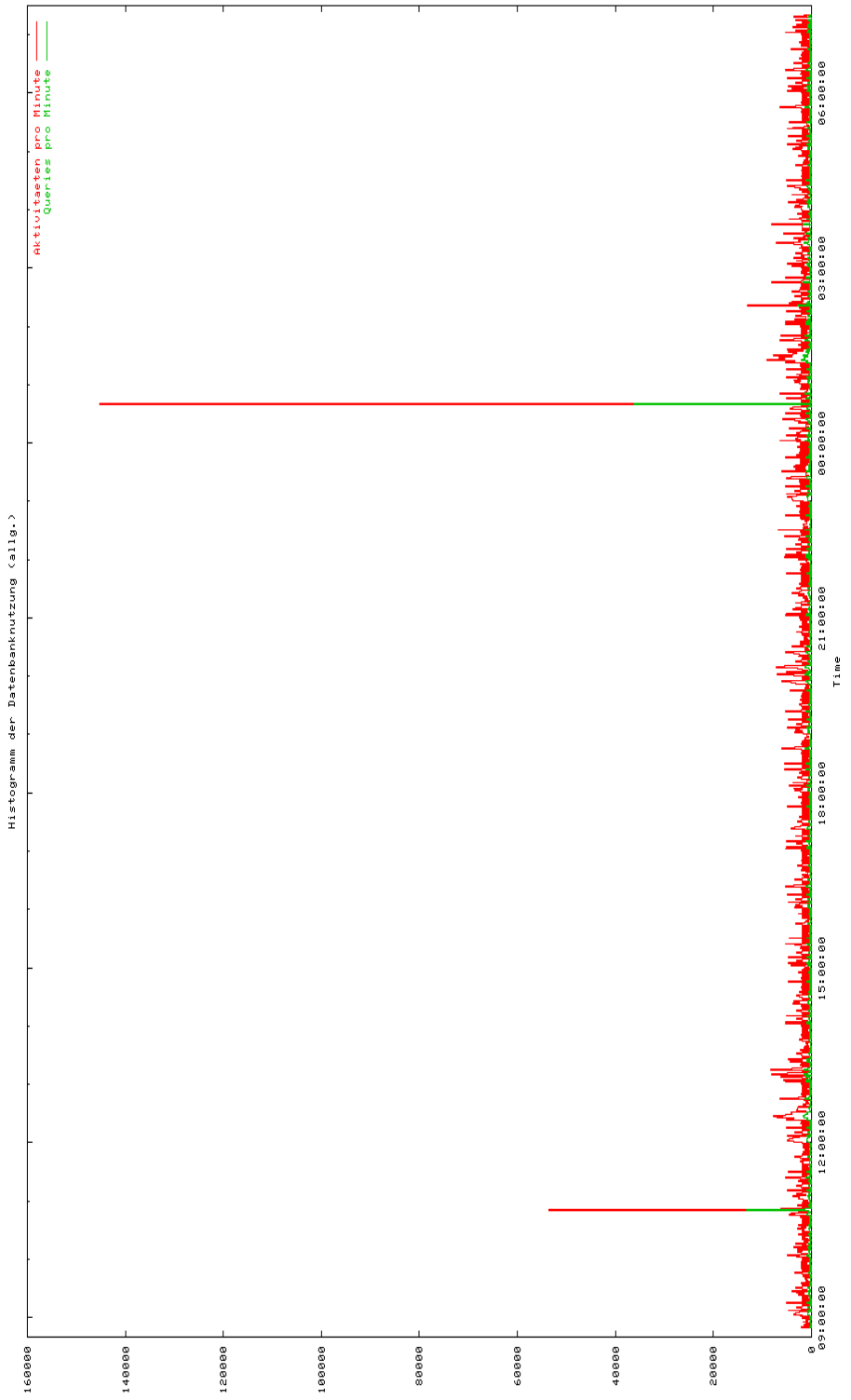
Mit Hilfe eines zu Beginn der Arbeit geschriebenen Perl-Skripts wurden die textuellen Debugging-Ausgaben des in der Produktion in Wettzell eingesetzten DBMS PostgreSQL eingelesen und als graphische Histogramme (hier für den 24.09.2001) im Bezug auf Aktionen je Zeiteinheit dargestellt.

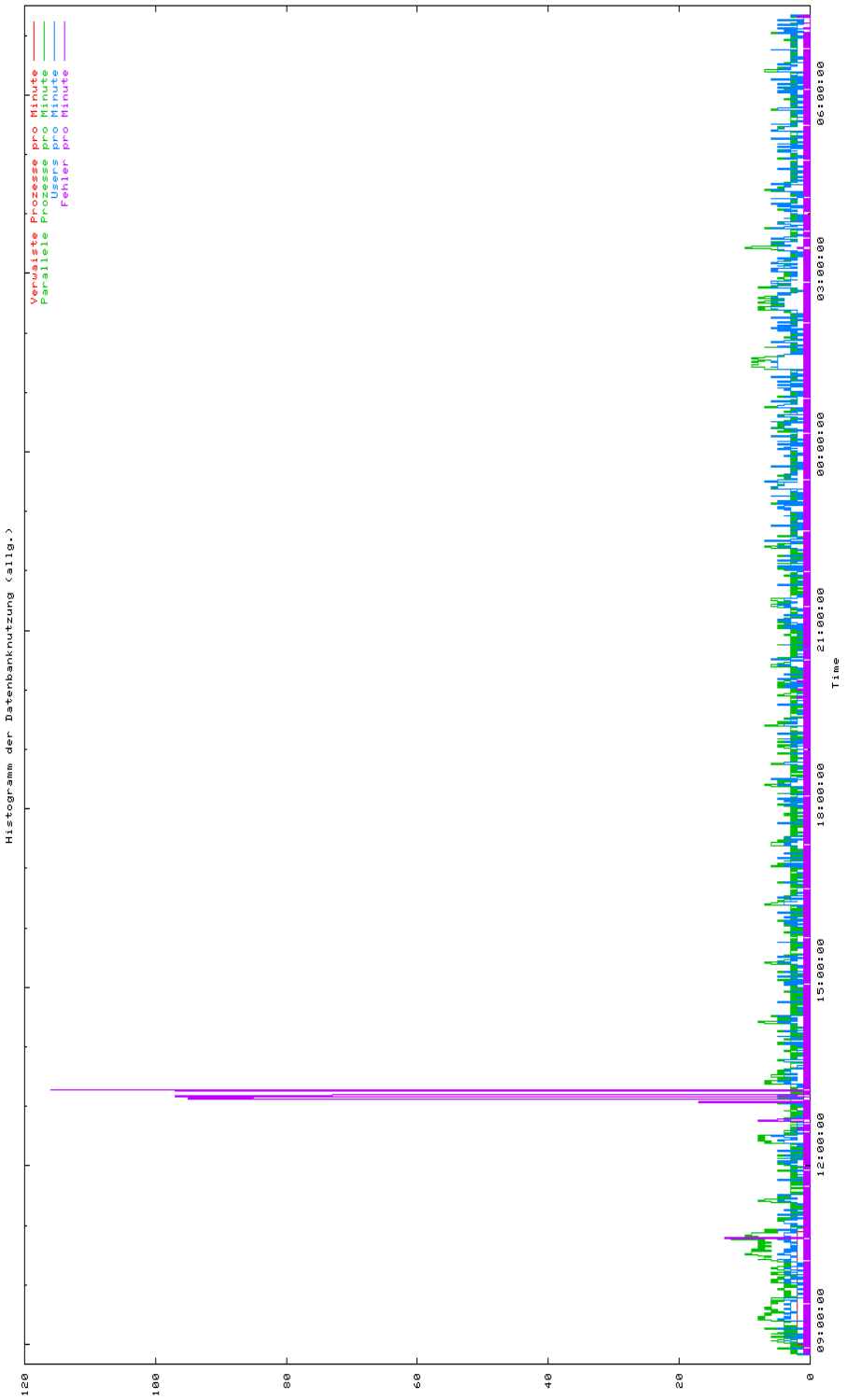
Es stellten sich zwei Erkenntnisse heraus:

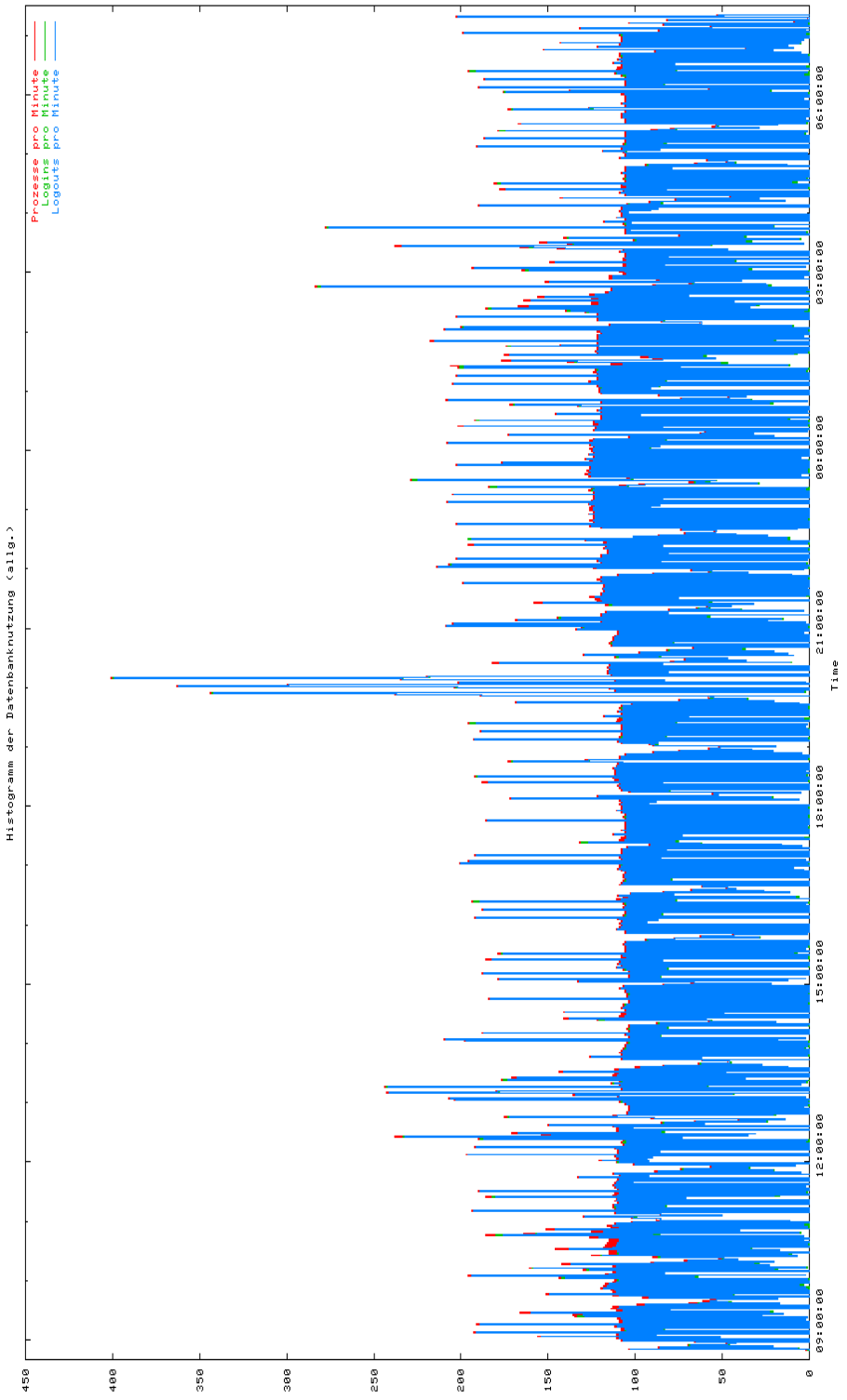
1. Es finden verhältnismäßig wenige Zugriffe statt, so dass diese ohne größere Einschränkungen über ein herkömmliches Dateisystem erledigt werden können.

2. Ein weitgehend unbetreutes, nebenher betriebenes Datenbanksystem wird keine optimalen Arbeitsergebnisse liefern, da der Verwaltungsaufwand für einen spezialisierten Administrator doch erheblich ist. Es zeigt sich, dass es zahlreichen „Wildwuchs“ bzgl. der Nutzerzugriffe gibt (z.B. greifen abfragende Nutzer mit den Rechten des eingerichteten administrativen Nutzers einer speziellen Datenbank zu, z.B. „gpsadmin“ statt „gps“). Zudem wird fast für jede Abfrage eine neue Verbindung auf und wieder zu gemacht, statt eine Kommunikation für die gesamte Dauer einer Abfragefolge offen zu lassen. Und aufretende Fehler gehen mangels Visualisierungsmethode in den Debugging-Ausgaben unter.

Zusammengefasst bedeutet dies, dass reine Datenbanken für das augenblickliche Umfeld in der Fundamentalstation Wettzell nicht ideal geeignet sind.


Histogramm der Datenbanknutzung (allg.)
Aktivitaeten pro Minute
Queries pro Minute
160000
140000
120000
100000
80000
60000
40000
20000
0
09:00:00
12:00:00
15:00:00
18:00:00
21:00:00
00:00:00
03:00:00
06:00:00
Time


Histogramm der Datenbanknutzung (allg.)
Verwaiste Prozesse pro Minute
Parallele Prozesse pro Minute
Users pro Minute
Fehler pro Minute
120
100
80
60
40
20
0
09:00:00
12:00:00
15:00:00
18:00:00
21:00:00
00:00:00
03:00:00
06:00:00
Time


Histogramm der Datenbanknutzung (allg.)
Prozesse pro Minute
Logins pro Minute
Logouts pro Minute
450
400
350
300
250
200
150
100
50
0
09:00:00
12:00:00
15:00:00
18:00:00
21:00:00
00:00:00
03:00:00
06:00:00
Time

Anhang B

Verschiedene Middlewarelösungen

(hauptsächlich entnommen aus [NEHM04])

Nachfolgend wird zu jeder gängigen und für diese Arbeit betrachteten Middleware-Lösung eine kurze Beschreibung ihrer Architekur und Charakteristika gegeben. Die jeweils folgende Bewertung der Nachteile eines Einsatzes in dieser Arbeit diente als Entscheidungshilfe.

B.1 Parallel Virtual Machine (PVM)

Virtuelle Parallelrechner aus PVM-Hosts

PVM [1] wird seit 1989 am Oak Rich National Laboratory entwickelt und hat sich die Aufgabe gestellt, mehr Rechenleistung durch Zusammenschluss von existierenden Netzwerkrechenressourcen zu schaffen. Zielgruppe dieser Middleware sind vor allem die sog. „Grand Challenges“, wie z.B. Wettervorhersagen oder Simulationen. Dabei schließt die frei für Windows- und Unix-Systeme verfügbare Software ein heterogenes Netz aus sog. PVM -Hosts zu einem virtuellen Parallelrechner zusammen.

Prozesskontrolle mit Synchronisation

Zentrale Komponenten sind hier die PVM -Dämonen, welche auf jedem Host als zentrale Verwaltungseinheit unter zugeteilten Nutzerrechten arbeiten. In ihrem Aufgabenbereich liegt neben dem Nachrichten-Routing auch die Authentifizierung, Prozesskontrolle, Fehlererkennung und Überwachung der Tasks. Eine Task ist eine Berechnungseinheit. Sie können netzwerkweit z.B. durch Barrierensynchronisation abgestimmt werden oder auf kritische Abschnitte reagieren.

Die Kommunikation erfolgt zu den vordefinierten Hosts über speziell überwachtes UDP (Dämon-Dämon-Nachrichten) und TCP (Dämon-Task- und Task-Task-Nachrichten). Dabei sorgt die Middleware für die Datentypwandlung zwischen den Systemen. Zum Aufsetzen der Kommunikationsstrukturen dienen Verallgemeinerungsroutinen aus der Benutzerbibliothek.

Gründe gegen einen Einsatz von PVM

- *Spezialisierung auf paralleles Rechnen*
- *Besser für lokale Umgebungen geeignet*
- *Quellen nur für C, C++ und Fortran erhältlich*
- *Datenzugriffsrechte von Nutzerrechten des Dämonen abhängig*
- *Kommunikationspartner müssen bekannt sein und in „hostfile“ eingetragen sein*
- *Fehlende Entwicklungswerkzeuge*

[1] vgl. zum nachfolgenden Abschnitt [NEHM04] a.a.O. S. 1 ff. und [ORN04]

B.2 Message Passing Interface (MPI)

MPI [2] wird seit den Jahren 1992/93 durch das MPI -Forum speziell zum Zwecke der parallelen, numerischen Datenverarbeitung in heterogenen Netzen als standardisierte API entwickelt. Dabei wurden die Ideen von ähnlichen Vorgängersystemen, wie z.B. PVM , erneut umgesetzt und teilweise verbessert.

Parallele, numerische Datenverarbeitung

MPI nutzt ein statisches Prozessmodell zur Verwaltung der in Gruppen organisierten Prozesse, welche untereinander über Kommunikatoren Nachrichten austauschen. Threading ist dabei nicht explizit unterstützt. Zentrale Einheit ist ein Laufzeitsystem ähnlich zu anderen virtuellen Parallelrechnersystemen, welches sich auch um die Datenkonvertierung (Marshalling) kümmert. Als Kommunikation gibt es die Punkt-zu-Punkt-Verbindung zwischen Prozessen und die Gruppenkommunikation von ganzen Prozessgruppen untereinander auf der Basis von Sockets.

Statisches Prozessmodell mit Laufzeitumgebung

Auf diese Weise sind neben den üblichen Funktionen zur dynamischen Prozessverwaltung auch folgende Dienste umgesetzt:

Zusatzdienste wie synch. Dateioperationen

- Remote Memory Access (RMA) : Lesender und schreibender Zugriff auf entfernten Prozessspeicher anderer Prozesse
- Extended Collective Operations: Verteilen und sammeln paralleler Aufgaben über eigene Kommunikationskanäle
- External Interfaces: Schnittstellen zur Objektorientierung
- Parallele Ein-/Ausgabe: Synchronisierte Dateioperationen der verteilten Prozesse untereinander

Deshalb bietet sich MPI auch für das „Grid-Computing“ (= „Gitter-Berechnung“) an, bei dem die Rechenleistung vieler Computer eines Netzwerks zur Berechnung von rechenintensiven Problemen zusammengefasst wird[3].

Gründe gegen einen Einsatz von MPI

- *Spezialisierung auf paralleles Rechnen*
- *Besser für lokale Umgebungen geeignet*
- *Quellen nur für C, C++ und Fortran erhältlich*
- *Kommunikationspartner müssen bekannt sein*

[2] vgl. zum nachfolgenden Abschnitt [NEHM04] a.a.O. S. 13 ff. und [MPI04]
[3] vgl. dazu [LEX04] unter . . . /Grid-Computing.html und [FRI04] a.a.O S. 62

B.3 Agentensysteme

Versenden von Code als mobile Agenten

Unter den Begriff Agentensystem[4] fallen verschiedene Middlewarelösungen, welche auf dem Prinzip mobiler Softwarevertreter basieren. Ausgangsüberlegung ist hier, dass häufig über langsame und teilweise kostenintensive Netzverbindungen große Datenmengen übertragen werden, von denen letztendlich nur Teile wirklich zur Erfüllung einer Aufgabe benötigt werden. Der Ansatz zur Lösung liegt nun darin, dass man statt der Daten den Code für die Lösung eines Auftrags zu den Daten sendet.

Verteilte Aufgabenerfüllung bei schlechten Infrastrukturen

Ein Client erzeugt deshalb einen Agenten seines Auftrags und schickt ihn über das Netz, wo er sich selbstständig über Rechner- und Netzressourcen (= Migration) fortbewegt und die Arbeiten erledigt. Damit sind auch die Vorteile offensichtlich: Agentensysteme sind durch ihre sichere, robuste, zuverlässige und skalierbare Infrastruktur eine echte Alternative zu herkömmlichen Middlewarelösungen für den Bereich der verteilten Aufgabenerfüllung (z.B. verteilte Berechnungen etc.) gerade auch im WAN bei schlechter Netzwerkqualität.

Ausgefeiltes Sicherheitskonzept

Wichtig ist, dass ein ausgefeiltes Sicherheitskonzept zum Einsatz kommt, welches keine unerlaubten Handlungen durch die Fremdprogramme zulässt. Zur Portabilität werden meist virtuelle Maschinen und Interpreter, wie z.B. unter Java, genutzt. Ein weiterer Aufwand ist es, ein gewisses Maß an Fehlertoleranz nach Systemausfällen zu gewährleisten, so dass Agenten selbstständig wiederanlaufen können.

Gründe gegen einen Einsatz von Agentensystemen

- *Ansatz widerspricht dem Transport von Datenmengen*
- *Komplette Infrastruktur muss umgebaut werden*
- *Quellen meist nur für Interpreter, wie Java, erhältlich*
- *Sicherheitsrelevante Aspekte für mobile Agenten widersprechen oft den Sicherheitsrichlinien*
- *Momentan keine einheitliche Entwicklungslinie erkennbar*
- *Ideen stecken im experimentellem Prototypstadium*

[4] vgl. dazu [NEHM04] a.a.O. S. 89 ff.

B.4 Open Database Connectivity (ODBC), Java Database Connectivity (JDBC)

Grundkonzept für ODBC und JDBC [5] ist, allen Datenklienten eine gemeinsame Datenbasis über eine einheitliche Schnittstelle zur Verfügung zu stellen. In Datenbanken werden die Informationen in Informationseinheiten aufgeteilt und in Tabellenform abgelegt. Verfeinerte, schnelle Zugriffsmechanismen, z.B. über Indizes, und durch die Datenbanksysteme realisierte Caches beschleunigen den gezielten Zugriff auf die strukturierten Informationen. Deshalb wurde die reine Datenbanktechnologie eine Zeit lang forciert und vorangetrieben. In weltweiten, heterogenen Systemen mit einer Vielzahl von Datenarten können aber reine Datenbanken nicht generell genutzt werden, da sich nicht alle Datenarten (z.B. Dokumente) abbilden lassen und Dateien als greifbarere Einheiten gehandhabt werden können, als z.B. mengenbasierende Abfragen. Datenbanken dienen dann häufig nur noch der Verwaltung von Zusatzinformationen.

Gemeinsame Datenbasis in strukturierter Form

Generell wurde zur Abfrage die SQL entwickelt, welche in einer eingebetteten Version von verschiedenen Programmiersprachen unterstützt wird. Um die umständliche Vorübersetzung der Einbettung zu vermeiden wurde 1992[6] von Microsoft die ODBC als API zu relationalen und nicht relationalen Datenbanksystmen entwickelt. Die meisten Datenbankhersteller unterstützen mittlerweile ODBC und es gibt zahlreiche Treiber, welche auf Clientseite die SQL -Befehle in datenbankspezifische Protokolle umsetzen. Der Zugang zu den vom sog. Treiber-Manager verwalteten Treibern wird über C++, Fortran oder Visual Basic geschaffen. Die Treiber selbst unterstützen dabei drei Typen: One-Tier- Modell (direkter, lokaler Zugriff z.B. auf indizierte, sequentielle Dateien über `Index Sequential Access Methode (ISAM)`), Two-Tier-Modell (Client kommuniziert mit einem Serverdämon, der die Datenbank anspricht) und Three-Tier-Modell (Client kommuniziert mit einem clientseitigen Server, welcher mit dem eigentlichen Serverdämonen für die Datenbankschnittstelle verbunden ist).

ODBC: Treiber im One-, Two- und Three-Tier-Modell

JDBC wurde auf ähnliche Weise seit 1996[7] von JavaSoft in der rein objektorientierten Sprache Java entwickelt. Die Umsetzung ist vergleichbar mit ODBC , wobei normalerweise ein Client-Server-Modell (vgl. zu Two-Tier-Modell) genutzt wird. Durch die Interpreternutzung wird eine hohe Portierbarkeit geschaffen, welche ohne großen Aufwand auch zur Nutzung in Webbrowser mittels sog. Applets führt. Trotzdem ist JDBC weniger verbreitet, was wohl zur Integration eines „JDBC - ODBC -Bridge“-Treibers geführt hat.

JDBC: Java-Treiber im Two-Tier-Modell, auch mit ODBC-Schnittstelle

Gründe gegen einen Einsatz von ODBC und JDBC

- *ODBC zu Windows-bezogen, auch wenn auf Unix und Macintosh portiert*
- *JDBC zu sprachfixiert auf Java*
- *Schnittstelle ODBC nur in C++, Fortran und Visual Basic, Schnittstelle JDBC nur in Java*
- *In Datenbanken lassen sich nicht alle Daten abbilden (z.B. Heterogene Dokumente)*

[5] vgl. dazu [NEHM04] a.a.O. S. 101 ff.
[6] vgl. [WEB104]
[7] vgl. [WEB204]

- *Zur reinen Datenablage zu kostspielig, besser für häufig veränderliche Daten geeignet*

B.5 Distributed Computing Environment (DCE)

DCE [8] wurde durch die Open Software Foundation (OSF) Anfang der 90er Jahre entwickelt und stellt Dienste zur einfachen Entwicklung und Bedienung von verteilten Anwendungen zur Verfügung. Grundlage ist das Client-/Server- Modell unter Nutzung von RPC . Die Middleware implementiert die Thread- Technologie, so dass sich innerhalb eines Prozesses mehrere Prozessfäden die Aufgaben teilen. Die Programmierung erfolgt über zwei Schritte. Zuerst wird mittels einer IDL die Schnittstelle definiert, aus welcher mit einem IDL -Compiler Programmiercode in C sowohl für Client als auch für Server entsteht. Die eigentliche Funktionalität der einzelnen Komponenten wird parallel in eigenen Programmquellen erstellt, welche anschließend mit Standardcompilern übersetzt werden können.

Einfache Entwicklung und Bedienung für verteilte Anwendungen

Interface Definition Language

DCE organisiert die Rechner in Zellen, welche zur Administration dienen. Server melden sich dabei zur Laufzeit beim Zellen-Verzeichnis-Server an, über den Clients die Zugangsinformationen der entsprechenden Server abfragen. Mit diesen wird die eigentliche Verbindung zum Server hergestellt. Auf diese Weise ist ein Zeit-Service (Verteilung der Zeit auf 10ms genau), ein Verzeichnis-Service (Lokalisierung von Ressourcen durch eindeutige Namen), ein Sicherheits-Service (Authentifizierung und Privilegienvergabe) und ein verteiltes Dateisystem (ähnlich zu NFS) realisiert. Gerade bzgl. des ausgefeilten Sicherheitskonzepts gibt es kaum eine andere Middleware mit ähnlichen Möglichkeiten.

Zellen-Verzeichnis für strukturierte Zellenverbände

Verteiltes Dateisystem ähnlich zu NFS

Gründe gegen einen Einsatz von DCE

- *Quellen sind nur in C geschrieben*
- *UNIX-lastiger Aufbau*
- *Server sind nur für UNIX-Derivate erhältlich, Clients auch für Windows und Macintosh*
- *Offizielle Lizenzen nicht frei erhältlich; letzte freie Version DCE1.2.2 1997*
- *Enormer Verwaltungsaufwand zur Laufendhaltung des Zusammenwirkens von DCE - Zellen*

[8] vgl. zum nachfolgenden Abschnitt [NEHM04] a.a.O. S. 27 ff. und [UZ04]

B.6 Common Object Request Broker Architecture (CORBA)

Ganzheitlicher Objektansatz als Standard

Die objektorientierte Middleware CORBA [9] ist eine seit Anfang der 90er standardisierte Definition der OMG , welche ein Konsortium aus mehr als 800 Mitgliedern ist. Grundlage für alle Definitionen der OMG ist eine einheitliche Terminologie, die Partitionierung des komplexen Problems und die Definition eines Objektmodells, was insgesamt in der Object Management Architecture (OMA) zusammengefasst ist. Dabei wird bei der Verteilung von Objekten besonders auf Orts-, Zugriffs- und Technologietransparenz (bzgl. Plattform und Programmiersprachen) wert gelegt (zu den Transparenzkriterien vgl. Abschnitt 2.4 auf Seite 31).

Verteilungsplattform mit Diensten und darauf aufbauenden Erweiterungen

Zentrale Komponente der Verteilungsplattform ist der sog. ORB , dessen wesentliche Aufgabe das Management der Methodenaufrufe (also die Abbildung auf das zugrundeliegende RPC -Kozept) ist. Für immer wiederkehrende Anforderungen in einer verteilten Anwendung wurden auf den ORB aufbauende Dienste (Namensdienst, Ereignisdienst, Traderdienst, etc.) in der COSS zusammengefasst. Auf diese setzen weitere „Common Facilities“ zur Administration der vernetzten Anwendung und die Frameworks für spezielle Anwendungsfälle auf.

CORBA ist eine klassische Middleware[10] mit Abstraktionsschichten in denen eigene „Repositories“ (Verwaltungsstrukturen mit Konvertierungsinformationen) für entfernte Objekte die einzelnen Vertreter (Client- und Server-Stubs (Skeletons)) verwalten. Die Schnittstellen werden sprachunabhängig in einer eigenen IDL deklariert und mittels eines speziellen IDL -Compilers in den Zielcode übersetzt. Zum Kontaktieren eines entfernten Objekts werden verallgemeinerte IOR genutzt, welche z.B. für das DII über den Namensdienst abgerufen werden können. Die Aufrufsemantik für entfernte Methoden ist „at-most-once“ (RPC) oder „best-effort“ (RPC ohne Antwort) (vgl. zu Aufrufsemantik Abschnitt 3.2.1 auf Seite 40). Auf Seite des Servers verbindet ein transparenter POA den ORB mit der Methodenrealisierung.

Interface Definition Language

Aufrufsemantik „at-most-once“ und „best-effort“

Sprachunabhängigkeit

Die Umsetzung ist sprachunabhängig und findet durch Übersetzung des Codes auf dem Zielsystem statt, indem eine Implementierung der CORBA -Spezifikation von einem beliebigen Hersteller individuell hinzu gebunden wird. Die definierte „InterORBability“ (Interoprable ORB s) sorgt dabei dafür, dass verschiedene Implementierungen, die sich an den Standard halten, miteinander kombinierbar sind. Die Kommunikation findet über ein eigens Protokoll, das IIOP , statt.

Gründe gegen einen Einsatz von CORBA

- *Keine Implementierung, sondern nur Vorgabe eines Standards => Umfang der Implementierung herstellerabhängig*
- *Konsens der OMG-Mitglieder ist meist kleinster, gemeinsamer Nenner und hat extrem lange Reaktionszeiten => hinkt Möglichkeiten hinterher*
- *Technische Notwendigkeiten (z.B. Firewallproblematik) nicht ausführlich genug betrachtet*
- *Entwicklungsschritte des Standards relativ dynamisch*

[9] vgl. dazu [NEHM04] a.a.O. S. 41 ff. und [PUD01] a.a.O. S. 37 ff.
[10] vgl. [PUD01] a.a.O. S. 41

B.7 Component Object Model (COM), Distributed Component Object Model (DCOM), COM+

Verknüpfung von Dokumenten unter Windows

Microsoft entwickelte[11] mit `Object Linking and Exchange (OLE)` in der Version 1 seit 1991 Möglichkeiten, Dokumente in unterschiedlichen Anwendungen komfortabel verknüpfen zu können. Dabei handelte es sich um rein lokale Methoden, welche zum einen die Zwischenablage und zum anderen den umständlichen `Dynamic Data Exchange (DDE)` ersetzen bzw. erweitern sollten. Mit der Erweiterung zu OLE2 konnten diese Einbettungen aus fremden Anwendungen heraus bearbeitet werden. Mit zunehmender Netzwerkanforderung wurden die lokalen Verfahren weiterentwickelt, wodurch im Jahre 1996 ein Komponentenmodell mit Namen `COM` entstand, welches später durch die Erweiterung durch `RPC` zum `Distributed Component Object Model (DCOM)` avancierte.

Interface Definition Language für „Client-Server-Verträge“

`DCOM` basiert dabei auf einem klassischen Client-Server-Modell, welches über eine `IDL` sog. Contracts (Verträge) zwischen Client und Server festlegt, welche somit implizit auch das Datenkonvertieren (Marshalling) definieren. Diese definieren die Art der Schnittstelle, also den Aufbau der Funktionsaufrufe. Die Umsetzung basiert auf den unter Windows üblichen Prozessfäden (Threads). Das zugrunde liegende Schichtenmodell der Komponenten bietet zudem die Möglichkeit, einzelne Softwareteile (Komponenten) vergleichbar einem Hardwarebus über einen sog. Softwarebus zu koppeln und sie einer Versionskontrolle unterzuordnen. Der Softwarebus bietet sogar Berührungspunkte zu `CORBA` . `DCOM` ist damit also ein auf Widerverwendbarkeit ausgelegtes, interface-orientiertes Programmierungsmodell, welches unabhängig von der Sprache ist, aber grundsätzlich für die Microsoft-Umgebung konzipiert wurde. Das ist z.B. auch darin ersichtlich, dass die Registry (Zentralregister) des Betriebssystems Windows eine zentrale Rolle zum Auffinden der `DCOM` -Server spielt.

Wiedrverwendbarer, interface-orientierter, sprachunabhängiger Softwarebus mit Berührungspunkten zu CORBA

Erweiterungen bzgl. Transaktionen, Sicherheit, Synchronisation

Im sog. ActiveX wurde eine abgespeckte Version speziel für Netzwerkanwendungen mit höheren Geschwindigkeiten integriert, in dem zudem Verbindungen zu `DCE` und `IIOP` existieren[12]. Ein weiterer Schritt zum Ausbau der Middleware konnte durch `COM` + geschaffen werden[13]. Dabei wurde der `Microsoft Transaction Server (MTS)` im Bezug auf die Transaktionsdienste, Sicherheitsdienste und Synchronisationsdienste verbessert. Zudem kamen weitere Basisfunktionalitäten hinzu: ein Ereignisdienst (Reaktionen auf definierte Ereignisse), ein Warteschlangenkonzept und Methoden zur Lastverteilung. Trotz alledem bleibt diese Middleware ein proprietärer Firmenstandrad von Microsoft und ist damit leider hauptsächlich in diesen Umgebungen nutzbar (es gibt mittlerweile auch Versionen für andere Betriebssysteme).

Gründe gegen einen Einsatz der `COM` -Familie

- *Proprietärer Microsoft-Standard*
- *`CORBA` bietet allgemeinen Standard von erheblich höherem Umfang*
- *Portierungen in andere Betriebssysteme existieren, sind aber mit erheblichen Lizenzkosten verbunden*

[11] vgl. dazu [NEHM04] a.a.O. S. 59 ff. und [EMS02] a.a.O. S. 6 ff.
[12] vgl. [MINT02]
[13] vgl. [EMS02] a.a.O. S. 45 f.

B.8 Java, Java Beans, Enterprise Java Beans (EJB)

Wenn im Zusammenhang moderner Softwareentwicklung das Wort Java[14] genannt wird, wird in erster Linie an eine Interpretersprache gedacht, welche durch ihre „Applet“-Konzeption insbesondere in heutigen Internetdiensten zum Einsatz kommt. Gerade darin liegt die Möglichkeit zur Nutzung als Middlewareschicht.

Interpreter und virtuelle Maschine zur Heterogenitätsverdeckung

Die Interpretereigenschaften in Verbindung mit Kommunikationsdiensten und einer zugrundeliegenden virtuellen Maschine verdecken die untergeordneten, heterogenen Schichten. Der enorme Vorteil hierbei liegt darin, dass der zu interpretierende Code auf allen Maschinen ohne Vorübersetzung lauffähig ist, welche die virtuelle Maschine zur Verfügung stellen. Bei Middlewarekomponenten auf der Basis von herkömmlichen Sprachen muss jeweils auf der Zielplattform eine Übersetzung stattfinden.

Internationalisierung durch Formatierungsvorgaben

Die Internationalisierung durch entsprechende Formatierungen im Unicode-Format trägt zur weltweiten Nutzung von Java bei, was sich seit der ersten Herausgabe des Java Development Kit (JDK) 1996[15] durch die Firma Sun bestätigt.

Strikte Objektorientierung

Ein weiterer Pluspunkt von Java liegt in der strikten Objektorientierung und den darauf aufbauenden Konzepten. Durch eine Serialisierung von ganzen Objekten ist es möglich, diese persistent abzuspeichern und später komplett mit den zuvor gegebenen Werten wieder zu laden. Die Objektstrukturierung ermöglicht es des weiteren, dass ganze Objekte über das Netz wandern können und als sog. Applets auf fremden Rechnern zu interpretieren sind.

Migration ideal für Agentensysteme

Diese Migration erfordert aber ein aufwändiges Sicherheitskonzept, in dem über Signaturen die Vertrauenswürdigkeit von Codesequenzen gewährleistet ist. Abgebildet wird die Kommunikation auf RPC und sog. Java Sockets auf TCP/IP-Basis.

Servlets werden durch Remote Methode Invocation aktiviert

Bei Java-Middleware aktiviert ein Java-Client auf dem Server ein sog. Servlet, welches als Dienstleister fungiert und über „Servlet Chaining“ weitere Servlets rufen kann. Dieser Kommunikation liegt die Remote Methode Invocation (RMI) zugrunde, welche es einer virtuellen Maschine auf einfache Weise gestattet, über Systemgrenzen hinaus Methoden zu aktivieren. Dabei werden Zugriffsreferenzen verwaltet und ungenutzte Objekte durch einen „Garbage Collector“ gelöscht. Da zudem Schnittstellen zu `CORBA` und auch zu `DCOM` nutzbar sind, kann von einer guten Interoperabilität zu anderen Systemen ausgegangen werden.

Softwarekomponentenmodell

Java Beans ist das auf Java aufbauende Softwarekomponentenmodell für wiederverwendbare Softwareteile, welche mit visuellen Tools gehandhabt werden können. `Enterprise Java Beans (EJB)` sind serverseitige Softwarekomponenten, welche die Kombination von Serverfunktionalität auf Komponentenbasis (d.h. als Schichtenmodell) ermöglichen und damit die eigentliche Middleware darstellen.

Serverkomponenten mit CORBA- Anbindung

Hier ist auch eine CORBA-Kompatibilität von Anfang an integriert worden, welche auf dem Java Transaktion Service beruht. Damit vereinen sich die Implementierungstransparenz von Java mit der Netzwerktransparenz von CORBA und der Komponentenstruktur von Java Beans.

[14] vgl. dazu [NEHM04] a.a.O. S. 71 ff.
[15] vgl. [WEB504]

Gründe gegen einen Einsatz von Java

- *Interpreter sind langsam in der Ausführung großer Quellprogramme*
- *Nur Java als Programmiersprache; Erweitert die meist existierende Anzahl an Sprachen um eine weitere*
- *Sicherheitsrelevante Bedenken können nicht komplett ausgeräumt werden*
- *Abhängigkeit von Entwicklungtools mit verschiedenen Schwerpunkten für EJB* [16]
- *Keine einheitliche Laufzeitumgebung vorhanden*[16]
- *Kosten vergleichbar mit der Einführung eines Datenbanksystems[*[16]*]*

[16] vgl. [MAND00] a.a.O. S. 108 ff.

B.9 Web-Services

Nutzung gängiger Standardtechnologie

In den letzten Jahren kam zu den herkömmlichen Middlewarelösungen ein weiterer Trend hinzu, welcher mittels gängiger Standardtechniken die Vorteile (z.B. Transparenz) von Middleware bietet und dabei die Nachteile (Probleme im WAN) behebt: die Web-Services[17]. Als Web-Services im Allgemeinen bezeichnet man

Netzwerkbasierte, dynamisch gekoppelte Anwendungen

netzwerkbasierte Anwendungen, welche dynamisch untereinander zusammenarbeiten. Dazu ist es nötig eine Beschreibung der eigenen Funktionalität allgemein verständlich zu publizieren, Funktionalitäten zu finden und zu aktivieren, sowie den erforderlichen Datenaustausch zu gewährleisten. Alles in allem sind dies klassische Middlewareanforderungen. Neu ist, dass dies von verschiedenen Firmen (Sun, IBM, Microsoft, etc.) firmenübergreifend mit Standardisierungen und allgemein zugänglichen Techniken realisiert werden soll.

Bei diesen Diensten handelt es sich letztendlich um nachrichtenorientierte Anwendungen, welche architektonisch in drei Gruppen zerfallen: die Anbieter (entwickeln und definieren Dienste), die Anfrager (nutzen Dienste) und die Register (enthalten Verzeichnisinformationen über Dienste und ermöglichen das Auffinden

Eigenes Kommunikationsprotokoll auf HTTP-Basis

von benötigten Diensten). Die Dienste selbst kommunizieren über ein eigenes Protokoll, das `SOAP` . Es setzt auf dem `HTTP` auf, kann damit Firewallgrenzen beliebig überwinden und soll Protokolle wie `IIOP` bei der Nutzung von `RPC` ersetzen. Die

Datenräpresentation in XML

Daten selbst werden mittels der `XML` im ASCII/ANSI-Format codiert und sind allgemein über geeignete Parser (Sprachumsetzer) nutzbar.

Zentrale Dienstverzeichnisse

Die standardisierten Verzeichnisse (`Universal Discovery, Description and Integration (UDDI)`) enthalten in `XML` strukturierte Informationen in Form der `WSDL` . Mit ihr lassen sich implementierungstransparent Funktionen, verwendete Protokolle und genutzte Formate beschreiben. Die `Web-Service Conservation Language (WSCL)` beschreibt erweiternd dazu die Reihenfolge von Nachrichten für einen ordnungsmäßigen Ablauf eines Dienstes. Für weitere Transparenz bzgl. Sprache und Technologie sorgt die Nutzung von Java und der dazugehörigen virtuellen Maschine.

Da jede publizierte Funktion von beliebigen Anfragern genutzt werden kann, sind Sicherheitsbedenken nicht komplett von der Hand zu weisen. Lösungen mit

Authentifizierungszentren für Nutzersicherheit

Identitäten in zentralen Authentifizierungs-Zentren (z.B. Trust-Center von Microsoft) sind wohl eher mit Vorsicht zu betrachten[18]. Des Weiteren handelt es sich um eine relativ neue Technologie, bei der trotz Standardisierung anzumerken ist, dass nur die eingesetzten Teiltechnologien festgelegt sind, nicht aber die Gesamtheit der Web-Services im Allgemeinen. Deshalb ist auch klar, dass verschiedene Firmen eigene Umgebungen anbieten (Java 2 Enterprise Edition von Sun, .NET von Microsoft, WebSphere von IBM). Die Ideen dahinter würden zu zahlreichen neuen Möglichkeiten beitragen, welche aber auch zu falschen Erwartungen im Bezug auf ein Allheilmittel für verteilte Anwendungen führen. Trotzdem zeigt sich in Umfragen eher Zurückhaltung vor allem wegen Sicherheitsbedenken und der fehlenden Vereinheitlichung, wobei versucht wird, den Anschluss nicht zu verpassen[19].

Individuelle Gesamtlösungen auf Basis der Standards

[17] vgl. dazu [ATTACH02] und [WOJ03] a.a.O. S. 5 ff.
[18] vgl. [SIER02]
[19] vgl. Umfrage in [BLE03] a.a.O. S. 15

Gründe gegen einen Einsatz von Web-Services

- *Zu neue Technologie*
- *Fehlende Vereinheitlichung in den Implementierungen*
- *Standards noch in der Entwicklung und damit Änderungen unterworfen*
- *Probleme z.B. bzgl. Sicherheit, Transaktionsintegrität oder Workflow noch nicht eindeutig gelöst*
- *Zusätzliche Sprachen: C# oder Java sind Hauptbestandteile*
- *Lösung hauptsächlich für Endnutzer-Dienste geeignet und weniger zur Automatisierungstechnik*[20]

[20] vgl. [BLE03] a.a.O. S. 18

Anhang C

CORBA-Implementierungen im Vergleich (Mai 2004)

(entnommen aus [PUDPRO04])

CORBA Product Matrix

Abbrev.	Description
Y	Feature is present
N	Feature is not present
P	Feature planned for a future release
?	Not known if feature is present
OS	Open Source ORB
St	Smalltalk
IIOP	Internet Inter-ORB Protocol
SOAP	Simple Object Access Protocol
COM	Common Object Model
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Interface
IFR	Interface Repository
BOA	Basic Object Adapter
POA	Portable Object Adapter
VTS	Value Type Semantics
Min	MinimumCORBA

Abbrev.	Description
AMI	Asynchronous Messaging Interface
CCM	CORBA Component Model
QoS	Quality of Service
FT	Fault tolerance
RT	Real time
PSS	Persistent State Service
CONC	Concurrency Service
PROP	Property Service
EVNT	Event Service
RLSH	Relationship Service
NAM	Naming Service
SEC	Security Service
TIME	Time Service
TRAD	Trading Service
NOTF	Notification Service
TRANS	Transaction Service

Last modified: Saturday, 15-May-2004 18:03:08 PDT *(AP)*

Profile		Language Mappings										Protocols		Core					
	OS	CPP	C	St	Ada	Java	Lisp	COBOL	Python	Tcl	Perl	SOAP	COM	DII	DSI	IFR	BOA	POA	VTS
Borland Enterprise Server Visibroker Ed.	-	Y	-	-	-	Y	-	-	-	-	-	Y	-	Y	Y	Y	Y	Y	Y
BusinessWare	-	Y	-	-	-	Y	-	-	-	-	-	P	Y	Y	Y	Y	P	P	P
DSTC CORBA Services	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
e*ORB C++ Edition	-	Y	Y	?	?	?	?	?	?	?	?	?	?	-	-	-	-	Y	-
e*ORB Java Edition	-	?	?	?	?	Y	?	?	?	?	?	?	?	P	P	-	-	Y	-
Egypt/Taiwan	-	P	Y	-	-	-	-	-	-	-	P	-	-	Y	Y	Y	-	Y	Y
EJCCM	Y	-	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	-	-
Eorb	-	Y	Y	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	Y	-
Gibraltar	Y	P	Y	-	-	-	-	-	-	-	P	-	-	-	-	-	-	Y	Y
Jacorb	Y	-	-	-	-	Y	-	-	-	-	-	-	-	Y	Y	Y	-	Y	Y
JacORB	Y	-	-	-	-	Y	-	-	-	-	-	-	-	Y	Y	Y	-	Y	Y
Java 2 Standard Edition 1.4.1	-	-	-	-	-	Y	-	-	-	-	-	P	-	Y	Y	P	Y	Y	Y
Jonathan	Y	-	-	-	-	Y	-	-	-	-	-	-	-	Y	Y	P	-	P	P
Mexico/Jordan	-	P	Y	-	-	P	-	-	-	-	P	-	-	Y	Y	Y	-	Y	Y
MICO	Y	Y	-	-	-	-	-	-	-	Y	-	-	-	Y	Y	Y	Y	Y	Y
mico/E	Y	-	-	-	-	-	-	-	-	-	-	-	-	Y	Y	Y	Y	Y	Y
Omniorb	Y	Y	-	-	-	-	-	-	Y	-	-	-	-	Y	Y	Y	Y	Y	P
omniORB 3	Y	Y	-	-	-	-	-	-	Y	-	-	?	?	Y	Y	Y	Y	Y	-
omniORB 4.0 preview	Y	Y	-	-	-	-	-	-	Y	-	-	-	-	Y	Y	Y	Y	Y	-
Openfusion TCS (Total CORBA Solution)	Y	Y	P	-	-	Y	-	-	-	-	-	Y	-	Y	Y	Y	Y	Y	Y
OpenORB	Y	-	-	-	-	Y	-	-	-	-	-	-	-	Y	Y	Y	Y	Y	Y
Orb2	-	Y	Y	-	-	Y	-	P	-	-	-	P	Y	Y	Y	Y	Y	Y	Y
Orbacus	-	Y	Y	-	-	Y	-	-	-	-	-	-	-	Y	Y	Y	-	Y	Y
ORBacus 3	-	Y	-	-	-	Y	-	-	-	-	-	-	-	Y	Y	Y	Y	-	-
ORBacus 4	-	Y	-	-	-	Y	-	-	-	-	-	-	-	Y	Y	Y	-	Y	Y
ORBacus/E 1.1	-	Y	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	Y	-
Orbexpress GT for C++	-	Y	-	-	-	-	-	-	-	-	-	-	-	P	P	P	-	Y	P
Orbexpress RT for Ada	-	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	Y	Y	P
Orbexpress RT for C++	-	Y	-	-	-	-	-	-	-	-	-	-	-	P	P	P	-	Y	P
Orbexpress ST for Ada	-	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	Y	Y	P
Orbexpress ST for C++	-	Y	-	-	-	-	-	-	-	-	-	-	-	P	P	P	-	Y	P
Orbit2 for Red Hat Linux 8.0	Y	Y	Y	-	-	-	Y	-	P	-	P	-	-	-	-	-	?	Y	Y
Orbix 2000	-	Y	-	-	-	Y	-	P	-	-	-	Y	Y	Y	Y	Y	-	Y	Y
Orbix E2A Application Server Platform	-	Y	Y	-	-	Y	-	Y	-	-	-	Y	Y	Y	Y	Y	-	Y	Y
Orbix/E 2.0	-	Y	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	-
Orbriver	-	Y	-	-	Y	Y	-	-	-	-	-	-	-	Y	P	Y	-	Y	Y
Orbriver RT	-	Y	-	-	Y	Y	-	-	-	-	-	-	-	Y	P	Y	-	Y	Y
Sankhya Varadhi	-	Y	-	-	-	Y	-	-	-	-	-	Y	-	Y	Y	Y	-	Y	P
SmalltalkBroker	-	-	-	Y	-	-	-	-	-	-	-	-	-	Y	Y	Y	-	-	-
Tao	Y	Y	-	-	-	P	-	-	-	-	-	-	-	Y	Y	Y	-	Y	Y
Tao (The ACE ORB)	Y	Y	-	-	-	-	-	-	-	-	-	-	-	Y	Y	Y	-	Y	Y
The ACE ORB (TAO) 1.1a	Y	Y	-	-	-	-	-	-	-	-	-	-	-	Y	Y	P	-	Y	Y
The Community Openorb	Y	-	-	-	-	Y	-	-	-	-	-	-	-	Y	Y	Y	Y	Y	Y
Tuxedo 8.0	-	Y	-	-	-	Y	-	-	-	-	-	-	Y	Y	Y	Y	-	Y	Y
VBOrb	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	-	Y
VisiBroker	-	Y	-	-	-	Y	-	-	-	-	-	-	-	Y	Y	Y	Y	Y	Y
Visibroker RT (RealTime)	Y	-	-	-	-	-	P	-	?	Y	-	-	Y	Y	Y	Y	Y	Y	

Profile	Core						Services										
	Min	AMI	CCM	QoS	FT	RT	PSS	CONC	PROP	EVNT	RLSH	NAM	SEC	TIME	TRAD	NOTF	TRANS
Borland Enterprise Server Visibroker Ed.	-	-	-	Y	Y	-	-	-	-	Y	-	Y	Y	-	-	Y	Y
BusinessWare	?	Y	?	?	?	?	?	?	?	?	?	Y	?	?	?	?	?
DSTC CORBA Services	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	-
e*ORB C++ Edition	Y	Y	Y	?	Y	Y	Y	-	-	Y	-	Y	Y	Y	-	Y	?
e*ORB Java Edition	Y	-	-	-	P	-	?	?	?	?	?	?	?	?	?	?	?
Egypt/Taiwan	Y	-	-	-	-	Y	Y	Y	Y	Y	-	Y	Y	Y	Y	Y	Y
EJCCM	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Eorb	Y	-	-	-	Y	Y	-	-	-	-	-	Y	-	-	-	Y	-
Gibraltar	Y	-	-	-	-	-	-	-	Y	Y	-	Y	Y	Y	Y	Y	-
Jacorb	-	P	-	-	-	-	-	Y	-	Y	-	Y	P	-	Y	P	Y
JacORB	-	Y	-	-	-	-	-	Y	-	Y	-	Y	P	-	Y	Y	Y
Java 2 Standard Edition 1.4.1	?	?	-	-	-	-	-	-	-	-	-	Y	-	-	-	-	Y
Jonathan	-	Y	P	-	-	-	P	-	-	Y	-	Y	-	-	-	-	P
Mexico/Jordan	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
MICO	Y	P	Y	-	P	-	-	-	Y	Y	Y	Y	Y	Y	Y	P	-
mico/E	-	-	P	-	-	-	P	-	P	Y	P	Y	-	-	Y	P	-
Omniorb	-	P	-	-	-	-	-	-	-	Y	-	Y	-	-	-	Y	-
omniORB 3	-	-	-	-	-	-	-	-	-	Y	-	Y	-	-	-	Y	-
omniORB 4.0 preview	-	P	-	-	-	-	-	-	-	Y	-	Y	-	-	-	Y	-
Openfusion TCS (Total CORBA Solution)	Y	Y	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P
OpenORB	-	-	-	-	P	-	Y	Y	Y	Y	-	Y	-	Y	Y	Y	Y
Orb2	P	P	-	P	P	-	-	-	-	Y	-	Y	Y	-	Y	-	-
Orbacus	-	Y	-	Y	-	-	-	-	-	Y	-	Y	-	-	-	Y	-
ORBacus 3	-	-	-	-	-	-	-	-	Y	Y	-	Y	-	Y	Y	-	-
ORBacus 4	-	P	-	-	-	-	-	-	Y	Y	-	Y	-	Y	Y	Y	Y
ORBacus/E 1.1	-	-	-	-	-	-	-	-	Y	Y	-	Y	-	Y	-	-	-
Orbexpress GT for C++	Y	Y	P	Y	-	-	-	-	-	Y	-	Y	P	P	-	P	-
Orbexpress RT for Ada	Y	Y	-	Y	P	Y	-	-	-	Y	-	Y	P	P	-	-	-
Orbexpress RT for C++	Y	Y	P	Y	P	Y	-	-	-	Y	-	Y	P	P	-	P	-
Orbexpress ST for Ada	Y	Y	-	Y	-	-	-	-	-	Y	-	Y	P	P	-	-	-
Orbexpress ST for C++	Y	Y	P	Y	-	-	-	-	-	Y	-	Y	P	P	-	P	-
Orbit2 for Red Hat Linux 8.0	?	Y	-	-	?	-	-	P	-	P	-	Y	-	-	-	-	P
Orbix 2000	-	Y	-	Y	Y	-	Y	-	-	Y	-	Y	-	-	Y	Y	Y
Orbix E2A Application Server Platform	-	Y	-	Y	Y	-	Y	-	-	Y	-	Y	Y	-	Y	Y	Y
Orbix/E 2.0	-	-	-	-	-	-	-	-	P	P	-	Y	-	-	-	-	-
Orbriver	P	-	P	-	P	-	-	-	-	Y	-	Y	P	P	-	P	-
Orbriver RT	P	P	P	P	P	P	-	-	-	Y	-	Y	P	P	-	P	-
Sankhya Varadhi	Y	P	-	P	Y	P	P	-	-	Y	-	Y	-	-	P	P	-
SmalltalkBroker	-	Y	-	-	-	-	-	Y	-	Y	-	Y	-	-	-	-	Y
Tao	Y	Y	P	Y	P	Y	Y	Y	Y	Y	-	Y	Y	Y	Y	Y	Y
Tao (The ACE ORB)	Y	Y	P	Y	P	Y	P	Y	Y	Y	-	Y	Y	Y	Y	Y	P
The ACE ORB (TAO) 1.1a	Y	Y	P	Y	P	P	-	Y	Y	Y	-	Y	P	Y	Y	Y	-
The Community Openorb	-	P	-	P	P	-	Y	Y	Y	Y	-	Y	-	Y	Y	Y	Y
Tuxedo 8.0	-	Y	-	Y	Y	-	-	Y	-	Y	-	Y	Y	-	-	Y	Y
VBOrb	-	-	-	-	-	-	-	-	-	-	-	Y	-	-	-	-	-
VisiBroker	-	-	Y	Y	Y	-	Y	-	-	Y	-	Y	Y	-	-	Y	Y
Visibroker RT (RealTime)	Y	Y	-	-	-	Y	-	-	?	Y	-	Y	-	-	P	-	-

Anhang D

Schnittstellendefinition in IDL

```
// ************************************************************************
// * CVS Concurrent Versions Control
// * ----------------------------------------------------------------------
// * RCSfile: WDMSInterface.idl,v
// * Revision: 1.1
// * ----------------------------------------------------------------------
// * Author: neidh
// * Date: 2002/05/31 13:26:48
// * Locker:
// * ----------------------------------------------------------------------
// * Log: WDMSInterface.idl,v
// * Revision 1.1  2002/05/31 13:26:48  neidh
// * Fundamental changes:
// * Basical interface-skeleton used for simple interfaces;
// * Using a connector offers advanced abstraction;
// * -> Upgrade of CFT as basis for ECFT
// *
// *
// ************************************************************************

// ************************************************************************
// * Description:                                                         *
// * IDL-definition of the WDMS-interface for automatical codegeneration.*
// * This interface describes the basical methodes of the WDMS-          *
// * Interfaces, which build the basic accesspoint to the service-       *
// * adapters (e.g. FileSystem, RemoteSystem, DatabaseSystem, ...)       *
// * It includes all other IDL-Definitions.                              *
// * ------------------------------------------------------------------- *
// * Used pre-compiler definitions:                                      *
// *                                                                     *
// * Used exception-handles:                                             *
// ************************************************************************

module WDMS
{
    // ********************************************************************
    // * GENERAL DEFINITIONS                                              *
    // ********************************************************************

    // Type of binary raw-data
    typedef sequence<octet> RawDataType;

    // Type of CFT-Infoquestion
    typedef sequence<string> InfoAnswerType;

    // CFT-Transferdescriptorstructure
    struct CFTFILE
    {
        unsigned short usUserID;
        unsigned short usStreamID;
        unsigned long ulTimeStamp;
    };

    // ********************************************************************
    // * INTERFACE INFO                                                   *
    // * ---------------------------------------------------------------- *
    // * Description:                                                     *
    // * Check remote data (e.g. "dir")                                   *
    // * -> the transmitted data are temporarily valid                    *
    // ********************************************************************
    interface CFTInfo
    {
        // --------------------------------------------------------------
        // > Class:     CFTInfo                                        <
        // > Methode:   cppGetInfo                                     <
        // >            Fetches corresponding information without a     <
        // >            intention (only reading access)                <
        // > Parameter: inout CFTFILE SCFTID -> TransferID             <
        // >            in unsigned short usOrderID -> Order-ID        <
        // >            in string cpParameterString -> Orderparameter  <
        // >            out InfoAnswerType CpAnswer -> Strings with ret. <
        // >            out unsigned short uspNumberOfLines -> How many  <
        // >                                                ret.-lines<
        // > Return:    unsigned short -> Errorcode (0=ok, 1=error)    <
```

```
    // > Author:    Alexander Neidhardt                              <
    // > Date:      07.12.2001                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // ---------------------------------------------------------------
    unsigned short usGetInfo (inout CFTFILE SCFTID,
                              in unsigned short usOrderID,
                              in string cpParameterString,
                              out InfoAnswerType CpAnswer,
                              out unsigned short uspNumberOfLines);
};

// ******************************************************************
// * INTERFACE CHANGE                                               *
// * -------------------------------------------------------------- *
// * Description:                                                   *
// * Propagate changes at the visited remote environment (e.g. "cd") *
// * -> reaching of a new environment-view                          *
// ******************************************************************
interface CFTContext
{
    // ---------------------------------------------------------------
    // > Class:     CFTContext                                       <
    // > Methode:   usContextOrder                                   <
    // >            Changes context view at remote server            <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            in unsigned short usOrderID -> Order-ID          <
    // >            in string cpParameterString -> Orderparameter    <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      07.12.2001                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // ---------------------------------------------------------------
    unsigned short usContextOrder (inout CFTFILE SCFTID,
                                   in unsigned short usOrderID,
                                   in string cpParameterString);
};

// ******************************************************************
// * INTERFACE TRANSFER                                             *
// * -------------------------------------------------------------- *
// * Description:                                                   *
// * Real transportation of data (in form of a filetransfer)        *
// * -> Duplication of data permanently                             *
// ******************************************************************
interface CFTTransfer
{
    // ---------------------------------------------------------------
    // > Class:     CFTTransfer                                      <
    // > Methode:   usOpenFile                                       <
    // >            Open a remote filestream                         <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            in string cpFilename -> Which file               <
    // >            in string cpMode -> Protection-mode              <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      07.12.2001                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // ---------------------------------------------------------------
    unsigned short usOpenFile (inout CFTFILE SCFTID,
                               in string cpFileName,
                               in string cpMode);
    // ---------------------------------------------------------------
    // > Class:     CFTTransfer                                      <
    // > Methode:   usCloseFile                                      <
    // >            Close a remote filestream                        <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      07.12.2001                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // ---------------------------------------------------------------
    unsigned short usCloseFile (inout CFTFILE SCFTID);
    // ---------------------------------------------------------------
    // > Class:     CFTTransfer                                      <
    // > Methode:   usGetLines                                       <
    // >            Get remote line/lines                            <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            out string cppLine -> Transmitted line           <
    // >            inout unisgned long ulpLength -> Length of line  <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      07.12.2001                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // ---------------------------------------------------------------
    unsigned short usGetLines (inout CFTFILE SCFTID,
                               out string cppLine,
                               inout unsigned long ulpLength);
    // ---------------------------------------------------------------
    // > Class:     CFTTransfer                                      <
    // > Methode:   usPutLines                                       <
    // >            Put remote line/lines                            <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            in string cpLine -> Transmitte line              <
    // >            in unisgned long ulLength -> Length of line      <
```

```
        // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
        // > Author:    Alexander Neidhardt                              <
        // > Date:      07.12.2001                                       <
        // > Revision:  -                                                <
        // > Info:      -                                                <
        // ---------------------------------------------------------------
        unsigned short usPutLines (inout CFTFILE SCFTID,
                                   in string cpLine,
                                   in unsigned long ulLength);
        // ---------------------------------------------------------------
        // > Class:     CFTTransfer                                      <
        // > Methode:   usGetBlock                                       <
        // >            Get remote block of 8bit-character-streams       <
        // > Parameter: inout CFTFILE SCFTID -> TransferID               <
        // >            out RawDataType cppBlock -> Transmitted block    <
        // >            inout unisgned long ulpLength -> Length of block <
        // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
        // > Author:    Alexander Neidhardt                              <
        // > Date:      22.01.2002                                       <
        // > Revision:  -                                                <
        // > Info:      -                                                <
        // ---------------------------------------------------------------
        unsigned short usGetBlock (inout CFTFILE SCFTID,
                                   out RawDataType CppBlock,
                                   inout unsigned long ulpLength);
        // ---------------------------------------------------------------
        // > Class:     CFTTransfer                                      <
        // > Methode:   usPutBlock                                       <
        // >            Put remote block of 8bit-character-streams       <
        // > Parameter: inout CFTFILE SCFTID -> TransferID               <
        // >            in RawDataType cpBlock -> Transmitte block       <
        // >            in unisgned long ulLength -> Length of block     <
        // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
        // > Author:    Alexander Neidhardt                              <
        // > Date:      22.01.2002                                       <
        // > Revision:  -                                                <
        // > Info:      -                                                <
        // ---------------------------------------------------------------
        unsigned short usPutBlock (inout CFTFILE SCFTID,
                                   in RawDataType CpBlock,
                                   in unsigned long ulLength);
        // ---------------------------------------------------------------
        // > Class:     CFTTransfer                                      <
        // > Methode:   usEOT                                            <
        // >            End Of Transmission (means End Of File)          <
        // > Parameter: inout CFTFILE SCFTID -> TransferID               <
        // > Return:    unsigned short -> Returncode (0=no, 1=yes)       <
        // > Author:    Alexander Neidhardt                              <
        // > Date:      07.12.2001                                       <
        // > Revision:  -                                                <
        // > Info:      -                                                <
        // ---------------------------------------------------------------
        unsigned short usEOT (in CFTFILE SCFTID);
    };

    // ****************************************************************
    // * INTERFACE CFTInterface (inherits CFTInfo, CFTContext,        *
    // *                         CFTTransfer)                         *
    // * ------------------------------------------------------------ *
    // * Description:                                                 *
    // * The real interface between server and client as combination of  *
    // * the three types of interfaces
    // ****************************************************************
    interface CFTInterface : CFTInfo, CFTContext, CFTTransfer
    {
        // ---------------------------------------------------------------
        // > Class:     CFTTransfer                                      <
        // > Methode:   usOpenReg                                        <
        // >            Registrate a remote connection                   <
        // > Parameter: out CFTFILE SCFTID -> TransferID                 <
        // >            in string cpUserName -> Username                 <
        // >            in string cpUserPwd -> Password                  <
        // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
        // > Author:    Alexander Neidhardt                              <
        // > Date:      07.12.2001                                       <
        // > Revision:  -                                                <
        // > Info:      -                                                <
        // ---------------------------------------------------------------
        unsigned short usOpenReg (out CFTFILE SCFTID,
                                  in string cpUserName,
                                  in string cpUserPwd);
        // ---------------------------------------------------------------
        // > Class:     CFTTransfer                                      <
        // > Methode:   usCloseReg                                       <
        // >            Close a registrate for remote connection         <
        // > Parameter: inout CFTFILE SCFTID -> TransferID               <
        // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
        // > Author:    Alexander Neidhardt                              <
        // > Date:      07.12.2001                                       <
        // > Revision:  -                                                <
        // > Info:      -                                                <
        // ---------------------------------------------------------------
        unsigned short usCloseReg (inout CFTFILE SCFTID);

    };

};
```

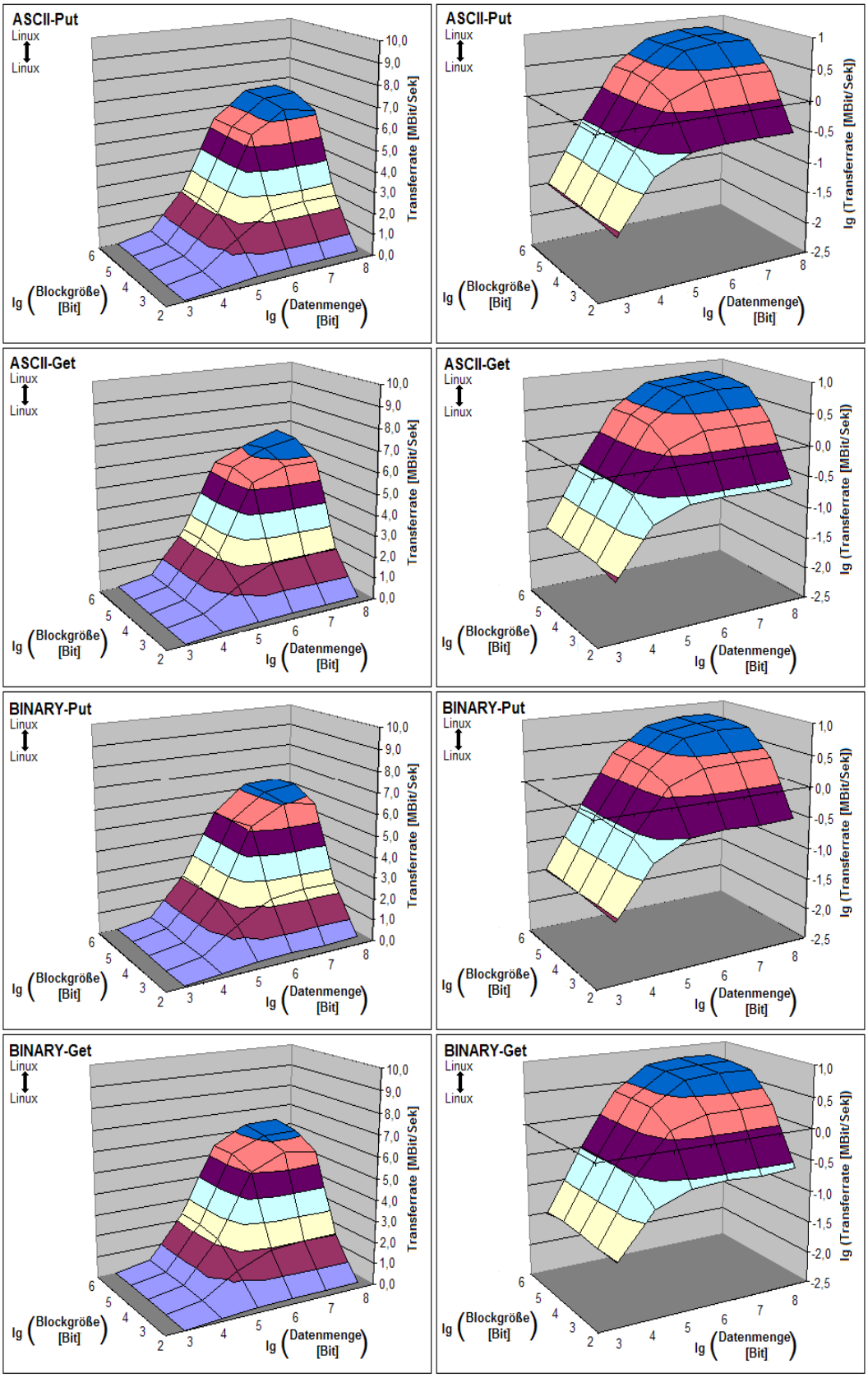
Anhang E

Ergebnisse lokaler Messungen

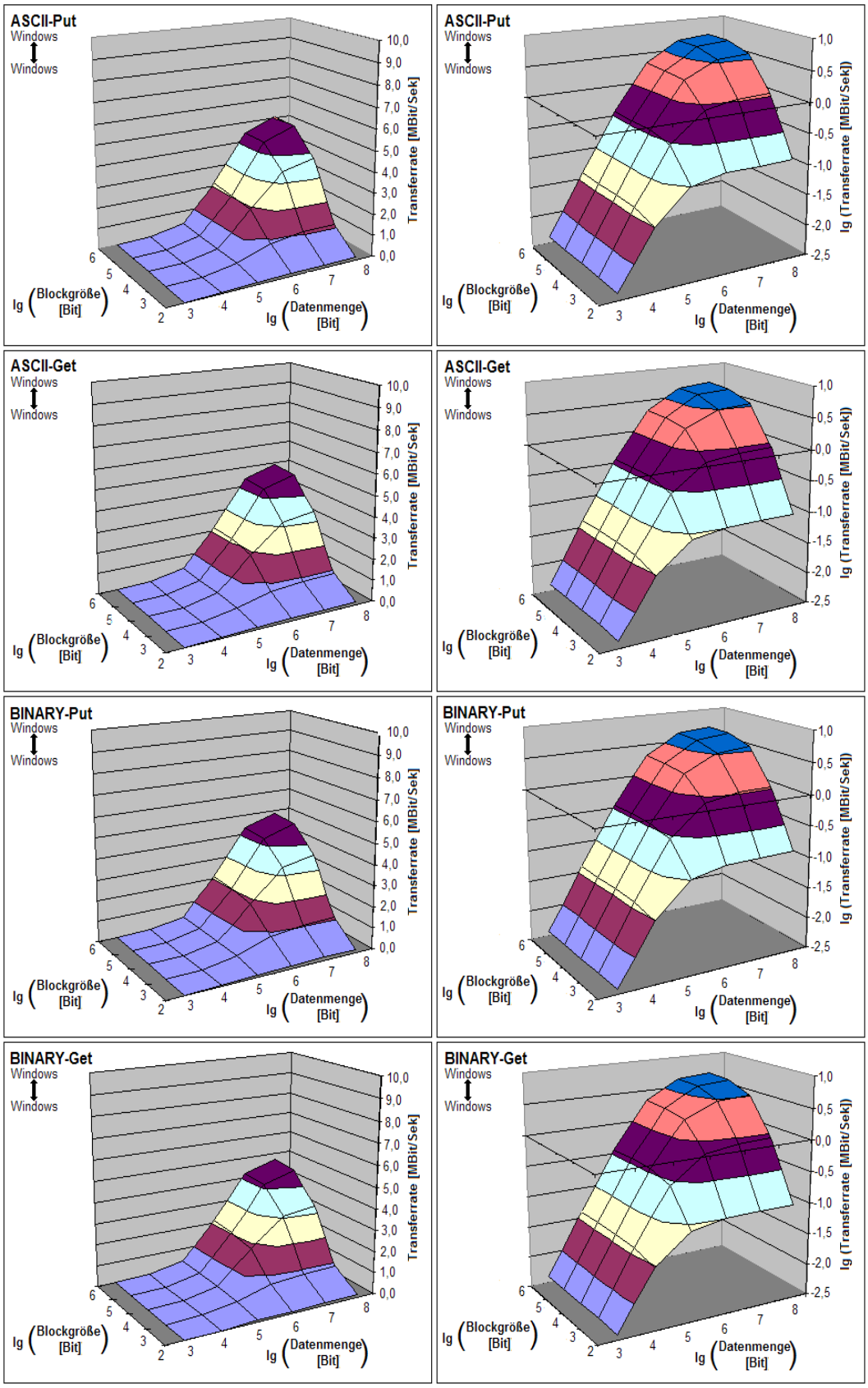
Die Messungen wurden innerhalb des lokalen Netzwerks der Fundamentalstaion Wettzell durchgeführt. Es wurden baugleiche Rechner des Typs Dell OptiPlex GX200 (P3, CU, 933/133, 256K, SKT) verwendet, welche mittels Hub mit dem LAN verbunden waren. Auf beiden Rechnern war sowohl Windows 2000 als auch SuSe Linux 7.2 installiert, so dass ein direkter Vergleich möglich wurde.

Bei den Versuchen zu CFT / ECFT wurde schrittweise für Dateien mit exponentiell zunehmender Dateigröße die Blockgröse der Übertragung erhöht. Ausgewertet wurden Transferraten und mittlerer Fehler der Messung. Bei den Versuchen mit FTP wurden stufenweise Dateien mit exponentiell zunehmender Dateigröße übertragen und ähnliche Auswertungen durchgeführt. Am Schluss konnten die Messungen zueinander in Relation gebracht werden.

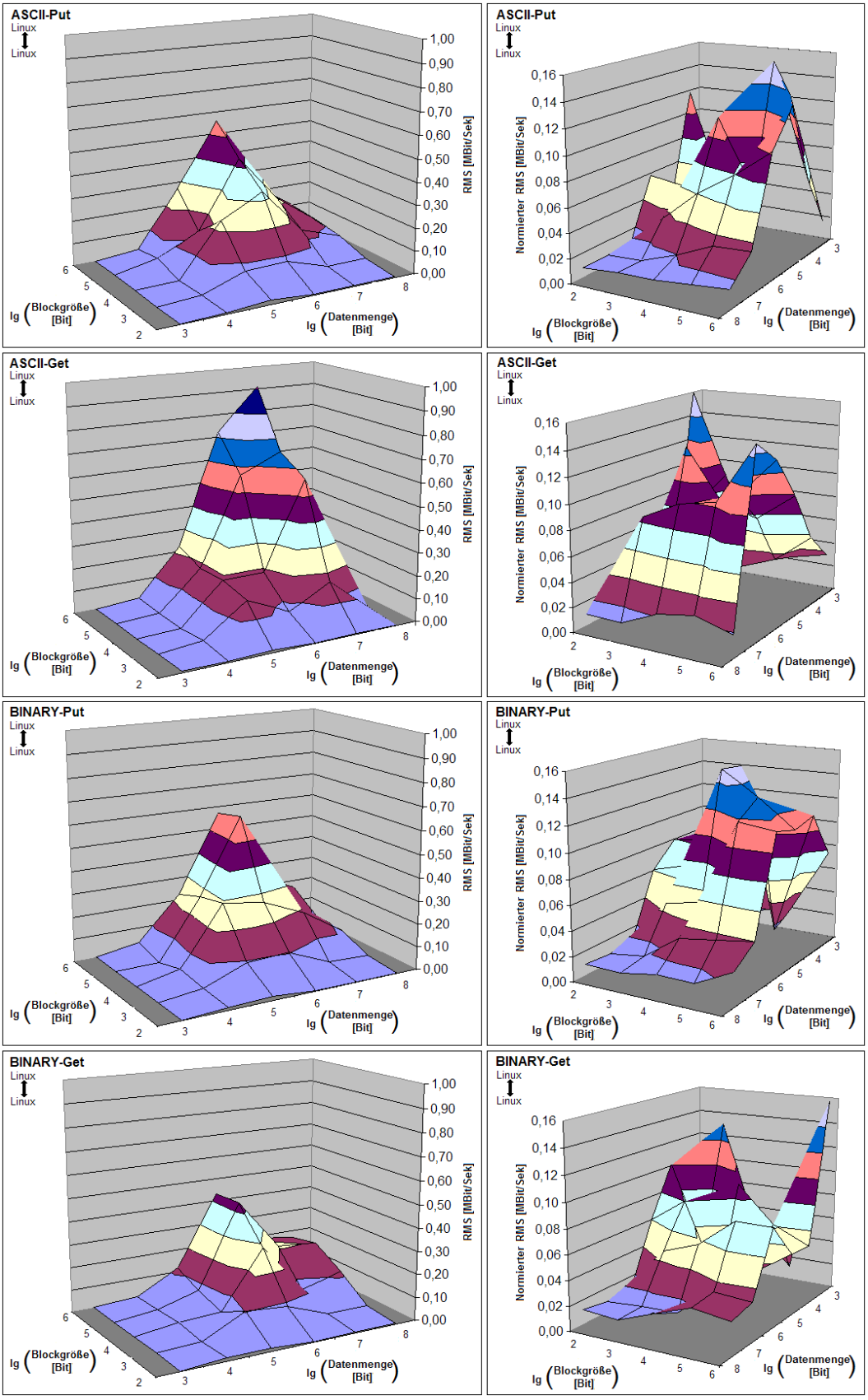
CFT unter SuSe Linux 7.2



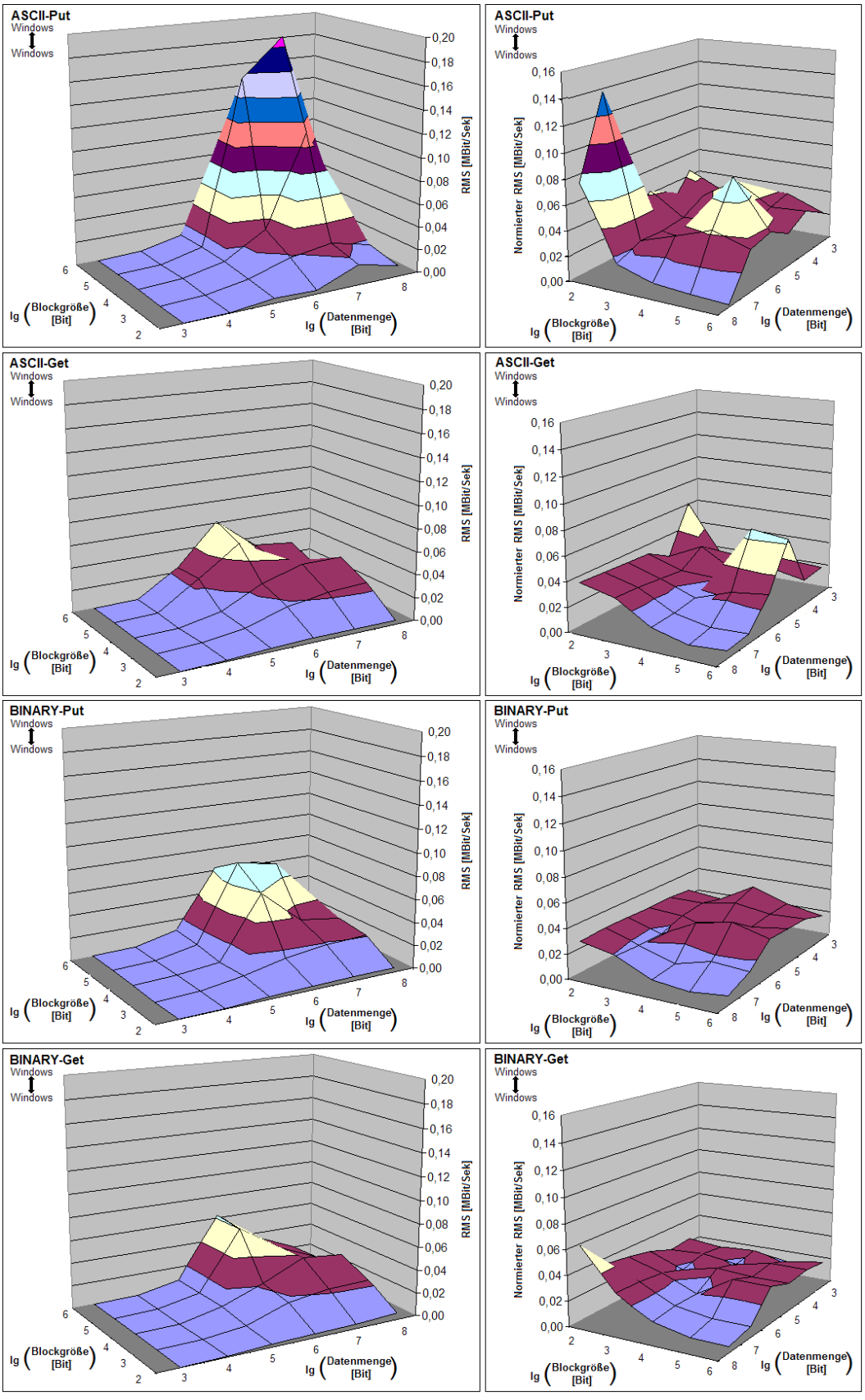
CFT unter Windows 2000

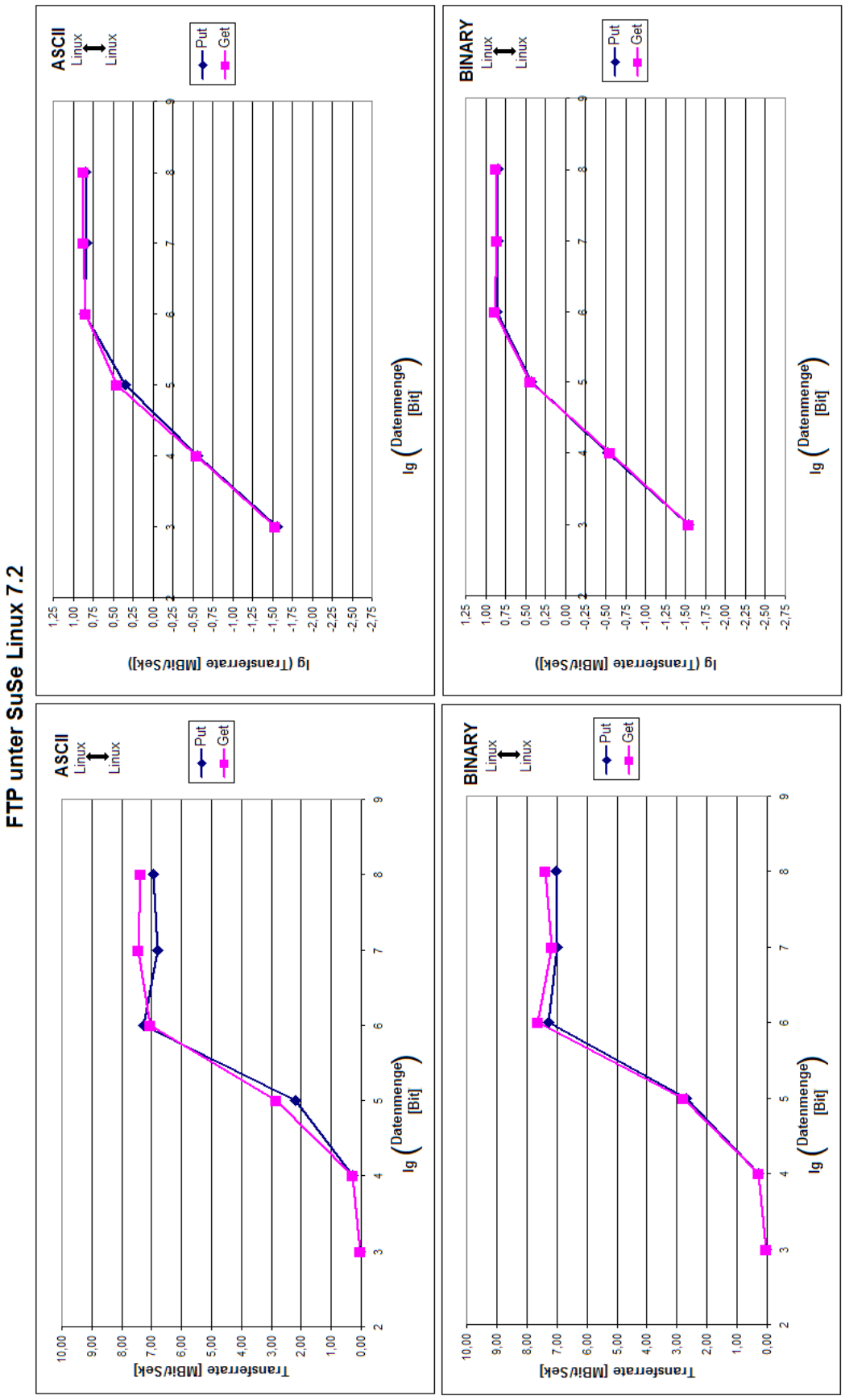


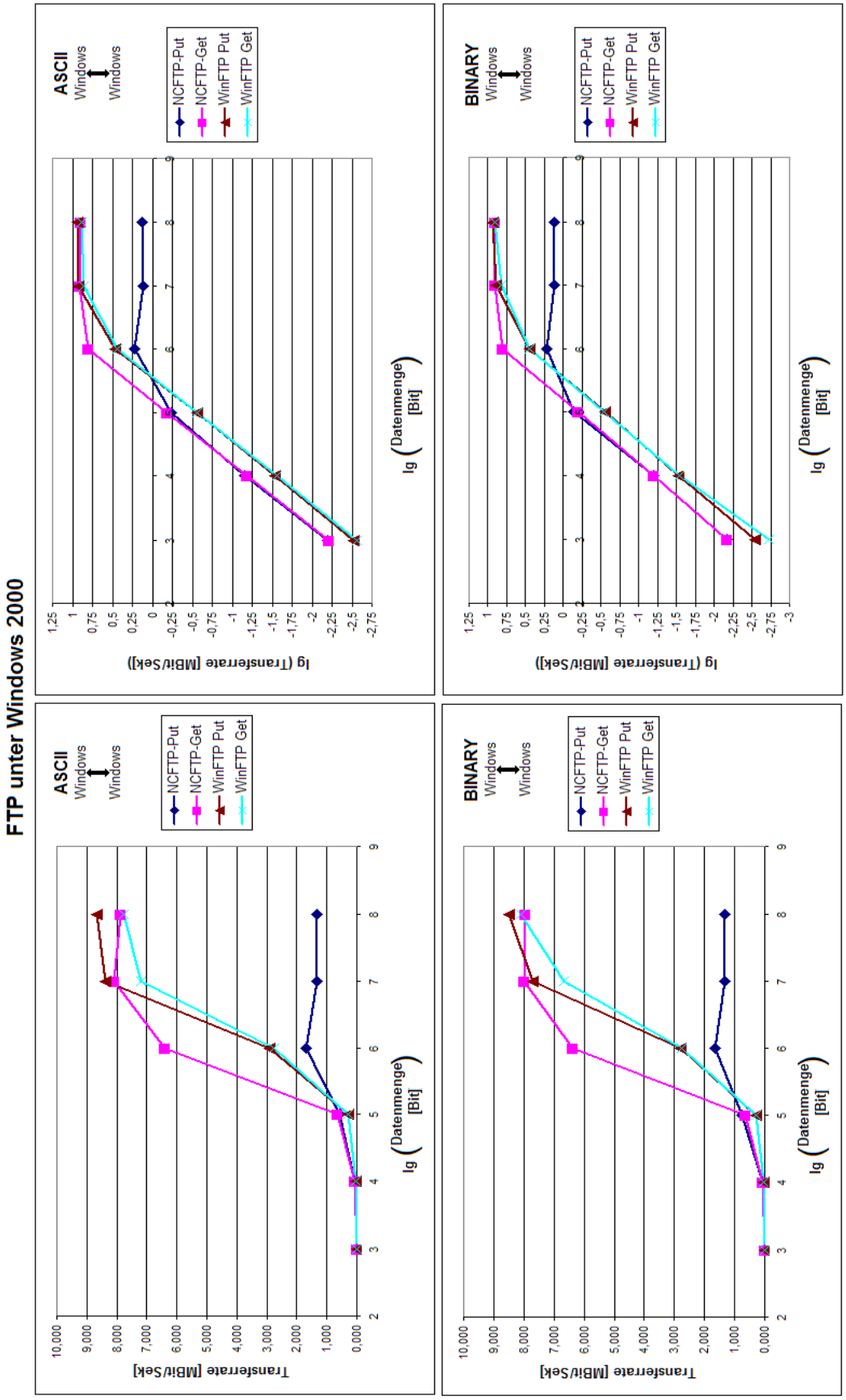
CFT unter SuSe Linux 7.2



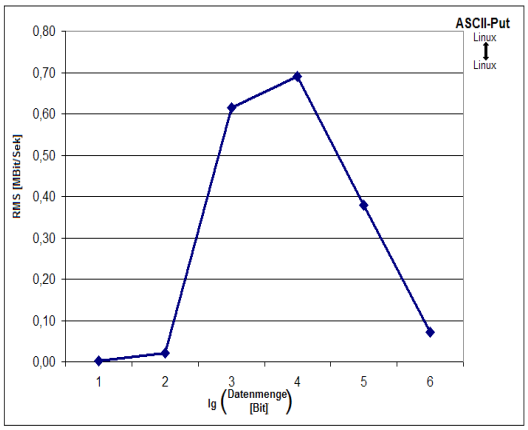
CFT unter Windows 2000

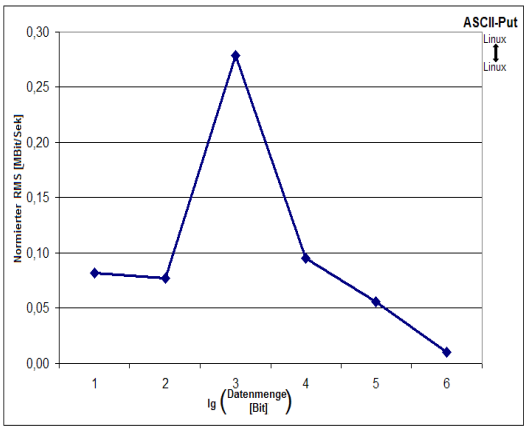


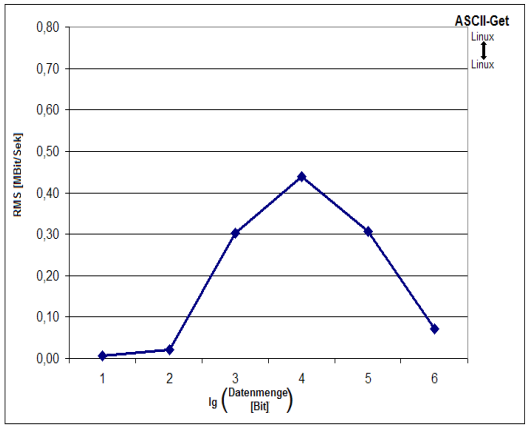

FTP unter SuSe Linux 7.2
ASCII
Linux
Linux
Put
Get
lg (Transferrate [MBit/Sek])
lg (Datenmenge [Bit])
BINARY
Linux
Linux
Put
Get
lg (Transferrate [MBit/Sek])
lg (Datenmenge [Bit])
ASCII
Linux
Linux
Put
Get
Transferrate [MBit/Sek]
lg (Datenmenge [Bit])
BINARY
Linux
Linux
Put
Get
Transferrate [MBit/Sek]
lg (Datenmenge [Bit])

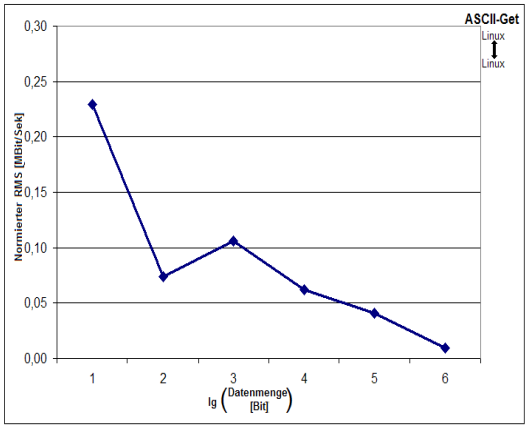

FTP unter Windows 2000
ASCII
Windows
Windows
NCFTP-Put
NCFTP-Get
WinFTP Put
WinFTP Get
Transferrate [MBit/Sek]
lg (Datenmenge [Bit])
BINARY
Windows
Windows
NCFTP-Put
NCFTP-Get
WinFTP Put
WinFTP Get
Transferrate [MBit/Sek]
lg (Datenmenge [Bit])
ASCII
Windows
Windows
NCFTP-Put
NCFTP-Get
WinFTP Put
WinFTP Get
lg (Transferrate [MBit/Sek])
lg (Datenmenge [Bit])
BINARY
Windows
Windows
NCFTP-Put
NCFTP-Get
WinFTP Put
WinFTP Get
lg (Transferrate [MBit/Sek])
lg (Datenmenge [Bit])

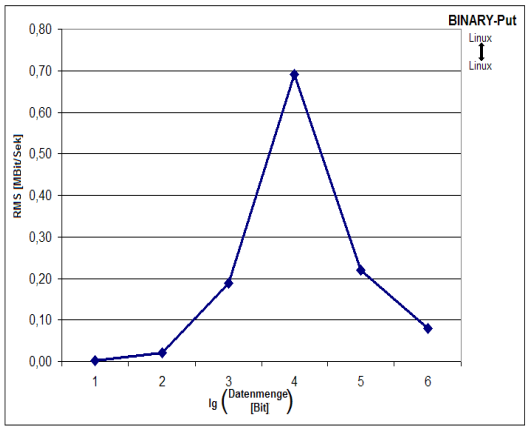
FTP unter SuSe Linux 7.2

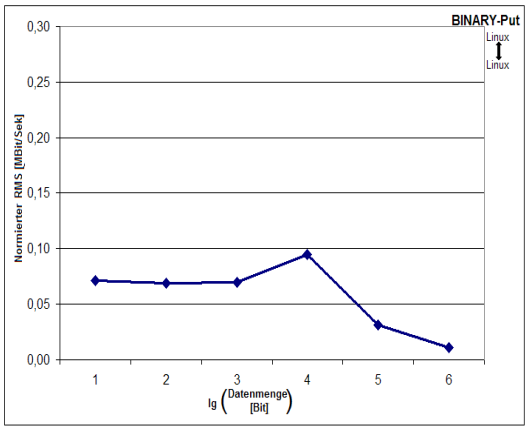


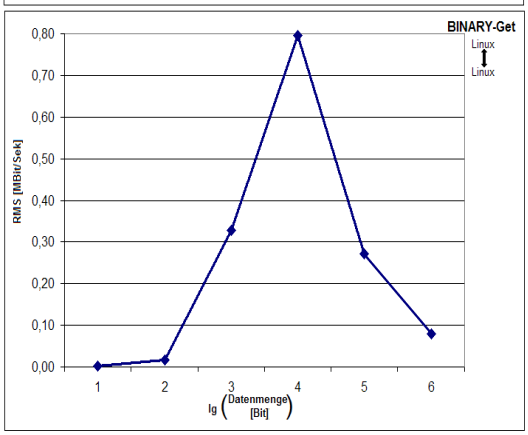


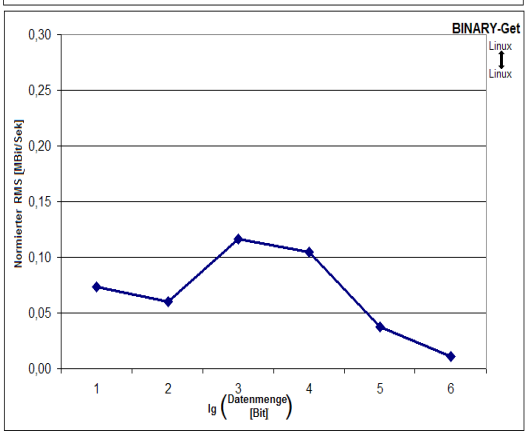




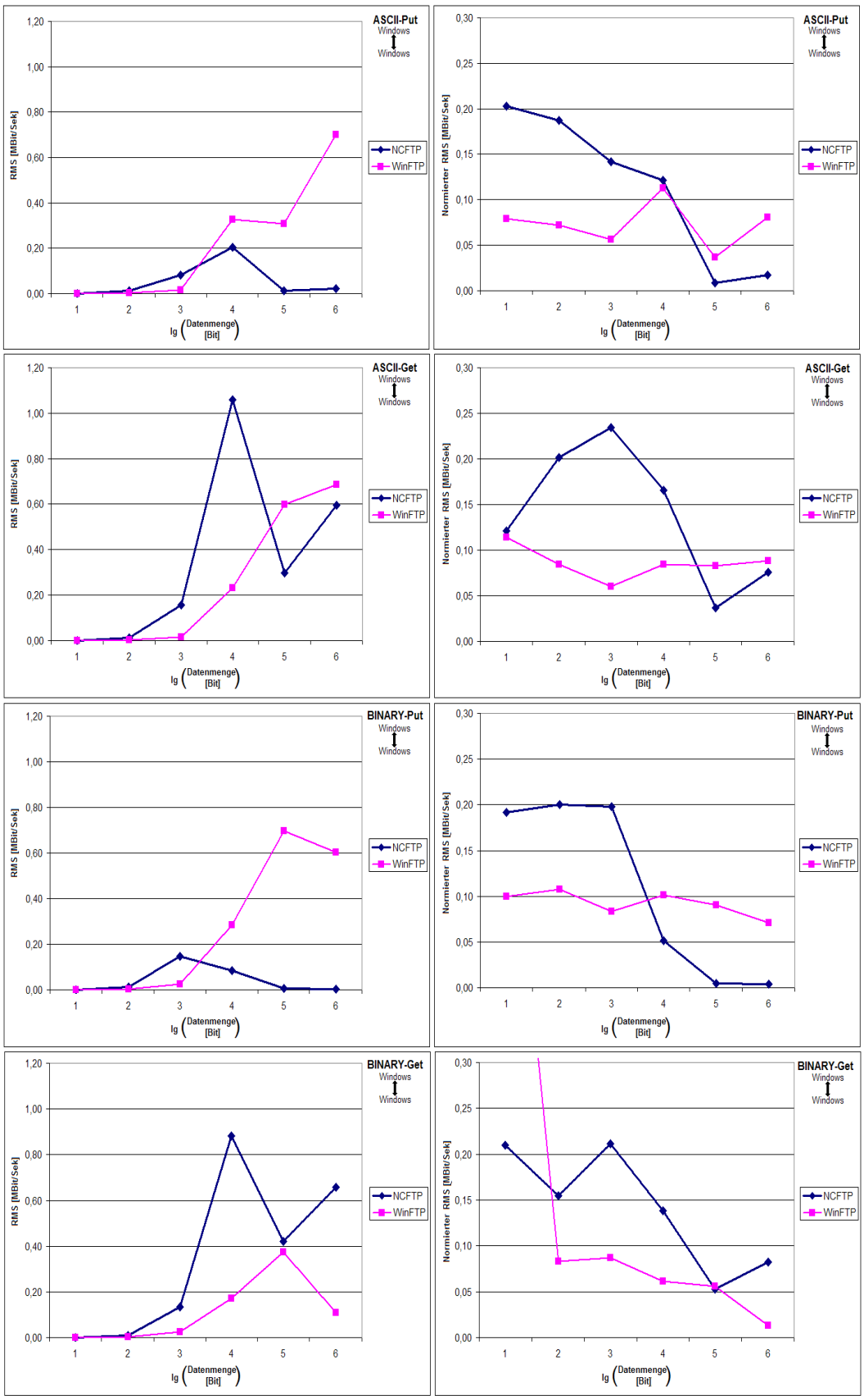




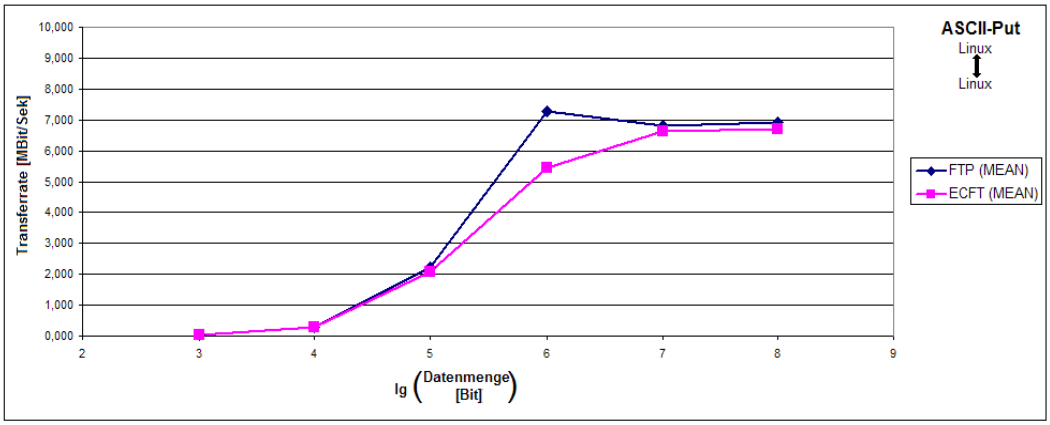


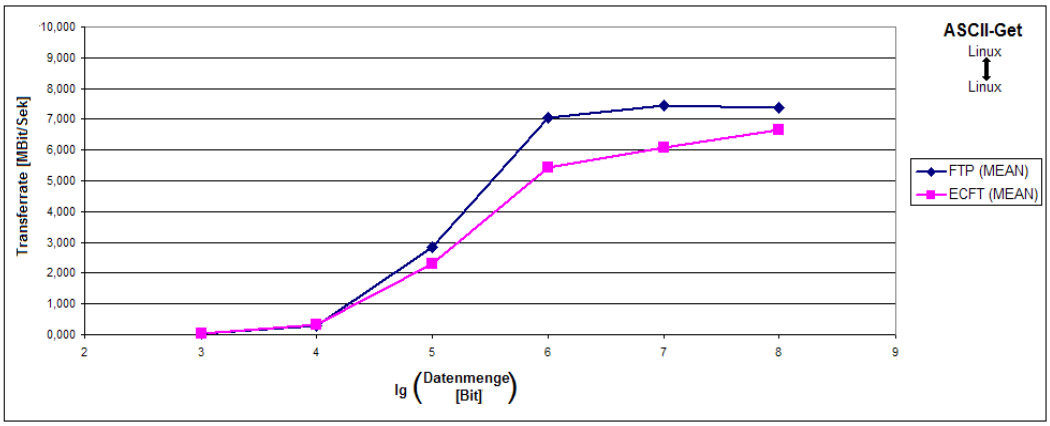


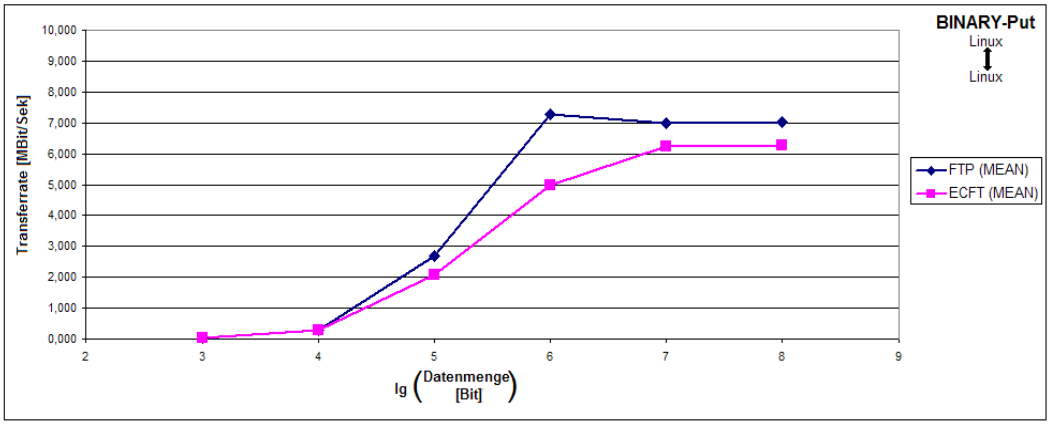
FTP unter Windows 2000

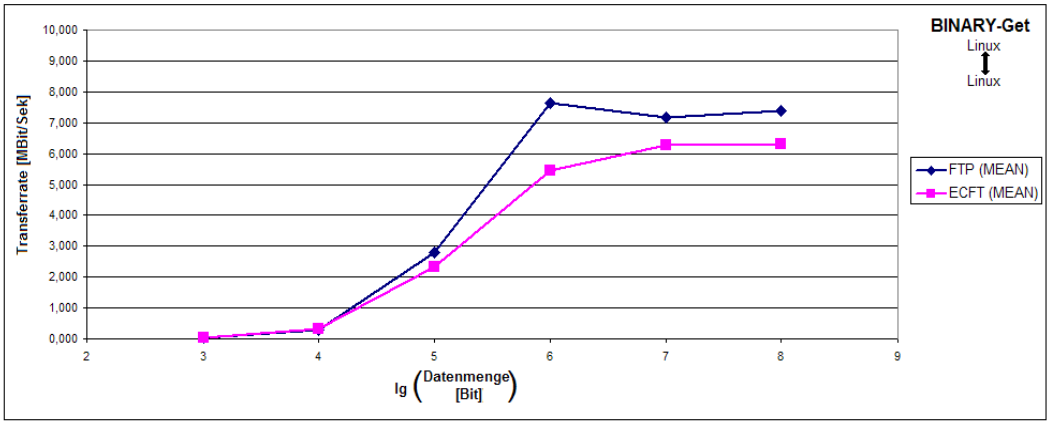


Vergleich von FTP und CFT unter SuSe Linux 7.2

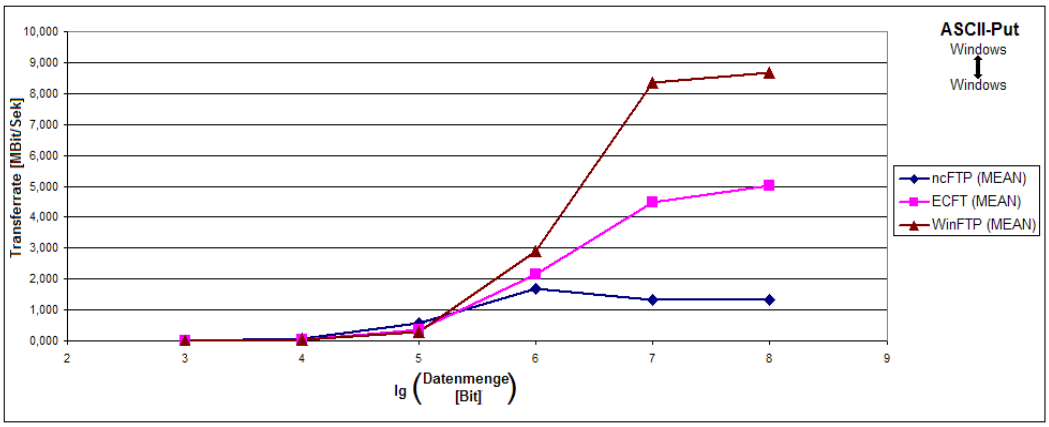


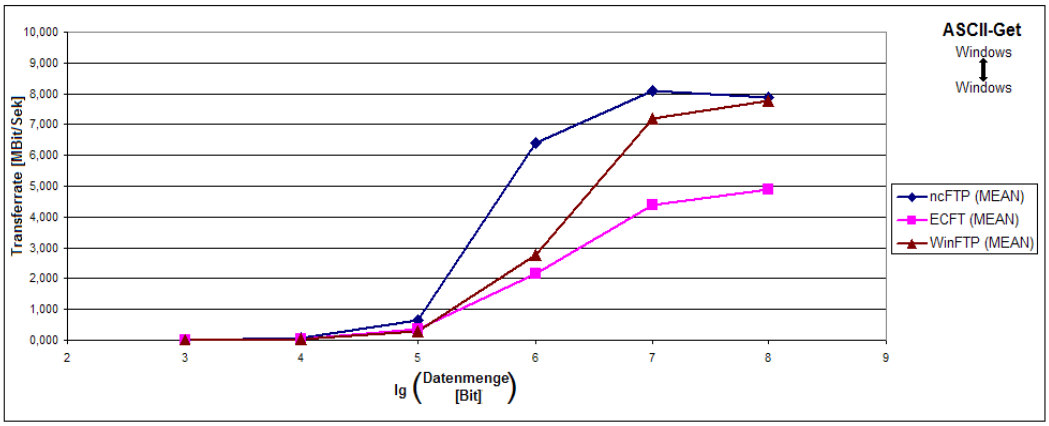


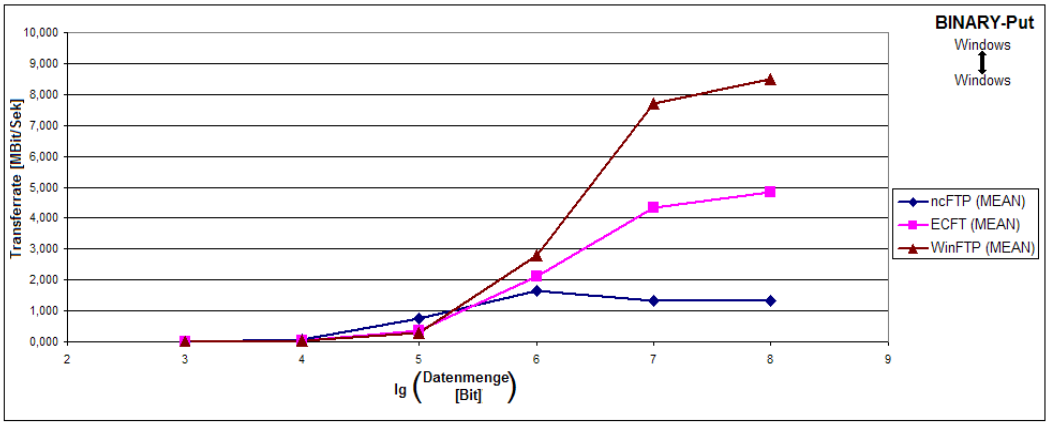


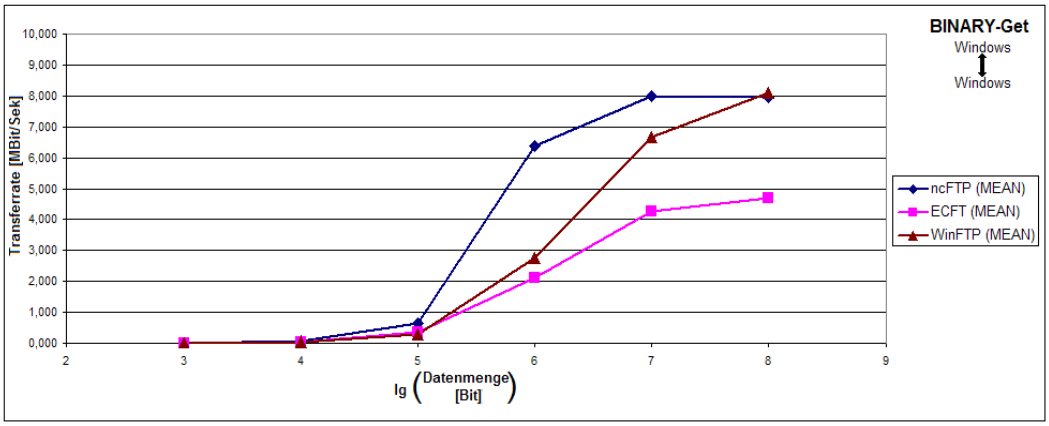


Vergleich von FTP und CFT unter Windows 2000





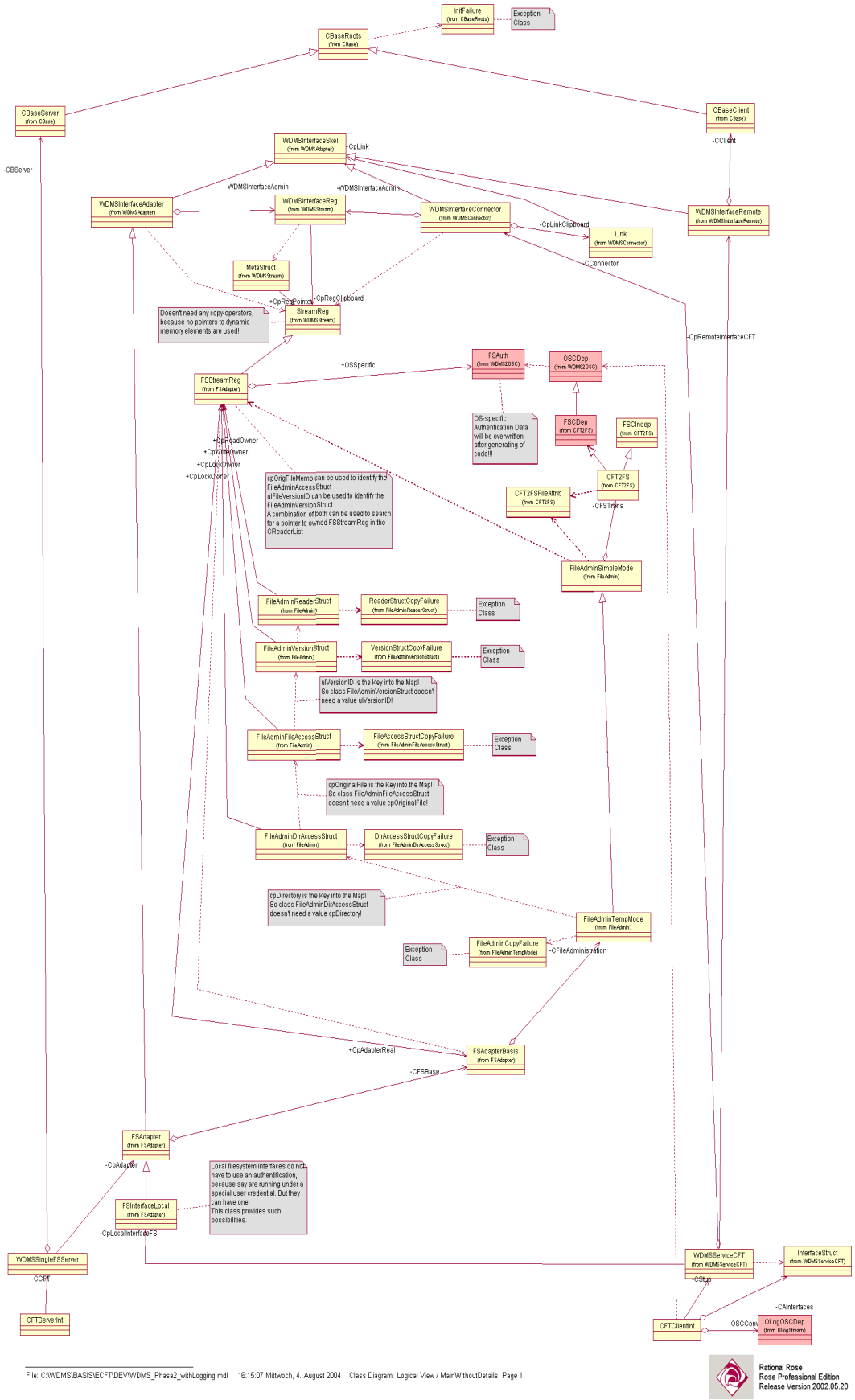


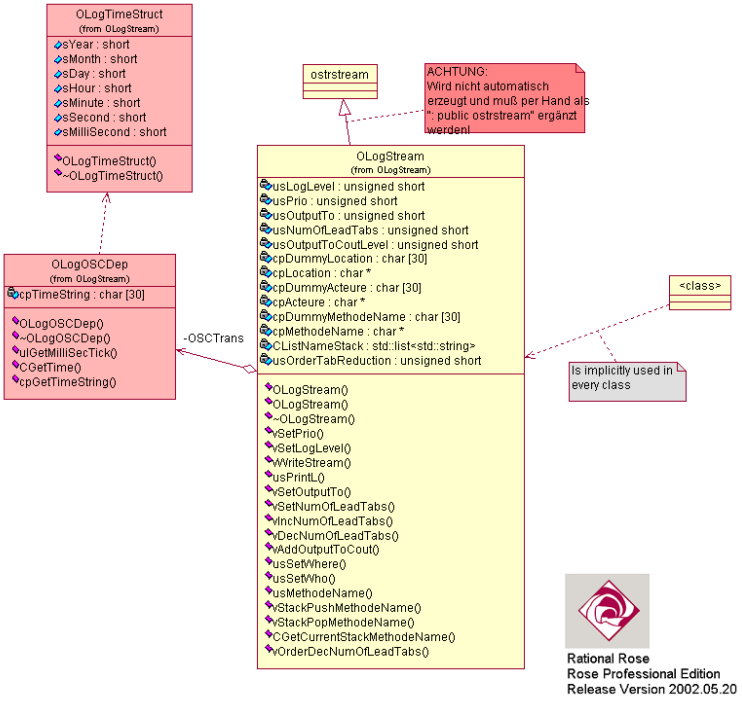


Anhang F

Klassendiagramme zum ECFT

Nachfolgende Diagramme wurden mit Rational Rose Professional Edition (Release Version 2002.05.20) erstellt und dienen zur automatischen Generierung des Codeskeletts (Code- und Headerdateien mit Aufbau und Struktur der einzelnen Klassen) für die Komponenten des Clients und des Servers des ECFT . Jedem Element (Methode, Klasse, etc.) ist dabei eine eindeutige Kennung zugeordnet, so dass ein schnelles Reengineering des Codes möglich wird. Dies ermöglicht die direkte Arbeit mit den Quellen mit anschließender, automatischer Übernahme in die logischen Entwicklungsdiagramme.



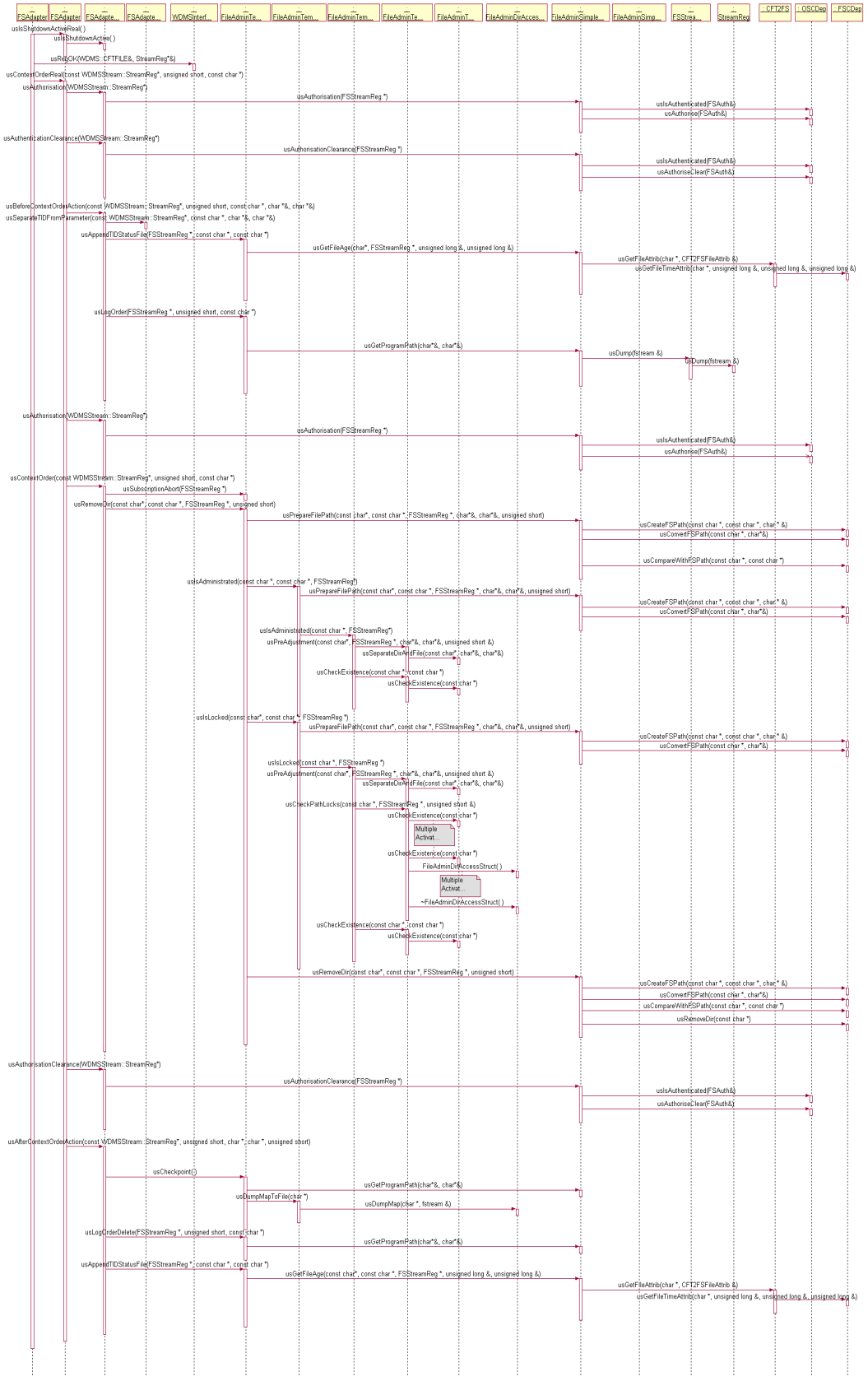


File: C:\WDMS\BASIS\ECFT\DEV\WDMS_Phase2_withLogging.mdl 16:26:14 Mittwoch, 4. August 2004 Class Diagram: Logical View / MainLogging Page 1

Anhang G

Beispiel eines Sequenzdiagramms zum ECFT: „mkdir“-Befehl

Nachfolgendes Diagramm wurde mit Rational Rose Professional Edition (Release Version 2002.05.20) erstellt. Es entstand aus einer Mitschrift des automatischen Logbuchs im „LOG_TRACE“-Modus, so dass damit eine Spur der Ein- und Aussprungpunkte der serverseitigen Methoden des „mkdir“-Befehls graphisch dargestellt ist. Diese Form von Diagrammen dient zur Validierung bzgl. der gewünschten Abfolge von Methodenaufrufen.



Anhang H

Die GPS-Permanentstation in Lhasa/Tibet

Die besonderen Bedingungen im klassischen Netz (d.h. im Nicht-Echtzeit-Netz) der GPS-Permanentstationen werden am besten durch die Station im Hauptgebäude des Tibet Autonomous Regional Buerau of Surveying and Mapping in Lhasa/Tibet (Höhe ca. 3600 Meter mitten im Himalaya) beschrieben. Obwohl sie von ihrem Aufbau in etwa anderen Stationen dieser Art entspricht, ist sie durch ihre exponierte Lage besonders problematisch. Im Rahmen einer 12-tägigen Instandsetzungsreise wurde die Station in der nachstehend beschriebenen Weise durch Klaus Röttcher und Alexander Neidhardt im Oktober 2003 errichtet[1].

Hauptbestandteil ist ein Datensammelrack. Darin befindet sich ein Sammelrechner (Standardindustrie-PC von der Firma SBS) mit zahlreichen seriellen Schnittstellen zur Kommunikation sowohl mit den GPS -Empfängern, der Wetterstation als auch mit entsprechenden Modems für den Datentransfer. In dem Rack sind zwei GPS -Empfänger (Javad, Turbo Rogue) mit jeweiligen Antennen auf dem Dach des Stationsgebäudes untergebracht. Auch die Wettersensoren befinden sich auf dem Dach oder direkt in der Wetterstation, welche ebenfalls im Rack integriert ist. Zur Sicherung der Stromversorgung ist eine Uninterruptable Power Supply (USP) mit 12V-Batterien am herkömmlichen Stromnetz und an einem Solarpanel auf dem Dach angeschlossen.

Die Daten werden stunden- (hourly) und tagesweise (daily) aufgezeichnet. Über eine ADSL -Verbindung erfolgt für die Stundendaten stündlich und für die Tagesdaten täglich eine FTP -Kommunikation (ncftp-Client) zum National Geomatics Center of China (NGCC) nach Peking und zur Fundamentalstation nach Wettzell. Ein Analogmodem erlaubt administrative Anwahlen direkt zum Mess-PC .

[1] vgl. dazu [NEI03]


GPS-Permanentstation
Lhasa/Tibet
GPS-Antennen und
Wettersensor
Solarzellen
Batteriepack
Datenaufzeichnungsrack
Unabhängige
Stromversorgung

Anhang I

Snapshot zur Übertragungsleistung von und nach Lhasa

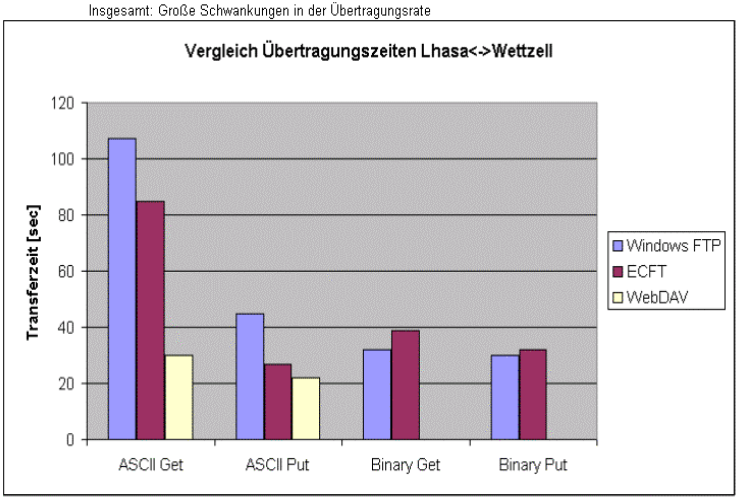
Nachfolgende Kurzversuche wurden im Rahmen des zeitlich eng begrenzten Reparaturaufenthalts in Lhasa im Oktober 2003 unternommen. Sie sollten einen generellen Funktionstest bilden und zur groben Abschätzung des Übertragungsverhaltens von und nach Lhasa dienen. Es wurden einige wenige Übertragungen jeweils manuell durchgeführt und die erhaltenen Übertragungszeiten gemittelt.

Als Testclient in Lhasa diente ein:
Dell Latitude CPx H, P3, CU, 500, MOBILE (ein von der Messstation unabhängiger Laptop) mit Windows 2000

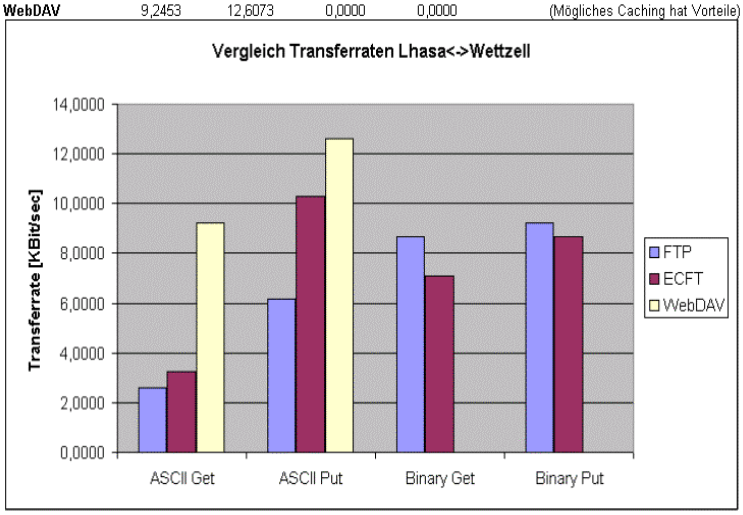
Als Testserver in Wettzell diente ein:
Dell OptiPlex GX1, P2, 400, DESHUTES mit Windows NT 4.0

Dateigröße:	277360Bit (= 34670 Byte)				
	ASCII Get	ASCII Put	Binary Get	Binary Put	
FTP	107,37	45	32	30	(4 Timeouts von 12 Versuchen)
ECFT	85	27	39	32	(Blocksize 5KB; keine Timeouts)
WebDAV	30	22	0	0	(Mögliches Caching hat Vorteile)

Insgesamt: Große Schwankungen in der Übertragungsrate



Dateigröße:	277360Bit (= 34670 Byte)				
	ASCII Get	ASCII Put	Binary Get	Binary Put	
FTP	2,5832	6,1636	8,6675	9,2453	(4 Timeouts von 12 Versuchen)
ECFT	3,2631	10,2726	7,1118	8,6675	(Blocksize 5KB; keine Timeouts)
WebDAV	9,2453	12,6073	0,0000	0,0000	(Mögliches Caching hat Vorteile)



Anhang J

Ergebnisse der Messungen im WAN

Die Messungen wurden mit Hilfe der weltweit verteilten GPS -Permanentstationen Concepción/Chile, Helgoland/Germany, Lhasa/Tibet und Reykjavik/Iceland im klassischen GPS -Netz durchgeführt. Die Kommunikation findet darin hauptsächlich via Internet statt. Für die Tests wurde prallel zum Produktionssystem ein Sammelserver eingerichtet, der die immer zur vollen Stunde verschickten „Hourly“-Daten neben dem regulären Produktionsbetrieb ebenfalls aufnahm. Die Übertragung wurde durch einen externen Scheduler gestartet und über den Skriptmechanismus von ECFT durchgeführt. Alle Aktionen und die Verarbeitungszeiten wurden mitprotokolliert. Bei FTP (mit ncftp- und Perl-Clients) wurden die Anfangs- und Endzeitpunkte einer kompletten Übertragung zum Server des Produktionssystems registriert. Die Messungen sollten den herkömmlichen Ablauf mit dem neuen Verfahren vergleichbar machen.

Gemessen wurden die Übertragungszeiten und die daraus resultierenden Transferraten für einen kompletten Übertragungslauf und bei ECFT zusätzlich für die eigentliche Dateiübertragung. Aufgrund eines Umsetzungsfehlers mussten die Ergebnisse um eine lineare Drift korrigiert werden. Der Fehler ist mittlerweile behoben. Die graphischen Darstellungen der Messreihen wurden miteinander verglichen.

Zur Verdeutlichung der Tagesstabilität wurden jeweils aufeinanderfolgende 24-Stunden-Abschnitte übereinander gelegt. Zudem rechnete Dr. Ulrich Schreiber mit Hilfe seines Programms unter National Instruments LabView 6.1 Leistungsspektren für die Messungen zu Helgoland (Auszüge der graphischen Darstellung des Programms und der zugehörigen Beschreibung zum Leistungsspektrum aus den Hilfstexten von LabView sind in Abschnitt J.2 ab Seite 259 abgedruckt).

Als Testclients dienten:
Verschiedene SBS-Industrierechner (AMD K6, 300 MHz, 256K) mit Windows NT 4.0

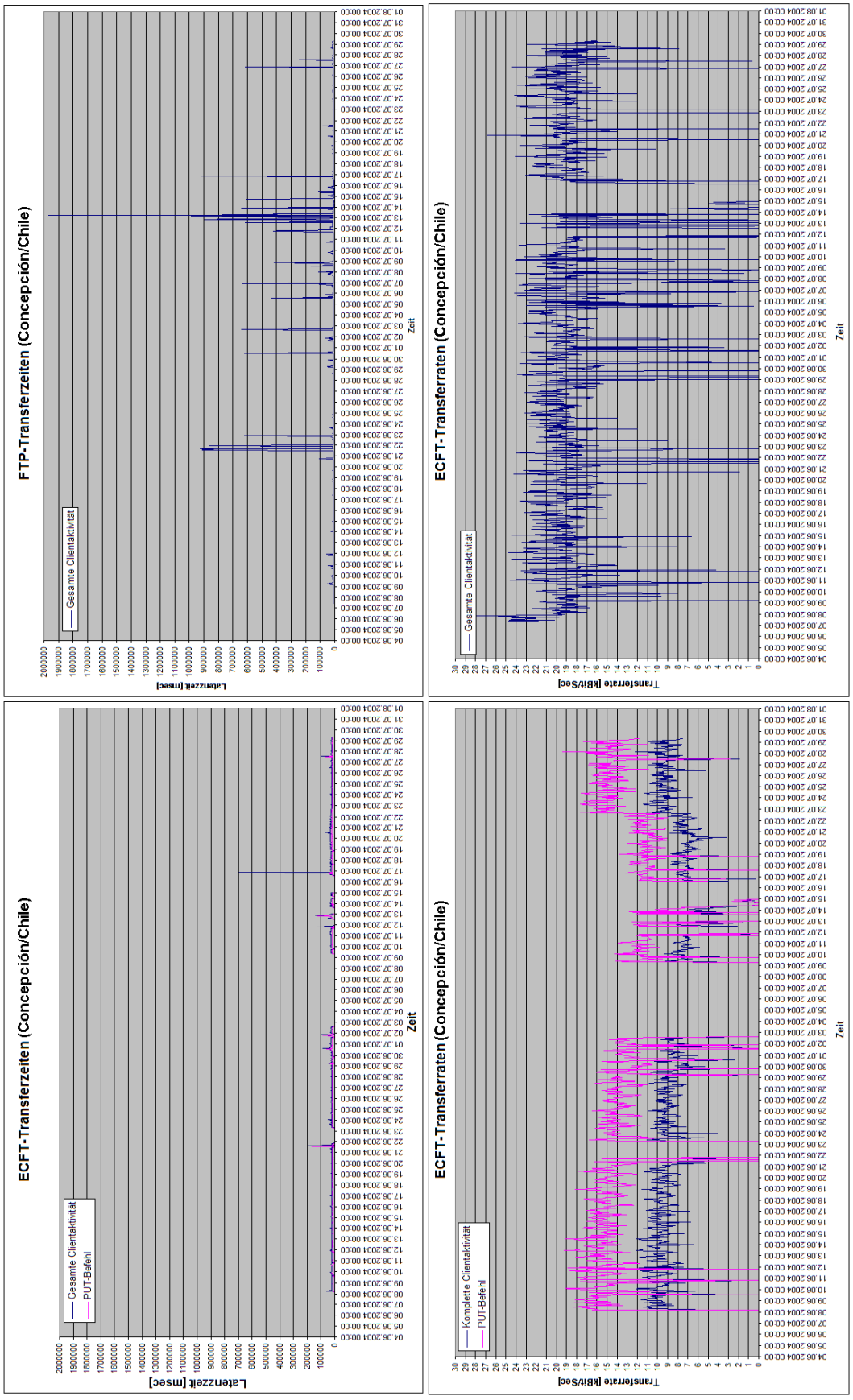
Als Testserver in Wettzell diente ein:
Dell OptiPlex GX1, P2, 400, DESHUTES mit SuSe Linux 7.2

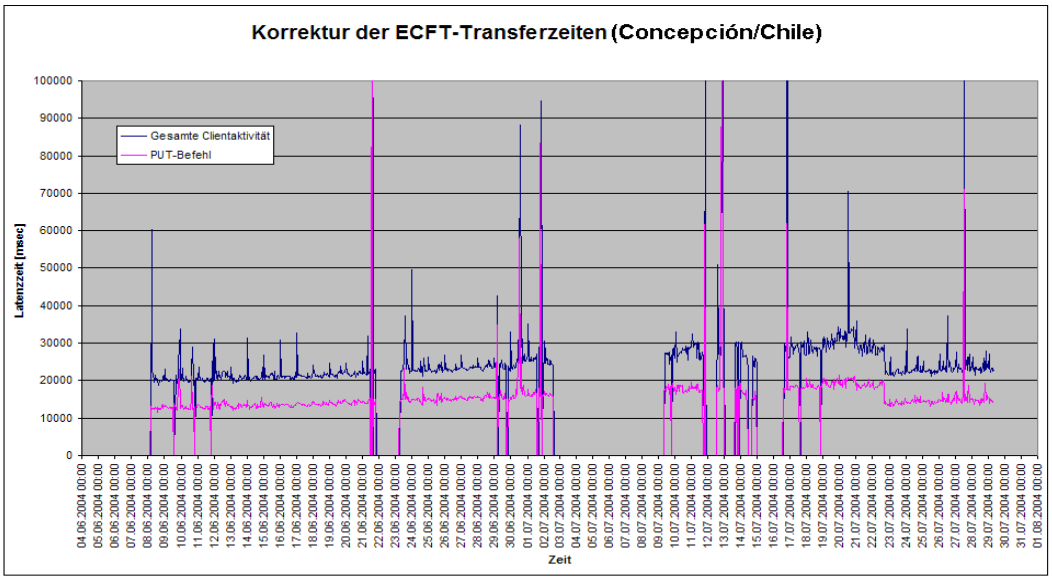
J.1 Concepción/Chile

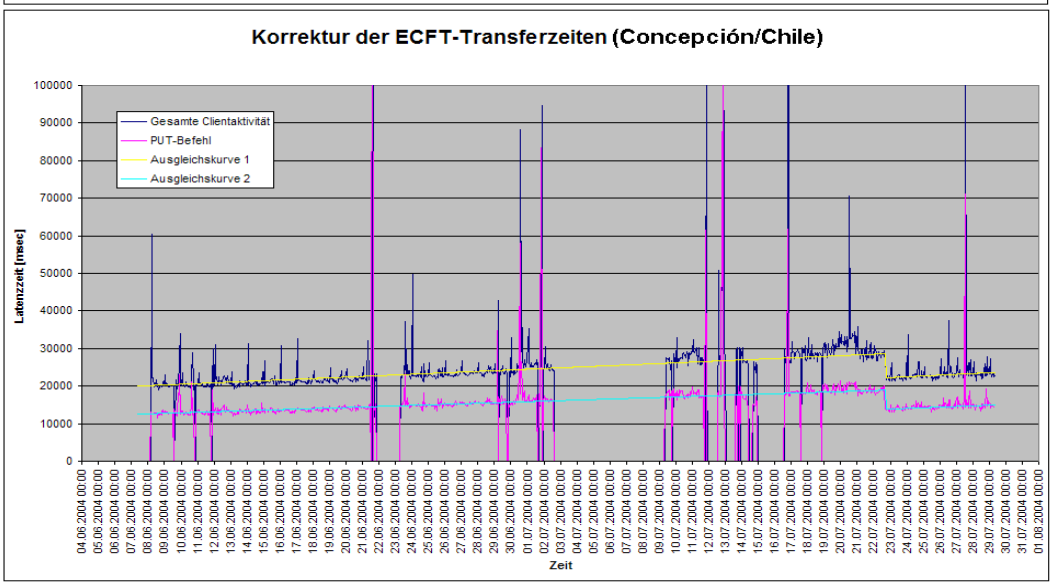
	ECFT				FTP			
Latenz: Gesamte Client-Aktivität [msec]	**MAX:**	688655,61	**RMS:**	**Normierter RMS:**	**MAX:**	440102,00	**RMS:**	**Normierter RMS:**
	MEAN:	22181,67	23006,06	1,04	**MEAN:**	13165,94	22205,51	1,69
	MIN:	16338,59			**MIN:**	7301,00		
Rate: Gesamte Client-Aktivität [KBit/Sec]]	**MAX:**	14,33	**RMS:**	**Normierter RMS:**	**MAX:**	28,27	**RMS:**	**Normierter RMS:**
	MEAN:	10,00	1,73	0,17	**MEAN:**	18,79	3,7	0,20
	MIN:	0,30			**MIN:**	0,40		

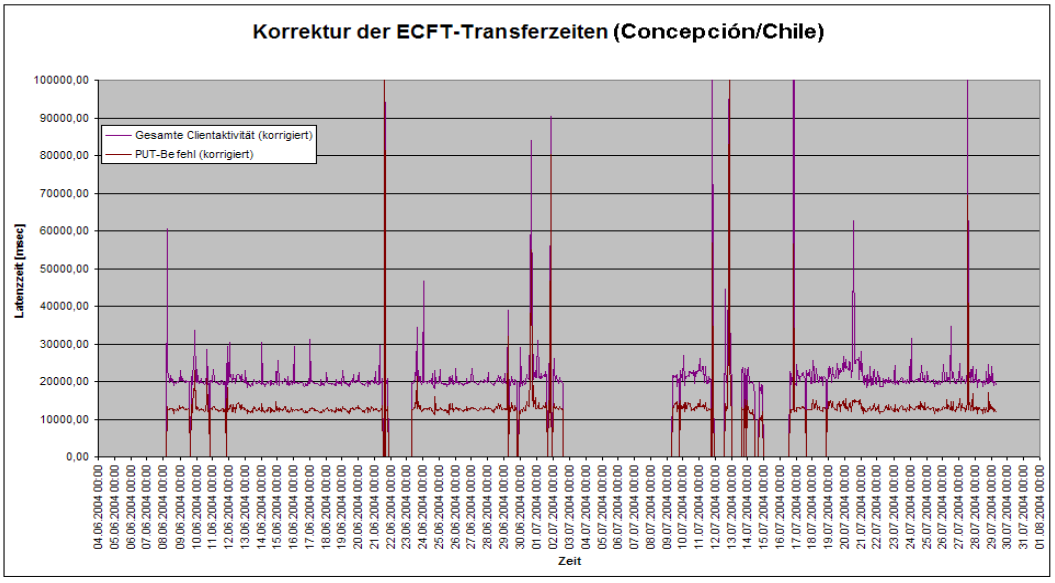
	ECFT				FTP			
Latenz: PUT-Befehl [msec]	**MAX:**	117967,28	**RMS:**	**Normierter RMS:**				
	MEAN:	13400,52	5449,98	0,41				
	MIN:	9566,50						
Rate: PUT-Befehl [KBit/Sec]]	**MAX:**	22,86	**RMS:**	**Normierter RMS:**				
	MEAN:	16,05	2,52	0,16				
	MIN:	1,07						

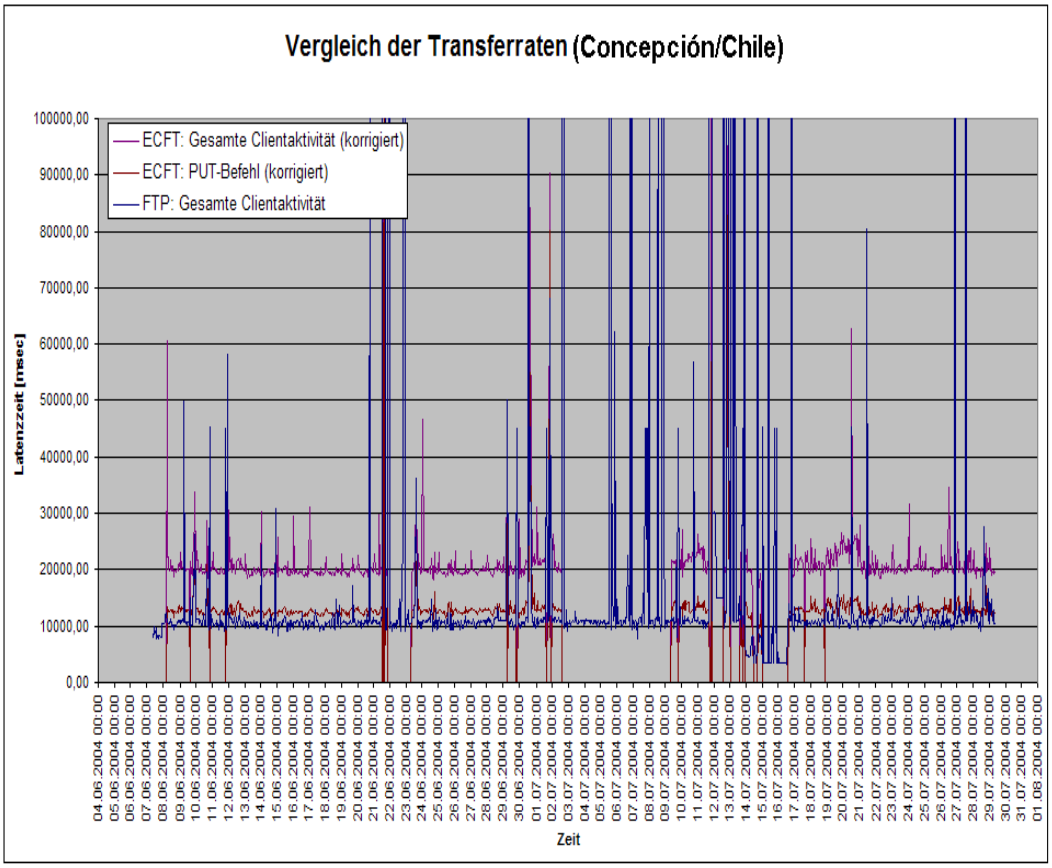
Datei-größen [Byte]	**MAX:**	33112,00	**RMS:**	**Normierter RMS:**
	MEAN:	26073,41	3587,85	0,14
	MIN:	1281,00		

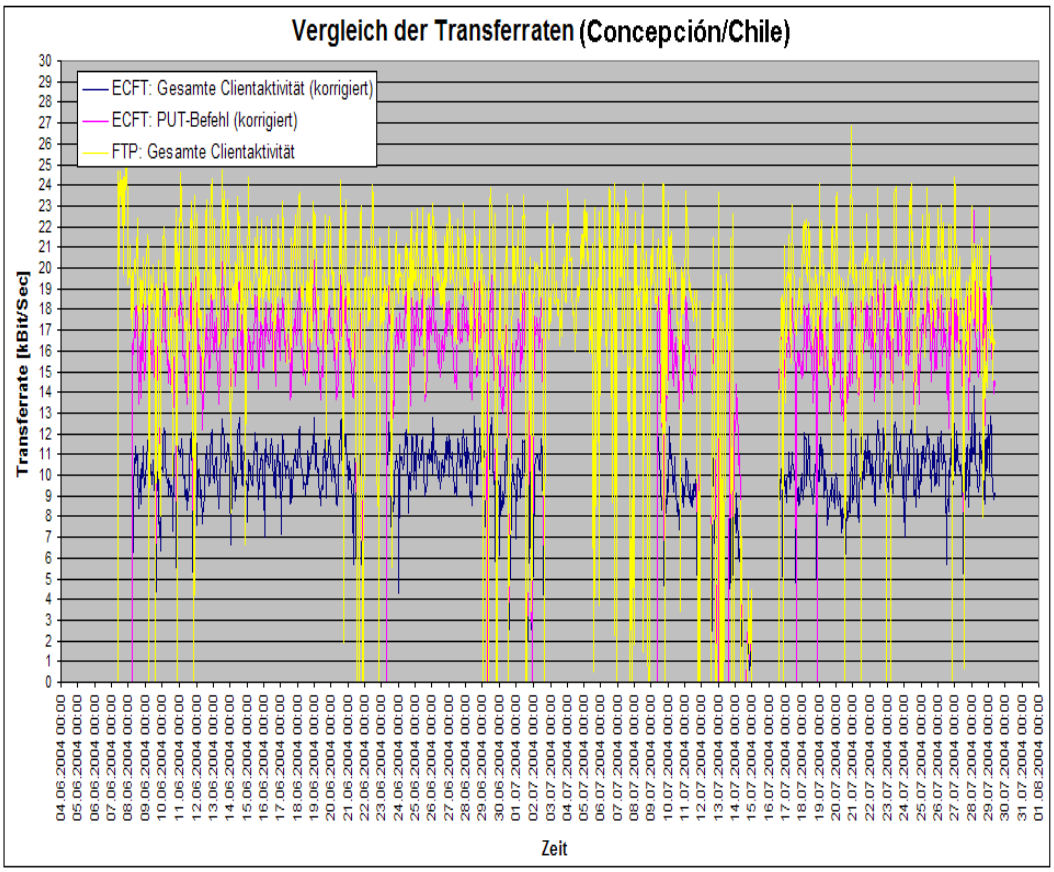

FTP-Transferzeiten (Concepción/Chile)
Gesamte Clientaktivität
Laufzeit [msec]
Zeit
ECFT-Transferraten (Concepción/Chile)
Gesamte Clientaktivität
Transferrate [kBit/Sec]
Zeit
ECFT-Transferzeiten (Concepción/Chile)
Gesamte Clientaktivität
PUT-Befehl
Laufzeit [msec]
Zeit
ECFT-Transferraten (Concepción/Chile)
Komplette Clientaktivität
PUT-Befehl
Transferrate [kBit/Sec]
Zeit

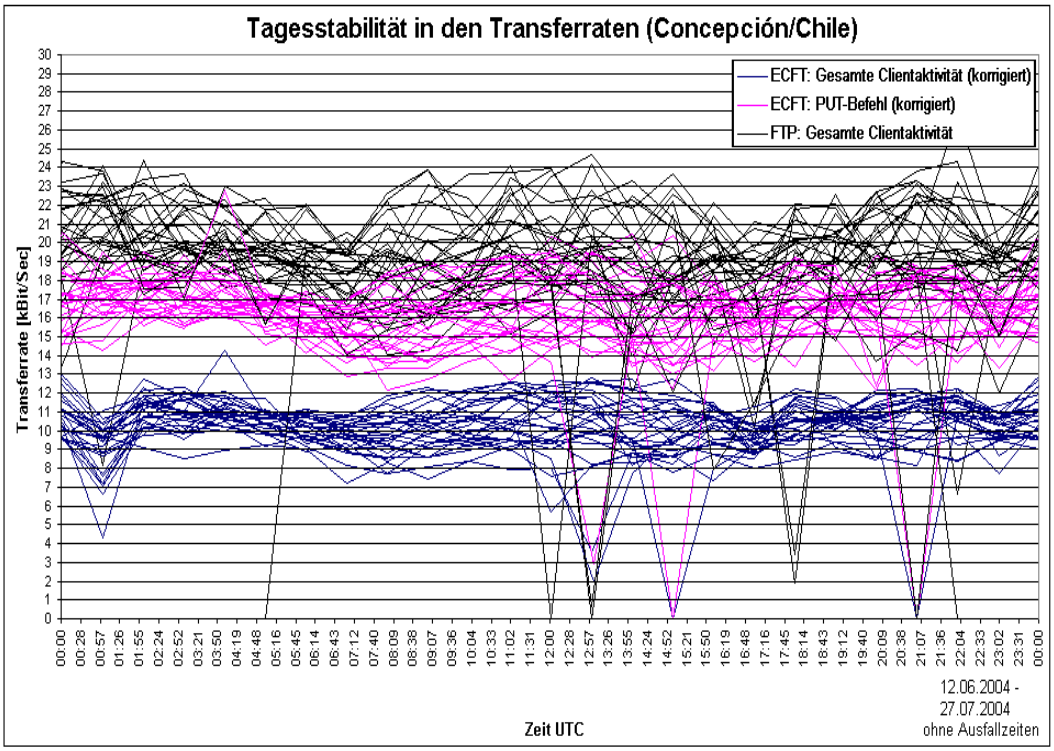

Korrektur der ECFT-Transferzeiten (Concepción/Chile)
Gesamte Clientaktivität
PUT-Befehl
Latenzzeit [msec]
Zeit


Korrektur der ECFT-Transferzeiten (Concepción/Chile)
Gesamte Clientaktivität
PUT-Befehl
Ausgleichskurve 1
Ausgleichskurve 2
Latenzzeit [msec]
Zeit


Korrektur der ECFT-Transferzeiten (Concepción/Chile)
Gesamte Clientaktivität (korrigiert)
PUT-Befehl (korrigiert)
Latenzzeit [msec]
Zeit


Vergleich der Transferraten (Concepción/Chile)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Latenzzeit [msec]
Zeit


Vergleich der Transferraten (Concepción/Chile)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Transferrate [kBit/Sec]
Zeit

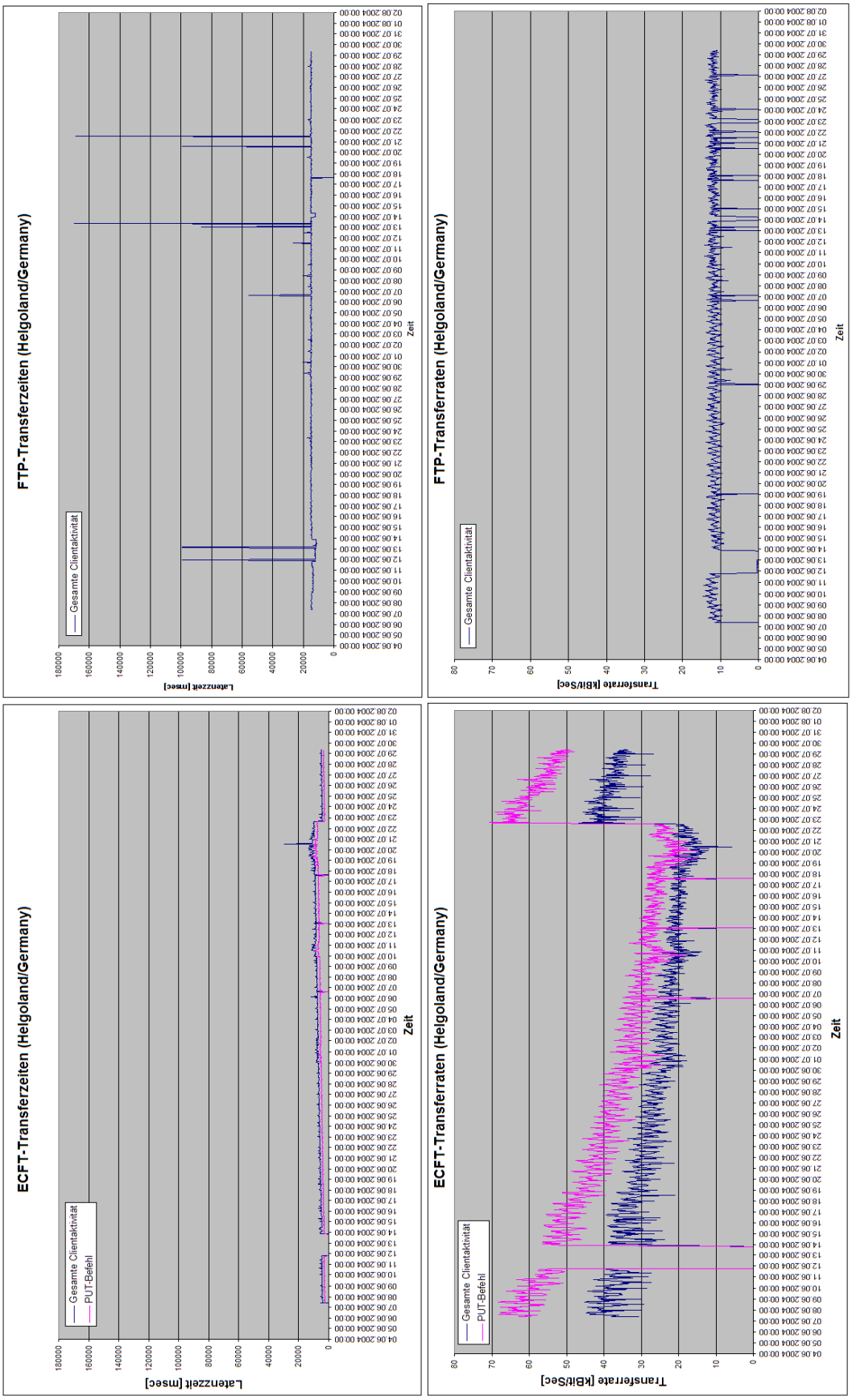

Tagesstabilität in den Transferraten (Concepción/Chile)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Transferrate [kBit/Sec]
Zeit UTC
12.06.2004 - 27.07.2004
ohne Ausfallzeiten

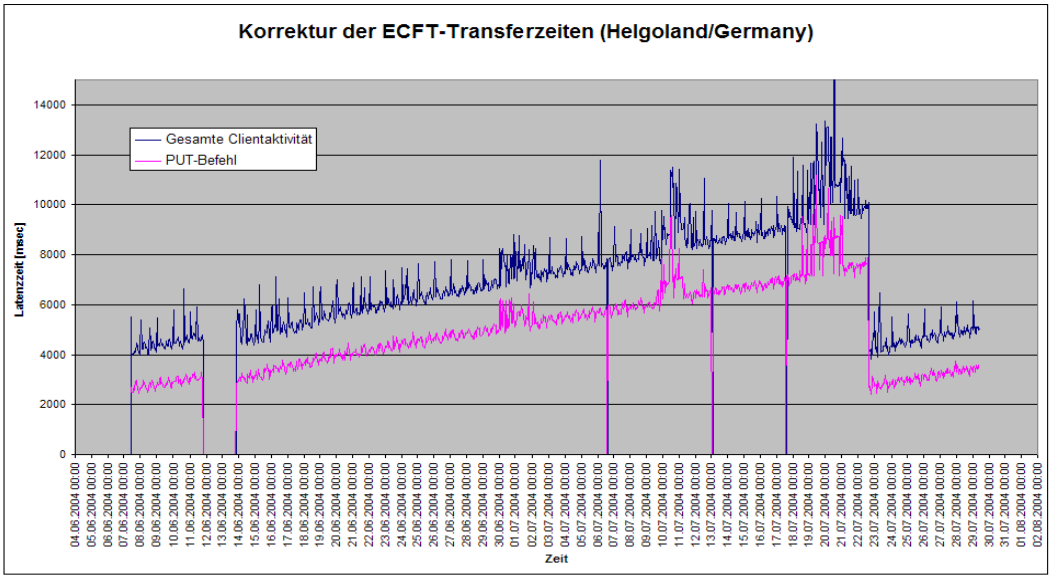
J.2 Helgoland/Germany

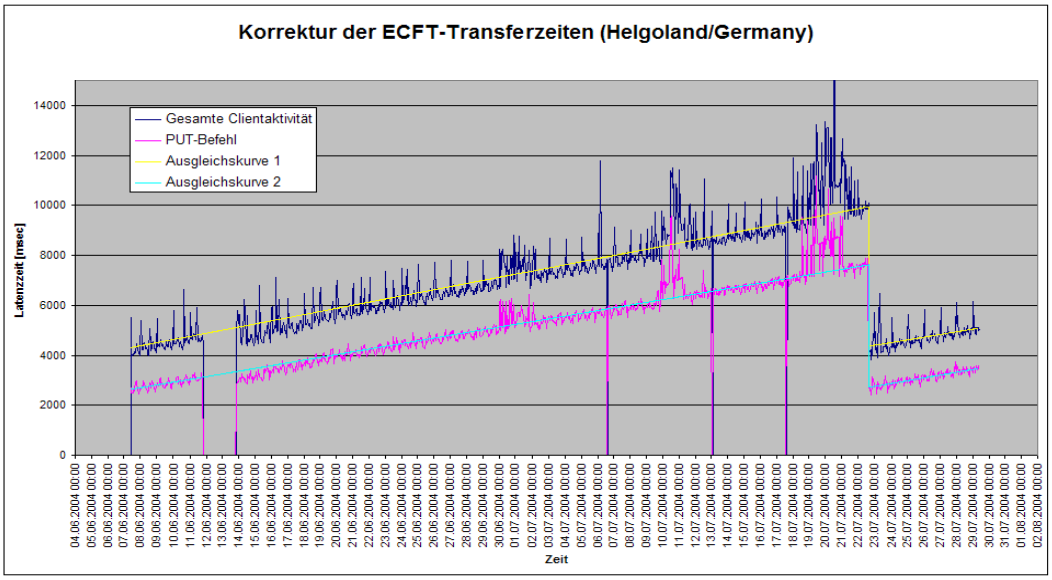
	ECFT				FTP			
Latenz: Gesamte Client-Aktivität [msec]	**MAX:**	24069,22	**RMS:**	**Normierter RMS:**	**MAX:**	169854,00	**RMS:**	**Normierter RMS:**
	MEAN:	4391,65	878,37	0,20	**MEAN:**	15088,01	5531,28	0,26
	MIN:	3490,85			**MIN:**	12027,00		
Rate: Gesamte Client-Aktivität [KBit/Sec]]	**MAX:**	49,81	**RMS:**	**Normierter RMS:**	**MAX:**	14,65	**RMS:**	**Normierter RMS:**
	MEAN:	40,97	5,28	0,13	**MEAN:**	11,35	2,34	0,21
	MIN:	5,61			**MIN:**	0,02		

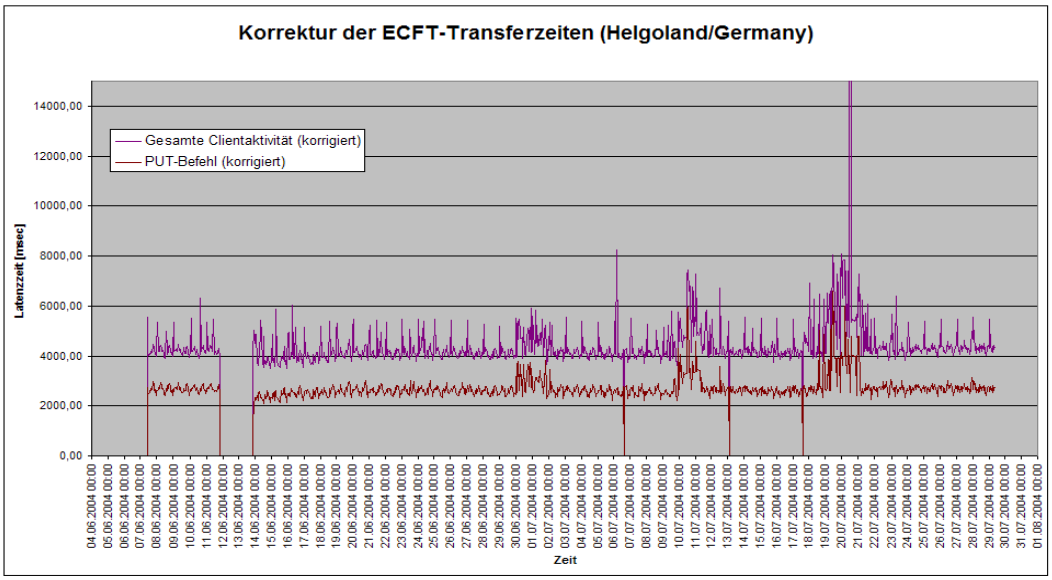
	ECFT				FTP			
Latenz: PUT-Befehl [msec]	**MAX:**	6593,45	**RMS:**	**Normierter RMS:**				
	MEAN:	2710,96	418,87	0,15				
	MIN:	1208,97						
Rate: PUT-Befehl [KBit/Sec]]	**MAX:**	78,14	**RMS:**	**Normierter RMS:**				
	MEAN:	65,94	6,83	0,10				
	MIN:	19,42						

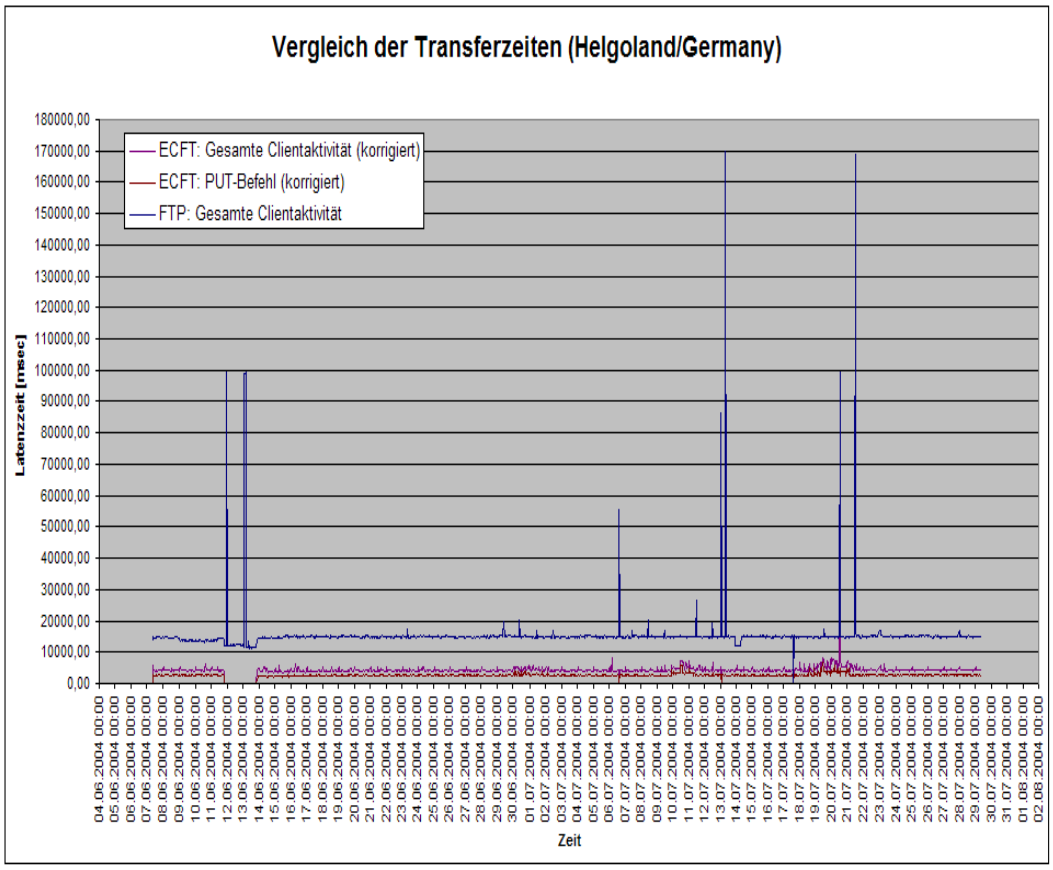
Datei-größen [Byte]	**MAX:**	27405,00	**RMS:**	**Normierter RMS:**
	MEAN:	22107,82	2132,29	0,10
	MIN:	2935,00		

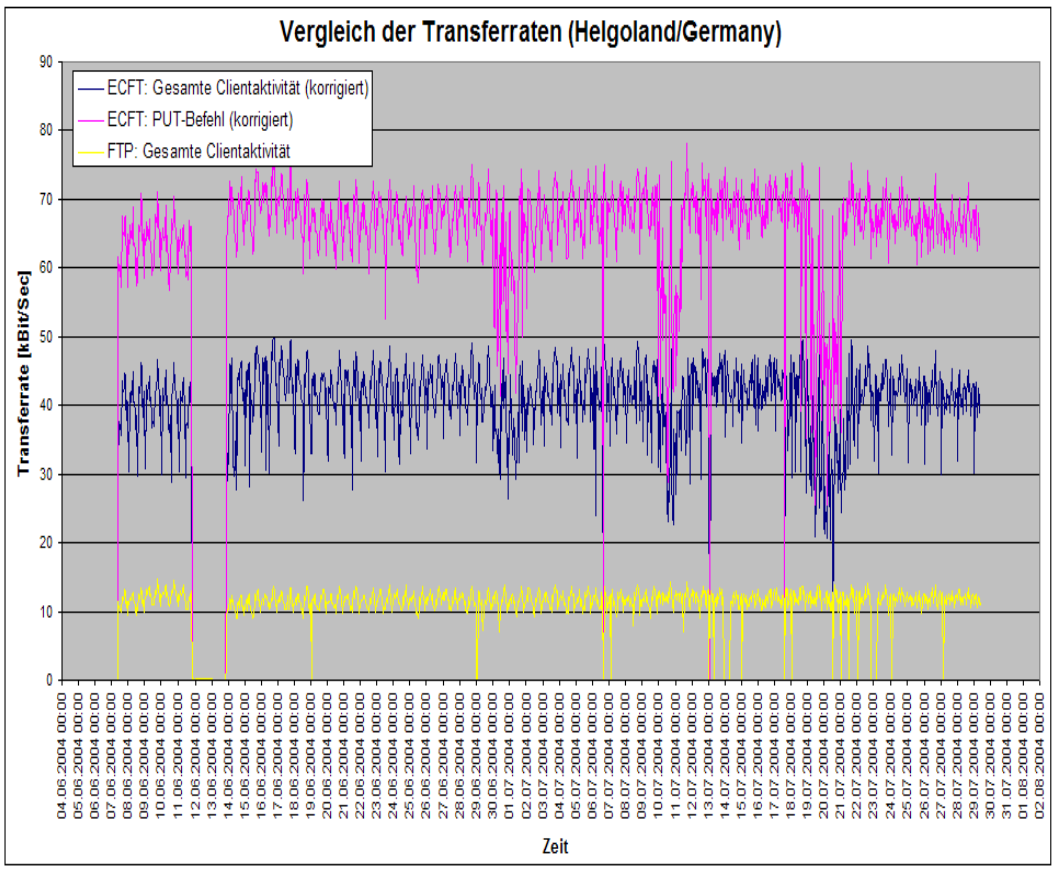

FTP-Transferzeiten (Helgoland/Germany)
Gesamte Clientaktivität
Latenzzeit [msec]
Zeit
FTP-Transferraten (Helgoland/Germany)
Gesamte Clientaktivität
Transferrate [kBit/Sec]
Zeit
ECFT-Transferzeiten (Helgoland/Germany)
Gesamte Clientaktivität
PUT-Befehl
Latenzzeit [msec]
Zeit
ECFT-Transferraten (Helgoland/Germany)
Gesamte Clientaktivität
PUT-Befehl
Transferrate [kBit/Sec]
Zeit

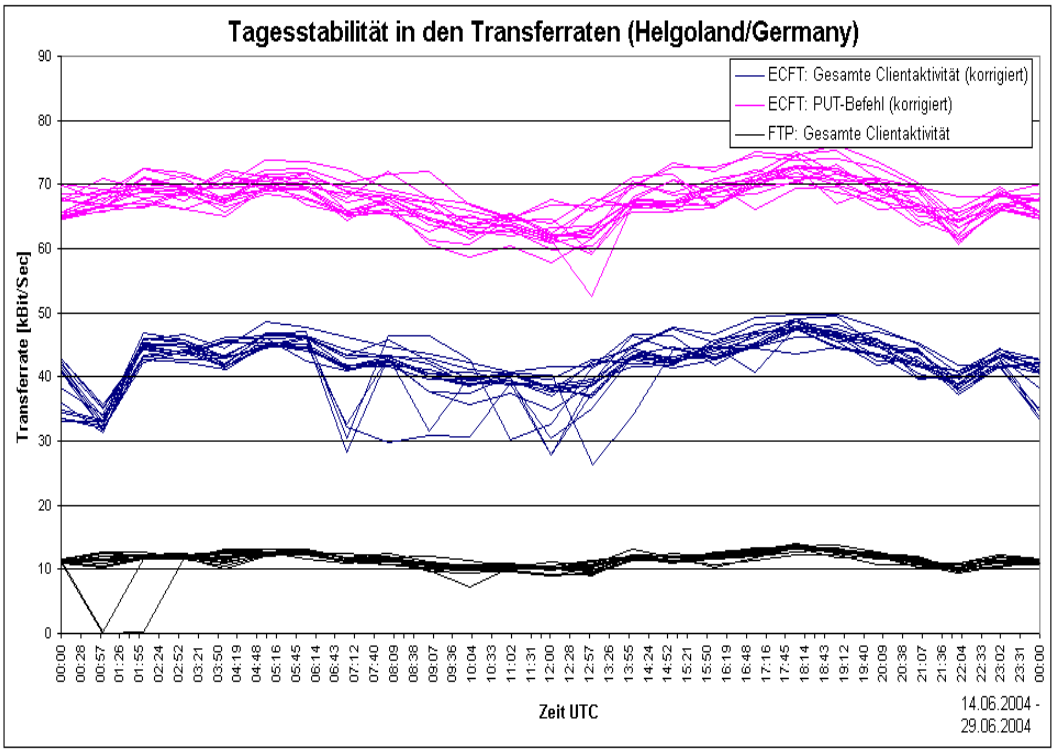

Korrektur der ECFT-Transferzeiten (Helgoland/Germany)
Gesamte Clientaktivität
PUT-Befehl
Latenzzeit [msec]
Zeit

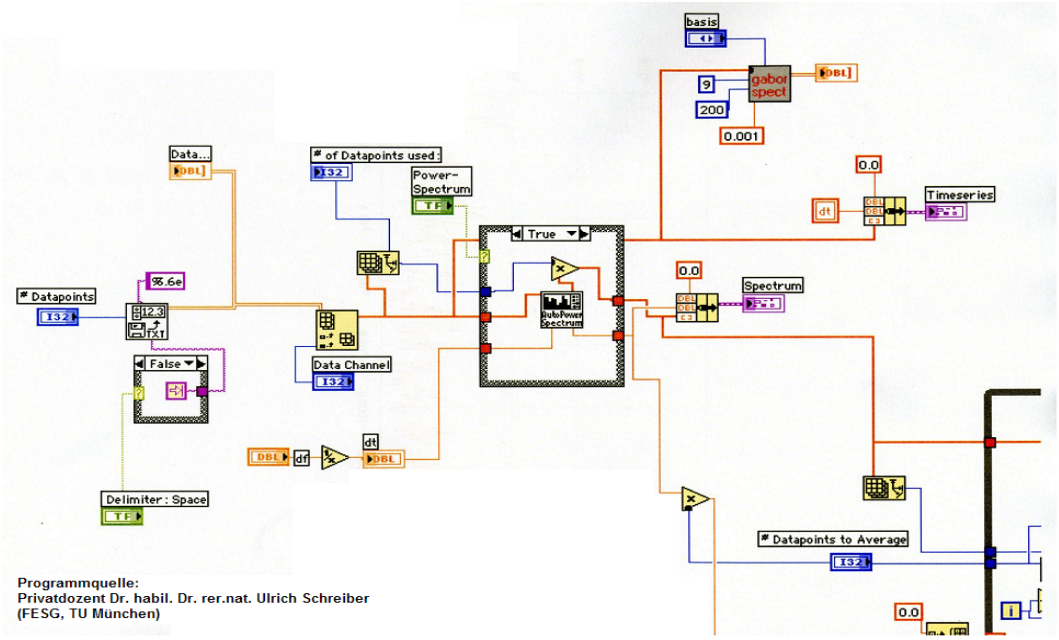

Korrektur der ECFT-Transferzeiten (Helgoland/Germany)
Gesamte Clientaktivität
PUT-Befehl
Ausgleichskurve 1
Ausgleichskurve 2
Latenzzeit [msec]
Zeit

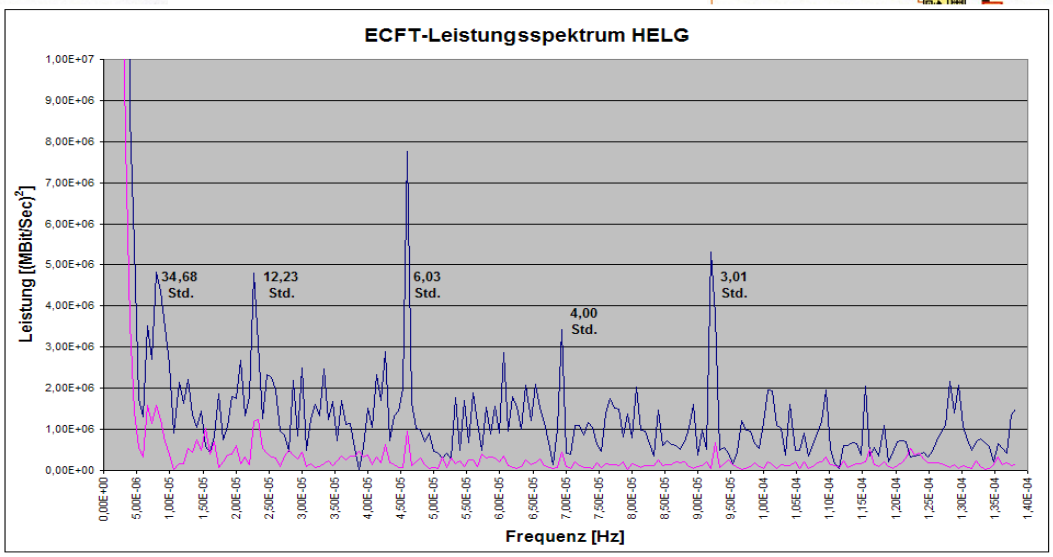

Korrektur der ECFT-Transferzeiten (Helgoland/Germany)
Gesamte Clientaktivität (korrigiert)
PUT-Befehl (korrigiert)
Latenzzeit [msec]
Zeit

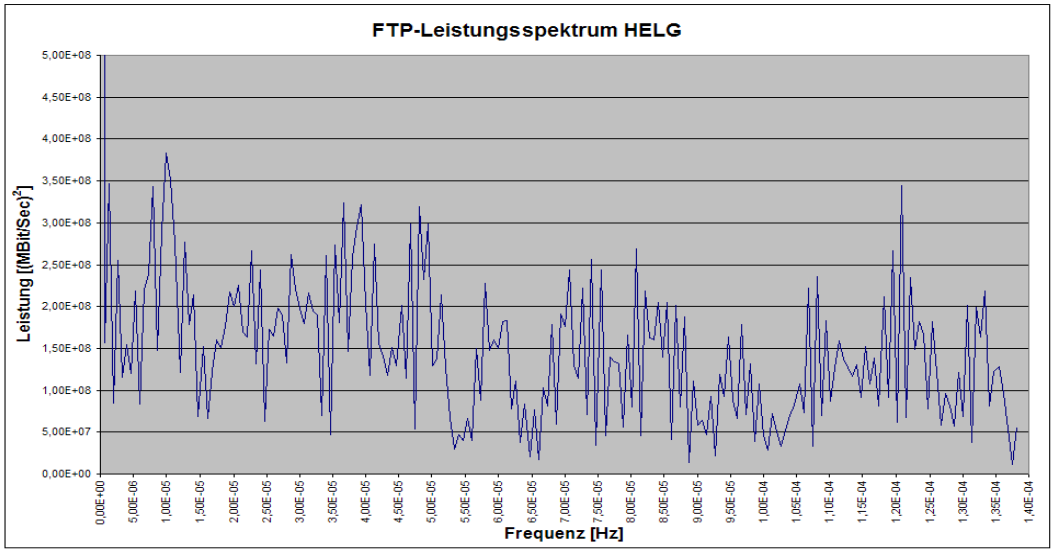

Vergleich der Transferzeiten (Helgoland/Germany)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Latenzzeit [msec]
Zeit


Vergleich der Transferraten (Helgoland/Germany)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Transferrate [kBit/Sec]
Zeit


Tagesstabilität in den Transferraten (Helgoland/Germany)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Transferrate [kBit/Sec]
90
80
70
60
50
40
30
20
10
0
00:00
00:28
00:57
01:26
01:55
02:24
02:52
03:21
03:50
04:19
04:48
05:16
05:45
06:14
06:43
07:12
07:40
08:09
08:38
09:07
09:36
10:04
10:33
11:02
11:31
12:00
12:28
12:57
13:26
13:55
14:24
14:52
15:21
15:50
16:19
16:48
17:16
17:45
18:14
18:43
19:12
19:40
20:09
20:38
21:07
21:36
22:04
22:33
23:02
23:31
00:00
Zeit UTC
14.06.2004 -
29.06.2004


basis
9
200
0.001
gabor spect
DBL
0.0
dt
Timeseries
Data...
# of Datapoints used
Power-Spectrum
TF
True
Spectrum
Datapoints
I32
%6.6e
12.3
FXT
False
Data Channel
AutoPower Spectrum
df
Delimiter : Space
Datapoints to Average
Programmquelle:
Privatdozent Dr. habil. Dr. rer.nat. Ulrich Schreiber
(FESG, TU München)

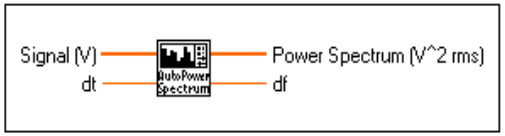

ECFT-Leistungsspektrum HELG
Leistung [(MBit/Sec)²]
Frequenz [Hz]
34,68 Std.
12,23 Std.
6,03 Std.
4,00 Std.
3,01 Std.


FTP-Leistungsspektrum HELG
Leistung [(MBit/Sec)²]
Frequenz [Hz]

(Auszug aus der Online-Dokumentation von LabView 6.1)

Auto Power Spectrum (Not in Base Package)

Computes the single-sided, scaled, auto power spectrum of a time-domain signal. Details



Signal is the input, time-domain signal, usually in volts. At least three cycles of the signal must be contained in the time-domain record for a valid estimate.

dt is the sample period of the time-domain signal, usually in seconds. **dt** is also $\frac{1}{f_s}$ where f_s is the sampling frequency of the time-domain signal.

Power Spectrum is the single-sided power spectrum in volts rms squared if the input signal is in volts. If the input signal is not in volts, the results are in input signal units rms squared.

df is the line frequency interval of the power spectrum in Hertz, if **dt** is in seconds.

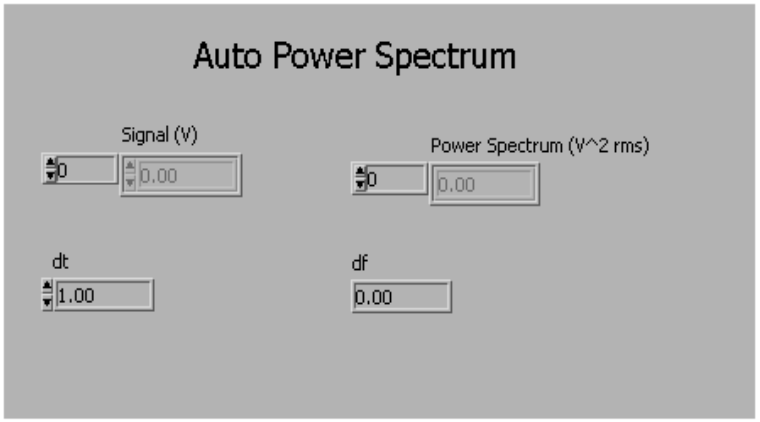
Auto Power Spectrum Details

This VI computes the power spectrum as

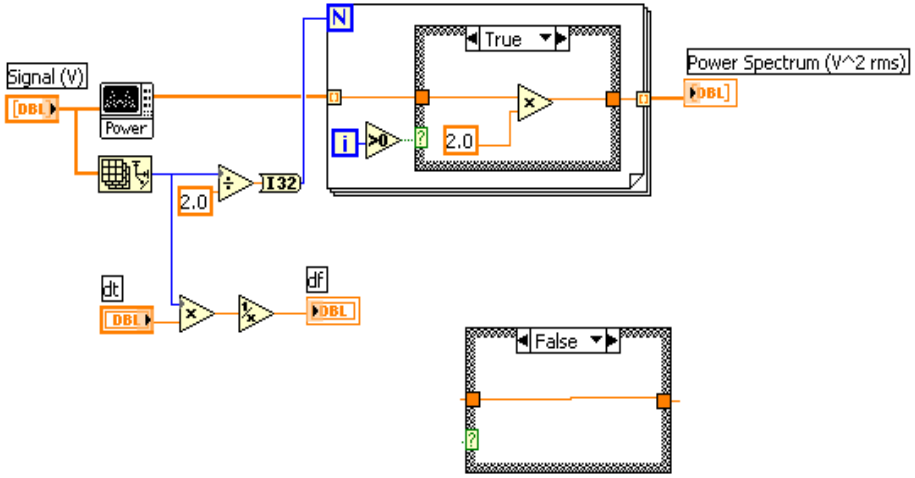
$$\frac{FFT^{*}(Signal) \times FFT(Signal)}{N^{2}},$$

where *N* is the number of points in the signal array and * denotes complex conjugate. The VI then converts the power spectrum into a single-sided power spectrum result.

Front Panel

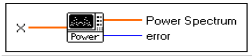


Block Diagram



Power Spectrum (Not in Base Package)

Computes the **Power Spectrum**, Sxx, of the input sequence **X**. Details



X is the input sequence.

Power Spectrum returns the harmonic power content of periodic signals. If **X** represents actual measurements in volts, the VI expresses the normalized units of the output sequence **Power Spectrum** in watts on a 1-Ω basis.

error returns any error or warning from the VI. Refer to Signal Processing Error Codes for more information about these conditions.

Power Spectrum Details

The **Power Spectrum** $S_{xx}(f)$ of a function $x(t)$ is defined as

$$S_{xx}(f) = X^*(f)X(f) = |X(f)|^2 ,$$

where $X(f) = F\{x(t)\}$, and $X^*(f)$ is the complex conjugate of $X(f)$.

The Power Spectrum VI uses the FFT and DFT routines to compute the power spectrum, which is given by

$$S_{xx} = \frac{1}{n^2 |F\{X\}|^2} ,$$

where S_{xx} represents the output sequence **Power Spectrum**, and n is the number of samples in the input sequence **X**.

When the number of samples, n, in the input sequence **X** is a valid power of 2

$$n = 2^m$$

for m = 1, 2, 3,...,23,

the Power Spectrum VI computes the fast Fourier transform of a real-valued sequence using the split-radix algorithm and efficiently scales the magnitude square. The largest **Power Spectrum** the VI can compute using the FFT is 2^23 (8,388,608 or 8M).

When the number of samples in the input sequence **X** is not a valid power of 2

$$n \neq 2^m$$

for m = 1, 2, 3,...,23,

where n is the number of samples, the Power Spectrum VI computes the discrete Fourier transform of a real-valued sequence using an efficient DFT algorithm and scales the magnitude square. The largest **Power Spectrum** the VI can compute using the fast DFT is

$$2^{22} - 1 (4,194,303 \text{ or } 4M - 1) .$$

Let Y be the Fourier transform of the input sequence **X** and n be the number of samples in it. You can show that

$$|Y_{n-i}|^2 = |Y_{-i}|^2 .$$

You can interpret the power in the $(n-1)^{th}$ element of Y as the power in the $-i^{th}$ element of the sequence, which represents the power in the negative $-i^{th}$ harmonic. You can find the total power for the i^{th} harmonic (DC and Nyquist component not included) using

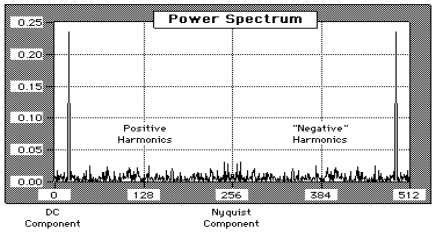
$$\text{Power in } i^{th} \text{ harmonic} = 2|Y_i|^2 = |Y_i|^2 + |Y_{n-i}|^2, \quad 0 < i < \frac{n}{2} .$$

The total power in the DC and Nyquist components are $|Y_0|^2$ and $|Y_{n/2}|^2$, respectively.

If n is even, let $k = \frac{n}{2}$. The following table shows the format of the output sequence S_{xx} corresponding to the **Power Spectrum**.

Array Element	Interpretation
Sxx_0	Power in DC component
$Sxx_1 = Sxx_{(n-1)}$	Power at frequency Δf
$Sxx_2 = Sxx_{(n-2)}$	Power at frequency 2Δf
$Sxx_3 = Sxx_{(n-3)}$	Power at frequency 3Δf
. . .	. . .
$Sxx_{(k-2)} = Sxx_{n-(k-2)}$	Power at frequency (k-2)Δf
$Sxx_{(k-1)} = Sxx_{n-(k-1)}$	Power at frequency (k-1)Δf
Sxx_k	Power in Nyquist harmonic

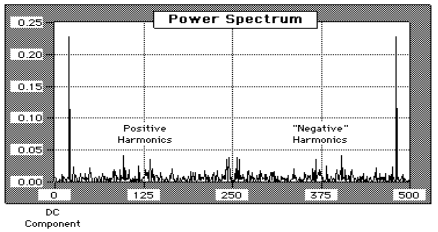
The following illustration represents the preceding table information.



If n is odd, let $k = \frac{n-1}{2}$. The following table shows the format of the output sequence S_{xx} corresponding to the **P Spectrum**.

Array Element	Interpretation
Sxx_0	Power in DC component
$Sxx_1 = Sxx_{(n-1)}$	Power at frequency Δf
$Sxx_2 = Sxx_{(n-2)}$	Power at frequency 2Δf
$Sxx_3 = Sxx_{(n-3)}$	Power at frequency 3Δf
. . .	. . .
$Sxx_{(k-2)} = Sxx_{n-(k-2)}$	Power at frequency (k-2)Δf
$Sxx_{(k-1)} = Sxx_{n-(k-1)}$	Power at frequency (k-1)Δf
$Sxx_k = Sxx_{n-k}$	Power at frequency kΔf

The following illustration represents the preceding table information.



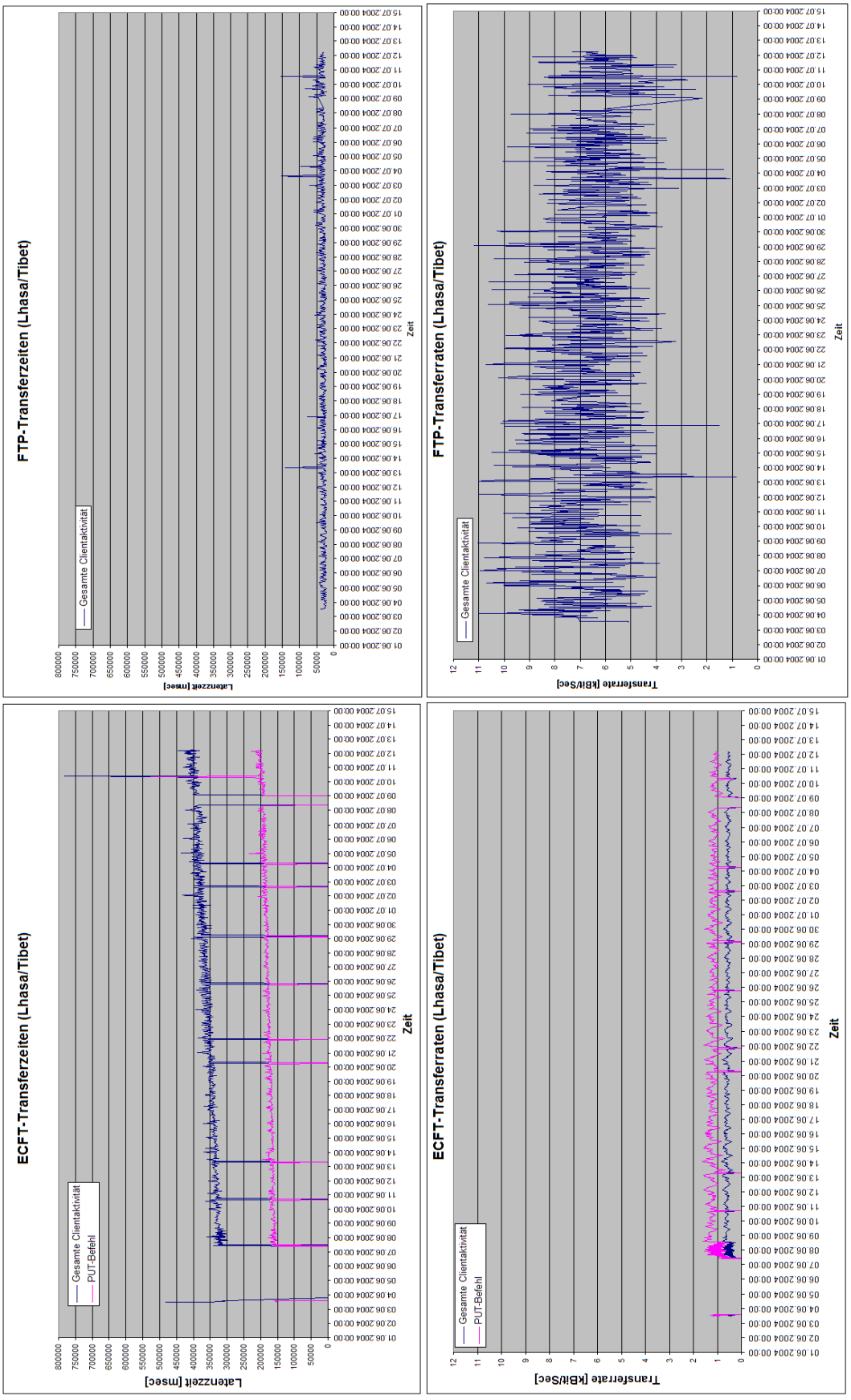
The format described in the preceding tables is an accepted standard in digital signal processing applications.

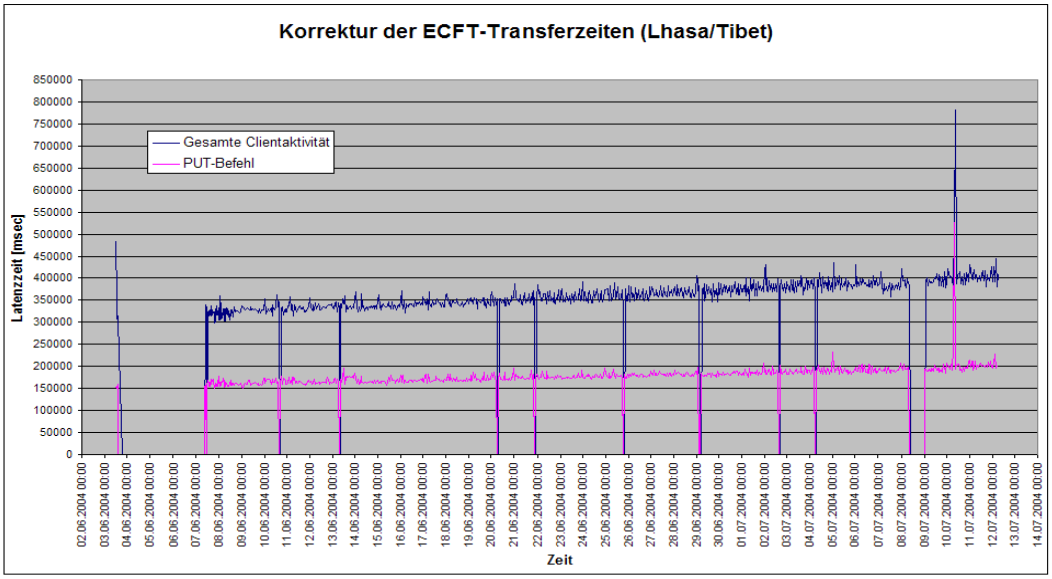
J.3 Lhasa/Tibet

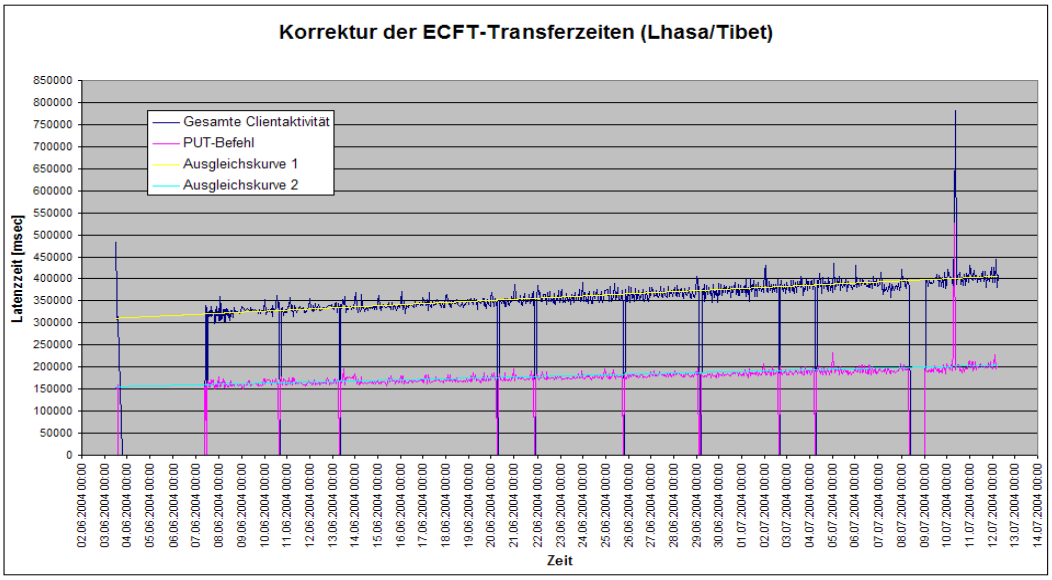
	ECFT				FTP (Lhas/Lhaz)			
Latenz: Gesamte Client-Aktivität [msec]	**MAX:**	701355,74	**RMS:**	**Normierter RMS:**	**MAX:**	152459,00	**RMS:**	**Normierter RMS:**
	MEAN:	320522,87	18764,52	0,06	**MEAN:**	35516,52	10506,76	0,30
	MIN:	285394,46			**MIN:**	15912,00		
Rate: Gesamte Client-Aktivität [KBit/Sec]]	**MAX:**	0,88	**RMS:**	**Normierter RMS:**	**MAX:**	11,26	**RMS:**	**Normierter RMS:**
	MEAN:	0,64	0,10	0,16	**MEAN:**	4,98	2,20	0,44
	MIN:	0,21			**MIN:**	0,76		

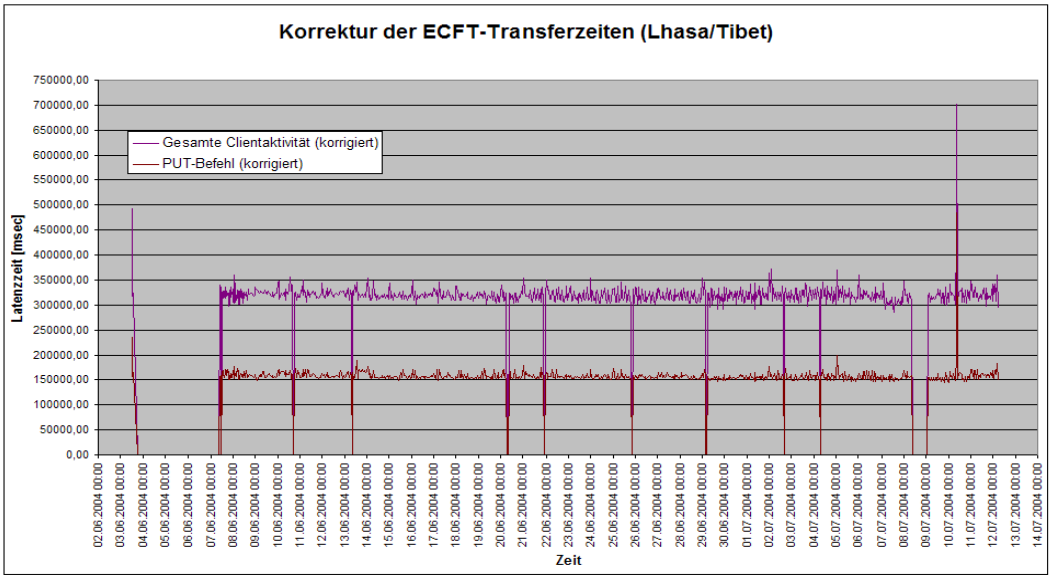
	ECFT				FTP (Lhas/Lhaz)			
Latenz: PUT-Befehl [msec]	**MAX:**	484178,24	**RMS:**	**Normierter RMS:**				
	MEAN:	157976,31	13132,66	0,08				
	MIN:	144881,96						
Rate: PUT-Befehl [KBit/Sec]]	**MAX:**	1,82	**RMS:**	**Normierter RMS:**				
	MEAN:	1,31	0,21	0,16				
	MIN:	0,35						

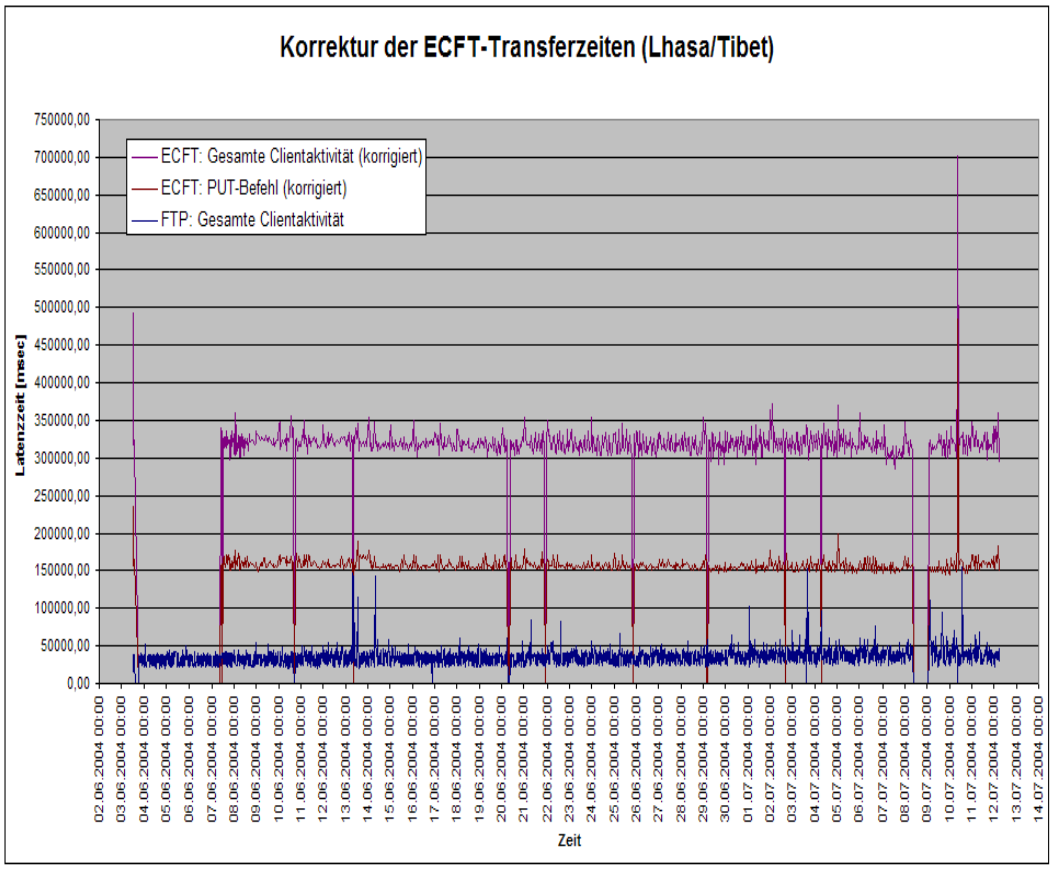
Datei-größen [Byte]	**MAX:**	34968,00	**RMS:**	**Normierter RMS:**
	MEAN:	25677,06	4049,92	0,16
	MIN:	10415,00		

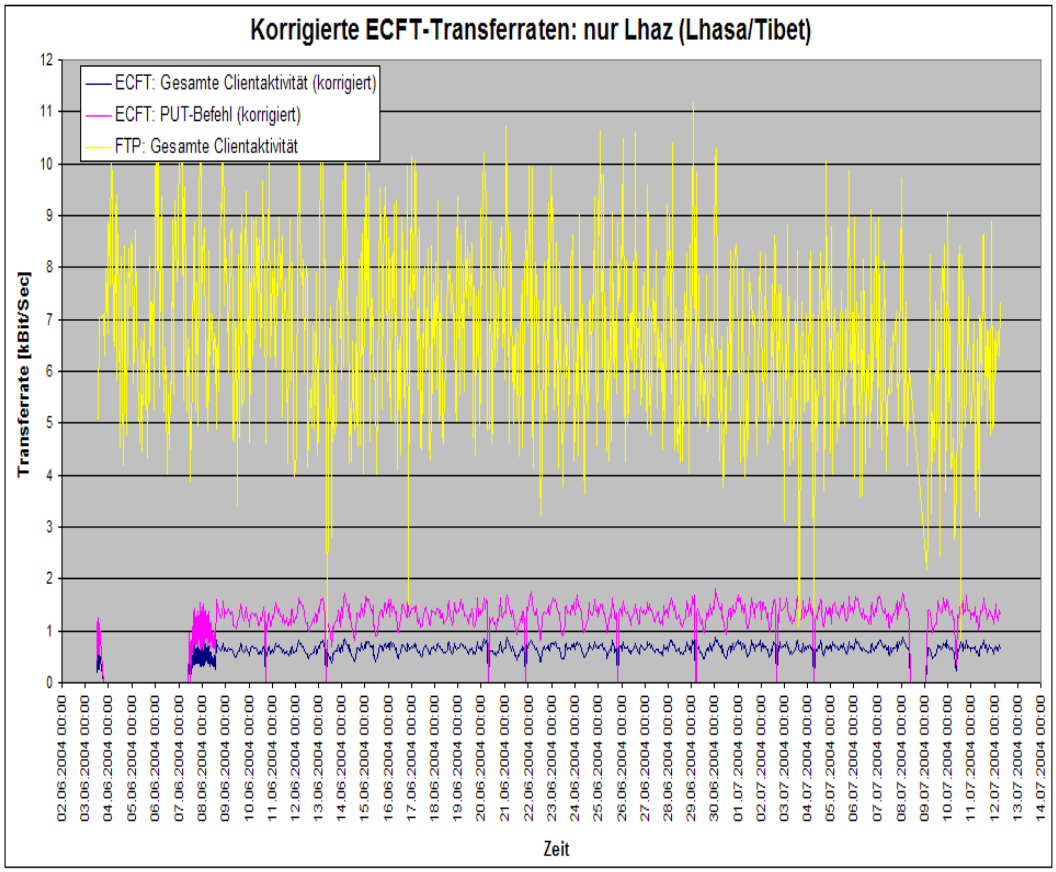

FTP-Transferzeiten (Lhasa/Tibet)
Gesamte Clientaktivität
Latenzzeit [msec]
Zeit
FTP-Transferraten (Lhasa/Tibet)
Gesamte Clientaktivität
Transferrate [KBit/Sec]
Zeit
ECFT-Transferzeiten (Lhasa/Tibet)
Gesamte Clientaktivität
PUT-Befehl
Latenzzeit [msec]
Zeit
ECFT-Transferraten (Lhasa/Tibet)
Gesamte Clientaktivität
PUT-Befehl
Transferrate [KBit/Sec]
Zeit

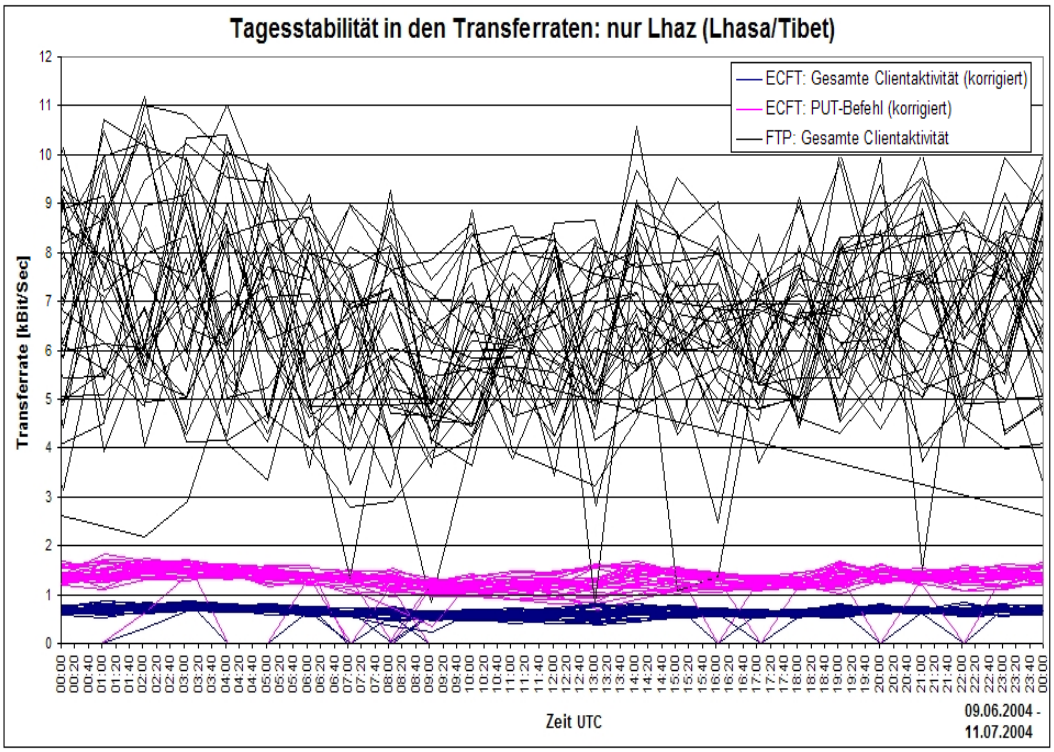

Korrektur der ECFT-Transferzeiten (Lhasa/Tibet)
Gesamte Clientaktivität
PUT-Befehl
Latenzzeit [msec]
Zeit


Korrektur der ECFT-Transferzeiten (Lhasa/Tibet)
Gesamte Clientaktivität
PUT-Befehl
Ausgleichskurve 1
Ausgleichskurve 2
Latenzzeit [msec]
Zeit


Korrektur der ECFT-Transferzeiten (Lhasa/Tibet)
Gesamte Clientaktivität (korrigiert)
PUT-Befehl (korrigiert)
Latenzzeit [msec]
Zeit


Korrektur der ECFT-Transferzeiten (Lhasa/Tibet)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Latenzzeit [msec]
Zeit


Korrigierte ECFT-Transferraten: nur Lhaz (Lhasa/Tibet)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Transferrate [kBit/Sec]
Zeit

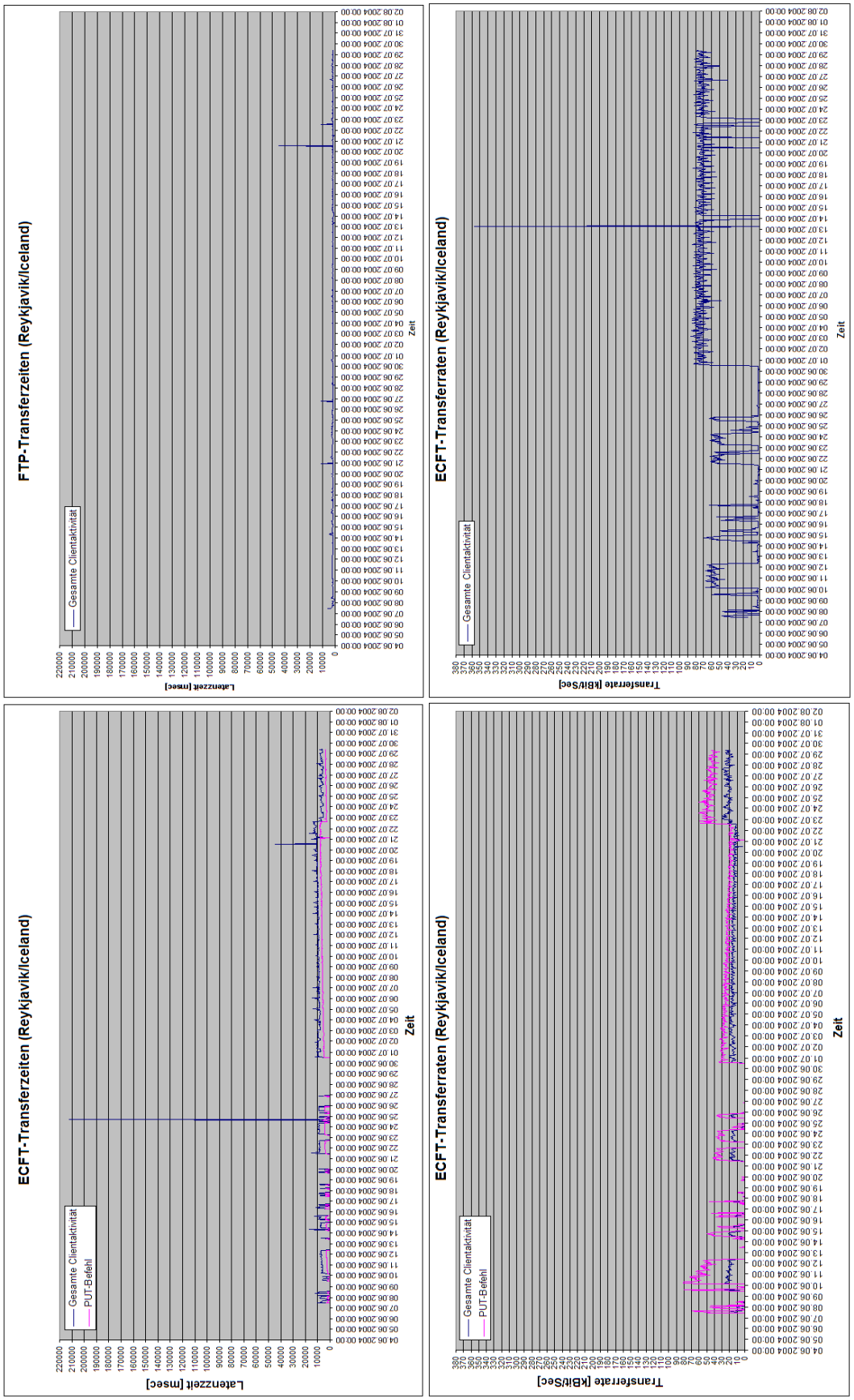

Tagesstabilität in den Transferraten: nur Lhaz (Lhasa/Tibet)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Transferrate [kBit/Sec]
Zeit UTC
09.06.2004 - 11.07.2004

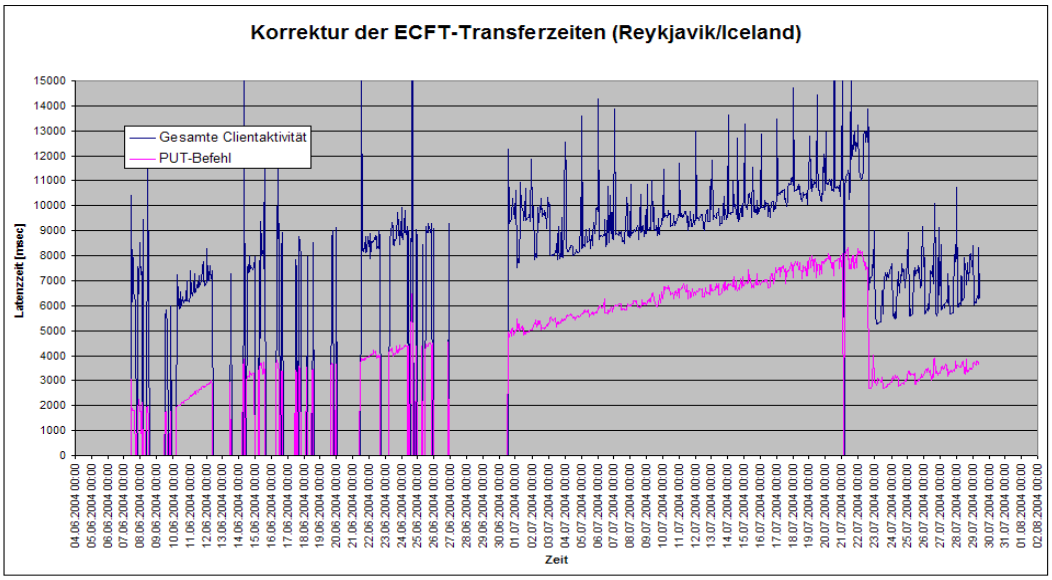
J.4 Reykjavik/Iceland

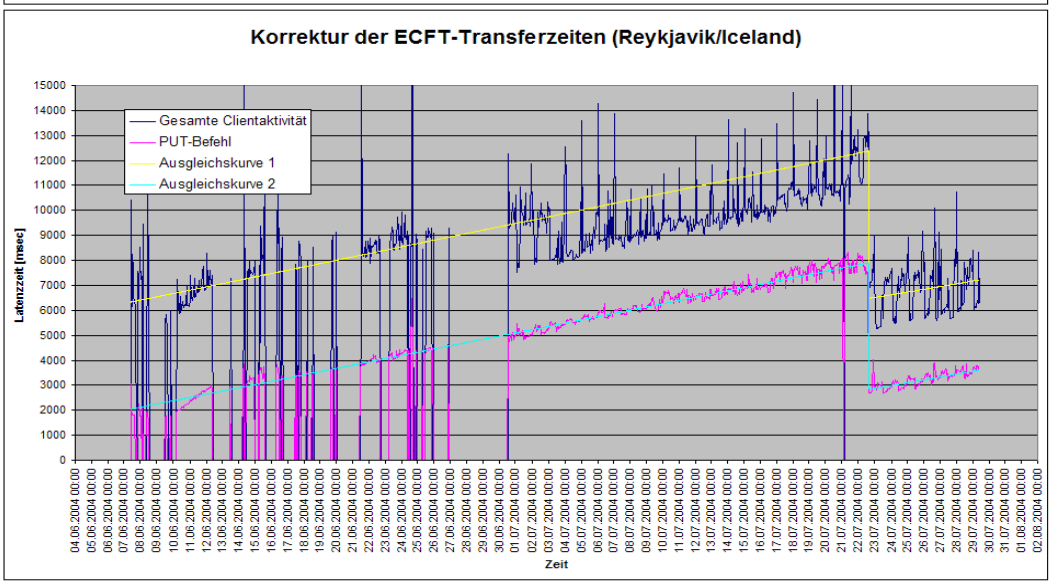
	ECFT				FTP			
Latenz: Gesamte Client-Aktivität [msec]	**MAX:**	39445,14	**RMS:**	**Normierter RMS:**	**MAX:**	3905,00	**RMS:**	**Normierter RMS:**
	MEAN:	6132,67	1654,12	0,27	**MEAN:**	2015,91	148,89	0,07
	MIN:	4823,98			**MIN:**	341,00		
Rate: Gesamte Client-Aktivität [KBit/Sec]]	**MAX:**	34,30	**RMS:**	**Normierter RMS:**	**MAX:**	357,26	**RMS:**	**Normierter RMS:**
	MEAN:	24,88	4,23	0,17	**MEAN:**	72,94	12,83	0,18
	MIN:	3,78			**MIN:**	39,95		

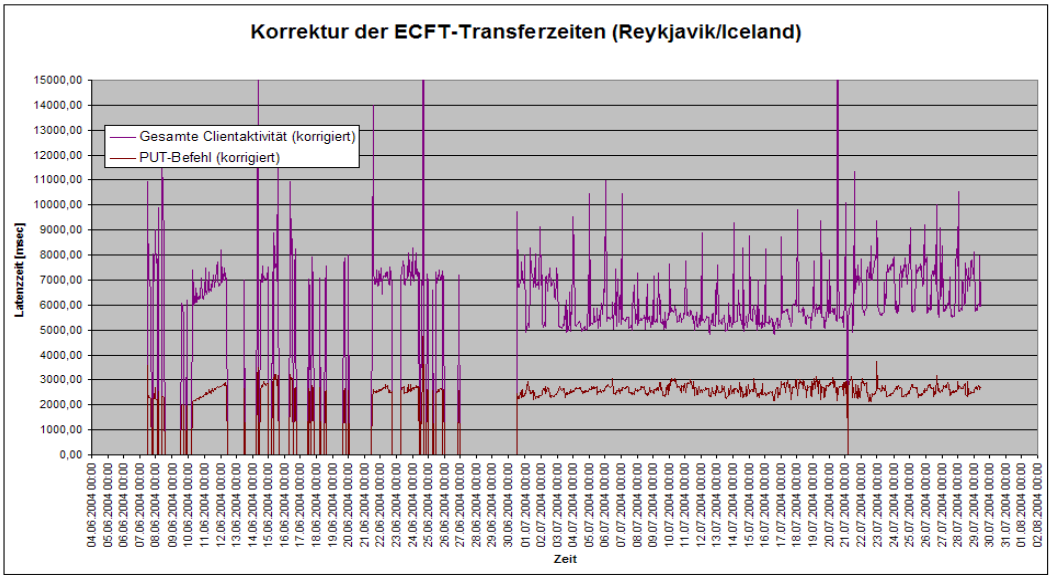
	ECFT				FTP			
Latenz: PUT-Befehl [msec]	**MAX:**	3735,58	**RMS:**	**Normierter RMS:**				
	MEAN:	2604,69	196,23	0,08				
	MIN:	2145,83						
Rate: PUT-Befehl [KBit/Sec]]	**MAX:**	74,48	**RMS:**	**Normierter RMS:**				
	MEAN:	57,12	6,47	0,11				
	MIN:	39,18						

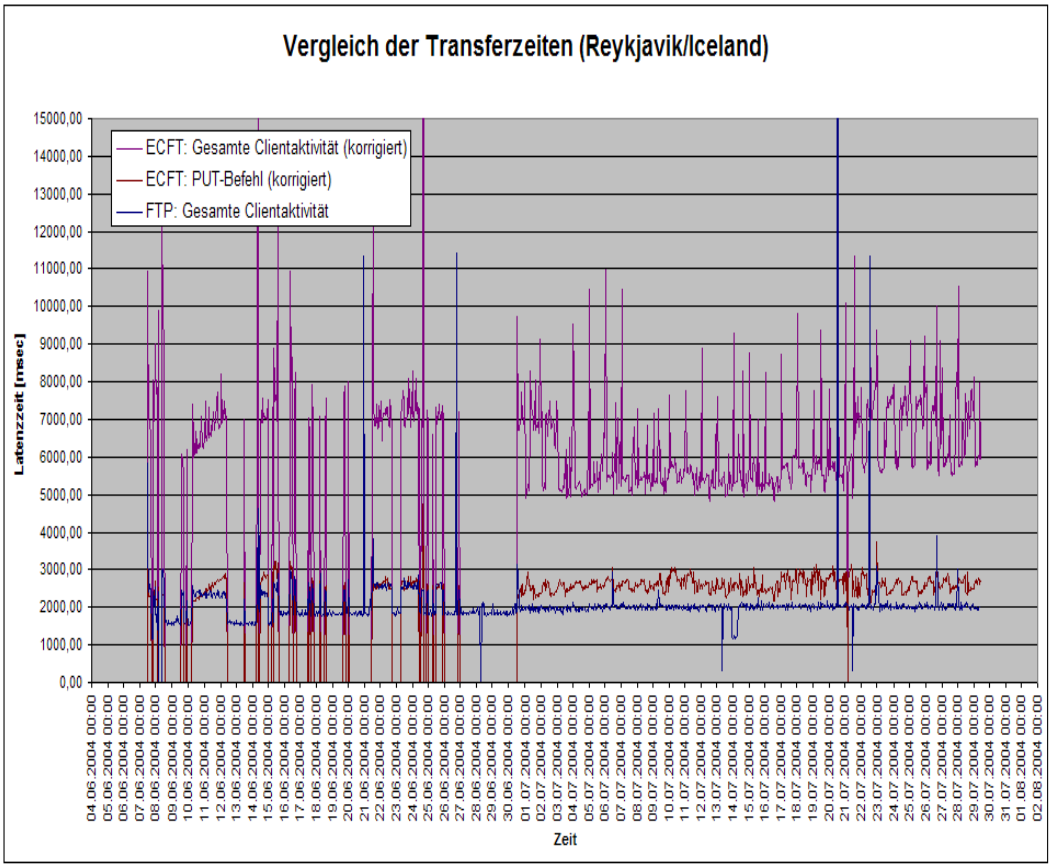
Datei-größen [Byte]	**MAX:**	21404,00	**RMS:**	**Normierter RMS:**
	MEAN:	18493,47	1574,01	0,09
	MIN:	13593,00		

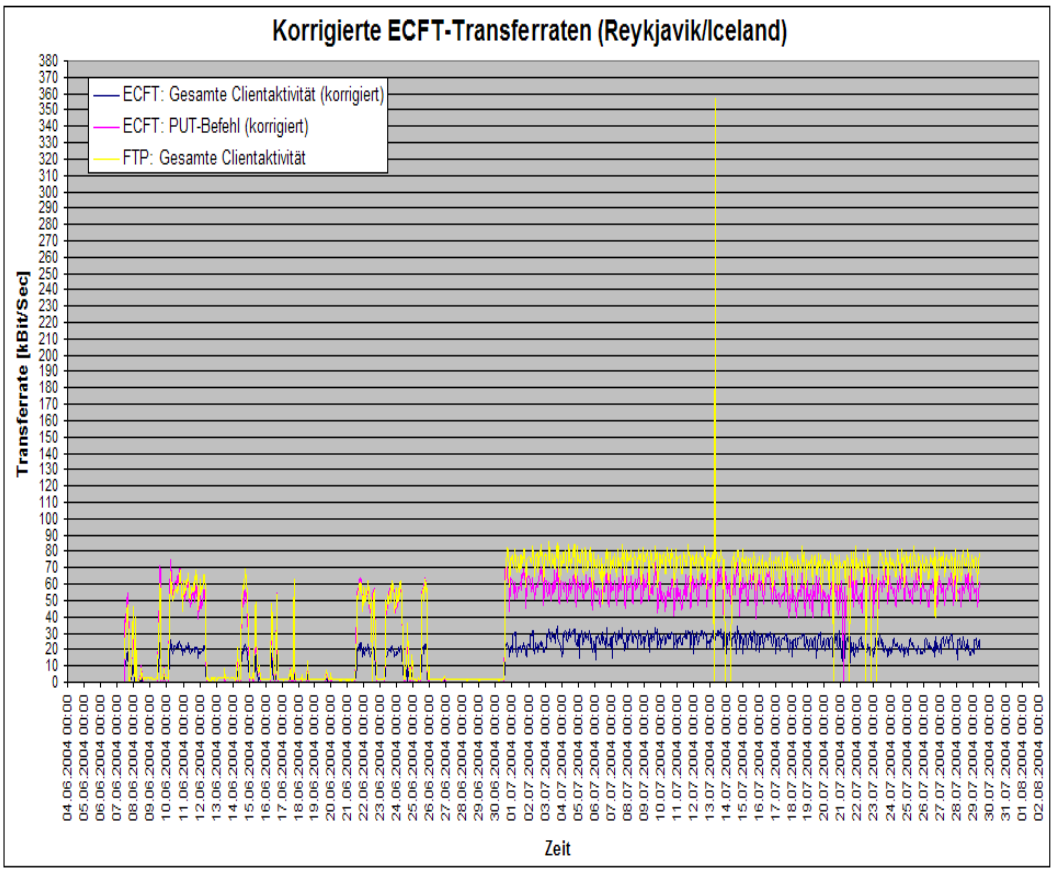

FTP-Transferzeiten (Reykjavik/Iceland)
Gesamte Clientaktivität
Latenzzeit [msec]
Zeit
ECFT-Transferraten (Reykjavik/Iceland)
Gesamte Clientaktivität
Transferrate [kBit/Sec]
Zeit
ECFT-Transferzeiten (Reykjavik/Iceland)
Gesamte Clientaktivität
PUT-Befehl
Latenzzeit [msec]
Zeit
ECFT-Transferraten (Reykjavik/Iceland)
Gesamte Clientaktivität
PUT-Befehl
Transferrate [kBit/Sec]
Zeit

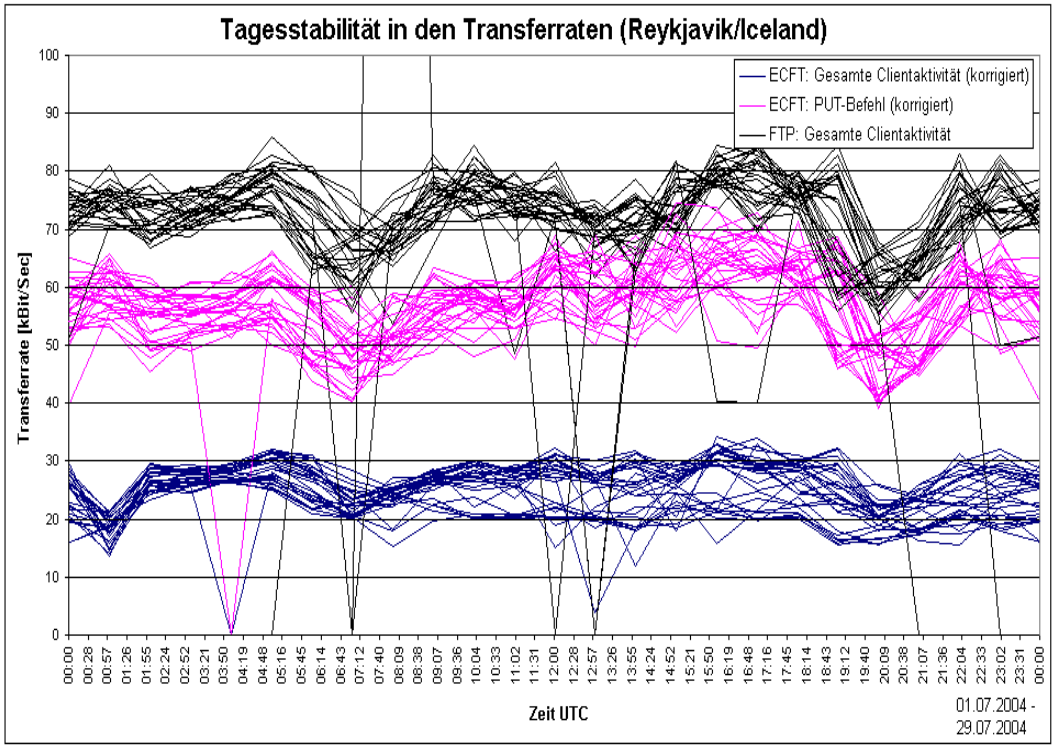

Korrektur der ECFT-Transferzeiten (Reykjavik/Iceland)
Gesamte Clientaktivität
PUT-Befehl
Latenzzeit [msec]
Zeit


Korrektur der ECFT-Transferzeiten (Reykjavik/Iceland)
Gesamte Clientaktivität
PUT-Befehl
Ausgleichskurve 1
Ausgleichskurve 2
Latenzzeit [msec]
Zeit


Korrektur der ECFT-Transferzeiten (Reykjavik/Iceland)
Gesamte Clientaktivität (korrigiert)
PUT-Befehl (korrigiert)
Latenzzeit [msec]
Zeit


Vergleich der Transferzeiten (Reykjavik/Iceland)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Latenzzeit [msec]
Zeit


Korrigierte ECFT-Transferraten (Reykjavik/Iceland)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Transferrate [kBit/Sec]
Zeit


Tagesstabilität in den Transferraten (Reykjavik/Iceland)
ECFT: Gesamte Clientaktivität (korrigiert)
ECFT: PUT-Befehl (korrigiert)
FTP: Gesamte Clientaktivität
Transferrate [kBit/Sec]
Zeit UTC
01.07.2004 - 29.07.2004

Anhang K

Sprachdokumentation A2X

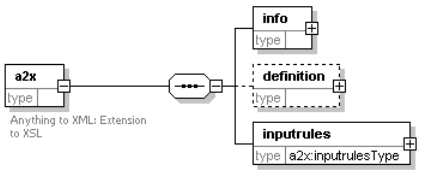
(Automatisch von der Entwicklungsumgebung XMLSpy von Altova erzeugte Dokumentation in HTML; Stand: 14.01.2005)

Schema **a2x.xsd**

schema location: C:\A2X\a2x.xsd
targetNamespace: **http://www.wettzell.ifag.de/a2x**

Elements	Complex types
a2x	comparisonType
applybreak	edgeType
comment	inputrulesType
const	knoteType
declare	typerefType
edge	varoperationType
operand	
typerep	

element **a2x**

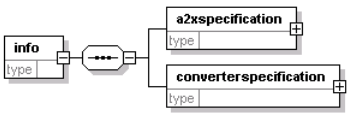
diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**info** **definition** **inputrules**
annotation	documentation Anything to XML: Extension to XSL

source

```
<xs:element name="a2x">
 <xs:annotation>
  <xs:documentation>Anything to XML: Extension to XSL</xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element name="info">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="a2xspecification">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="revision">
          <xs:complexType>
           <xs:attribute name="version" use="required">
            <xs:simpleType>
             <xs:restriction base="xs:string">
              <xs:enumeration value="1.0beta"/>
             </xs:restriction>
            </xs:simpleType>
           </xs:attribute>
           <xs:attribute name="date" type="xs:date" use="required"/>
           <xs:attribute name="homeurl" type="xs:anyURI" use="required"/>
          </xs:complexType>
         </xs:element>
         <xs:element name="comment" type="xs:string" minOccurs="0"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="convspecification">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="revision">
          <xs:complexType>
           <xs:attribute name="name" type="xs:string" use="required"/>
           <xs:attribute name="roottag" type="xs:string" use="required"/>
           <xs:attribute name="version" use="required">
            <xs:simpleType>
             <xs:restriction base="xs:string"/>
            </xs:simpleType>
           </xs:attribute>
           <xs:attribute name="date" type="xs:date" use="required"/>
           <xs:attribute name="institution" type="xs:string" use="optional"/>
           <xs:attribute name="developer" type="xs:string" use="required"/>
          </xs:complexType>
         </xs:element>
         <xs:element name="comment" type="xs:string" minOccurs="0"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="definition" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="a2x:typerep" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="a2x:declare" minOccurs="0"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="inputrules" type="a2x:inputrulesType"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **a2x/info**

diagram



namespace: http://www.wettzell.ifag.de/a2x

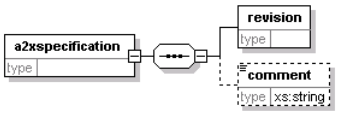
children: **a2xspecification** **convspecification**

source

```
<xs:element name="info">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="a2xspecification">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="revision">
       <xs:complexType>
        <xs:attribute name="version" use="required">
         <xs:simpleType>
          <xs:restriction base="xs:string">
           <xs:enumeration value="1.0beta"/>
          </xs:restriction>
         </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="date" type="xs:date" use="required"/>
        <xs:attribute name="homeurl" type="xs:anyURI" use="required"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="comment" type="xs:string" minOccurs="0"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="convspecification">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="revision">
       <xs:complexType>
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="roottag" type="xs:string" use="required"/>
        <xs:attribute name="version" use="required">
         <xs:simpleType>
          <xs:restriction base="xs:string"/>
         </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="date" type="xs:date" use="required"/>
        <xs:attribute name="institution" type="xs:string" use="optional"/>
        <xs:attribute name="developer" type="xs:string" use="required"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="comment" type="xs:string" minOccurs="0"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **a2x/info/a2xspecification**

diagram



namespace: http://www.wettzell.ifag.de/a2x

children: **revision** **comment**

source

```
<xs:element name="a2xspecification">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="revision">
    <xs:complexType>
     <xs:attribute name="version" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="1.0beta"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="date" type="xs:date" use="required"/>
     <xs:attribute name="homeurl" type="xs:anyURI" use="required"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="comment" type="xs:string" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

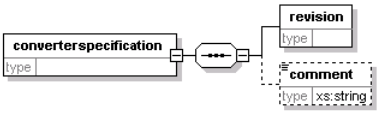
element **a2x/info/a2xspecification/revision**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
attributes	Name Type Use Default Fixed Annotation version xs:string required date xs:date required homeurl xs:anyURI required
source	<xs:element name="revision"> <xs:complexType> <xs:attribute name="version" use="required"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="1.0beta"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="date" type="xs:date" use="required"/> <xs:attribute name="homeurl" type="xs:anyURI" use="required"/> </xs:complexType> </xs:element>

element **a2x/info/a2xspecification/comment**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	**xs:string**
source	<xs:element name="comment" type="xs:string" minOccurs="0"/>

element **a2x/info/convspecification**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**revision comment**
source	<xs:element name="convspecification"> <xs:complexType> <xs:sequence> <xs:element name="revision"> <xs:complexType> <xs:attribute name="name" type="xs:string" use="required"/> <xs:attribute name="roottag" type="xs:string" use="required"/> <xs:attribute name="version" use="required"> <xs:simpleType> <xs:restriction base="xs:string"/> </xs:simpleType> </xs:attribute> <xs:attribute name="date" type="xs:date" use="required"/> <xs:attribute name="institution" type="xs:string" use="optional"/> <xs:attribute name="developer" type="xs:string" use="required"/> </xs:complexType> </xs:element> <xs:element name="comment" type="xs:string" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>

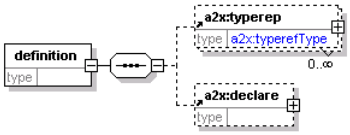
element **a2x/info/convspecification/revision**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
attributes	Name Type Use Default Fixed Annotation name xs:string required roottag xs:string required version xs:string required date xs:date required institution xs:string optional developer xs:string required
source	<xs:element name="revision"> <xs:complexType> <xs:attribute name="name" type="xs:string" use="required"/> <xs:attribute name="roottag" type="xs:string" use="required"/> <xs:attribute name="version" use="required"> <xs:simpleType> <xs:restriction base="xs:string"/> </xs:simpleType> </xs:attribute> <xs:attribute name="date" type="xs:date" use="required"/> <xs:attribute name="institution" type="xs:string" use="optional"/> <xs:attribute name="developer" type="xs:string" use="required"/> </xs:complexType> </xs:element>

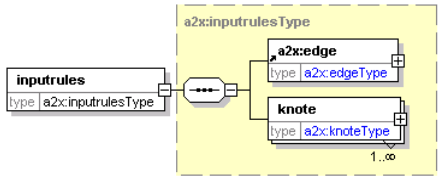
element **a2x/info/convspecification/comment**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	**xs:string**
source	<xs:element name="comment" type="xs:string" minOccurs="0"/>

element **a2x/definition**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**a2x:typerep** **a2x:declare**
source	<xs:element name="definition" minOccurs="0"> <xs:complexType> <xs:sequence> <xs:element ref="a2x:typerep" minOccurs="0" maxOccurs="unbounded"/> <xs:element ref="a2x:declare" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>

element **a2x/inputrules**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	**a2x:inputrulesType**
children	**a2x:edge** **knote**
source	<xs:element name="inputrules" type="a2x:inputrulesType"/>

element **applybreak**

diagram	applybreak type
namespace	http://www.wettzell.ifag.de/a2x
used by	complexType **edgeType**
source	<xs:element name="applybreak"/>

element **comment**

diagram	comment type xs:string
namespace	http://www.wettzell.ifag.de/a2x
type	**xs:string**
used by	elements **edgeType/dummy** **edgeType/field** **edgeType/pattern** **declare/variable**
source	<xs:element name="comment" type="xs:string"/>

element **const**

diagram	const type
namespace	http://www.wettzell.ifag.de/a2x
used by	complexTypes **comparisonType** **comparisonType** **varoperationType** **varoperationType**

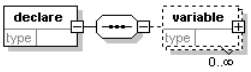
attributes

Name	Type	Use	Default	Fixed	Annotation
value	xs:anySimpleType	required			
cast	xs:string	optional			

source

```
<xs:element name="const">
  <xs:complexType>
    <xs:attribute name="value" type="xs:anySimpleType" use="required"/>
    <xs:attribute name="cast" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="ushort"/>
          <xs:enumeration value="uint"/>
          <xs:enumeration value="ulong"/>
          <xs:enumeration value="short"/>
          <xs:enumeration value="int"/>
          <xs:enumeration value="long"/>
          <xs:enumeration value="float"/>
          <xs:enumeration value="double"/>
          <xs:enumeration value="char"/>
          <xs:enumeration value="string"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

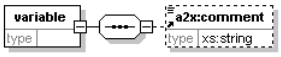
element **declare**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**variable**
used by	element **a2x/definition**

source

```
<xs:element name="declare">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="variable" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="a2x:comment" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="name" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:ID">
        <xs:pattern value="VAR.*"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="tag" type="xs:NMTOKEN" use="optional"/>
     <xs:attribute name="type" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="ushort"/>
        <xs:enumeration value="uint"/>
        <xs:enumeration value="ulong"/>
        <xs:enumeration value="short"/>
        <xs:enumeration value="int"/>
        <xs:enumeration value="long"/>
        <xs:enumeration value="float"/>
        <xs:enumeration value="double"/>
        <xs:enumeration value="char"/>
        <xs:enumeration value="string"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="init" type="xs:anySimpleType" use="optional"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

element **declare/variable**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**a2x:comment**

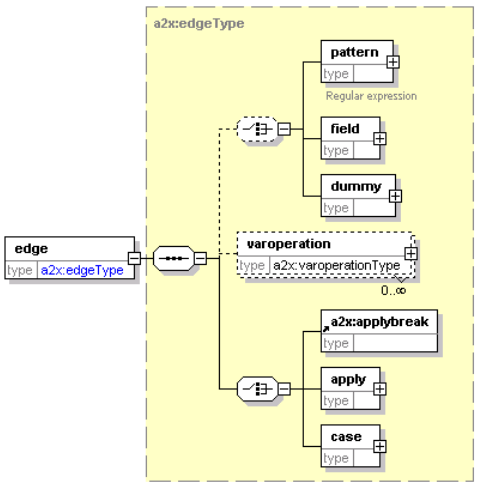
attributes

Name	Type	Use	Default	Fixed	Annotation
name	xs:ID	required			
tag	xs:NMTOKEN	optional			
type	xs:string	required			
init	xs:anySimpleType	optional			

source

```
<xs:element name="variable" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="a2x:comment" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="name" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:ID">
     <xs:pattern value="VAR.*"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="tag" type="xs:NMTOKEN" use="optional"/>
  <xs:attribute name="type" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="ushort"/>
     <xs:enumeration value="uint"/>
     <xs:enumeration value="ulong"/>
     <xs:enumeration value="short"/>
     <xs:enumeration value="int"/>
     <xs:enumeration value="long"/>
     <xs:enumeration value="float"/>
     <xs:enumeration value="double"/>
     <xs:enumeration value="char"/>
     <xs:enumeration value="string"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="init" type="xs:anySimpleType" use="optional"/>
 </xs:complexType>
</xs:element>
```

element **edge**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	extension of **a2x:edgeType**
children	**pattern field dummy varoperation a2x:applybreak apply case**
used by	complexTypes **inputrulesType knoteType**

attributes

Name	Type	Use	Default	Fixed	Annotation
TaggingOn	xs:boolean	optional	true		

source

```
<xs:element name="edge">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="a2x:edgeType">
    <xs:attribute name="TaggingOn" use="optional" default="true">
     <xs:simpleType>
      <xs:restriction base="xs:boolean"/>
     </xs:simpleType>
    </xs:attribute>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```

element **operand**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
used by	complexTypes **varoperationType varoperationType**

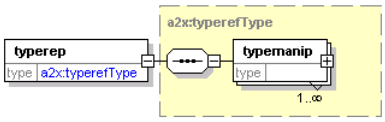
attributes

Name	Type	Use	Default	Fixed	Annotation
variable	xs:IDREF	required			
cast	xs:string	optional			

source

```
<xs:element name="operand">
 <xs:complexType>
  <xs:attribute name="variable" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:IDREF">
     <xs:pattern value="VAR.*"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="cast" use="optional">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="ushort"/>
     <xs:enumeration value="uint"/>
     <xs:enumeration value="ulong"/>
     <xs:enumeration value="short"/>
     <xs:enumeration value="int"/>
     <xs:enumeration value="long"/>
     <xs:enumeration value="float"/>
     <xs:enumeration value="double"/>
     <xs:enumeration value="char"/>
     <xs:enumeration value="string"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:element>
```

element **typerep**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	extension of **a2x:typerefType**
children	**typemanip**
used by	element **a2x/definition**

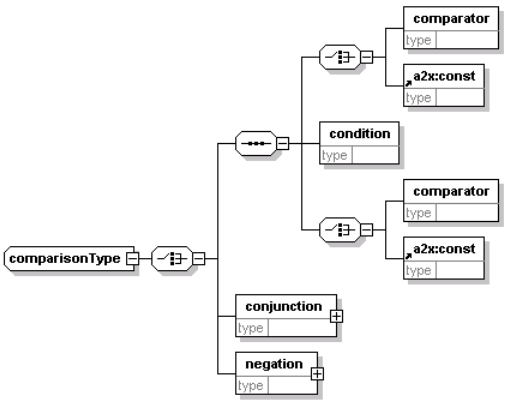
attributes

Name	Type	Use	Default	Fixed	Annotation
representation	xs:ID	required			
exttyperep	xs:anyURI	optional			

source

```
<xs:element name="typerep">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="a2x:typerefType">
    <xs:attribute name="representation" use="required">
     <xs:simpleType>
      <xs:restriction base="xs:ID">
       <xs:pattern value="REP.*"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="exttyperep" type="xs:anyURI" use="optional"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```

complexType **comparisonType**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**comparator** **a2x:const** **condition** **comparator** **conjunction** **negation**
used by	elements **edgeType/case/comparison** **comparisonType/conjunction/composition** **comparisonType/conjunction/composition** **comparisonType/negation/composition**

source

```
<xs:complexType name="comparisonType">
 <xs:choice>
  <xs:sequence>
   <xs:choice>
    <xs:element name="comparator">
     <xs:complexType>
      <xs:attribute name="variable" type="xs:IDREF" use="required"/>
      <xs:attribute name="cast" use="optional">
       <xs:simpleType>
        <xs:restriction base="xs:string">
         <xs:enumeration value="ushort"/>
         <xs:enumeration value="uint"/>
         <xs:enumeration value="ulong"/>
         <xs:enumeration value="short"/>
         <xs:enumeration value="int"/>
         <xs:enumeration value="long"/>
         <xs:enumeration value="float"/>
         <xs:enumeration value="double"/>
         <xs:enumeration value="char"/>
         <xs:enumeration value="string"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
     </xs:complexType>
    </xs:element>
    <xs:element ref="a2x:const"/>
   </xs:choice>
   <xs:element name="condition">
    <xs:complexType>
     <xs:attribute name="constraint" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="lt"/>
        <xs:enumeration value="eq"/>
        <xs:enumeration value="gt"/>
        <xs:enumeration value="!lt"/>
        <xs:enumeration value="!eq"/>
        <xs:enumeration value="!gt"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
    </xs:complexType>
   </xs:element>
   <xs:choice>
    <xs:element name="comparator">
     <xs:complexType>
      <xs:attribute name="variable" type="xs:IDREF" use="required"/>
      <xs:attribute name="cast" use="optional">
       <xs:simpleType>
        <xs:restriction base="xs:string">
         <xs:enumeration value="ushort"/>
         <xs:enumeration value="uint"/>
         <xs:enumeration value="ulong"/>
         <xs:enumeration value="short"/>
         <xs:enumeration value="int"/>
         <xs:enumeration value="long"/>
         <xs:enumeration value="float"/>
         <xs:enumeration value="double"/>
         <xs:enumeration value="char"/>
         <xs:enumeration value="string"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
     </xs:complexType>
    </xs:element>
    <xs:element ref="a2x:const"/>
   </xs:choice>
  </xs:sequence>
  <xs:element name="conjunction">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="composition" type="a2x:comparisonType"/>
     <xs:choice>
      <xs:element name="and"/>
      <xs:element name="or"/>
     </xs:choice>
     <xs:element name="composition" type="a2x:comparisonType"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="negation">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="not">
      <xs:complexType/>
     </xs:element>
     <xs:element name="composition" type="a2x:comparisonType"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:choice>
</xs:complexType>
```

element **comparisonType/comparator**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
attributes	Name Type Use Default Fixed Annotation variable xs:IDREF required cast xs:string optional
source	<xs:element name="comparator"> <xs:complexType> <xs:attribute name="variable" type="xs:IDREF" use="required"/> <xs:attribute name="cast" use="optional"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="ushort"/> <xs:enumeration value="uint"/> <xs:enumeration value="ulong"/> <xs:enumeration value="short"/> <xs:enumeration value="int"/> <xs:enumeration value="long"/> <xs:enumeration value="float"/> <xs:enumeration value="double"/> <xs:enumeration value="char"/> <xs:enumeration value="string"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element>

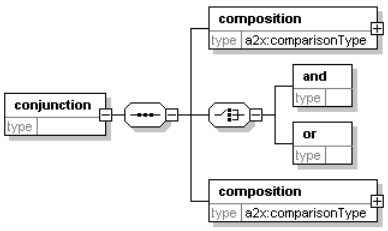
element **comparisonType/condition**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
attributes	Name Type Use Default Fixed Annotation constraint xs:string required
source	<xs:element name="condition"> <xs:complexType> <xs:attribute name="constraint" use="required"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="lt"/> <xs:enumeration value="eq"/> <xs:enumeration value="gt"/> <xs:enumeration value="!lt"/> <xs:enumeration value="!eq"/> <xs:enumeration value="!gt"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element>

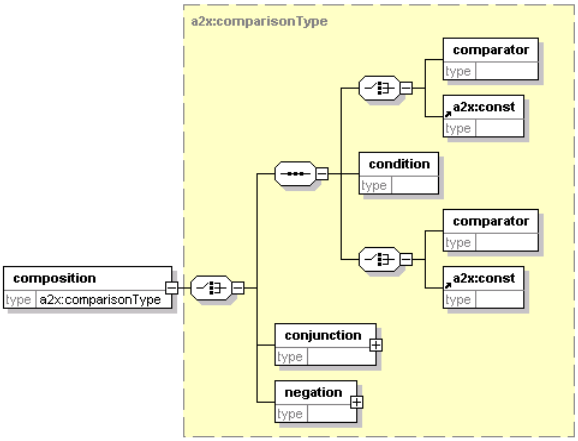
element **comparisonType/comparator**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
attributes	Name Type Use Default Fixed Annotation variable xs:IDREF required cast xs:string optional
source	<xs:element name="comparator"> <xs:complexType> <xs:attribute name="variable" type="xs:IDREF" use="required"/> <xs:attribute name="cast" use="optional"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="ushort"/> <xs:enumeration value="uint"/> <xs:enumeration value="ulong"/> <xs:enumeration value="short"/> <xs:enumeration value="int"/> <xs:enumeration value="long"/> <xs:enumeration value="float"/> <xs:enumeration value="double"/> <xs:enumeration value="char"/> <xs:enumeration value="string"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element>

element **comparisonType/conjunction**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	composition and or composition
source	<xs:element name="conjunction"> <xs:complexType> <xs:sequence> <xs:element name="composition" type="a2x:comparisonType"/> <xs:choice> <xs:element name="and"/> <xs:element name="or"/> </xs:choice> <xs:element name="composition" type="a2x:comparisonType"/> </xs:sequence> </xs:complexType> </xs:element>

element **comparisonType/conjunction/composition**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	a2x:comparisonType
children	comparator a2x:const condition comparator conjunction negation
source	<xs:element name="composition" type="a2x:comparisonType"/>

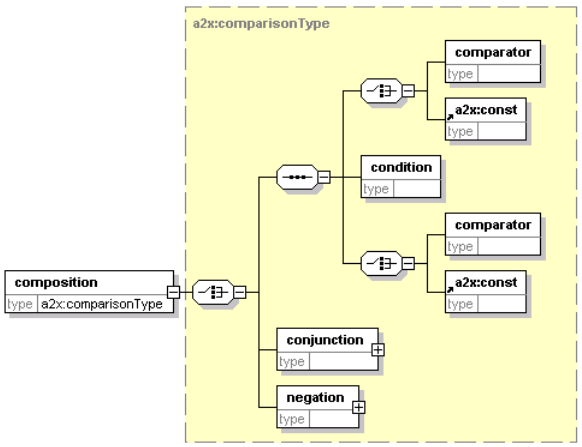
element **comparisonType/conjunction/and**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
source	<xs:element name="and"/>

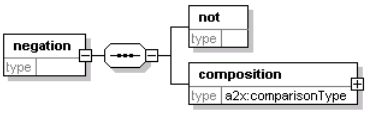
element **comparisonType/conjunction/or**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
source	<xs:element name="or"/>

element **comparisonType/conjunction/composition**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	a2x:comparisonType
children	comparator a2x:const condition comparator conjunction negation
source	<xs:element name="composition" type="a2x:comparisonType"/>

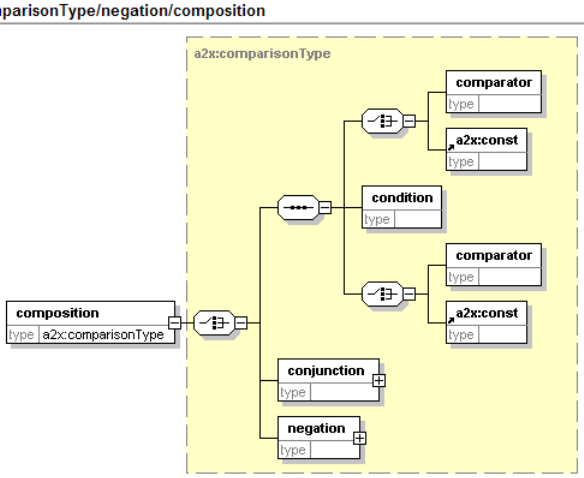
element **comparisonType/negation**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	not composition
source	<xs:element name="negation"> <xs:complexType> <xs:sequence> <xs:element name="not"> <xs:complexType/> </xs:element> <xs:element name="composition" type="a2x:comparisonType"/> </xs:sequence> </xs:complexType> </xs:element>

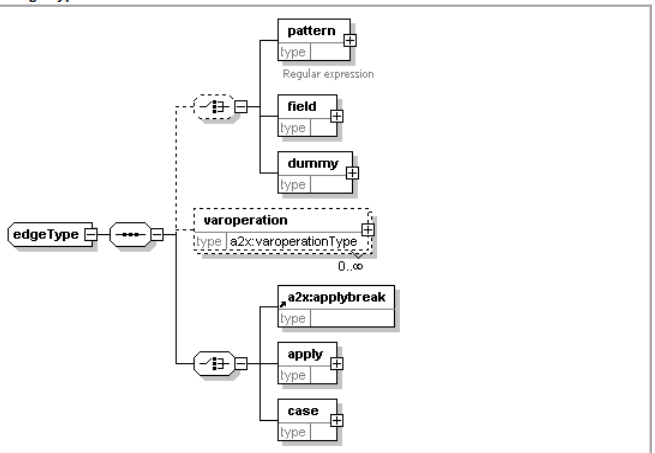
element **comparisonType/negation/not**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
source	<xs:element name="not"> <xs:complexType/> </xs:element>

element **comparisonType/negation/composition**

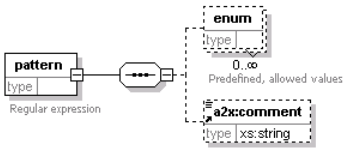
diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	a2x:comparisonType
children	comparator a2x:const condition comparator conjunction negation
source	<xs:element name="composition" type="a2x:comparisonType"/>

complexType **edgeType**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	pattern field dummy varoperation a2x:applybreak apply case
used by	elements edge edgeType/case/else edgeType/case/if
source	<xs:complexType name="edgeType"> <xs:sequence> <xs:choice minOccurs="0"> <xs:element name="pattern"> <xs:annotation> <xs:documentation>Regular expression</xs:documentation> </xs:annotation> <xs:complexType> <xs:sequence>

```
        <xs:element name="enum" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>Predefined, allowed values</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:attribute name="value" type="xs:anySimpleType" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element ref="a2x:comment" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="expression" type="xs:string" use="required"/>
      <xs:attribute name="variable" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:IDREF">
            <xs:pattern value="VAR.*"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="typerep" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:IDREF">
            <xs:pattern value="REP.*"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="field">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="enum" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="value" type="xs:anySimpleType" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element ref="a2x:comment" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="length" type="xs:long" use="required"/>
      <xs:attribute name="variable" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:IDREF">
            <xs:pattern value="VAR.*"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="typerep" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:IDREF">
            <xs:pattern value="REP.*"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="dummy">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="a2x:comment" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="event" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="EOF"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="variable" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:IDREF">
            <xs:pattern value="VAR.*"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
 </xs:choice>
 <xs:element name="varoperation" type="a2x:varoperationType" minOccurs="0" maxOccurs="unbounded"/>
 <xs:choice>
  <xs:element ref="a2x:applybreak"/>
  <xs:element name="apply">
    <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="varoperation" type="a2x:varoperationType"/>
      </xs:sequence>
      <xs:attribute name="next" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:IDREF">
            <xs:pattern value="ID.*"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="case">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="comparison" type="a2x:comparisonType"/>
        <xs:element name="if" type="a2x:edgeType"/>
        <xs:element name="else" type="a2x:edgeType" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
 </xs:choice>
</xs:sequence>
</xs:complexType>
```

element **edgeType/pattern**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**enum** **a2x:comment**

attributes

Name	Type	Use	Default	Fixed	Annotation
expression	xs:string	required			
variable	xs:IDREF	optional			
typerep	xs:IDREF	optional			

annotation: documentation Regular expression

source

```
<xs:element name="pattern">
 <xs:annotation>
  <xs:documentation>Regular expression</xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element name="enum" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
     <xs:documentation>Predefined, allowed values</xs:documentation>
    </xs:annotation>
    <xs:complexType>
     <xs:attribute name="value" type="xs:anySimpleType" use="required"/>
    </xs:complexType>
   </xs:element>
   <xs:element ref="a2x:comment" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="expression" type="xs:string" use="required"/>
  <xs:attribute name="variable" use="optional">
   <xs:simpleType>
    <xs:restriction base="xs:IDREF">
     <xs:pattern value="VAR.*"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="typerep" use="optional">
   <xs:simpleType>
    <xs:restriction base="xs:IDREF">
     <xs:pattern value="REP.*"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:element>
```

element **edgeType/pattern/enum**

diagram	
namespace	http://www.wettzell.ifag.de/a2x

attributes

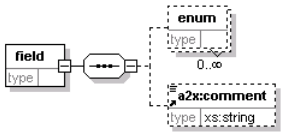
Name	Type	Use	Default	Fixed	Annotation
value	xs:anySimpleType	required			

annotation: documentation Predefined, allowed values

source

```
<xs:element name="enum" minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:documentation>Predefined, allowed values</xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:attribute name="value" type="xs:anySimpleType" use="required"/>
 </xs:complexType>
</xs:element>
```

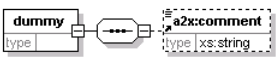
element **edgeType/field**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	enum a2x:comment
attributes	Name Type Use Default Fixed Annotation length xs:long required variable xs:IDREF optional typerep xs:IDREF optional
source	<xs:element name="field"> <xs:complexType> <xs:sequence> <xs:element name="enum" minOccurs="0" maxOccurs="unbounded"> <xs:complexType> <xs:attribute name="value" type="xs:anySimpleType" use="required"/> </xs:complexType> </xs:element> <xs:element ref="a2x:comment" minOccurs="0"/> </xs:sequence> <xs:attribute name="length" type="xs:long" use="required"/> <xs:attribute name="variable" use="optional"> <xs:simpleType> <xs:restriction base="xs:IDREF"> <xs:pattern value="VAR.*"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="typerep" use="optional"> <xs:simpleType> <xs:restriction base="xs:IDREF"> <xs:pattern value="REP.*"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element>

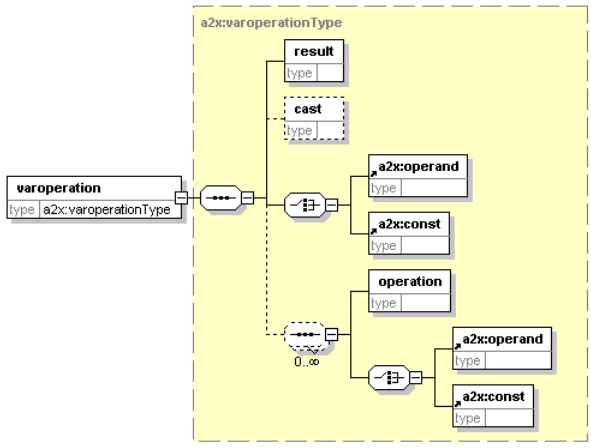
element **edgeType/field/enum**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
attributes	Name Type Use Default Fixed Annotation value xs:anySimpleType required
source	<xs:element name="enum" minOccurs="0" maxOccurs="unbounded"> <xs:complexType> <xs:attribute name="value" type="xs:anySimpleType" use="required"/> </xs:complexType> </xs:element>

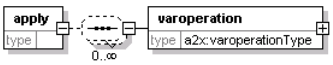
element **edgeType/dummy**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**a2x:comment**
attributes	Name Type Use Default Fixed Annotation event xs:string optional variable xs:IDREF optional
source	<xs:element name="dummy"> <xs:complexType> <xs:sequence> <xs:element ref="a2x:comment" minOccurs="0"/> </xs:sequence> <xs:attribute name="event" use="optional"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="EOF"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="variable" use="optional"> <xs:simpleType> <xs:restriction base="xs:IDREF"> <xs:pattern value="VAR.*"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element>

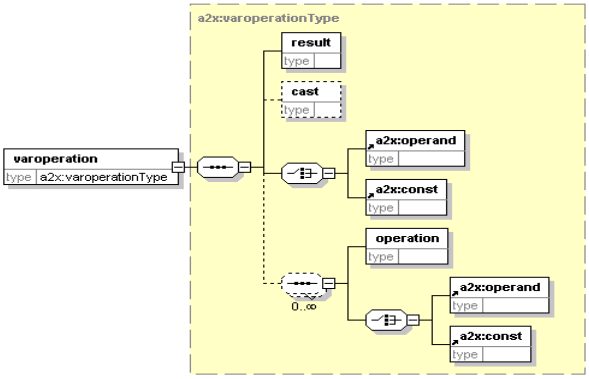
element **edgeType/varoperation**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	**a2x:varoperationType**
children	**result** **cast** **a2x:operand** **a2x:const** **operation**
source	<xs:element name="varoperation" type="a2x:varoperationType" minOccurs="0" maxOccurs="unbounded"/>

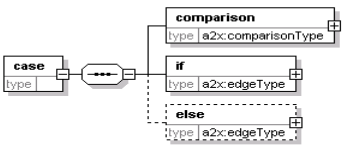
element **edgeType/apply**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	varoperation
attributes	Name Type Use Default Fixed Annotation next xs:IDREF required
source	<xs:element name="apply"> <xs:complexType> <xs:sequence minOccurs="0" maxOccurs="unbounded"> <xs:element name="varoperation" type="a2x:varoperationType"/> </xs:sequence> <xs:attribute name="next" use="required"> <xs:simpleType> <xs:restriction base="xs:IDREF"> <xs:pattern value="ID.*"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element>

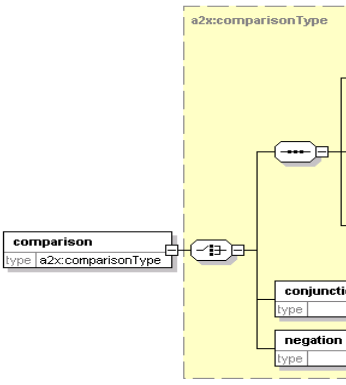
element **edgeType/apply/varoperation**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	a2x:varoperationType
children	result cast a2x:operand a2x:const operation
source	<xs:element name="varoperation" type="a2x:varoperationType"/>

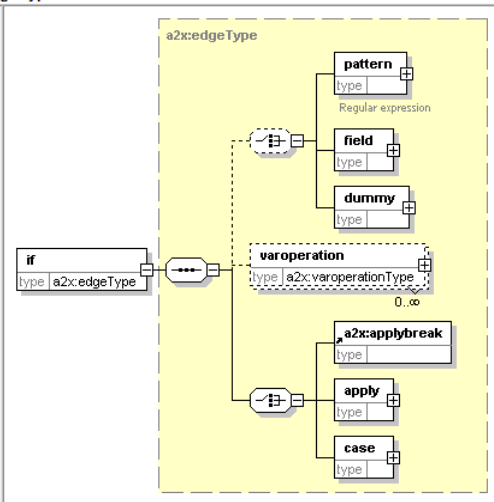
element **edgeType/case**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	comparison if else
source	<xs:element name="case"> <xs:complexType> <xs:sequence> <xs:element name="comparison" type="a2x:comparisonType"/> <xs:element name="if" type="a2x:edgeType"/> <xs:element name="else" type="a2x:edgeType" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>

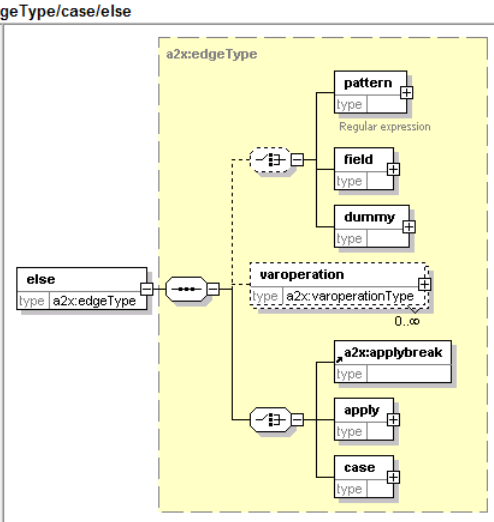
element **edgeType/case/comparison**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	a2x:comparisonType
children	comparator a2x:const condition comparator conjunction negation
source	<xs:element name="comparison" type="a2x:comparisonType"/>

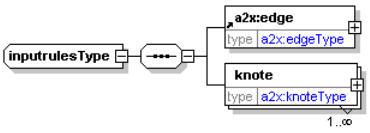
element **edgeType/case/if**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	a2x:edgeType
children	pattern field dummy varoperation a2x:applybreak apply case
source	<xs:element name="if" type="a2x:edgeType"/>

element **edgeType/case/else**

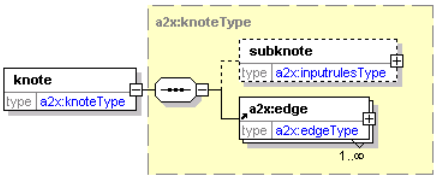
diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	a2x:edgeType
children	pattern field dummy varoperation a2x:applybreak apply case
source	<xs:element name="else" type="a2x:edgeType" minOccurs="0"/>

complexType **inputrulesType**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**a2x:edge** **knote**
used by	elements **a2x/inputrules** **knoteType/subknote**

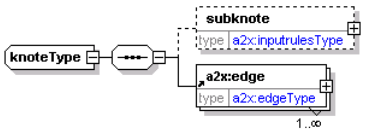
source	`<xs:complexType name="inputrulesType">`
	`<xs:sequence>`
	`<xs:element ref="a2x:edge"/>`
	`<xs:element name="knote" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:complexContent>`
	`<xs:extension base="a2x:knoteType">`
	`<xs:attribute name="knoteID" use="required">`
	`<xs:simpleType>`
	`<xs:restriction base="xs:ID">`
	`<xs:pattern value="ID.*"/>`
	`</xs:restriction>`
	`</xs:simpleType>`
	`</xs:attribute>`
	`<xs:attribute name="name" type="xs:string" use="optional"/>`
	`<xs:attribute name="identifier" type="xs:ID" use="optional"/>`
	`</xs:extension>`
	`</xs:complexContent>`
	`</xs:complexType>`
	`</xs:element>`
	`</xs:sequence>`
	`</xs:complexType>`

element **inputrulesType/knote**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	extension of **a2x:knoteType**
children	**subknote** **a2x:edge**
attributes	Name Type Use Default Fixed Annotation knoteID xs:ID required name xs:string optional identifier xs:ID optional

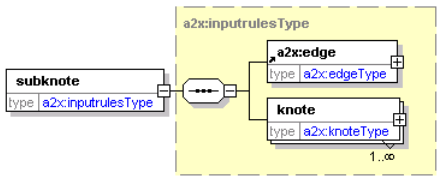
source	`<xs:element name="knote" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:complexContent>`
	`<xs:extension base="a2x:knoteType">`
	`<xs:attribute name="knoteID" use="required">`
	`<xs:simpleType>`
	`<xs:restriction base="xs:ID">`
	`<xs:pattern value="ID.*"/>`
	`</xs:restriction>`
	`</xs:simpleType>`
	`</xs:attribute>`
	`<xs:attribute name="name" type="xs:string" use="optional"/>`
	`<xs:attribute name="identifier" type="xs:ID" use="optional"/>`
	`</xs:extension>`
	`</xs:complexContent>`
	`</xs:complexType>`
	`</xs:element>`

complexType **knoteType**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**subknote** **a2x:edge**
used by	element **inputrulesType/knote**

source	`<xs:complexType name="knoteType">`
	`<xs:sequence>`
	`<xs:element name="subknote" minOccurs="0">`
	`<xs:complexType>`
	`<xs:complexContent>`
	`<xs:extension base="a2x:inputrulesType">`
	`<xs:attribute name="tagblock" type="xs:NMTOKEN" use="optional"/>`
	`<xs:attribute name="ReadFromVariable" use="optional">`
	`<xs:simpleType>`
	`<xs:restriction base="xs:IDREF">`
	`<xs:pattern value="VAR.*"/>`
	`</xs:restriction>`
	`</xs:simpleType>`
	`</xs:attribute>`
	`</xs:extension>`
	`</xs:complexContent>`
	`</xs:complexType>`
	`</xs:element>`
	`<xs:element ref="a2x:edge" maxOccurs="unbounded"/>`
	`</xs:sequence>`
	`</xs:complexType>`

element **knoteType/subknote**

diagram	
namespace	http://www.wettzell.ifag.de/a2x
type	extension of **a2x:inputrulesType**
children	**a2x:edge** **knote**

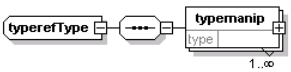
attributes

Name	Type	Use	Default	Fixed	Annotation
tagblock	xs:NMTOKEN	optional			
ReadFromVariable	xs:IDREF	optional			

source


```
<xs:element name="subknote" minOccurs="0">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="a2x:inputrulesType">
    <xs:attribute name="tagblock" type="xs:NMTOKEN" use="optional"/>
    <xs:attribute name="ReadFromVariable" use="optional">
     <xs:simpleType>
      <xs:restriction base="xs:IDREF">
       <xs:pattern value="VAR.*"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```

complexType **typerefType**

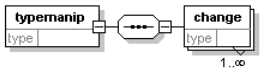
diagram	
namespace	http://www.wettzell.ifag.de/a2x
children	**typemanip**
used by	element **typerep**

source

```
<xs:complexType name="typerefType">
 <xs:sequence>
  <xs:element name="typemanip" maxOccurs="unbounded">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="change" maxOccurs="unbounded">
      <xs:complexType>
       <xs:attribute name="srcbytepos" use="required">
        <xs:simpleType>
         <xs:restriction base="xs:nonNegativeInteger">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="7"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="srcbitpos" use="required">
        <xs:simpleType>
         <xs:restriction base="xs:nonNegativeInteger">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="7"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="dstbytepos" use="required">
        <xs:simpleType>
         <xs:restriction base="xs:nonNegativeInteger">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="7"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="dstbitpos" use="required">
        <xs:simpleType>
         <xs:restriction base="xs:nonNegativeInteger">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="7"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:attribute>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
    <xs:attribute name="type" use="required">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:pattern value="ushort"/>
       <xs:pattern value="uint"/>
       <xs:pattern value="ulong"/>
       <xs:pattern value="short"/>
       <xs:pattern value="int"/>
       <xs:pattern value="long"/>
       <xs:pattern value="float"/>
       <xs:pattern value="double"/>
       <xs:pattern value="char"/>
       <xs:pattern value="string"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="srctypedef" use="optional">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:pattern value="WINx86"/>
       <xs:pattern value="LINUXx86"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="dsttypedef" use="optional">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:pattern value="WINx86"/>
       <xs:pattern value="LINUXx86"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="exttyperep" type="xs:anyURI" use="optional"/>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
```

element **typerefType/typemanip**

diagram



namespace: http://www.wettzell.ifag.de/a2x

children: change

attributes

Name	Type	Use	Default	Fixed	Annotation
type	xs:string	required			
srctypedef	xs:string	optional			
dsttypedef	xs:string	optional			
exttyperep	xs:anyURI	optional			

source

```
<xs:element name="typemanip" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="change" maxOccurs="unbounded">
    <xs:complexType>
     <xs:attribute name="srcbytepos" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:nonNegativeInteger">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="7"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="srcbitpos" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:nonNegativeInteger">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="7"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="dstbytepos" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:nonNegativeInteger">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="7"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="dstbitpos" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:nonNegativeInteger">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="7"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="type" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:pattern value="ushort"/>
     <xs:pattern value="uint"/>
     <xs:pattern value="ulong"/>
     <xs:pattern value="short"/>
     <xs:pattern value="int"/>
     <xs:pattern value="long"/>
     <xs:pattern value="float"/>
     <xs:pattern value="double"/>
     <xs:pattern value="char"/>
     <xs:pattern value="string"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="srctypedef" use="optional">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:pattern value="WINx86"/>
     <xs:pattern value="LINUXx86"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="dsttypedef" use="optional">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:pattern value="WINx86"/>
     <xs:pattern value="LINUXx86"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="exttyperep" type="xs:anyURI" use="optional"/>
 </xs:complexType>
</xs:element>
```

element **typereflType/typemanip/change**

diagram

namespace: http://www.wettzell.ifag.de/a2x

attributes

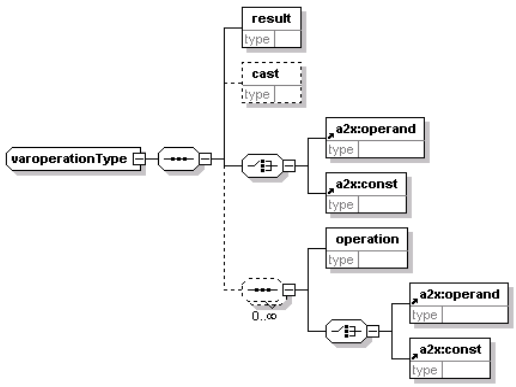
Name	Type	Use	Default	Fixed	Annotation
srcbytepos	xs:nonNegativeInteger	required			
srcbitpos	xs:nonNegativeInteger	required			
dstbytepos	xs:nonNegativeInteger	required			
dstbitpos	xs:nonNegativeInteger	required			

source

```
<xs:element name="change" maxOccurs="unbounded">
 <xs:complexType>
  <xs:attribute name="srcbytepos" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:nonNegativeInteger">
     <xs:minInclusive value="0"/>
     <xs:maxInclusive value="7"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="srcbitpos" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:nonNegativeInteger">
     <xs:minInclusive value="0"/>
     <xs:maxInclusive value="7"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="dstbytepos" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:nonNegativeInteger">
     <xs:minInclusive value="0"/>
     <xs:maxInclusive value="7"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="dstbitpos" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:nonNegativeInteger">
     <xs:minInclusive value="0"/>
     <xs:maxInclusive value="7"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:element>
```

complexType **varoperationType**

diagram



namespace	http://www.wettzell.ifag.de/a2x
children	**result** **cast** **a2x:operand** **a2x:const** **operation**
used by	elements **edgeType/varoperation** **edgeType/apply/varoperation**

source

```
<xs:complexType name="varoperationType">
 <xs:sequence>
  <xs:element name="result">
   <xs:complexType>
    <xs:attribute name="variable" use="required">
     <xs:simpleType>
      <xs:restriction base="xs:IDREF">
       <xs:pattern value="VAR.*"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
   </xs:complexType>
  </xs:element>
  <xs:element name="cast" minOccurs="0">
   <xs:complexType>
    <xs:attribute name="type" use="optional">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="ushort"/>
       <xs:enumeration value="uint"/>
       <xs:enumeration value="ulong"/>
       <xs:enumeration value="short"/>
       <xs:enumeration value="int"/>
       <xs:enumeration value="long"/>
       <xs:enumeration value="float"/>
       <xs:enumeration value="double"/>
       <xs:enumeration value="char"/>
       <xs:enumeration value="string"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
   </xs:complexType>
  </xs:element>
  <xs:choice>
   <xs:element ref="a2x:operand"/>
   <xs:element ref="a2x:const"/>
  </xs:choice>
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
   <xs:element name="operation">
    <xs:complexType>
     <xs:attribute name="type" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="+"/>
        <xs:enumeration value="-"/>
        <xs:enumeration value="*"/>
        <xs:enumeration value="/"/>
        <xs:enumeration value="%"/>
        <xs:enumeration value="cat"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
    </xs:complexType>
   </xs:element>
   <xs:choice>
    <xs:element ref="a2x:operand"/>
    <xs:element ref="a2x:const"/>
   </xs:choice>
  </xs:sequence>
 </xs:sequence>
</xs:complexType>
```

element **varoperationType/result**

diagram	result type
namespace	http://www.wettzell.ifag.de/a2x
attributes	Name: variable; Type: xs:IDREF; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="result"> <xs:complexType> <xs:attribute name="variable" use="required"> <xs:simpleType> <xs:restriction base="xs:IDREF"> <xs:pattern value="VAR.*"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element>

element **varoperationType/cast**

diagram	cast type
namespace	http://www.wettzell.ifag.de/a2x
attributes	Name: type; Type: xs:string; Use: optional; Default: ; Fixed: ; Annotation:
source	<xs:element name="cast" minOccurs="0"> <xs:complexType> <xs:attribute name="type" use="optional"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="ushort"/> <xs:enumeration value="uint"/> <xs:enumeration value="ulong"/> <xs:enumeration value="short"/> <xs:enumeration value="int"/> <xs:enumeration value="long"/> <xs:enumeration value="float"/> <xs:enumeration value="double"/> <xs:enumeration value="char"/> <xs:enumeration value="string"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element>

element **varoperationType/operation**

diagram	operation type
namespace	http://www.wettzell.ifag.de/a2x
attributes	Name: type; Type: xs:string; Use: required; Default: ; Fixed: ; Annotation:
source	<xs:element name="operation"> <xs:complexType> <xs:attribute name="type" use="required"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="+"/> <xs:enumeration value="-"/> <xs:enumeration value="*"/> <xs:enumeration value="/"/> <xs:enumeration value="%"/> <xs:enumeration value="cat"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element>

Anhang L

Schema-Definition zu A2X

(Stand: 14.01.2005)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com)
     by A. Neidhardt (Fundamentalstation Wettzell) -->
<xs:schema targetNamespace="http://www.wettzell.ifag.de/a2x"
           xmlns:a2x="http://www.wettzell.ifag.de/a2x"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="a2x">
    <xs:annotation>
      <xs:documentation>Anything to XML: Extension to XSL</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="info">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="a2xspecification">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="revision">
                      <xs:complexType>
                        <xs:attribute name="version" use="required">
                          <xs:simpleType>
                            <xs:restriction base="xs:string">
                              <xs:enumeration value="1.0beta"/>
                            </xs:restriction>
                          </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="date" type="xs:date" use="required"/>
                        <xs:attribute name="homeurl" type="xs:anyURI" use="required"/>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="comment" type="xs:string" minOccurs="0"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="converterspecification">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="revision">
                      <xs:complexType>
                        <xs:attribute name="name" type="xs:string" use="required"/>
                        <xs:attribute name="roottag" type="xs:string" use="required"/>
                        <xs:attribute name="version" use="required">
                          <xs:simpleType>
                            <xs:restriction base="xs:string"/>
                          </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="date" type="xs:date" use="required"/>
                        <xs:attribute name="institution" type="xs:string" use="optional"/>
                        <xs:attribute name="developer" type="xs:string" use="required"/>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="comment" type="xs:string" minOccurs="0"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="definition" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="a2x:typerep" minOccurs="0" maxOccurs="unbounded"/>
              <xs:element ref="a2x:declare" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="inputrules" type="a2x:inputrulesType"/>
      </xs:sequence>
    </xs:complexType>
```

```
</xs:element>
<xs:complexType name="typerefType">
  <xs:sequence>
    <xs:element name="typemanip" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="change" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="srcbytepos" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:nonNegativeInteger">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="7"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="srcbitpos" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:nonNegativeInteger">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="7"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="dstbytepos" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:nonNegativeInteger">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="7"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="dstbitpos" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:nonNegativeInteger">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="7"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="type" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="ushort"/>
              <xs:pattern value="uint"/>
              <xs:pattern value="ulong"/>
              <xs:pattern value="short"/>
              <xs:pattern value="int"/>
              <xs:pattern value="long"/>
              <xs:pattern value="float"/>
              <xs:pattern value="double"/>
              <xs:pattern value="char"/>
              <xs:pattern value="string"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="srctypedef" use="optional">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="WINx86"/>
              <xs:pattern value="LINUXx86"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="dsttypedef" use="optional">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="WINx86"/>
              <xs:pattern value="LINUXx86"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="exttyperep" type="xs:anyURI" use="optional"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:element name="typerep">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="a2x:typerefType">
        <xs:attribute name="representation" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:ID">
              <xs:pattern value="REP.*"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="exttyperep" type="xs:anyURI" use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="declare">
  <xs:complexType>
    <xs:sequence>
```

```
      <xs:element name="variable" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="a2x:comment" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="name" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:ID">
                <xs:pattern value="VAR.*"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="tag" type="xs:NMTOKEN" use="optional"/>
          <xs:attribute name="type" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="ushort"/>
                <xs:enumeration value="uint"/>
                <xs:enumeration value="ulong"/>
                <xs:enumeration value="short"/>
                <xs:enumeration value="int"/>
                <xs:enumeration value="long"/>
                <xs:enumeration value="float"/>
                <xs:enumeration value="double"/>
                <xs:enumeration value="char"/>
                <xs:enumeration value="string"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="init" type="xs:anySimpleType" use="optional"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="applybreak"/>
<xs:complexType name="edgeType">
  <xs:sequence>
    <xs:choice minOccurs="0">
      <xs:element name="pattern">
        <xs:annotation>
          <xs:documentation>Regular expression</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="enum" minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Predefined, allowed values</xs:documentation>
              </xs:annotation>
              <xs:complexType>
                <xs:attribute name="value" type="xs:anySimpleType" use="required"/>
              </xs:complexType>
            </xs:element>
            <xs:element ref="a2x:comment" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="expression" type="xs:string" use="required"/>
          <xs:attribute name="variable" use="optional">
            <xs:simpleType>
              <xs:restriction base="xs:IDREF">
                <xs:pattern value="VAR.*"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="typerep" use="optional">
            <xs:simpleType>
              <xs:restriction base="xs:IDREF">
                <xs:pattern value="REP.*"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
      <xs:element name="field">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="enum" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:attribute name="value" type="xs:anySimpleType" use="required"/>
              </xs:complexType>
            </xs:element>
            <xs:element ref="a2x:comment" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="length" type="xs:long" use="required"/>
          <xs:attribute name="variable" use="optional">
            <xs:simpleType>
              <xs:restriction base="xs:IDREF">
                <xs:pattern value="VAR.*"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="typerep" use="optional">
            <xs:simpleType>
              <xs:restriction base="xs:IDREF">
                <xs:pattern value="REP.*"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
```

```
      <xs:element name="dummy">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="a2x:comment" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="event" use="optional">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="EOF"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="variable" use="optional">
            <xs:simpleType>
              <xs:restriction base="xs:IDREF">
                <xs:pattern value="VAR.*"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:choice>
    <xs:element name="varoperation" type="a2x:varoperationType"
                minOccurs="0" maxOccurs="unbounded"/>
    <xs:choice>
      <xs:element ref="a2x:applybreak"/>
      <xs:element name="apply">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="varoperation" type="a2x:varoperationType"/>
          </xs:sequence>
          <xs:attribute name="next" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:IDREF">
                <xs:pattern value="ID.*"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
      <xs:element name="case">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="comparison" type="a2x:comparisonType"/>
            <xs:element name="if" type="a2x:edgeType"/>
            <xs:element name="else" type="a2x:edgeType" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<xs:element name="edge">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="a2x:edgeType">
        <xs:attribute name="TaggingOn" use="optional" default="true">
          <xs:simpleType>
            <xs:restriction base="xs:boolean"/>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:complexType name="inputrulesType">
  <xs:sequence>
    <xs:element ref="a2x:edge"/>
    <xs:element name="knote" maxOccurs="unbounded">
      <xs:complexType>
        <xs:complexContent>
          <xs:extension base="a2x:knoteType">
            <xs:attribute name="knoteID" use="required">
              <xs:simpleType>
                <xs:restriction base="xs:ID">
                  <xs:pattern value="ID.*"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="name" type="xs:string" use="optional"/>
            <xs:attribute name="identifier" type="xs:ID" use="optional"/>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="knoteType">
  <xs:sequence>
    <xs:element name="subknote" minOccurs="0">
      <xs:complexType>
        <xs:complexContent>
          <xs:extension base="a2x:inputrulesType">
            <xs:attribute name="tagblock" type="xs:NMTOKEN" use="optional"/>
            <xs:attribute name="ReadFromVariable" use="optional">
              <xs:simpleType>
                <xs:restriction base="xs:IDREF">
                  <xs:pattern value="VAR.*"/>
                </xs:restriction>
```

```
              </xs:simpleType>
            </xs:attribute>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
    <xs:element ref="a2x:edge" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="varoperationType">
  <xs:sequence>
    <xs:element name="result">
      <xs:complexType>
        <xs:attribute name="variable" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:IDREF">
              <xs:pattern value="VAR.*"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
    <xs:element name="cast" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="type" use="optional">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="ushort"/>
              <xs:enumeration value="uint"/>
              <xs:enumeration value="ulong"/>
              <xs:enumeration value="short"/>
              <xs:enumeration value="int"/>
              <xs:enumeration value="long"/>
              <xs:enumeration value="float"/>
              <xs:enumeration value="double"/>
              <xs:enumeration value="char"/>
              <xs:enumeration value="string"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
    <xs:choice>
      <xs:element ref="a2x:operand"/>
      <xs:element ref="a2x:const"/>
    </xs:choice>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="operation">
        <xs:complexType>
          <xs:attribute name="type" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="+"/>
                <xs:enumeration value="-"/>
                <xs:enumeration value="*"/>
                <xs:enumeration value="/"/>
                <xs:enumeration value="%"/>
                <xs:enumeration value="cat"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
      <xs:choice>
        <xs:element ref="a2x:operand"/>
        <xs:element ref="a2x:const"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
</xs:complexType>
<xs:element name="const">
  <xs:complexType>
    <xs:attribute name="value" type="xs:anySimpleType" use="required"/>
    <xs:attribute name="cast" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="ushort"/>
          <xs:enumeration value="uint"/>
          <xs:enumeration value="ulong"/>
          <xs:enumeration value="short"/>
          <xs:enumeration value="int"/>
          <xs:enumeration value="long"/>
          <xs:enumeration value="float"/>
          <xs:enumeration value="double"/>
          <xs:enumeration value="char"/>
          <xs:enumeration value="string"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="operand">
  <xs:complexType>
    <xs:attribute name="variable" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:IDREF">
          <xs:pattern value="VAR.*"/>
        </xs:restriction>
      </xs:simpleType>
```

```
    </xs:attribute>
    <xs:attribute name="cast" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="ushort"/>
          <xs:enumeration value="uint"/>
          <xs:enumeration value="ulong"/>
          <xs:enumeration value="short"/>
          <xs:enumeration value="int"/>
          <xs:enumeration value="long"/>
          <xs:enumeration value="float"/>
          <xs:enumeration value="double"/>
          <xs:enumeration value="char"/>
          <xs:enumeration value="string"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:complexType name="comparisonType">
  <xs:choice>
    <xs:sequence>
      <xs:choice>
        <xs:element name="comparator">
          <xs:complexType>
            <xs:attribute name="variable" type="xs:IDREF" use="required"/>
            <xs:attribute name="cast" use="optional">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="ushort"/>
                  <xs:enumeration value="uint"/>
                  <xs:enumeration value="ulong"/>
                  <xs:enumeration value="short"/>
                  <xs:enumeration value="int"/>
                  <xs:enumeration value="long"/>
                  <xs:enumeration value="float"/>
                  <xs:enumeration value="double"/>
                  <xs:enumeration value="char"/>
                  <xs:enumeration value="string"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
        <xs:element ref="a2x:const"/>
      </xs:choice>
      <xs:element name="condition">
        <xs:complexType>
          <xs:attribute name="constraint" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="lt"/>
                <xs:enumeration value="eq"/>
                <xs:enumeration value="gt"/>
                <xs:enumeration value="!lt"/>
                <xs:enumeration value="!eq"/>
                <xs:enumeration value="!gt"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
      <xs:choice>
        <xs:element name="comparator">
          <xs:complexType>
            <xs:attribute name="variable" type="xs:IDREF" use="required"/>
            <xs:attribute name="cast" use="optional">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="ushort"/>
                  <xs:enumeration value="uint"/>
                  <xs:enumeration value="ulong"/>
                  <xs:enumeration value="short"/>
                  <xs:enumeration value="int"/>
                  <xs:enumeration value="long"/>
                  <xs:enumeration value="float"/>
                  <xs:enumeration value="double"/>
                  <xs:enumeration value="char"/>
                  <xs:enumeration value="string"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
        <xs:element ref="a2x:const"/>
      </xs:choice>
    </xs:sequence>
    <xs:element name="conjunction">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="composition" type="a2x:comparisonType"/>
          <xs:choice>
            <xs:element name="and"/>
            <xs:element name="or"/>
          </xs:choice>
          <xs:element name="composition" type="a2x:comparisonType"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

```
      <xs:element name="negation">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="not">
              <xs:complexType/>
            </xs:element>
            <xs:element name="composition" type="a2x:comparisonType"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
  <xs:element name="comment" type="xs:string"/>
</xs:schema>
```

Anhang M

Beispiel einer A2X-Beschreibung für das Normalpunkteformat

Die Messergebnisse der Laserentfernungsmessung werden nach der lokalen Auswertung in Wettzell als Normalpunkte (statistisch charakterisierende Auswertungspunkte einer Satellitenpassage) in einer Datei abgespeichert. Diese Datei ist in einem standardisierten Format zu erstellen, welches vom ILRS vorgegeben ist. Dabei handelt es sich um eine Art ASCII -Tabelle mit festgelegten Spaltenbreiten.

Nachfolgend ist ein Abdruck der Vorgaben für das Format von der offiziellen Internetseite des ILRS gegeben[1]. Die anschließend folgende A2X -Umsetzung ist eine fragmenthafte, jedoch als Beispiel gedachte Beschreibung für eine reale Konvertierung.

[1] vgl. dazu [HUS04]

M.1 Die Formatbeschreibung

(Abdruck entnommen aus [HUS04])

ILRS Normal Point Format

This page presents the latest version of the ILRS normal point format. Fields revised in March 1997 are shown in blue.

- ILRS NP Header Record Format (Revision 1)
- ILRS NP Data Record Format (Revision 1)
- ILRS Sampled Engineering Data Record Format

Other related links:

- History of the Normal Point Format
- The Normal Point Algorithm

Header Record
(Revision 1 - March 1997)

Column	Description	Example
1-7	ILRS Satellite identifier - 7-digit identification number (based on COSPAR) *See Note below:*	'7603901'
8-9	Year of century	'89'
10-12	Day of year	'079'
13-16	Crustal Dynamics Project Pad ID- a 4-digit monument identification	'7105'
17-18	Crustal Dynamics Project 2-digit system number	'07'
19-20	Crustal Dynamics Project 2-digit occupancy sequence number	'02'
21-24	Wavelength of the laser The user of the data should interpret the value given as follows: 3000 - 9999: units are 0.1 nanometer 1000 - 2999: units are 1.0 nanometer For the station generating the data, the rule is: Wavelength in rate 0.3000 - 0.9999 microns: unit 0.1 nanometer Wavelength in rate 1.000 - 2.999 microns: unit 1.0 nanometer	'5321'
25-32	Calibration system delay (two-way value in picoseconds)	'00095942'
33-38	Calibration delay shift (two-way value in picoseconds)	'000033'
39-42	Root Mean Square (RMS) of raw system delay values from the mean. Two-way value in picoseconds. If pre- and post- pass calibrations are made,use the mean of the two RMS values, or the RMS of the combined data set.	'0040'
43	Normal Point window indicator (an integer from 0 to 9) 0: not a normal point 1: 5-second normal point (GFZ-1) 2: LLR normal point 3: 15-second normal point (TOPEX) 4: 20-second normal point 5: 30-second normal point 6: 1-minute normal point 7: 2-minute normal point (LAGEOS) 8: 3-minute normal point 9: 5-minute normal point (ETALON)	'7'
44	Epoch time scale indicator 3: UTC (USNO) 4: UTC (GPS) 7: UTC (BIPM) (BIH prior to 1988)	'3'

45	System calibration method and delay shift indicator. Indicates the type of calibration and the type of calibration shift given in columns 33-38 Pre- to Post-Pass Calibration Shift / Minimum to Maximum Calibration Shift External cal: 0 / 5 Internal cal: 1 / 6 Burst cal: 2 / 7 Some other cal: 3 / 8 Not used: 4 / 9	'0'
46	System CHange indicator (SCH). A flag to increment for every major change to the system (hardware or software). After the value '9' return to '0', and then continue incrementing. The station and data centers should keep a log in a standard format of the value used, the date of the change, and a description of the change.	'0'
47	System Configuration Indicator (SCI). A flag used to indicate alternative modes of operation for a system (e.g., choice of alternative timers or detectors, or use of a different mode of operation for high satellites). Each value of the flag indicates a particular configuration, which is described in a log file held at the station and at the data centers. If only a single configuration is used then use a fixed value. If a new configuration is introduced then use the next higher flag value. If value exceeds '9' then return to '0', overwriting a previous configuration flag (it is not likely that a station will have 10 current possible configurations).	'1'
48-51	Pass RMS from the mean of raw range values minus the trend function, for accepted ranges (two-way value in picoseconds).	'0065'
52	Data quality assessment indicator For LLR data: 0: Undefined or no comment. 1: Clear, easily filtered data, with little or no noise. 2: Clear data with some noise; filtering is slightly compromised by noise level. 3: Clear data with a significant amount of noise, or weak data with little noise. Data are certainly present, but filtering is difficult. 4: Un-clear data; data appear marginally to be present, but are very difficult to separate from noise during filtering. Signal to noise ratio can be less than 1:1. 5: No data apparent.	'0'
53-54	Checksum - an integer value equal to the sum of integers in columns 1-52, modulo 100 (optional)	'53'
55	Format revision number indicator. Value '1' for this 1997 revision. Implied value '0' or 'space' for original 1990 release.	'1'

Note: COSPAR ID to ILRS Satellite Identification Algorithm

COSPAR ID Format: (YYYY-XXXA)

YYYY is the four digit year of when the launch vehicle was put in orbit
XXX is the sequential launch vehicle number for that year
A is the alpha numeric sequence number within a launch
Example: LAGEOS-1 COSPAR ID is ***1976-039A***
Explanation: LAGEOS-1 launch vehicle wasplaced in orbit in 1976; was the 39th launch in that year; and LAGEOS-1 was the first object injected into orbit from this launch.

ILRS Satellite Identification Format: (YYXXXAA), based on the COSPAR ID

Where YY is the two digit year of when the launch vehicle was put in orbit
Where XXX is the sequential launch vehicle number for that year
AA is the numeric sequence number within a launch
Example: LAGEOS-1 ILRS Satellite ID is ***7603901***

Data Record
(Revision 1 - March 1997)

Column	Description	Example
1-12	Time of day of laser firing, from 0 hours UTC in units of 0.1 microseconds. Value is given module 864000000000 if pass crosses 24 hours UTC	'214360786545'
13-24	Two-way time-of-flight corrected for system delay, in picoseconds. Not corrected for atmospheric delay, nor to the center-of-mass of the satellite.	'052035998000'
25-31	Bin RMS from the mean of raw range values minus the trend function, for accepted ranges. Two-way value in picoseconds. If point is a single raw data point, then use pass RMS.	'0000066'
32-36	Surface pressure, in units of 0.1 millibar	'10052'
37-40	Surface temperature in units of 0.1 degree Kelvin	'2932'
41-43	Relative humidity at surface in percent	'092'
44-47	Number of raw ranges (after editing) compressed into the normal point. *See Note 1: below*	'0108'
48	A flag to indicate the data release: 0: first release of data 1: first replacement release of the data, 2: second replacement release, etc.	'0'
49	For SLR data: not used For LLR data: integer seconds of the two-way time of flight (columns 13-24 contain the fractional part).	'2'
50	For SLR data: not used For LLR data: normal point window indicator. Indicates the time span of the normal point (can be variable from point to point). 1: <= 5 minutes 2: 1 minutes 3: 15 minutes 4: 20 minutes 5: 25 minutes 6: 30 minutes 7: 35 minutes 8: 40 minutes 9: >= 50 minutes	'1'
51-52	For SLR data: not used For LLR data: signal to noise ratio, in units of 0.1, e.g., 00: No information 01: Signal/noise = 0.1 ... 99: Signal/noise = 9.9 or greater	'00'
53-54	Checksum - integer value = to the sum of digits in columns 1-52, modulo 100 (optional)	'51'

Note 1: In September 1999, the Jaguar Team concluded "*That ILRS make **NO RESTRICTION** on the minimum number of returns used to generate Normal Points.*"

Sampled Engineering Data Record
(last change December 1995)

Column	Description	Example
1-12	Time of day of laser firing, from 0 hours UTC, in units of 0.1 microseconds	'214360786545'
13-24	Two-way time of flight with no corrections applied, in picoseconds	'052035998000'
25-29	Surface pressure, in units of 0.1 millibar	'10052'
30-33	Surface temperature in units of 0.1 degree Kelvin	'2932'
34-36	Relative humidity at surface in percent	'092'
37-44	Internal burst calibration system delay.	'00003124'
45-48	Relative signal strength for the return (unit of measure determined by individual stations).	'0789'
49	Angle origin indicator - source of angle values in columns 50-62: 0: Unknown 1: Computed (from range) 2: Command angles - predicted angles with refraction correction and crew biases, if any, applied 3: Measured angles - encoder readings with mount model corrections removed to give actual azimuth and elevation as affected by refraction	'3'
50-56	Azimuth angle in units of 0.0001 degree, using local reference system (north 0, east = 90)	'0981501'
57-62	Elevation angle in units of 0.0001 degree, using local reference system (zenith = 90)	'292501'
63-67	Unused (zero-filled)	'00000'
68-69	Checksum - an integer value equal to the sum of the digits in columns 147, module 100	'07'

Responsible Government Official: Carey Noll
NASA's Privacy, Security, Notices

Send us your comments

Last modified date: Monday, September 15, 2003
Author: Van Husson
Maintained by: Carey Noll

M.2 Schematische Darstellung einer A2X-Beschreibung für das Format

(Stand: 19.01.2005)

Nachfolgend wird eine schematische Graphdarstellung für die Beschreibung des Normalpunkte-Formats mittels A2X gegeben. Im Wesentlichen sind dabei drei Blöcke zu erkennen, welche als Subgraphen (subknotes) definiert wurden.

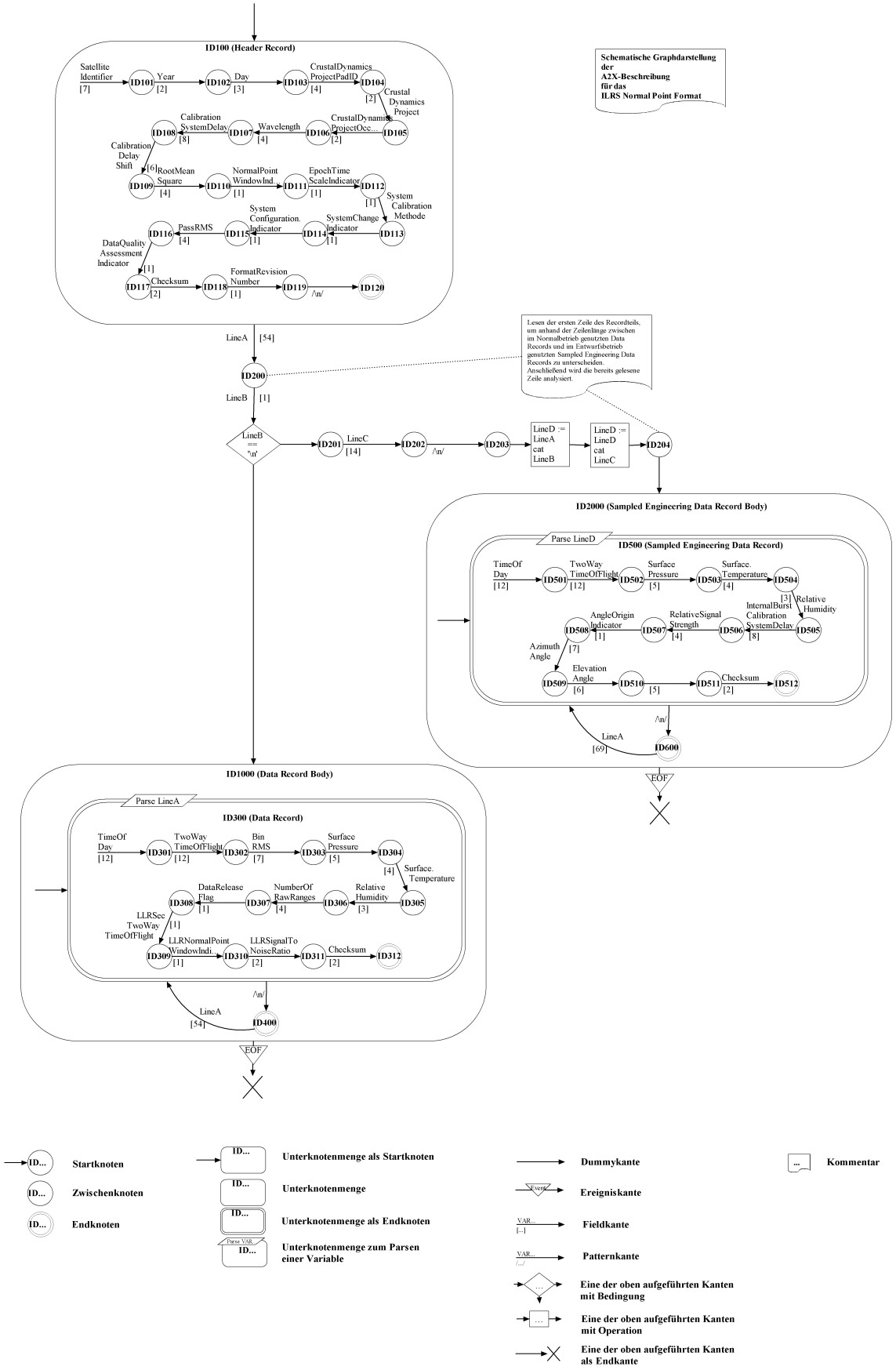
Der erste Subgraph dient zur Beschreibung des Header Records und besteht aus einer Reihung zur Erfassungen von Werten mit jeweils festgelegter Zeichenzahl. Der Header endet mit einem Zeilenumbruch.

Da im eigentlichen Datenkörper der Datei zwei verschiedene Typen von Einträgen enthalten sein können, welche nur anhand der Zeichenzahl einer Zeile erkennbar sind, folgt ein Zwischenschritt zur Ermittlung des Typs. Hierbei werden 54 Zeichen in eine Textvariable übernommen, was der Länge eines herkömmlichen Data Records für den produktionsmäßigen Messbetrieb entspricht. Folgt ein Zeilenumbruch, der ebenfalls in einer Textvariable abgelegt wird, kann daraus abgeleitet werden, dass alle weiteren Einträge Data Records sind. Trifft dies nicht zu, so handelt es sich um ein Sampled Engineering Data Record. Es müssen weitere 14 Zeichen in eine Textvariable eingelesen werden, um die Zeile zu komplettieren. Die drei Textvariablen werden anschließend zu einer zusammengefasst, welche dann eine komplette Zeile eines Sampled Engineering Data Records beinhaltet.

Die folgende, zweite Hauptunterteilung in einen Subgraph beschreibt den Datenbereich (Data Record Body). Er besteht aus einer weiteren Unterteilung zur Beschreibung des Aufbaus eines herkömmlichen Data Records. Sie bildet sich aus einer Reihung zur Erfassung der Werte mit jeweils festgelegter Zeichenzahl aus der vorher gelesenen Zwischenvariable mit der kompletten „Data Record“-Zeile. Nach jeder Zeile wird direkt ein Zeilenumbruch erfasst und der nächste Zeilenabschnitt in die Textzwischenvariable eingelesen, um erneut die Record-Analyse zu starten.

Der dritte Block zur Erfassung der Sampled Engineering Data Records gleicht dem Aufbau der Aufteilung zum Parsen der herkömmlichen Data Records mit dem Unterschied, dass die Record-Zeile nach den spezifischen Vorgaben des Sampled Engineering Data Records behandelt wird. Der Parsing-Durchlauf der beiden Datenbereiche für Records endet jeweils mit einer Endkante, welche noch das Ereignis des Dateiendes (End Of File) prüft.

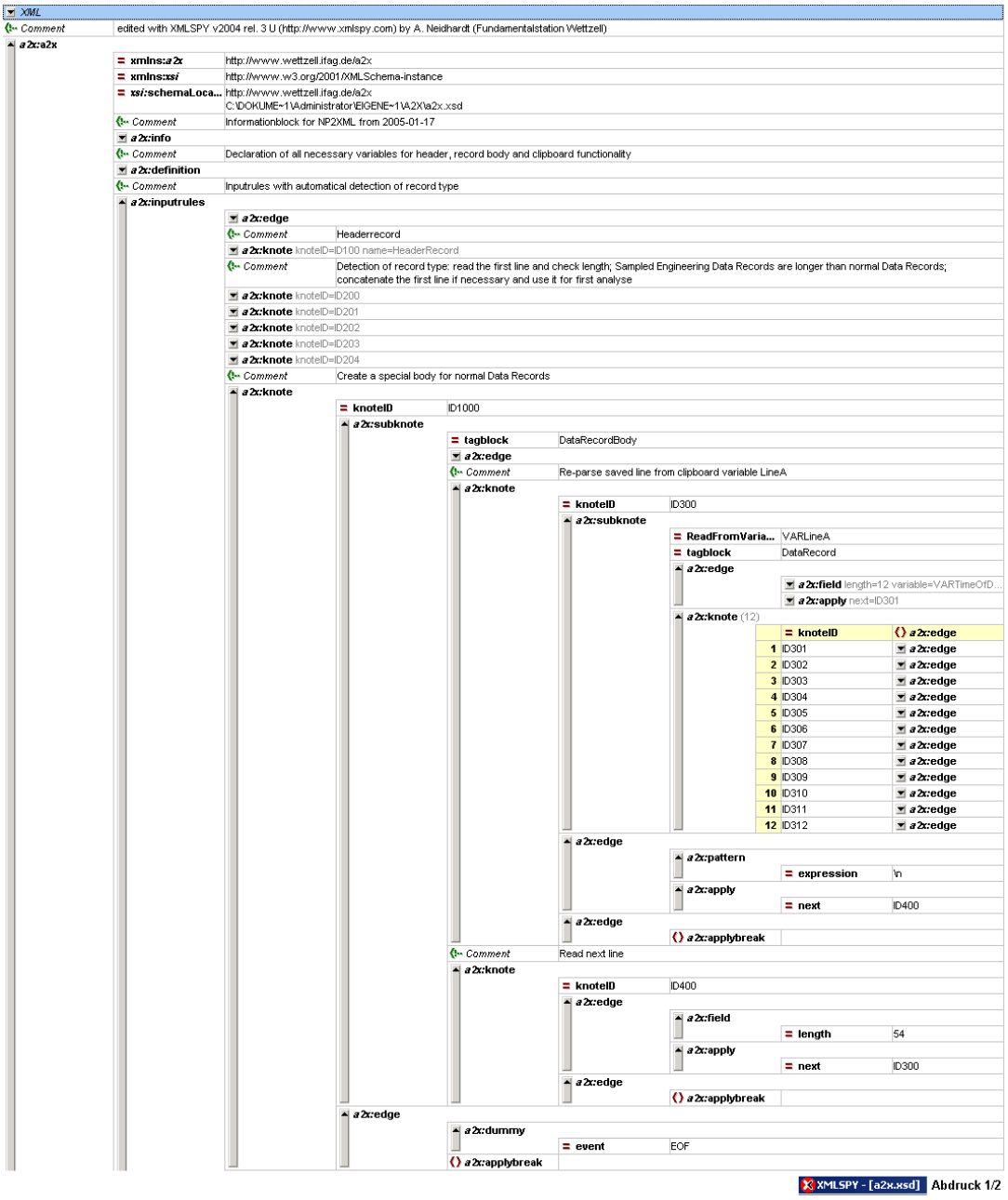
Aus dem Beispiel wird deutlich, dass mit A2X Beschreibungen zum Parsing auf eine strukturierte Art und Weise anhand spezieller Graphen möglich ist. Würden hierzu Entwicklungstools existieren, so könnte direkt aus der graphischen Beschreibung ein XML -Äquivalent erzeugt werden, wodurch eine Art graphische Programmierung von Konvertierungen geschaffen wäre. Die XML -Beschreibung kann dann von beliebigen, standardisierten XSLT -Transformatoren benutzt werden.

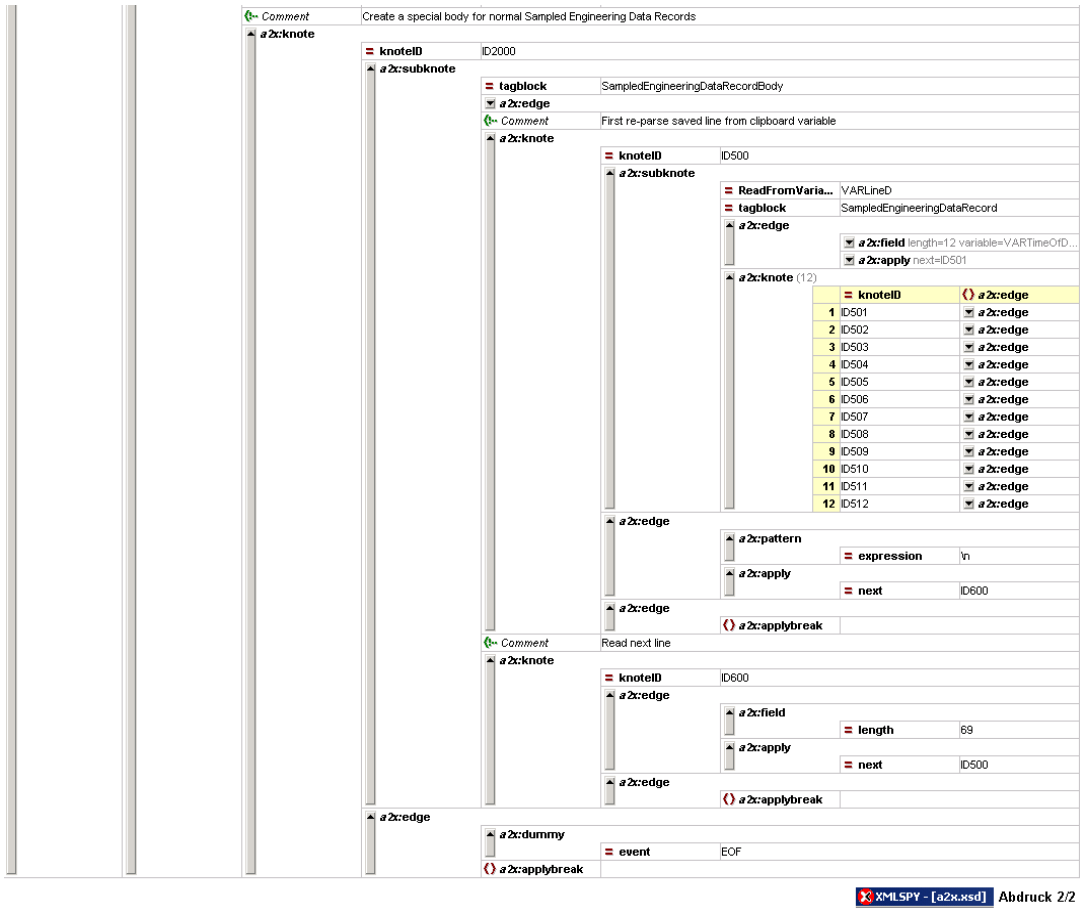

Schematische Graphdarstellung der A2X-Beschreibung für das ILRS Normal Point Format
ID100 (Header Record)
Satellite Identifier [7]
Year [2]
Day [3]
CrustalDynamics ProjectPadID [4]
Crustal Dynamics Project [2]
CrustalDynamics ProjectOcc... [2]
Wavelength [4]
Calibration SystemDelay [8]
Calibration Delay Shift [6]
RootMean Square [4]
NormalPoint WindowInd... [1]
EpochTime ScaleIndicator [1]
System Calibration Methode [1]
SystemChange Indicator [1]
System Configuration. Indicator [1]
PassRMS [4]
DataQuality Assessment Indicator [1]
Checksum [2]
FormatRevision Number [1]
/\n/
LineA [54]
Lesen der ersten Zeile des Recordteils, um anhand der Zeilenlänge zwischen im Normalbetrieb genutzten Data Records und im Entwurfsbetrieb genutzten Sampled Engineering Data Records zu unterscheiden. Anschließend wird die bereits gelesene Zeile analysiert.
LineB [1]
LineB == '\n'
LineC [14]
LineD := LineA cat LineB
LineD := LineD cat LineC
ID2000 (Sampled Engineering Data Record Body)
Parse LineD
ID500 (Sampled Engineering Data Record)
TimeOf Day [12]
TwoWay TimeOfFlight [12]
Surface Pressure [5]
Surface. Temperature [4]
Relative Humidity [3]
InternalBurst Calibration SystemDelay [8]
RelativeSignal Strength [4]
AngleOrigin Indicator [1]
Azimuth Angle [7]
Elevation Angle [6]
Checksum [2]
LineA [69]
ID1000 (Data Record Body)
Parse LineA
ID300 (Data Record)
TimeOf Day [12]
TwoWay TimeOfFlight [12]
Bin RMS [7]
Surface Pressure [5]
Surface. Temperature [4]
Relative Humidity [3]
NumberOf RawRanges [4]
DataRelease Flag [1]
LLRSec TwoWay TimeOfFlight [1]
LLRNormalPoint WindowInd... [1]
LLRSignalTo NoiseRatio [2]
Checksum [2]
LineA [54]
EOF
Startknoten
Zwischenknoten
Endknoten
Unterknotenmenge als Startknoten
Unterknotenmenge
Unterknotenmenge als Endknoten
Unterknotenmenge zum Parsen einer Variable
Dummykante
Ereigniskante
Fieldkante
Patternkante
Eine der oben aufgeführten Kanten mit Bedingung
Eine der oben aufgeführten Kanten mit Operation
Eine der oben aufgeführten Kanten als Endkante
Kommentar

M.3 Graphische Form einer A2X-Beschreibung für das Format

(Stand: 19.01.2005)

Nachfolgend ist die A2X -Beschreibung der Umwandlung für das Normalpunkteformat als ein XML -Äquivalent in einer unter XMLSpy von Altova üblichen Form in Teilen abgedruckt.




Comment
Create a special body for normal Sampled Engineering Data Records
a2x:knote
knoteID
ID2000
a2x:subknote
tagblock
SampledEngineeringDataRecordBody
a2x:edge
Comment
First re-parse saved line from clipboard variable
a2x:knote
knoteID
ID500
a2x:subknote
ReadFromVaria...
VARLineD
tagblock
SampledEngineeringDataRecord
a2x:edge
a2x:field length=12 variable=VARTimeOfD...
a2x:apply next=ID501
a2x:knote (12)
knoteID
a2x:edge
1 ID501 a2x:edge
2 ID502 a2x:edge
3 ID503 a2x:edge
4 ID504 a2x:edge
5 ID505 a2x:edge
6 ID506 a2x:edge
7 ID507 a2x:edge
8 ID508 a2x:edge
9 ID509 a2x:edge
10 ID510 a2x:edge
11 ID511 a2x:edge
12 ID512 a2x:edge
a2x:edge
a2x:pattern
expression
\n
a2x:apply
next
ID600
a2x:edge
a2x:applybreak
Comment
Read next line
a2x:knote
knoteID
ID600
a2x:edge
a2x:field
length
69
a2x:apply
next
ID500
a2x:edge
a2x:applybreak
a2x:edge
a2x:dummy
event
EOF
a2x:applybreak
XMLSPY - [a2x.xsd]
Abdruck 2/2

M.4 XML-Schreibweise der A2X-Umsetzung für das Format

(Stand: 19.01.2005)

Nachfolgend ist die A2X -Beschreibung der Umwandlung für das Normalpunkteformat als ein XML -Äquivalent abgedruckt.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com)
     by A. Neidhardt (Fundamentalstation Wettzell) -->
<a2x:a2x xmlns:a2x="http://www.wettzell.ifag.de/a2x"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.wettzell.ifag.de/a2x
C:\DOKUME~1\Administrator\EIGENE~1\A2X\a2x.xsd">
  <!--Informationblock for NP2XML from 2005-01-17-->
  <a2x:info>
    <a2x:a2xspecification>
      <a2x:revision version="1.0beta" date="2005-01-15"
                    homeurl="http://www.wettzell.ifag.de/a2x"/>
    </a2x:a2xspecification>
    <a2x:convspecification>
      <a2x:revision name="NP2XML" roottag="NP2XML"
                    version="1beta" date="2005-01-17"
                    institution="Bundesamt fuer Kartographie und Geodaesie, Wettzell"
                    developer="Neidhardt, Alexander"/>
      <a2x:comment>This file describes the definitions for a conversion from
                   normal point format file into XML equivalent</a2x:comment>
    </a2x:convspecification>
  </a2x:info>
  <!--Declaration of all necessary variables for header, record body and
      clipboard functionality-->
  <a2x:definition>
    <a2x:declare>
      <a2x:variable name="VARSatelliteIdentifier"
                    tag="SatelliteIdentifier" type="ulong">
        <a2x:comment>ILRS Satellite identifier - 7-digit identification
                     number (based on COSPAR)</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARYear" tag="Year" type="ushort">
        <a2x:comment>Year of century</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARDay" tag="Day" type="ushort">
        <a2x:comment>Day of year</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARCrustalDynamicsProjectPadID"
                    tag="CrustalDynamicsProjectPadID" type="ushort">
        <a2x:comment>Crustal Dynamics Project Pad ID- a 4-digit
                     monument identification</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARCrustalDynamicsProject"
                    tag="CrustalDynamicsProject" type="ushort">
        <a2x:comment>Crustal Dynamics Project 2-digit
                     system number</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARCrustalDynamicsProjectOccupancySequenceNumber"
                    tag="CrustalDynamicsProjectOccupancySequenceNumber"
                    type="ushort">
        <a2x:comment>Crustal Dynamics Project 2-digit occupancy
                     sequence number</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARWavelength" tag="Wavelength" type="ushort">
        <a2x:comment>Wavelength of the laser
                     The user of the data should interpret the value
                          given as follows:
                     3000 - 9999: units are 0.1 nanometer
                     1000 - 2999: units are 1.0 nanometer
                     For the station generating the data, the rule is:
                     Wavelength in rate 0.3000 - 0.9999 microns:
                          unit 0.1 nanometer
                     Wavelength in rate 1.000 - 2.999 microns:
                          unit 1.0 nanometer</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARCalibrationSystemDelay"
                    tag="CalibrationSystemDelay" type="ulong">
        <a2x:comment>Calibration system delay (two-way value in
                     picoseconds)</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARCalibrationDelayShift"
                    tag="CalibrationDelayShift" type="ulong">
        <a2x:comment>Calibration delay shift (two-way value in
                     picoseconds)</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARRootMeanSquare" tag="RootMeanSquare"
                    type="ushort">
        <a2x:comment>Root Mean Square (RMS) of raw system delay
                     values from the mean. Two-way value in picoseconds.
                     If pre- and post- pass calibrations are made,use
                     the mean of the two RMS values, or the RMS of the
```

```
                    combined data set</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARNormalPointWindowIndicator"
                    tag="NormalPointWindowIndicator" type="ushort">
        <a2x:comment>Normal Point window indicator (an integer from 0 to 9)
0: not a normal point
1: 5-second normal point (GFZ-1)
2: LLR normal point
3: 15-second normal point (TOPEX)
4: 20-second normal point
5: 30-second normal point
6: 1-minute normal point
7: 2-minute normal point (LAGEOS)
8: 3-minute normal point
9: 5-minute normal point (ETALON</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VAREpochTimeScaleIndicator"
                    tag="EpochTimeScaleIndicator" type="ushort">
        <a2x:comment>Epoch time scale indicator
3: UTC (USNO)
4: UTC (GPS)
7: UTC (BIPM) (BIH prior to 1988)</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARSystemCalibrationMethode"
                    tag="SystemCalibrationMethode" type="ushort">
        <a2x:comment>System calibration method and delay shift indicator.
                     Indicates the type of calibration and the type of
                     calibration shift given in columns 33-38</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARSystemChangeIndicator" tag="SystemChangeIndicator"
                    type="ushort">
        <a2x:comment>System CHange indicator (SCH). A flag to increment
                     for every major change to the system (hardware or software).
                     After the value '9' return to '0', and then continue incrementing.
                     The station and data centers should keep a log in a standard
                     format of the value used, the date of the change, and a
                     description of the change.</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARSystemConfigurationIndicator"
                    tag="SystemConfigurationIndicator" type="ushort">
        <a2x:comment>System Configuration Indicator (SCI). A flag used to
                     indicate alternative modes of operation for a system (e.g.,
                     choice of alternative timers or detectors, or use of a
                     different mode of operation for high satellites). Each value
                     of the flag indicates a particular configuration, which is
                     described in a log file held at the station and at the data
                     centers. If only a single configuration is used then use a
                     fixed value. If a new configuration is introduced then use
                     the next higher flag value. If value exceeds '9' then return
                     to '0', overwriting a previous configuration flag (it is not
                     likely that a station will have 10 current possible
                     configurations).</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARPassRMS" tag="PassRMS" type="ushort">
        <a2x:comment>Pass RMS from the mean of raw range values minus the trend
                     function, for accepted ranges (two-way value in picoseconds).
                     </a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARDataQualityAssessmentIndicator"
                    tag="DataQualityAssessmentIndicator" type="ushort">
        <a2x:comment>Data quality assessment indicator
                     For LLR data:
                     0: Undefined or no comment.
                     1: Clear, easily filtered data, with little or no noise.
                     2: Clear data with some noise; filtering is slightly
                        compromised by noise level.
                     3: Clear data with a significant amount of noise, or
                        weak data with little noise. Data are certainly present,
                        but filtering is difficult.
                     4: Un-clear data; data appear marginally to be present, but
                        are very difficult to separate from noise during filtering.
                        Signal to noise ratio can be less than 1:1.
                     5: No data apparent.</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARChecksum" tag="Checksum" type="ushort">
        <a2x:comment>Checksum - an integer value equal to the sum of integers
                     in columns 1-52, modulo 100 (optional)</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARFormatRevisionNumber" tag="FormatRevisionNumber"
                    type="ushort">
        <a2x:comment>Format revision number indicator. Value '1' for this 1997
                     revision. Implied value '0' or 'space' for original 1990
                     release.</a2x:comment>
      </a2x:variable>
      <a2x:variable name="VARTimeOfDay" tag="TimeOfDay" type="ulong"/>
      <a2x:variable name="VARTwoWayTimeOfFlight" tag="TwoWayTimeOfFlight"
                    type="ulong"/>
      <a2x:variable name="VARBinRMS" tag="BinRMS" type="ulong"/>
      <a2x:variable name="VARSurfacePressure" tag="SurfacePressure"
                    type="ulong"/>
      <a2x:variable name="VARSurfaceTemperature" tag="SurfaceTemperature"
                    type="ushort"/>
      <a2x:variable name="VARRelativeHumidity" tag="RelativeHumidity"
                    type="ushort"/>
      <a2x:variable name="VARNumberOfRawRanges" tag="NumberOfRawRanges"
                    type="ushort"/>
      <a2x:variable name="VARDataReleaseFlag" tag="DataReleaseFlag"
                    type="ushort"/>
      <a2x:variable name="VARLLRSecTwoWayTimeOfFlight"
```

```
                tag="LLRSecTwoWayTimeOfFlight" type="ushort"/>
    <a2x:variable name="VARLLRNormalPointWindowInd"
                tag="LLRNormalPointWindowInd" type="ushort"/>
    <a2x:variable name="VARLLRSignalToNoiseRatio"
                tag="LLRSignalToNoiseRatio" type="ushort"/>
    <a2x:variable name="VARInternalBurstCalibrationSystemDelay"
                tag="InternalBurstCalibrationSystemDelay" type="ulong"/>
    <a2x:variable name="VARRelativeSignalStrength" tag="RelativeSignalStrength"
                type="ushort"/>
    <a2x:variable name="VARAngleOriginIndicator" tag="AngleOriginIndicator"
                type="ushort"/>
    <a2x:variable name="VARAzimuthAngle" tag="AzimuthAngle"
                type="ulong"/>
    <a2x:variable name="VARElevationAngle" tag="ElevationAngle"
                type="ulong"/>
    <a2x:variable name="VARLineA" type="string"/>
    <a2x:variable name="VARLineB" type="string"/>
    <a2x:variable name="VARLineC" type="string"/>
    <a2x:variable name="VARLineD" type="string"/>
  </a2x:declare>
</a2x:definition>
<!--Inputrules with automatical detection of record type-->
<a2x:inputrules>
  <a2x:edge>
    <a2x:apply next="ID100"/>
  </a2x:edge>
  <!--Headerrecord-->
  <a2x:knote knoteID="ID100" name="HeaderRecord">
    <a2x:subknote tagblock="HeaderRecord">
      <a2x:edge>
        <a2x:field length="7" variable="VARSatelliteIdentifier"/>
        <a2x:apply next="ID101"/>
      </a2x:edge>
      <a2x:knote knoteID="ID101">
        <a2x:edge>
          <a2x:field length="2" variable="VARYear"/>
          <a2x:apply next="ID102"/>
        </a2x:edge>
      </a2x:knote>
      <a2x:knote knoteID="ID102">
        <a2x:edge>
          <a2x:field length="3" variable="VARDay"/>
          <a2x:apply next="ID103"/>
        </a2x:edge>
      </a2x:knote>
      <a2x:knote knoteID="ID103">
        <a2x:edge>
          <a2x:field length="4" variable="VARCrustalDynamicsProjectPadID"/>
          <a2x:apply next="ID104"/>
        </a2x:edge>
      </a2x:knote>
      <a2x:knote knoteID="ID104">
        <a2x:edge>
          <a2x:field length="2" variable="VARCrustalDynamicsProject"/>
          <a2x:apply next="ID105"/>
        </a2x:edge>
      </a2x:knote>
      <a2x:knote knoteID="ID105">
        <a2x:edge>
          <a2x:field length="2"
                     variable="VARCrustalDynamicsProjectOccupancySequenceNumber"/>
          <a2x:apply next="ID106"/>
        </a2x:edge>
      </a2x:knote>
      <a2x:knote knoteID="ID106">
        <a2x:edge>
          <a2x:field length="4" variable="VARWavelength"/>
          <a2x:apply next="ID107"/>
        </a2x:edge>
      </a2x:knote>
      <a2x:knote knoteID="ID107">
        <a2x:edge>
          <a2x:field length="8" variable="VARCalibrationSystemDelay"/>
          <a2x:apply next="ID108"/>
        </a2x:edge>
      </a2x:knote>
      <a2x:knote knoteID="ID108">
        <a2x:edge>
          <a2x:field length="6" variable="VARCalibrationDelayShift"/>
          <a2x:apply next="ID109"/>
        </a2x:edge>
      </a2x:knote>
      <a2x:knote knoteID="ID109">
        <a2x:edge>
          <a2x:field length="4" variable="VARRootMeanSquare"/>
          <a2x:apply next="ID110"/>
        </a2x:edge>
      </a2x:knote>
      <a2x:knote knoteID="ID110">
        <a2x:edge>
          <a2x:field length="1" variable="VARNormalPointWindowIndicator"/>
          <a2x:apply next="ID111"/>
        </a2x:edge>
      </a2x:knote>
      <a2x:knote knoteID="ID111">
        <a2x:edge>
          <a2x:field length="1" variable="VAREpochTimeScaleIndicator"/>
          <a2x:apply next="ID112"/>
        </a2x:edge>
```

```
    </a2x:knote>
    <a2x:knote knoteID="ID112">
      <a2x:edge>
        <a2x:field length="1" variable="VARSystemCalibrationMethode"/>
        <a2x:apply next="ID113"/>
      </a2x:edge>
    </a2x:knote>
    <a2x:knote knoteID="ID113">
      <a2x:edge>
        <a2x:field length="1" variable="VARSystemChangeIndicator"/>
        <a2x:apply next="ID114"/>
      </a2x:edge>
    </a2x:knote>
    <a2x:knote knoteID="ID114">
      <a2x:edge>
        <a2x:field length="1" variable="VARSystemConfigurationIndicator"/>
        <a2x:apply next="ID115"/>
      </a2x:edge>
    </a2x:knote>
    <a2x:knote knoteID="ID115">
      <a2x:edge>
        <a2x:field length="4" variable="VARPassRMS"/>
        <a2x:apply next="ID116"/>
      </a2x:edge>
    </a2x:knote>
    <a2x:knote knoteID="ID116">
      <a2x:edge>
        <a2x:field length="1" variable="VARDataQualityAssessmentIndicator"/>
        <a2x:apply next="ID117"/>
      </a2x:edge>
    </a2x:knote>
    <a2x:knote knoteID="ID117">
      <a2x:edge>
        <a2x:field length="2" variable="VARChecksum"/>
        <a2x:apply next="ID118"/>
      </a2x:edge>
    </a2x:knote>
    <a2x:knote knoteID="ID118">
      <a2x:edge>
        <a2x:field length="1" variable="VARFormatRevisionNumber"/>
        <a2x:apply next="ID119"/>
      </a2x:edge>
    </a2x:knote>
    <a2x:knote knoteID="ID119">
      <a2x:edge>
        <a2x:pattern expression="\n"/>
        <a2x:apply next="ID120"/>
      </a2x:edge>
    </a2x:knote>
    <a2x:knote knoteID="ID120">
      <a2x:edge>
        <a2x:applybreak/>
      </a2x:edge>
    </a2x:knote>
  </a2x:subknote>
  <a2x:edge>
    <a2x:field length="54" variable="VARLineA"/>
    <a2x:apply next="ID200"/>
  </a2x:edge>
</a2x:knote>
<!--Detection of record type: read the first line and check length;
    Sampled Engineering Data Records are longer than normal Data Records;
    concatenate the first line if necessary and use it for first analyse-->
<a2x:knote knoteID="ID200">
  <a2x:edge>
    <a2x:field length="1" variable="VARLineB"/>
    <a2x:case>
      <a2x:comparison>
        <a2x:comparator variable="VARLineB"/>
        <a2x:condition constraint="eq"/>
        <a2x:const value="\n"/>
      </a2x:comparison>
      <a2x:if>
        <a2x:apply next="ID1000"/>
      </a2x:if>
      <a2x:else>
        <a2x:apply next="ID201"/>
      </a2x:else>
    </a2x:case>
  </a2x:edge>
</a2x:knote>
<a2x:knote knoteID="ID201">
  <a2x:edge>
    <a2x:field length="14" variable="VARLineC"/>
    <a2x:apply next="ID202"/>
  </a2x:edge>
</a2x:knote>
<a2x:knote knoteID="ID202">
  <a2x:edge>
    <a2x:pattern expression="\n"/>
    <a2x:apply next="ID203"/>
  </a2x:edge>
</a2x:knote>
<a2x:knote knoteID="ID203">
  <a2x:edge>
    <a2x:varoperation>
      <a2x:result variable="VARLineD"/>
      <a2x:operand variable="VARLineA"/>
      <a2x:operation type="cat"/>
```

```
      <a2x:operand variable="VARLineB"/>
    </a2x:varoperation>
    <a2x:varoperation>
      <a2x:result variable="VARLineD"/>
      <a2x:operand variable="VARLineD"/>
      <a2x:operation type="cat"/>
      <a2x:operand variable="VARLineC"/>
    </a2x:varoperation>
    <a2x:apply next="ID204"/>
  </a2x:edge>
</a2x:knote>
<a2x:knote knoteID="ID204">
  <a2x:edge>
    <a2x:apply next="ID2000"/>
  </a2x:edge>
</a2x:knote>
<!--Create a special body for normal Data Records-->
<a2x:knote knoteID="ID1000">
  <a2x:subknote tagblock="DataRecordBody">
    <a2x:edge>
      <a2x:apply next="ID300"/>
    </a2x:edge>
    <!--Re-parse saved line from clipboard variable LineA-->
    <a2x:knote knoteID="ID300">
      <a2x:subknote ReadFromVariable="VARLineA" tagblock="DataRecord">
        <a2x:edge>
          <a2x:field length="12" variable="VARTimeOfDay"/>
          <a2x:apply next="ID301"/>
        </a2x:edge>
        <a2x:knote knoteID="ID301">
          <a2x:edge>
            <a2x:field length="12" variable="VARTwoWayTimeOfFlight"/>
            <a2x:apply next="ID302"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID302">
          <a2x:edge>
            <a2x:field length="7" variable="VARBinRMS"/>
            <a2x:apply next="ID303"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID303">
          <a2x:edge>
            <a2x:field length="5" variable="VARSurfacePressure"/>
            <a2x:apply next="ID304"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID304">
          <a2x:edge>
            <a2x:field length="4" variable="VARSurfaceTemperature"/>
            <a2x:apply next="ID305"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID305">
          <a2x:edge>
            <a2x:field length="3" variable="VARRelativeHumidity"/>
            <a2x:apply next="ID306"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID306">
          <a2x:edge>
            <a2x:field length="4" variable="VARNumberOfRawRanges"/>
            <a2x:apply next="ID307"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID307">
          <a2x:edge>
            <a2x:field length="1" variable="VARDataReleaseFlag"/>
            <a2x:apply next="ID308"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID308">
          <a2x:edge>
            <a2x:field length="1" variable="VARLLRSecTwoWayTimeOfFlight"/>
            <a2x:apply next="ID309"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID309">
          <a2x:edge>
            <a2x:field length="1" variable="VARLLRNormalPointWindowInd"/>
            <a2x:apply next="ID310"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID310">
          <a2x:edge>
            <a2x:field length="1" variable="VARLLRSignalToNoiseRatio"/>
            <a2x:apply next="ID311"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID311">
          <a2x:edge>
            <a2x:field length="2" variable="VARChecksum"/>
            <a2x:apply next="ID312"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID312">
          <a2x:edge>
            <a2x:applybreak/>
          </a2x:edge>
```

```
        </a2x:knote>
      </a2x:subknote>
      <a2x:edge>
        <a2x:pattern expression="\n"/>
        <a2x:apply next="ID400"/>
      </a2x:edge>
      <a2x:edge>
        <a2x:applybreak/>
      </a2x:edge>
    </a2x:knote>
    <!--Read next line-->
    <a2x:knote knoteID="ID400">
      <a2x:edge>
        <a2x:field length="54"/>
        <a2x:apply next="ID300"/>
      </a2x:edge>
      <a2x:edge>
        <a2x:applybreak/>
      </a2x:edge>
    </a2x:knote>
  </a2x:subknote>
  <a2x:edge>
    <a2x:dummy event="EOF"/>
    <a2x:applybreak/>
  </a2x:edge>
</a2x:knote>
<!--Create a special body for normal Sampled Engineering Data Records-->
<a2x:knote knoteID="ID2000">
  <a2x:subknote tagblock="SampledEngineeringDataRecordBody">
    <a2x:edge>
      <a2x:apply next="ID500"/>
    </a2x:edge>
    <!--First re-parse saved line from clipboard variable-->
    <a2x:knote knoteID="ID500">
      <a2x:subknote ReadFromVariable="VARLineD" tagblock="SampledEngineeringDataRecord">
        <a2x:edge>
          <a2x:field length="12" variable="VARTimeOfDay"/>
          <a2x:apply next="ID501"/>
        </a2x:edge>
        <a2x:knote knoteID="ID501">
          <a2x:edge>
            <a2x:field length="12" variable="VARTwoWayTimeOfFlight"/>
            <a2x:apply next="ID502"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID502">
          <a2x:edge>
            <a2x:field length="5" variable="VARSurfacePressure"/>
            <a2x:apply next="ID503"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID503">
          <a2x:edge>
            <a2x:field length="4" variable="VARSurfaceTemperature"/>
            <a2x:apply next="ID504"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID504">
          <a2x:edge>
            <a2x:field length="3" variable="VARRelativeHumidity"/>
            <a2x:apply next="ID505"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID505">
          <a2x:edge>
            <a2x:field length="8" variable="VARInternalBurstCalibrationSystemDelay"/>
            <a2x:apply next="ID506"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID506">
          <a2x:edge>
            <a2x:field length="4" variable="VARRelativeSignalStrength"/>
            <a2x:apply next="ID507"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID507">
          <a2x:edge>
            <a2x:field length="1" variable="VARAngleOriginIndicator"/>
            <a2x:apply next="ID508"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID508">
          <a2x:edge>
            <a2x:field length="7" variable="VARAzimuthAngle"/>
            <a2x:apply next="ID509"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID509">
          <a2x:edge>
            <a2x:field length="7" variable="VARElevationAngle"/>
            <a2x:apply next="ID510"/>
          </a2x:edge>
        </a2x:knote>
        <a2x:knote knoteID="ID510">
          <a2x:edge>
            <a2x:field length="5"/>
            <a2x:apply next="ID511"/>
          </a2x:edge>
        </a2x:knote>
```

```
                <a2x:knote knoteID="ID511">
                  <a2x:edge>
                    <a2x:field length="2" variable="VARChecksum"/>
                    <a2x:apply next="ID512"/>
                  </a2x:edge>
                </a2x:knote>
                <a2x:knote knoteID="ID512">
                  <a2x:edge>
                    <a2x:applybreak/>
                  </a2x:edge>
                </a2x:knote>
              </a2x:subknote>
              <a2x:edge>
                <a2x:pattern expression="\n"/>
                <a2x:apply next="ID600"/>
              </a2x:edge>
              <a2x:edge>
                <a2x:applybreak/>
              </a2x:edge>
            </a2x:knote>
            <!--Read next line-->
            <a2x:knote knoteID="ID600">
              <a2x:edge>
                <a2x:field length="69"/>
                <a2x:apply next="ID500"/>
              </a2x:edge>
              <a2x:edge>
                <a2x:applybreak/>
              </a2x:edge>
            </a2x:knote>
          </a2x:subknote>
          <a2x:edge>
            <a2x:dummy event="EOF"/>
            <a2x:applybreak/>
          </a2x:edge>
        </a2x:knote>
      </a2x:inputrules>
    </a2x:a2x>
```

M.5 Der Aufbau der in XML-gewandelten Normalpunktedatei in DTD-Schreibweise

(Stand: 19.01.2005)

Nachfolgend ist der Aufbau einer durch eine Umwandlung mit Hilfe der A2X -Beschreibung für das Normalpunkteformat entstandenen XML -Zwischendatei in DTD -Darstellung abgedruckt.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com)
     by A. Neidhardt (Fundamentalstation Wettzell) -->
<!ELEMENT NP2XML (HeaderRecord, (DataRecordBody | SampledEngineeringDataRecordBody))>
<!ELEMENT DataRecordBody (DataRecord+)>
<!ELEMENT SampledEngineeringDataRecordBody (SampledEngineeringDataRecord+)>
<!ELEMENT HeaderRecord (SatelliteIdentifier, Year, Day, CrustalDynamicsProjectPadID,
                        CrustalDynamicsProject, CrustalDynamicsProjectOccupancySequenceNumber,
                        Wavelength, CalibrationSystemDelay, CalibrationDelayShift,
                        RootMeanSquare, NormalPointWindowIndicator, EpochTimeScaleIndicator,
                        SystemCalibrationMethode, SystemChangeIndicator, SystemConfigurationIndicator,
                        PassRMS, DataQualityAssessmentIndicator, Checksum, FormatRevisionNumber)>
<!ELEMENT DataRecord (TimeOfDay, TwoWayTimeOfFlight, BinRMS, SurfacePressure, SurfaceTemperature,
                      RelativeHumidity, NumberOfRawRanges, DataReleaseFlag, LLRSecTwoWayTimeOfFlight,
                      LLRNormalPointWindowInd, LLRSignalToNoiseRatio, Checksum)>
<!ELEMENT SampledEngineeringDataRecord (TimeOfDay, TwoWayTimeOfFlight, BinRMS, SurfacePressure,
                                        SurfaceTemperature, RelativeHumidity,
                                        InternalBurstCalibrationSystemDelay, RelativeSignalStrength,
                                        AngleOriginIndicator, AzimuthAngle, ElevationAngle, Checksum)>
<!ELEMENT SatelliteIdentifier (#PCDATA)>
<!ELEMENT Year (#PCDATA)>
<!ELEMENT Day (#PCDATA)>
<!ELEMENT CrustalDynamicsProjectPadID (#PCDATA)>
<!ELEMENT CrustalDynamicsProject (#PCDATA)>
<!ELEMENT CrustalDynamicsProjectOccupancySequenceNumber (#PCDATA)>
<!ELEMENT Wavelength (#PCDATA)>
<!ELEMENT CalibrationSystemDelay (#PCDATA)>
<!ELEMENT CalibrationDelayShift (#PCDATA)>
<!ELEMENT RootMeanSquare (#PCDATA)>
<!ELEMENT NormalPointWindowIndicator (#PCDATA)>
<!ELEMENT EpochTimeScaleIndicator (#PCDATA)>
<!ELEMENT SystemCalibrationMethode (#PCDATA)>
<!ELEMENT SystemChangeIndicator (#PCDATA)>
<!ELEMENT SystemConfigurationIndicator (#PCDATA)>
<!ELEMENT PassRMS (#PCDATA)>
<!ELEMENT DataQualityAssessmentIndicator (#PCDATA)>
<!ELEMENT Checksum (#PCDATA)>
<!ELEMENT FormatRevisionNumber (#PCDATA)>
<!ELEMENT TimeOfDay (#PCDATA)>
<!ELEMENT TwoWayTimeOfFlight (#PCDATA)>
<!ELEMENT BinRMS (#PCDATA)>
<!ELEMENT SurfacePressure (#PCDATA)>
<!ELEMENT SurfaceTemperature (#PCDATA)>
<!ELEMENT RelativeHumidity (#PCDATA)>
<!ELEMENT NumberOfRawRanges (#PCDATA)>
<!ELEMENT DataReleaseFlag (#PCDATA)>
<!ELEMENT LLRSecTwoWayTimeOfFlight (#PCDATA)>
<!ELEMENT LLRNormalPointWindowInd (#PCDATA)>
<!ELEMENT LLRSignalToNoiseRatio (#PCDATA)>
<!ELEMENT InternalBurstCalibrationSystemDelay (#PCDATA)>
<!ELEMENT RelativeSignalStrength (#PCDATA)>
<!ELEMENT AngleOriginIndicator (#PCDATA)>
<!ELEMENT AzimuthAngle (#PCDATA)>
<!ELEMENT ElevationAngle (#PCDATA)>
```

Anhang N

Indeenskizze eines Berechenbarkeitsnachweises mit A2X

Algorithmische Fähigkeiten einer Sprache im Vergleich zu einer Turingmaschine

Die Mächtigkeit einer Sprache kann unter anderem dadurch belegt werden, welche algorithmischen Fähigkeiten sie für einen Berechnungsprozess zur Verfügung stellt. Eine Möglichkeit ist damit der Nachweis, dass die Sprache ebenso mächtig ist, wie eine Turingmaschine.

Begriffsdefinition DEFN.1 „Turingmaschine“ nach [SCHOE97][1]:
Das von A. M. Turing 1937 vorgeschlagene und heute als Turing Maschine bezeichnete Automatenmodell ist eine mathematisch klar beschreibbare Maschine, welche in ihrer Allgemeinheit jeden beliebigen, mathematischen Berechnungsprozess nachbilden kann.

Aufbau einer Turingmaschine

Schematisch ist die Maschine aus einer endlichen Kontrolleinheit und einem unendlichen Band aufgebaut, auf dem ein Schreib-/Lesekopf Einträge bearbeitet. Im mathematischen Sinne besteht die Turingmaschine aus einer Zustandsmenge, einem Eingabealphabet, einem Arbeitsalphabet (in dem das Eingabealphabet eine Teilmenge darstellt), einem Startzustand, einer Endzustandsmenge, einem Leerzeichen (Blank) und einer Menge von Zustandsübergängen. Diese Übergänge wechseln von einem Zustand und einer Arbeitsalphabetkombination in einen anderen Zustand mit einer neuen Arbeitsalphabetkombination, verbunden mit einer Bewegungsanweisung an den Schreib-/Lesekopf (links, rechts, stand).

Formale Definition der intuitiv verstandenen Berechenbarkeit

Warum gerade diese Definition von solcher Bedeutung ist, ergibt sich aus der Tatsache, dass sie die intuitiv verstandene Berechenbarkeit mathematisch formal definiert. Es gibt zahlreiche weitere Vorschläge zur Definition von Berechenbarkeit dazu (WHILE-, GOTO- Programme oder μ-rekursive Funktionen). Erstaunlicher weise hat sich gezeigt, dass alle untereinander äquivalent sind [2]. Auf den Punkt wird dies in der Churchschen These gebracht.

Begriffsdefinition DEFN.2 „Churchsche These“ nach [SCHOE97][3]:
„DIE DURCH DIE FORMALE DEFINITION DER Turing-Berechenbarkeit (ÄQUIVALENT: WHILE-Berechenbarkeit, GOTO-Berechenbarkeit, μ-Rekursivität) ERFASSTE KLASSE VON FUNKTIONEN STIMMT GENAU MIT DER KLASSE DER IM INTUITIVEN SINNE BERECHENBAREN FUNKTIONEN ÜBEREIN.“

[1] vgl. [SCHOE97] a.a.O. S. 80
[2] vgl. [SCHOE97] a.a.O. S. 93 f.

Nachweis der Mächtigkeit einer Sprache durch Nachbildung einer der „Churchschen Funktionenklassen“

Im übertragenen Sinne, auf die vorliegende Sprache angewandt, bedeutet dies: Wenn nachgewiesen werden kann, dass eine der genannten Modellvorschläge zur Berechenbarkeit äquivalent mit der neuen Sprache nachgebildet werden kann, so können mit ihr dieselben Funktionen berechnet werden, wie mit der Turingmaschine. Da nicht turingmaschinen-berechenbare Funktionen überhaupt nicht berechenbar sind [4], können somit mit der neuen Sprache alle intuitiv berechenbaren Funktionen nachgebildet und berechnet werden. Dies ist ein Nachweis für die Mächtigkeit einer Sprache.

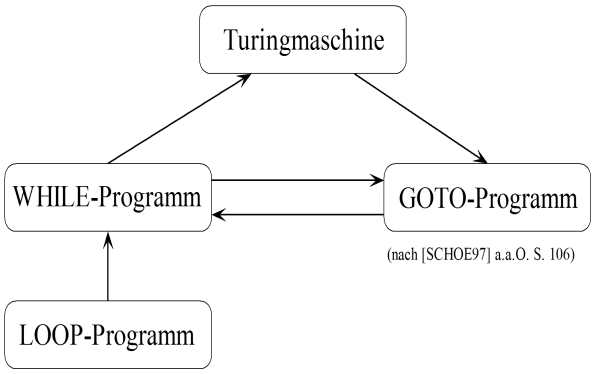


Abbildung N.1: Ringschluss der Berechenbarkeitsbeschreibungen

Ringschluss

Die Vorschläge zur Berechenbarkeit können über den in Abb. N.1 auf Seite 334 dargestellten Ringschluss mit einander in eine Äquivalenzbeziehung gebracht werden, so dass nachgewiesen ist, dass sie alle denselben Berechenbarkeitsbegriff darstellen[5]. Nachfolgend wird deshalb aufgezeigt, dass mit A2X exakt die WHILE-Programme nachgebildet werden können. Dadurch ist dann mit A2X die gleiche algorithmische Berechenbarkeit möglich, wie mit WHILE-Programmen (und äquivalent GOTO-Programmen, bzw. Turingmaschinen).

Der Nachweis dass A2X WHILE-Programme simulieren kann[6]:

1. **Variablen:** Es existieren Variablen. Diese dürfen nur positive Werte annehmen, so dass an gegebener Stelle (z.B. bei Subtraktionen) eine Bedingungsabfrage durchgeführt werden muss. Im Falle eines negativen Ergebnisses wird das Resultat auf 0 gesetzt (modifizierte Subtraktion).

Syntaktische Komponenten WHILE-Programm	**Syntaktische Komponenten A2X**
x0, x1, x2, ...	<a2x:declare> <a2x:variable name=“VARx0“ .../> <a2x:variable name=“VARx1“ .../> <a2x:variable name=“VARx2“ .../> ... </a2x:declare>

[3] [SCHOE97] a.a.O. S. 94
[4] vgl. [SCHOE97] a.a.O. S. 94
[5] vgl. [SCHOE97] a.a.O. S. 106
[6] vgl. zu LOOP-Syntax als Teil der WHILE-Syntax [SCHOE97] a.a.O. S. 100 und zur Erweiterung zur WHILE-Syntax [SCHOE97] a.a.O. S. 103

2. **Konstanten:** Es existieren Konstanten (innerhalb von Operationen, bzw. zur Initialisierung einer Variable) und damit auch Zuweisungen

Syntaktische Komponenten WHILE-Programm	**Syntaktische Komponenten A2X**
1, 2, 3, ...	<a2x:const value="1"/> <a2x:const value="2"/> <a2x:const value="3"/> ...

3. **Trennsymbole:** Es existieren Symbole zur syntaktischen Trennung

Syntaktische Komponenten WHILE-Programm	**Syntaktische Komponenten A2X**
; (Trennung von Programmteilen)	Tags trennen Programmteile
:= (Zuweisung)	Implizite Zuweisung bei der Konstruktion von <varoperation>

4. **Operationszeichen:** Es existieren Operationszeichen (innerhalb von Operationen)

Syntaktische Komponenten WHILE-Programm	**Syntaktische Komponenten A2X**
+, -	<a2x:operation type="+"/> <a2x:operation type="-"/>

5. **Schlüsselwörter:** Es existieren eindeutige Schlüsselwörter

Syntaktische Komponenten WHILE-Programm	**Syntaktische Komponenten A2X**
WHILE, DO, END	Alle auszeichnenden Tags sind Schlüsselwörter

6. **Wertzuweisung:** Eine Wertzuweisung ist ein WHILE-Programm

Syntaktische Komponenten WHILE-Programm	**Syntaktische Komponenten A2X**
xi := xj + c	<a2x:varoperation> <a2x:result variable="VARxi"/> <a2x:operand variable="VARxj"/> <a2x:operation type="+"/> <a2x:const value="c"/> </a2x:varoperation>
xi := xj - c	<a2x:varoperation> <a2x:result variable="VARxi"/> <a2x:operand variable="VARxj"/> <a2x:operation type="-"/> <a2x:const value="c"/> </a2x:varoperation>

7. **Sequenzbildung:** Eine Folge von WHILE-Programmen wird nacheinander ausgeführt und ist ein WHILE-Programm

Syntaktische Komponenten WHILE-Programm	Syntaktische Komponenten A2X
P1; P2	<a2x:inputrules> <a2x:edge> <a2x:apply next="IDP1"/> </a2x:edge> <a2x:knote knoteID="IDP1"> <a2x:subknote> <!– P1 –> </a2x:subknote> <a2x:edge> <a2x:apply next="IDP2"/> </a2x:edge> </a2x:knote> <a2x:knote knoteID="IDP2"> <a2x:subknote> <!– P2 –> </a2x:subknote> <a2x:edge> <a2x:applybreak/> </a2x:edge> </a2x:knote> </a2x:inputrules>

8. **WHILE-Schleife:** Eine WHILE-Schleife ist ein WHILE-Programm

Syntaktische Komponenten WHILE-Programm	Syntaktische Komponenten A2X
WHILE xi $\neq 0$ DO P END	<a2x:inputrules> <a2x:edge> <a2x:apply next="IDP1"/> </a2x:edge> <a2x:knote knoteID="ID1"> <a2x:subknote> <!– P –> </a2x:subknote> <a2x:edge> <a2x:case> <a2x:comparison> <a2x:comparator variable="VARxi"/> <a2x:condition constraint="!eq"/> <a2x:const value="0"/> </a2x:comparison> <a2x:if> <a2x:apply next="ID1"/> </a2x:if> <a2x:else> <a2x:apply next="ID2"/> </a2x:else> </a2x:case> </a2x:edge> </a2x:knote> <a2x:knote knoteID="ID2"> <a2x:edge> <a2x:applybreak/> </a2x:edge> </a2x:knote> </a2x:inputrules>

Alle intuitiv berechenbaren Funktionen sind mit A2X abbildbar

Mit diesen elementaren Syntaxformen sind somit alle intuitiv berechenbaren Funktionen darstellbar. Komplexere Strukturen lassen sich darauf zurückführen[7]. Da gezeigt wurde, dass A2X -Konstrukte jede syntaktische Komponente eines

[7] vgl. [SCHOE97] a.a.O. S. 100 ff.

WHILE-Programms nachbilden können, ist die gleiche Berechenbarkeit der beiden Sprachen gegeben. Damit können A2X -Programme auch den kompletten Berechenbarkeitsumfang von Turingmaschinen nachbilden, was A2X zu einer kompletten Programmiersprache macht. Alle weiteren, eingeführten syntaktischen Komponenten dienen nur noch der vereinfachten Handhabung und komfortableren Programmierung.

Anhang O

Erweiterte Schnittstellendefinition des WDAP in IDL

```
// ************************************************************************
// * CVS Concurrent Versions Control
// * ----------------------------------------------------------------------
// * RCSfile: WDMSInterface.idl,v
// * Revision: 1.1
// * ----------------------------------------------------------------------
// * Author: neidh
// * Date: 2002/05/31 13:26:48
// * Locker:
// * ----------------------------------------------------------------------
// * Log: WDMSInterface.idl,v
// * Revision 1.1  2002/05/31 13:26:48  neidh
// * Fundamental changes:
// * Basical interface-skeleton used for simple interfaces;
// * Using a connector offers advanced abstraction;
// * -> Upgrade of CFT as basis for ECFT
// *
// *
// ************************************************************************

// ************************************************************************
// * Description:                                                         *
// * IDL-definition of the WDMS-interface for automatical codegeneration.*
// * This interface describes the basical methodes of the WDMS-          *
// * Interfaces, which build the basic accesspoint to the service-       *
// * adapters (e.g. FileSystem, RemoteSystem, DatabaseSystem, ...)       *
// * It includes all other IDL-Definitions.                              *
// * ------------------------------------------------------------------- *
// * Used pre-compiler definitions:                                      *
// *                                                                     *
// * Used exception-handles:                                             *
// ************************************************************************

module WDMS
{
    // ********************************************************************
    // * GENERAL DEFINITIONS                                              *
    // ********************************************************************

    // Type of binary raw-data
    typedef sequence<octet> RawDataType;

    // Type of CFT-Infoquestion
    typedef sequence<string> InfoAnswerType;

    // CFT-Transferdescriptorstructure
    struct CFTFILE
    {
        unsigned short usUserID;
        unsigned short usStreamID;
        unsigned long ulTimeStamp;
    };

    // ********************************************************************
    // * INTERFACE INFO                                                   *
    // * ---------------------------------------------------------------- *
    // * Description:                                                     *
    // * Check remote data (e.g. "dir")                                   *
    // * -> the transmitted data are temporarily valid                    *
    // ********************************************************************
    interface CFTInfo
    {
        // ------------------------------------------------------------
        // > Class:     CFTInfo                                       <
        // > Methode:   cppGetInfo                                    <
        // >            Fetches corresponding information without a   <
        // >            intention (only reading access)               <
        // > Parameter: inout CFTFILE SCFTID -> TransferID            <
        // >            in unsigned short usOrderID -> Order-ID       <
        // >            in string cpParameterString -> Orderparameter <
        // >            out InfoAnswerType CpAnswer -> Strings with ret. <
```

```
    // >            out unsigned short uspNumberOfLines -> How many  <
    // >                                                   ret.-lines<
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      07.12.2001                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // -----------------------------------------------------------------
    unsigned short usGetInfo (inout CFTFILE SCFTID,
                              in unsigned short usOrderID,
                              in string cpParameterString,
                              out InfoAnswerType CpAnswer,
                              out unsigned short uspNumberOfLines);
};

// ********************************************************************
// * INTERFACE CHANGE                                                 *
// * ---------------------------------------------------------------- *
// * Description:                                                     *
// * Propagate changes at the visited remote environment (e.g. "cd") *
// * -> reaching of a new environment-view                            *
// ********************************************************************
interface CFTContext
{
    // -----------------------------------------------------------------
    // > Class:     CFTContext                                       <
    // > Methode:   usContextOrder                                   <
    // >            Changes context view at remote server            <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            in unsigned short usOrderID -> Order-ID          <
    // >            in string cpParameterString -> Orderparameter    <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      07.12.2001                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // -----------------------------------------------------------------
    unsigned short usContextOrder (inout CFTFILE SCFTID,
                                   in unsigned short usOrderID,
                                   in string cpParameterString);
};

// ********************************************************************
// * INTERFACE TRANSFER                                               *
// * ---------------------------------------------------------------- *
// * Description:                                                     *
// * Real transportation of data (in form of a filetransfer)          *
// * -> Duplication of data permanently                               *
// ********************************************************************
interface CFTTransfer
{
    // -----------------------------------------------------------------
    // > Class:     CFTTransfer                                      <
    // > Methode:   usOpenFile                                       <
    // >            Open a remote filestream                         <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            in string cpFilename -> Which file               <
    // >            in string cpMode -> Protection-mode              <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      07.12.2001                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // -----------------------------------------------------------------
    unsigned short usOpenFile (inout CFTFILE SCFTID,
                               in string cpFileName,
                               in string cpMode);
    // -----------------------------------------------------------------
    // > Class:     CFTTransfer                                      <
    // > Methode:   usCloseFile                                      <
    // >            Close a remote filestream                        <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      07.12.2001                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // -----------------------------------------------------------------
    unsigned short usCloseFile (inout CFTFILE SCFTID);
    // -----------------------------------------------------------------
    // > Class:     CFTTransfer                                      <
    // > Methode:   usGetLines                                       <
    // >            Get remote line/lines                            <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            out string cppLine -> Transmitted line           <
    // >            inout unisgned long ulpLength -> Length of line  <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      07.12.2001                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // -----------------------------------------------------------------
    unsigned short usGetLines (inout CFTFILE SCFTID,
                               out string cppLine,
                               inout unsigned long ulpLength);
    // -----------------------------------------------------------------
    // > Class:     CFTTransfer                                      <
    // > Methode:   usPutLines                                       <
    // >            Put remote line/lines                            <
```

```
    // > Parameter: inout CFTFILE SCFTID -> TransferID              <
    // >            in string cpLine -> Transmitte line            <
    // >            in unisgned long ulLength -> Length of line    <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)    <
    // > Author:    Alexander Neidhardt                            <
    // > Date:      07.12.2001                                     <
    // > Revision:  -                                              <
    // > Info:      -                                              <
    // -------------------------------------------------------------
    unsigned short usPutLines (inout CFTFILE SCFTID,
                               in string cpLine,
                               in unsigned long ulLength);
    // -------------------------------------------------------------
    // > Class:     CFTTransfer                                    <
    // > Methode:   usGetBlock                                     <
    // >            Get remote block of 8bit-character-streams     <
    // > Parameter: inout CFTFILE SCFTID -> TransferID              <
    // >            out RawDataType cppBlock -> Transmitted block  <
    // >            inout unisgned long ulpLength -> Length of block <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)    <
    // > Author:    Alexander Neidhardt                            <
    // > Date:      22.01.2002                                     <
    // > Revision:  -                                              <
    // > Info:      -                                              <
    // -------------------------------------------------------------
    unsigned short usGetBlock (inout CFTFILE SCFTID,
                               out RawDataType CppBlock,
                               inout unsigned long ulpLength);
    // -------------------------------------------------------------
    // > Class:     CFTTransfer                                    <
    // > Methode:   usPutBlock                                     <
    // >            Put remote block of 8bit-character-streams     <
    // > Parameter: inout CFTFILE SCFTID -> TransferID              <
    // >            in RawDataType cpBlock -> Transmitte block     <
    // >            in unisgned long ulLength -> Length of block   <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)    <
    // > Author:    Alexander Neidhardt                            <
    // > Date:      22.01.2002                                     <
    // > Revision:  -                                              <
    // > Info:      -                                              <
    // -------------------------------------------------------------
    unsigned short usPutBlock (inout CFTFILE SCFTID,
                               in RawDataType CpBlock,
                               in unsigned long ulLength);
    // -------------------------------------------------------------
    // > Class:     CFTTransfer                                    <
    // > Methode:   usEOT                                          <
    // >            End Of Transmission (means End Of File)        <
    // > Parameter: inout CFTFILE SCFTID -> TransferID              <
    // > Return:    unsigned short -> Returncode (0=no, 1=yes)     <
    // > Author:    Alexander Neidhardt                            <
    // > Date:      07.12.2001                                     <
    // > Revision:  -                                              <
    // > Info:      -                                              <
    // -------------------------------------------------------------
    unsigned short usEOT (in CFTFILE SCFTID);
};

// ****************************************************************
// * INTERFACE CFTInterface (inherits CFTInfo, CFTContext,        *
// *                         CFTTransfer)                         *
// * ------------------------------------------------------------ *
// * Description:                                                 *
// * The real interface between server and client as combination of  *
// * the three types of interfaces
// ****************************************************************
interface CFTInterface : CFTInfo, CFTContext, CFTTransfer
{
    // -------------------------------------------------------------
    // > Class:     CFTTransfer                                    <
    // > Methode:   usOpenReg                                      <
    // >            Registrate a remote connection                 <
    // > Parameter: out CFTFILE SCFTID -> TransferID                <
    // >            in string cpUserName -> Username               <
    // >            in string cpUserPwd -> Password                <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)    <
    // > Author:    Alexander Neidhardt                            <
    // > Date:      07.12.2001                                     <
    // > Revision:  -                                              <
    // > Info:      -                                              <
    // -------------------------------------------------------------
    unsigned short usOpenReg (out CFTFILE SCFTID,
                              in string cpUserName,
                              in string cpUserPwd);
    // -------------------------------------------------------------
    // > Class:     CFTTransfer                                    <
    // > Methode:   usCloseReg                                     <
    // >            Close a registrate for remote connection       <
    // > Parameter: inout CFTFILE SCFTID -> TransferID              <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)    <
    // > Author:    Alexander Neidhardt                            <
    // > Date:      07.12.2001                                     <
    // > Revision:  -                                              <
    // > Info:      -                                              <
    // -------------------------------------------------------------
    unsigned short usCloseReg (inout CFTFILE SCFTID);

};
```

```
// ********************************************************************
// * GENERAL DEFINITIONS                                              *
// ********************************************************************

// ********************************************************************
// * INTERFACE ECFTInterface                                          *
// * ---------------------------------------------------------------- *
// * Description:                                                     *
// * Extended interface-definitions.                                  *
// ********************************************************************
interface ECFTInterface : CFTInterface
{
    // ----------------------------------------------------------------
    // > Class:     WDMSInterface                                    <
    // > Methode:   usOpenFile                                       <
    // >            Open a remote filestream                         <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            in string cpFilename -> Which file               <
    // >            in string cpMode -> Protection-mode              <
    // >            in unsigned int uiDummy -> Use only as dummy     <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      28.09.2004                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // ----------------------------------------------------------------
    unsigned short usOpenFile (inout CFTFILE SCFTID,
                               in string cpFileName,
                               in string cpMode,
                               in unsigned int uiDummy);
    // ----------------------------------------------------------------
    // > Class:     WDMSInterface                                    <
    // > Methode:   usCloseFile                                      <
    // >            Close a remote filestream                        <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            in unsigned int uiDummy -> Use only as dummy     <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      28.09.2004                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // ----------------------------------------------------------------
    unsigned short usCloseFile (inout CFTFILE SCFTID,
                                in unsigned int uiDummy);
    // ----------------------------------------------------------------
    // > Class:     WDMSInterface                                    <
    // > Methode:   usGetLines                                       <
    // >            Get remote line/lines                            <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            out string cppLine -> Transmitted line           <
    // >            inout unisgned long ulpLength -> Length of line  <
    // >            in unsigned int uiDummy -> Use only as dummy     <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      28.09.2004                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // ----------------------------------------------------------------
    unsigned short usGetLines (inout CFTFILE SCFTID,
                               out string cppLine,
                               inout unsigned long ulpLength,
                               in unsigned int uiDummy);
    // ----------------------------------------------------------------
    // > Class:     WDMSInterface                                    <
    // > Methode:   usPutLines                                       <
    // >            Put remote line/lines                            <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            in string cpLine -> Transmitte line              <
    // >            in unisgned long ulLength -> Length of line      <
    // >            in unsigned int uiDummy -> Use only as dummy     <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      28.09.2004                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // ----------------------------------------------------------------
    unsigned short usPutLines (inout CFTFILE SCFTID,
                               in string cpLine,
                               in unsigned long ulLength,
                               in unsigned int uiDummy);
    // ----------------------------------------------------------------
    // > Class:     WDMSInterface                                    <
    // > Methode:   usGetBlock                                       <
    // >            Get remote block of 8bit-character-streams       <
    // > Parameter: inout CFTFILE SCFTID -> TransferID               <
    // >            out RawDataType cppBlock -> Transmitted block    <
    // >            inout unisgned long ulpLength -> Length of block <
    // >            in unsigned int uiDummy -> Use only as dummy     <
    // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
    // > Author:    Alexander Neidhardt                              <
    // > Date:      28.09.2004                                       <
    // > Revision:  -                                                <
    // > Info:      -                                                <
    // ----------------------------------------------------------------
    unsigned short usGetBlock (inout CFTFILE SCFTID,
                               out RawDataType CppBlock,
                               inout unsigned long ulpLength,
                               in unsigned int uiDummy);
    // ----------------------------------------------------------------
```

```
        // > Class:     WDMSInterface                                    <
        // > Methode:   usPutBlock                                       <
        // >            Put remote block of 8bit-character-streams       <
        // > Parameter: inout CFTFILE SCFTID -> TransferID                <
        // >            in RawDataType cpBlock -> Transmitte block       <
        // >            in unisgned long ulLength -> Length of block     <
        // >            in unsigned int uiDummy -> Use only as dummy     <
        // > Return:    unsigned short -> Errorcode (0=ok, 1=error)      <
        // > Author:    Alexander Neidhardt                              <
        // > Date:      28.09.2004                                       <
        // > Revision:  -                                                <
        // > Info:      -                                                <
        // ----------------------------------------------------------------
        unsigned short usPutBlock (inout CFTFILE SCFTID,
                                   in RawDataType CpBlock,
                                   in unsigned long ulLength,
                                   in unsigned int uiDummy);
        // ----------------------------------------------------------------
        // > Class:     WDMSInterface                                    <
        // > Methode:   usEOT                                            <
        // >            End Of Transmission (means End Of File)          <
        // > Parameter: inout CFTFILE SCFTID -> TransferID                <
        // >            in unsigned int uiDummy -> Use only as dummy     <
        // > Return:    unsigned short -> Returncode (0=no, 1=yes)       <
        // > Author:    Alexander Neidhardt                              <
        // > Date:      28.09.2004                                       <
        // > Revision:  -                                                <
        // > Info:      -                                                <
        // ----------------------------------------------------------------
        unsigned short usEOT (in CFTFILE SCFTID,
                              in unsigned int uiDummy);
    };

};
```

Anhang P

Die Headerdateien der DLL

P.1 Headerdatei für C++

```
// ************************************************************************
// * CVS Concurrent Versions Control
// * ---------------------------------------------------------------------
// * $RCSfile: ECFT_RI.H,v $
// * $Revision: 1.1 $
// * ---------------------------------------------------------------------
// * $Author: neidh $
// * $Date: 2004/10/11 09:47:26 $
// * $Locker:  $
// * ---------------------------------------------------------------------
// * $Log: ECFT_RI.H,v $
// * Revision 1.1  2004/10/11 09:47:26  neidh
// * Anpassungen und Fertigstellung bis fast Kapitel WDMS
// *
// * Revision 1.4  2004/03/08 16:20:43  neidh
// * Change: Separated Interfaces possible
// * -> Use own interface for several access points not only one global
// * -> Extend SCFTID and add interfaceinfo
// *
// * Revision 1.3  2004/01/16 13:32:48  neidh
// * Change: Use DLL for C and C++ sources
// * - set extern "C" by choice
// * - ECFT_RIC.h (C) and ECFT_RI.h (C++)
// *
// * Revision 1.2  2004/01/15 17:33:12  neidh
// * BugFix: Working DLL
// * - correct usage of extern "C", __declspec(dllexport) and __declspec(dllimport)
// * - first test with ApacheHTTPD
// *
// * Revision 1.1  2004/01/15 15:20:58  neidh
// * AddOn: DLL-Creation for external applications e.g. ApcheHTTPD
// * -> DllMain
// * -> exported C-functions
// * -> usage of remote interface
// *
// *
// *
// ************************************************************************

// ************************************************************************
// * Description:                                                         *
// * This DLL is written to be used by foreign programs to access ECFT-   *
// * interfaces.                                                          *
// * It is the realisation of the jump-in point for DLL-applications.     *
// * -------------------------------------------------------------------- *
// * Used pre-compiler definitions:                                       *
// *                                                                      *
// * Used exception-handles:                                              *
// ************************************************************************

#ifndef __CPP2CWDMSINTERFACEREMOTE_H_HEADER__
#define __CPP2CWDMSINTERFACEREMOTE_H_HEADER__

// Chose porting for import and export
#ifdef ECFT_RIDLL
#define DLL_PORTER __declspec(dllexport)
#else
#define DLL_PORTER __declspec(dllimport)
#endif

// Chose external C tagging
#ifndef EXTERN_CONV
#define EXTERN_CONV extern "C"
#endif

typedef struct STRUCT_CFTFILEDEF
{
    unsigned short usUserID;        // UserID
    unsigned short usStreamID;      // ID of Filestream
    unsigned long  ulTimeStamp;     // Timing information
```

```
    void* vpInterface;              // Pointer to the interface
    char* cpCallingProg;            // Name of the calling program
    char* cpInetAddr;               // IP-Adress
    char* cpPort;                   // Port
    char* cpRemoteName;             // CORBA-Applicationname
    char* cpRemoteKind;             // CORBA-Applicationkind
    unsigned long ulRoundTripDelay; // CORBA Time for a round trip
    char* cpUsername;               // Username for registration
    char* cpPassword;               // Password for registration
} STRUCT_CFTFILE;

// Tracing in a protocol
EXTERN_CONV DLL_PORTER
void vPrintP(char* cpMethode,
             int iLineNumber,
             char* cpFileName);

// CFTInterface-Methode: Activate debug into file
EXTERN_CONV DLL_PORTER
void vActivateLogFile (unsigned short usLogLevel,
                       char * cpFilename);

// CFTInterface-Methode: Set the debug-level
EXTERN_CONV DLL_PORTER
unsigned short usSetDebugLevel (unsigned short usLevel);

// CFTInterface-Methode: Open a connection
EXTERN_CONV DLL_PORTER
unsigned short usConnectToRemote (char* cpCallingProg,
                                  char* cpInetAddr,
                                  char* cpPort,
                                  char* cpRemoteName,
                                  char *cpRemoteKind,
                                  unsigned long ulRoundTripDelay,
                                  STRUCT_CFTFILE * SCFTID);

// CFTInterface-Methode: Close a connection (destructor-call)
EXTERN_CONV DLL_PORTER
unsigned short usDisconnectFromRemote (STRUCT_CFTFILE * SCFTID);

// CFTInterface-Methode: Duplicate connection
EXTERN_CONV DLL_PORTER
unsigned short usDuplicateConnection (STRUCT_CFTFILE * SCFTIDNew,
                                      STRUCT_CFTFILE * SCFTIDOld);

// CFTInterface-Methode: Open a registration
EXTERN_CONV DLL_PORTER
unsigned short usOpenReg(STRUCT_CFTFILE * SCFTID,
                         char * cpUserName,
                         char * cpUserPwd);

// CFTInterface-Methode: Close a registration
EXTERN_CONV DLL_PORTER
unsigned short usCloseReg(STRUCT_CFTFILE * SCFTID);

// CFTInfo-Methode: Get a specified information
EXTERN_CONV DLL_PORTER
unsigned short usGetInfo(STRUCT_CFTFILE * SCFTID,
                         unsigned short usOrderID,
                         char * cpParameterString,
                         char *** cppAnswer,
                         unsigned short * uspNumberOfLines);

// CFTContext-Methode: Set a context-changing order
EXTERN_CONV DLL_PORTER
unsigned short usContextOrder(STRUCT_CFTFILE * SCFTID,
                              unsigned short usOrderID,
                              const char * cpParameterString);

// CFTTransfer-Methode: Open a file
EXTERN_CONV DLL_PORTER
unsigned short usOpenFile(STRUCT_CFTFILE * SCFTID,
                          char * cpFileName,
                          char * cpMode);

// CFTTransfer-Methode: Close a file
EXTERN_CONV DLL_PORTER
unsigned short usCloseFile(STRUCT_CFTFILE * SCFTID);

// CFTTransfer-Methode: Get ASCII-lines
EXTERN_CONV DLL_PORTER
unsigned short usGetLines(STRUCT_CFTFILE * SCFTID,
                          char ** cppLine,
                          unsigned long * ulpLength);

// CFTTransfer-Methode: Put ASCII-lines
EXTERN_CONV DLL_PORTER
unsigned short usPutLines(STRUCT_CFTFILE * SCFTID,
                          char * cpLine,
                          unsigned long ulLength);

// CFTTransfer-Methode: Get binary-block
EXTERN_CONV DLL_PORTER
unsigned short usGetBlock(STRUCT_CFTFILE * SCFTID,
                          char ** cpBlock,
                          unsigned long * ulpLength);

// CFTTransfer-Methode: Put binary-block
EXTERN_CONV DLL_PORTER
```

```
unsigned short usPutBlock(STRUCT_CFTFILE * SCFTID,
                          char * cpBlock,
                          unsigned long ulLength);

// CFTTransfer-Methode: Check End-Of-Transfer
EXTERN_CONV DLL_PORTER
unsigned short usEOT(const STRUCT_CFTFILE * SCFTID);

#endif /* __CPP2CWDMSINTERFACEREMOTE_H_HEADER__ */
```

P.2 Headerdatei für C

```
// ************************************************************************
// * CVS Concurrent Versions Control
// * ---------------------------------------------------------------------
// * $RCSfile: ECFT_RIC.H,v $
// * $Revision: 1.1 $
// * ---------------------------------------------------------------------
// * $Author: neidh $
// * $Date: 2004/10/11 09:47:26 $
// * $Locker:  $
// * ---------------------------------------------------------------------
// * $Log: ECFT_RIC.H,v $
// * Revision 1.1  2004/10/11 09:47:26  neidh
// * Anpassungen und Fertigstellung bis fast Kapitel WDMS
// *
// * Revision 1.3  2004/03/08 16:20:43  neidh
// * Change: Separated Interfaces possible
// * -> Use own interface for several access points not only one global
// * -> Extend SCFTID and add interfaceinfo
// *
// * Revision 1.2  2004/01/16 15:28:10  neidh
// * BugFix: Memory error in methode usCreateCorbaArg
// * - To few memory was allocated for cppCORBAArgv
// *
// * Revision 1.1  2004/01/16 13:32:48  neidh
// * Change: Use DLL for C and C++ sources
// * - set extern "C" by choice
// * - ECFT_RIC.h (C) and ECFT_RI.h (C++)
// *
// ************************************************************************

// ************************************************************************
// * Description:                                                         *
// * This DLL is written to be used by foreign programs to access ECFT-  *
// * interfaces.                                                          *
// * It is the realisation of the jump-in point for DLL-applications.    *
// * -------------------------------------------------------------------- *
// * Used pre-compiler definitions:                                       *
// *                                                                      *
// * Used exception-handles:                                              *
// ************************************************************************

#ifndef __CCODECPP2CWDMSINTERFACEREMOTE_H_HEADER__
#define __CCODECPP2CWDMSINTERFACEREMOTE_H_HEADER__

// Chose external C tagging
#define EXTERN_CONV

#include "ECFT_RI.H"

#undef EXTERN_CONV

#endif /* __CCODECPP2CWDMSINTERFACEREMOTE_H_HEADER__ */
```

Anhang Q

Testszenario einer Vererbung

Nachfolgende Testprogramme entstanden zu Beginn der Arbeiten noch während der Analysephase. Damit sollte verifiziert werden, ob die Ideen zu vererbten Schnittstellen mit den Möglichkeiten von CORBA umgesetzt werden können. Der Versuch bestätigte dies.

Q.1 IDL-Beschreibung der Testschnittstelle

```
interface TextCopy {
    short sOpen();
    string cpGetLine ();
    short sEOF ();
    void vClose ();
    short sRebind (in string cpNSDObjectIOR);
};
```

Q.2 Servercode zum Vererbungstest

```
#include <iostream.h>
#include <fstream.h>
#include <string.h>

#include <CORBA.h>
#include <mico/CosNaming.h>
#include <mico/poa.h>

#include "TextCopy3.h"

// ********************************************************************
// Implementierung der Klassenmethoden
// ********************************************************************

class TextCopy_Skel : virtual public POA_TextCopy
{
private:
    ifstream fsInputFile;
    unsigned short usCorbaSemaphore;
    CORBA::ORB_var COrb;
    CORBA::Object_var CTransferObject;
    CosNaming::Name CServiceName;

public:
    void set (CORBA::ORB_var COrb_new,
              CORBA::Object_var CTransferObject_new,
              CosNaming::Name CServiceName_new);
public:
    TextCopy_Skel ();
    short sOpen ();
    char * cpGetLine ();
    short sEOF ();
    void vClose ();
    short sRebind (const char * cpNSDObjectIOR);
};

TextCopy_Skel::TextCopy_Skel ()
{
    usCorbaSemaphore = 0;
}

void TextCopy_Skel::set (CORBA::ORB_var COrb_new,
                         CORBA::Object_var CTransferObject_new,
```

```
                          CosNaming::Name CServiceName_new)
{
    COrb = COrb_new;
    CTransferObject = CTransferObject_new;
    CServiceName = CServiceName_new;
}

short TextCopy_Skel::sOpen ()
{
    if (usCorbaSemaphore)
    return 2;
    usCorbaSemaphore = 1;
    if (fsInputFile)
    fsInputFile.close ();
    fsInputFile.open ("eingabe.dat");
    if (!fsInputFile)
    return 1;
    else
        return 0;
}

char * TextCopy_Skel::cpGetLine ()
{
    unsigned long ulSize = 256;   // Zeilengroesse
    char * cpLine;        // Efingelesene Zeile
    char * cpDepot;       // Zweite Zeile als Kopierpuffer
    char cIn;             // ein Zeichen

    // Zeile mit Startgroesse erstellen
    cpLine = CORBA::string_alloc (ulSize);
    // Zeile einlesen
    fsInputFile.get (cpLine, ulSize-1, '\n');
    // Pruefen ob naechstes Zeichen '\n', d.h., Zeile vollstaendig
    fsInputFile.get (cIn);
    cpLine[strlen(cpLine)+1] = '\0';
    cpLine[strlen(cpLine)] = cIn;
    while (cIn != '\n' && !fsInputFile.eof ())
    {
        // Zeile nicht vollstaendig
        // Depot anlegen
        ulSize += 256;
    cpDepot = CORBA::string_alloc (ulSize);
        // Bereits gelesenen Zeileninhalt umkopieren
        strcpy (cpDepot, cpLine);
        // Weiteren Inhalt wie gehabt einlesen
        fsInputFile.get (cpLine, ulSize-1, '\n');
        // Gelesenes anfuegen
        strcat (cpDepot, cpLine);
        // Alte Zeilenspeicher freigeben und neuen umhaengen
        CORBA::string_free (cpLine);
        cpLine = cpDepot;
        // Pruefen ob Zeile vollstaendig, wie gehabt
        fsInputFile.get (cIn);
        cpLine[strlen(cpLine)+1] = '\0';
        cpLine[strlen(cpLine)] = cIn;
    }
    // Zeile zurueckgeben
    return cpLine;
}

short TextCopy_Skel::sEOF ()
{
    if (!usCorbaSemaphore)
    return 1;
    return fsInputFile.eof ();
}

void TextCopy_Skel::vClose ()
{
    if (!usCorbaSemaphore)
    return;
    fsInputFile.close ();
    usCorbaSemaphore = 0;
}

short TextCopy_Skel::sRebind (const char * cpNSDObjectIOR)
{
    CORBA::String_var ref = cpNSDObjectIOR;
    CORBA::Object_var CNameServiceObject = COrb->string_to_object (ref);
    CosNaming::NamingContext_var CNamingContext =
    CosNaming::NamingContext::_narrow (CNameServiceObject);
    if (CORBA::is_nil (CNamingContext))
    {
    return 1;
    }
    CNamingContext->rebind (CServiceName, CTransferObject);
    CNameServiceObject->_ior()->print(cerr);
    return 0;
}


int main (int argc, char * argv[])
{
    CORBA::ORB_var COrb;                  // ORB
    CORBA::Object_var CPoaObject;         // POA-Referenz
    PortableServer::POA_var CPoa;         // Klassenreferenz
    PortableServer::POAManager_var CPoaMgr; // POA-Manager
    PortableServer::ObjectId_var CTextTransferOid;    // Object ID
    CORBA::Object_var CTextTransferObject; // Objectreferenz
```

```
    CORBA::Object_var CNameServiceObject; // NameService-Referenz
    CORBA::Object_var CTextCopyObject;    // Dienst-Referenz
    CosNaming::NamingContext_var CNamingContext; // Namenskontext
    CosNaming::Name CTextCopyServiceName; // Name des Dienstes
    TextCopy_Skel * TextTransfer;

    // ORB initialisieren
    COrb = CORBA::ORB_init (argc, argv);

    // POA nutzen
    CPoaObject = COrb->resolve_initial_references ("RootPOA");
    CPoa = PortableServer::POA::_narrow (CPoaObject);
    CPoaMgr = CPoa->the_POAManager ();

    // Servicemethoden an POA eingliedern
    TextTransfer = new TextCopy_Skel;
    CTextTransferOid =
         CPoa->activate_object (TextTransfer);
    CTextTransferObject = CPoa->id_to_reference (CTextTransferOid.in());

    // NameService referenzieren
    CNameServiceObject = COrb->resolve_initial_references ("NameService");
    // NameServiceContext downcasten
    CNamingContext = CosNaming::NamingContext::_narrow (CNameServiceObject);
    if (CORBA::is_nil (CNamingContext))
    {
    cerr << "[ERROR] Can't get access to Name Server" << endl;
        exit (1);
    }


//*****************************
//    CNameServiceObject->_ior()->print(cout);
//*****************************



    // Dienstanbieter vom Nameserver erfragen
    cerr << "Binding TextTransfer in the Name Service ..." << flush;
    CTextCopyServiceName.length (1);
    CTextCopyServiceName[0].id = CORBA::string_dup ("TextCopy");
    CTextCopyServiceName[0].kind = CORBA::string_dup ("");
    CNamingContext->rebind (CTextCopyServiceName, CTextTransferObject);
    cerr << " done." << endl;

    TextTransfer->set (COrb, CTextTransferObject, CTextCopyServiceName);

    // Starten des Serverbetriebs
    cerr << "Running!" << endl;
    CPoaMgr->activate ();
    COrb->run();

    // Herunterfahren
    CPoa->destroy (TRUE,TRUE);
    delete TextTransfer;

    return 0;
}
```

Q.3 Clientcode zum Vererbungstest

```
#include <CORBA.h>
#include <mico/CosNaming.h>
#include "TextCopy3.h"

int main (int argc, char * argv[])
{
    unsigned int uiErrorCode;            // Fehler-Code
    CORBA::ORB_var COrb;                 // ORB
    CORBA::Object_var CFatherNameServiceObject; // NameService-Referenz Vater
    CORBA::Object_var CChildNameServiceObject;  // NameService-Referenz Child
    CORBA::Object_var CTextCopyObject;   // Dienst-Referenz
    CosNaming::NamingContext_var CNamingContext; // Namenskontext
    CosNaming::Name CTextCopyServiceName; // Name des Dienstes
    CORBA::String_var ref;               // Stringbeschreibung der Referenz
    TextCopy_var CTextCopy;              // Klassenreferenz auf Dienste
                                         // (gecastete Dienst-Referenz)
    CosNaming::BindingList_var CBlist;   // Bindingliste des NSD
    CosNaming::BindingIterator_var CBiter;// BindingIterator fuer den NSD
    CosNaming::Binding_var CBind;        // Binding-Variable
    CosNaming::NamingContext * CpNC;     // Zeiger auf NamingContext


    // ORB initialisieren
    COrb = CORBA::ORB_init (argc, argv);

// ------------------------------------------------------------------------------
// Vater-NSD referenzieren
// ------------------------------------------------------------------------------
    cerr << "----> Referentiate Father-NSD " << endl;
    try
    {
        // NameService referenzieren
        CFatherNameServiceObject = COrb->resolve_initial_references ("FatherNameService");
        // NameServiceContext downcasten
        CNamingContext = CosNaming::NamingContext::_narrow (CFatherNameServiceObject);
        if (CORBA::is_nil (CNamingContext))
        {
        cerr << "\n[ERROR] PROG: Can't get access to Name Server" << endl;
            exit (1);
        }
    }
    catch (CORBA::COMM_FAILURE)
    {
        cerr << "\n[ERROR] CORBA: Trouble with communication to NameServer" << endl;
        return 1;
    }
    catch (CORBA::TRANSIENT)
    {
        cerr << "\n[ERROR] CORBA: Can't get full answer from NameServer" << endl;
        return 1;
    }
    catch (CORBA::ORB::InvalidName)
    {
        cerr << "\n[ERROR] CORBA: Can't find adress-setting for 'FatherNameService'!" << endl;
        return 1;
    }

// ------------------------------------------------------------------------------
// Kind-NSD referenzieren und Adressreferenz holen
// ------------------------------------------------------------------------------
    cerr << "----> Referentiate own NSD " << endl;
    try
    {
    CChildNameServiceObject =
                    COrb->resolve_initial_references ("ChildNameService");
        ref = COrb->object_to_string (CChildNameServiceObject);
    }
    catch (CORBA::COMM_FAILURE)
    {
        cerr << "\n[ERROR] CORBA: Trouble with communication to NameServer" << endl;
        return 1;
    }
    catch (CORBA::TRANSIENT)
    {
        cerr << "\n[ERROR] CORBA: Can't get full answer from NameServer" << endl;
        return 1;
    }
    catch (CORBA::ORB::InvalidName)
    {
        cerr << "\n[ERROR] CORBA: Can't find adress-setting for 'ChildNameService'!" << endl;
        return 1;
    }

// ------------------------------------------------------------------------------
// Fuer alle Vatereigenschaften Vererbung anstossen
// ------------------------------------------------------------------------------
    cerr << "----> Inheritance " << endl;
    CpNC = CNamingContext.in();
    CORBA::ULong k = 0;
    CpNC->list (0, CBlist, CBiter);
    if (CORBA::is_nil (CBiter))
    {
    cerr << "Nothing could be inherited" << endl;
    }
    else
```

```
    {
    while (CBiter->next_one (CBind.out()))
    {
            for (k = 0; k < CBind->binding_name.length (); k++)
        {
            cerr << CBind->binding_name[k].id.in() << ": Inheritance is ... ";
                try
                {
            CTextCopyServiceName.length (1);
            CTextCopyServiceName[0].id = CORBA::string_dup (CBind->binding_name[k].id.in());
            CTextCopyServiceName[0].kind = CORBA::string_dup ("");
            CTextCopyObject = CNamingContext->resolve (CTextCopyServiceName);
                    CTextCopy = TextCopy::_narrow (CTextCopyObject);
                    CTextCopy->sRebind (ref.in());
                }
                catch (CosNaming::NamingContext::NotFound)
                {
                    cerr << "\n[ERROR] NameServer: Namecontext not found" << endl;
                    return 1;
                }
                catch (CosNaming::NamingContext::CannotProceed)
                {
                    cerr << "\n[ERROR] NameServer: Trouble with Naming-Context" << endl;
                    return 1;
                }
                catch (CosNaming::NamingContext::InvalidName)
                {
                    cerr << "\n[ERROR] NameServer: Name invalid!" << endl;
                    return 1;
                }
                catch (CORBA::COMM_FAILURE)
                {
                    cerr << "\n[ERROR] CORBA: Trouble with communication to NameServer" << endl;
                    return 1;
                }
                catch (CORBA::TRANSIENT)
                {
                    cerr << "\n[ERROR] CORBA: Can't get full answer from NameServer" << endl;
                    return 1;
                }

                cerr << "done!" << endl;
        }
    }
    }

// ------------------------------------------------------------------------------
// Datenuebertragung anstossen ueber eigenen NSD
// ------------------------------------------------------------------------------
    cerr << "----> Datatransmission using own NSD" << endl;
    // Dienstanbieter vom Nameserver erfragen
    try
    {
        cerr << "Looking up TextCopy ..." << flush;
        CNamingContext = CosNaming::NamingContext::_narrow (CChildNameServiceObject);
        if (CORBA::is_nil (CNamingContext))
        {
        cerr << "\n[ERROR] PROG: Can't get access to Name Server" << endl;
            exit (1);
        }
        CTextCopyServiceName.length (1);
        CTextCopyServiceName[0].id = CORBA::string_dup ("TextCopy");
        CTextCopyServiceName[0].kind = CORBA::string_dup ("");
        CTextCopyObject = CNamingContext->resolve (CTextCopyServiceName);
        cerr << "done!" << endl;
    }
    catch (CosNaming::NamingContext::NotFound)
    {
        cerr << "\n[ERROR] NameServer: Namecontext not found" << endl;
        return 1;
    }
    catch (CosNaming::NamingContext::CannotProceed)
    {
        cerr << "\n[ERROR] NameServer: Trouble with Naming-Context" << endl;
        return 1;
    }
    catch (CosNaming::NamingContext::InvalidName)
    {
        cerr << "\n[ERROR] NameServer: Name invalid!" << endl;
        return 1;
    }
    catch (CORBA::COMM_FAILURE)
    {
        cerr << "\n[ERROR] CORBA: Trouble with communication to NameServer" << endl;
        return 1;
    }
    catch (CORBA::TRANSIENT)
    {
        cerr << "\n[ERROR] CORBA: Can't get full answer from NameServer" << endl;
        return 1;
    }

    // Downcast der Dienstreferenz auf Klassenreferenz
    CTextCopy = TextCopy::_narrow (CTextCopyObject);
    // Empfangenen Text ausgeben
    try
    {
        cerr << "Try to open stream ..." << flush;
        while ((uiErrorCode = CTextCopy->sOpen ()) == 2) {cerr << ".";};
```

```
        if (uiErrorCode == 1)
        {
        cerr << "\n[ERROR] PROG: Can't open file!" << endl;
            return 1;
        }
        cerr << "done." << endl;
        cerr << "Receiving data ..." << flush;
        while (!CTextCopy->sEOF())
        {
            cout << CTextCopy->cpGetLine ();
    }
        cerr << "done." << endl;
        cerr << "Close stream" << endl;
        CTextCopy->vClose ();
    }
    catch (CORBA::COMM_FAILURE)
    {
        cerr << "\n[ERROR] CORBA: Trouble with communication to TextCopyServer" << endl;
        return 1;
    }
    catch (CORBA::TRANSIENT)
    {
        cerr << "\n[ERROR] CORBA: Can't get full answer from TextCopyServer" << endl;
        return 1;
    }


    return 0;
}
```

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

A2X Anything to XML

ACE Adaptive Communication Environment

ADSL Asymmetric Digital Subscriber Line

ANSI American National Standards Institute

API Application Programming Interface

ASCII American Standard Code for Information Interchange

BKG Bundesamt für Kartographie und Geodäsie

CDDIS Crustal Dynamics Data Information System

CDR Common Data Representation

CFT CORBA File Transfer

CGI Common Gateway Interface

CMS Content Management System

COM Component Object Model

CPU Central Processing Unit

CORBA Common Object Request Broker Architecture

COSS Common Object Service Specification

CSS Cascading Style Sheets

CVS Concurrent Versions System

DBMS Database Management System

DCE Distributed Computing Environment

DCMI Dublin Core Metadata Initiatie

DCOM Distributed Component Object Model

DDE Dynamic Data Exchange

DII Dynamic Invocation Interface

DLL Dynamic Link Library

DOM Document Object Model

DTD Document Type Definition

ECFT Extended CORBA File Transfer

EDC EUROLAS Data Center

EJB Enterprise Java Beans

EOF End Of File

ESIOP Environment Specific Inter-ORB Protocol

EUREF European Reference Frame

eVLBI Electronic Very Long Baseline Interferometry

FESG Forschungseinrichtung Satellitengeodäsie

FFT Fast Fourier Transformation

FGS Forschungsgruppe Satellitengeodäsie

FIFO First In First Out Buffer

FTAM File Transfer , Access and Management

FTP File Transfer Protocol

G Großringlaser Wettzell

GIOP General Inter-ORB Protocol

GIS Geographic Information Systems

GPS Global Positioning System

GREF German Reference Frame

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IDL Interface Definition Language

IERS International Earth Rotation and Reference Systems Service

IETF Internet Engineering Task Force

IGS International GPS Service for Geodynamics

IIOP Internet Inter-ORB Protocol

ILRS International Laser Ranging Service

IOR Interoperable Object Reference

IP Internet Protocol

ISAM Index Sequential Access Methode

ISDN Integrated Services Digital Network

IT Information Technologie

JDBC Java Database Connectivity

JIT Just In Time

LAN Local Area Network

LLR Lunar Laser Ranging

NASA National Aeronautics and Space Adminstration

MICO MICO Is CORBA

NISO National Information Standards Organisation

MPI Message Passing Interface

MTS Microsoft Transaction Server

NAT Network Address Translation

NFS Network File System

NGCC National Geomatics Center of China

NSD Naming Service Daemon

NTP Network Time Protocol

ODBC Open Database Connectivity

OLE Object Linking and Exchange

OMA Object Management Architecture

OMG Object Management Group

OOP Objektorientierte Programmierung

ORB Object Request Broker

OSF Open Software Foundation

PC Personal Computer

POA Portable Object Adapter

PVM Parallel Virtual Machine

QoS Quality of Service

RCS Revision Control System

RFC Request For Comment

RINEX Receiver Independent Exchange Format

RMA Remote Memory Access

RMS Root Mean Square Error

RPC Remote Procedure Call

RSI Remote Service Invocation

RTW Radioteleskop Wettzell

SAX Simple API for XML

SGML Standard Generalized Markup Language

SHMIOP Shared Memory Inter-ORB Protocol

SLR Satellite Laser Ranging

SMTP Simple Mail Transfer Protocol

SOAP Simple Object Access Protocol

SOCKS Sockets Secure

SOS-W Satellite Observing System Wettzell

SQL Structured Query Language

SSL Secure Socket Layer

SSLIOP Secure Socket Layer Inter-ORB Protocol

STL Standard Template Library

SVG Scalable Vector Graphics

Tamino Transaktionsarchitektur zum Management von Internet-Objekten

TAN Transaction Number

TAO The ACE Object Request Broker

TCP Transmission Control Protocol

TDSL Telekom Digital Subscriber Line

TID Transaction Identifier

UDDI Universal Discovery, Description and Integration

UDP User Datagram Protocol

UIOP Unix Domain Socket Inter-ORB Protocol

UML Unified Modelling Language

USP Uninterruptable Power Supply

UTC Universal Time Coordinated

VLAN Virtual Local Area Network

VLBI Very Long Baseline Interferometry

W3C World Wide Web Consortium

WAN Wide Area Network

WebDAV Web-based Distributed Authoring and Versioning

WDAP Wettzell Data Access Point

WDMS Wettzell Data Management System

WfMC Workflow Management Coalition

Wf-XML Workflow XML

WLRS Wettzell Laser Ranging System

WSCL Web-Service Conservation Language

WSDL Web-Service Description Language

WWW World Wide Web

XDR External Data Representation

XML Extensible Markup Language

XPath XML Path Language

XPDL XML Process Definition Language

XSD XML Schema Definition

XSL Extensible Stylesheet Language

XSL-FO Extensible Stylesheet Language - Formatting Objects

XSLT Extensible Stylesheet Language for Transformations

Literaturverzeichnis

[ABI02] Abie, Habtamu: *CORBA Firewall Security: Increasing the Security of CORBA Applications*, http://www.ifi.nio.no/~abie/fw.pdf, Download 24.10.2002

[ANS02] Anspann, Claus-Peter: *Nutzung des Audio/Video-Streaming-Service der CORBA-Implementierung TAO für den Transport allgemeiner Daten und vergleichende Leistungsbewertung durch geeignetes Monitoring. Diplomarbeit zur Erlangung des akademischen Grades Diplominformatiker (FH)*, Fachhochschule Regensburg 2002

[ANSI01] ANSI: *The Dublin Core Metadata Element Set*, http://www.niso.org/standards/resources/Z39-85.pdf, NISO Press Bethesda Maryland (USA) 2001, Download 26.09.2004

[APACH04] Apache Software Foundation: *xml.apache.org*, http://xml.apache.org, Download 23.07.2004

[ATTACH02] Attachemate: *Der Nutzen von Web-Services*, http://ch.attachmate.com/Downloads/download_item_new/1,1627,2802_12,00.html, Attachmate International Sales GmbH Mai 2002, Download 08.04.2004

[BEN05] Benz, Benjamin; Reimann, Lars: *Divide et impera!. Absichern des Heimnetzes mit VLANs*, Zeitschrift c't Nr. 1 2005

[BKGW04] BKG Wettzell: *Fundamentalstation Wettzell*, http://www.wettzell.ifag.de, Fundamentalstation Wettzell 2004, Download 01.07.2004

[BLE03] Bleckat, Burkhard; Haberland, Nils; Puls, Stefan: *EAI und Web Services. Rivalen oder Partner?*, http://akea.iwi.unisg.ch/downloads/20020726-eai-ws.pdf, Avinci Region West GmbH 2003, Download 14.04.2003

[DASS04] Dassing, Reiner; Lauber, Pierre; Neidhardt, Alexander; Riepl, Stefan: *Design-Rules für die objektorientierte Programmierung in C++ und die strukturierte Programmierung in C*, Fundamentalstation Wettzell 2004

[DOC04] Center for Distributed Object Computing, Distributed Object Computing Laboratory: *CORBA Security Conformance Statement*, http://www.cs.wustl.edu/~schmidt/ACE_wrappers/TAO/docs/Security/Conformance.html, Center for Distributed Object

Computing, Washington University, Distributed Object Computing Laboratory, University of California at Irvine, Download 19.05.2004

[EMS02] Emsenhuber, Christoph: *COM+ in der Prozessautomatisierung. Diplomarbeit zur Erlangung des akademischen Grades Diplomingenieur*, Institut für Informationsverarbeitung und Mikroprozessortechnik, Johannes Kepler Universität Linz 2002

[ENG02] Engel, Franz-Josef: *Certification Authority*, http://sunfje.rz.uni-sb.de/CA/X509/short.html, Download 10.04.2002

[FGS01] Forschungsgruppe Satellitengeodäsie: *Forschungs- und Entwicklungsprogramm 2001 - 2005*, München/Frankfurt/Bonn Juni 2000

[FIN04] Finkenzeller, Stefan: *Internet-Festplatte. Microsofts IIS als WebDAV-Server*, Zeitschrift c't Nr. 24 November 2004

[FOG02] Fogel, Karl; Bar, Moshe: *Open Source-Projekte mit CVS. Verteilte Softwareentwicklung mit dem Concurrent Versions System*, 2. Auflage. mitp-Verlag Bonn 2002

[FOW00] Fowler, Martin; Scott, Kendall: *UML konzentriert. Eine strukturierte Einführung in die Standard-Objektmodellierungssprache*, 2. Auflage, Addison-Wesley Verlag München 2000

[FRI04] Friedl, Stefan; Ludwig, Thomas: *Kartenkontrolle. Job-Management in Rechnerclustern*, Zeitschrift iX Nr. 11 November 2004

[HERP02] Herpers, Franz-Josef; Sebestyen, Thomas J.: *XSL. Das Einsteigerseminar*, verlag moderne industrie Buch AG & Co. KG Landsberg 2002

[HOLM04] Holm, Frode; Wolf, Ahmi; Iovino, Francisco: *ATON Report 2001.06.4: CORBA Measurements and Programmer Support*, http://www.create.ucsb.edu/ATON/01.06/Bench_Templ.pdf, Download 26.04.2004

[HUS04] Husson, Van: *ILRS Normal Point Format*, http://ilrs.gsfc.nasa.gov/products_formats_procedures/normal_point/np_format.html, ILRS 2003, Download 24.08.2004

[JOB92] Jobst, Fritz: *Compilerbau. Von der Quelle zum professionellen Assemblertext*, Carl Hanser Verlag München Wien 1992

[KAZ02] Kazakos, Wassilios; Schmidt, Andreas; Tomczyk, Peter: *Datenbanken und XML*, Springer-Verlag Berlin Heidelberg 2002

[KIR04] Kirchmair, Stefan: *Dynamische Webseiten und Kommunikationsmethoden. Bachelor thesis*, http://homepage.uibk.ac.at/homepage/csad/csad2633/thesis_t.pdf, Universität Innsbruck (A) Sep. 2004, Download 12.11.2004

[KLAR95] Klar, Rainer: *Messung und Modellierung paralleler und verteilter Rechensysteme*, B. G. Teubner Stuttgart 1995

[KOP97] Kopp, Herbert: *Bildverarbeitung interaktiv. Eine Einführung mit multimedialem Lernsystem auf CD-ROM*, B. G. Teubner Stuttgart 1997

[KOW03] Kowalski, Marta: *Das WebDav Protokoll. Eine Erweiterung von HTTP 1.1*, http://www.fh-wedel.de/~si/seminare/ws01/Ausarbeitung/a.webdav/WebDav0.htm, Fachhochschule Wedel, Download 10.06.2003

[KREKL04] Krekel, Holger: *Virtual Interview of Douglas C. Schmidt*, http://www.heise.de/ix/artikel/E/2000/10/062/, iX 10/2000, Download 29.03.2004

[LANG02] Lang, Ulrich: *Using CORBA and SSL in a large Banking Environment*, http://www.objectsecurity.com/whitepapers/corba/banking/docsec_final.ppt, University of Cambridge, Download 24.10.2002

[LEO05] Leo Dictionary Team: *LEO - Ein Online-Service der Informatik der Technischen Universität München*, http://dict.leo.org/, Technische Universität München, Download 16.01.2005

[LEX04] Net-Lexikon: *Net-Lexikon*, http://www.net-lexikon.de/, akademie.de, Download Juni/Juli/November 2004 und Januar 2005

[LIB00] Liberty, Jesse: *C++ in 21 Tagen*, Markt&Technik Verlag München 2000

[MAG93] Maguire, Steve: *Writing solid code. Microsoft's Techniques for Developing Bug-Free C Programs*, Microsoft Press Redmond, Washington (USA) 1993

[MAND00] Mandl, Peter; Bauer, Nikolai: *Rollenspiel. Chancen und Risiken der EJB-Komponententechnik*, Zeitschrift iX Nr. 1 Januar 2000

[MANG05] Mangione, Carmine: *Performance tests show Java as fast as C++. Java has endured criticism for its laggard performance (relative to C++) since its birth, but the performance gap is closing*, http://www.javaworld.com/javaworld/jw-02-1998/jw-02-jperf_p.html, Java World, Download 05.01.2005

[MEI04] Meinberg, Werner: *NTP - Network Time Protocol*, http://www.ntp-zeit.de, Meinberg Funkuhren Bad Pyrmont, Download 21.11.2004

[MERKL02] Merkle, Bernhard: *IIOP: Alternative zu HTTP*, http://www.heise.de/ix/artikel/1997/07/136, iX 7/1997, Download 22.10.2002

[MINT02] Minton, Gabriel: *IIOP Specification: A Closer Look*, http://www.blackmagic.com/people/gabe/iiop.html, UNIX Review 1997, Download 21.10.2002

[MOE04] Möller, Ralf: *Vorlesung „Software-Engineering“*, http://www.sts.tu-harburg.de/~r.f.moeller/lectures/se-ss-04/08-ProjMngmt-Versioning.pdf, TU Harburg, Download 26.09.2004

[MPI04] MPI Forum: *The Message Passing Interface (MPI) standard*, http://www-unix.mcs.anl.gov/mpi/, Download 07.04.2004

[MUNG04] Mungee, Sumedh; Surendran, Nagarajan; Schmidt, Douglas C.: *The Design and Performance of a CORBA Audio/Video Streaming Service*, Department of Computer Science, Washington University 1999, http://www.cs.wustl.edu/~schmidt/PDF/av.pdf, Download 21.07.2004

[NEHM04] Nehmer, J. (Hrsg.), Sturm, P. (Hrsg.): *Middleware - Betriebssysteme der Zukunft.* , http://www.syssoft.uni-trier.de/systemsoftware/Download/Fruehere_Veranstaltungen/Seminare/Middleware/middleware.book.pdf 1998, Download 06.04.2004

[NEI03] Neidhardt, Alexander: *Working Report about the installation of a GPS-permanent-site at Lhasa/Tibet (2003-10-13 – 2003-10-24)*, Interner Bericht, Fundamentalstation Wettzell 2003

[NWG04] Network Working Group: *Hypertext Transfer Protocol – HTTP/1.1*, http://www.faqs.org/rfcs/rfc2068.html, Download 26.09.2004

[OCI00] OCI: *TAO Developer's Guide. Building a standard in performance*, Version 1.1a, Object Computing, Inc. (USA) 2000

[OGIS04] Open GIS Consortium: *The OpenGIS™ Abstract Specification. Topic 11: OpenGIS(tm) Metadata (ISO/TC 211 DIS 19115)*, http://www.opengis.org/docs/01-111.pdf,Open GIS Consortium Wayland (USA), Download 08.01.2004

[OMGF04] OMG: *CORBA Firewall Traversal Specification*, http://www.omg.org/docs/ptc/04-03-01.pdf, May 2004, Download 31.07.2004

[OMGFT04] OMG: *CORBA-FTAM/FTP Interworking Specification*, http://www.omg.org/docs/formal/02-03-13.pdf, March 2002, Download 31.07.2004

[OMGSOAP04] OMG: *CORBA to WSDL/SOAP Interworking Specification*, http://www.omg.org/docs/formal/03-11-02.pdf, 2003, Download 31.07.2004

[ORN04] Oak Rich National Laboratory: *The PVM System*, http://www.netlib.org/pvm3/book/node17.html, Download 07.04.2004

[ORYAN02] O'Ryan, Carlos: *Building a Stock Quoter with TAO - A Tutorial*, http://www.cs.wustl.edu/~schmidt/ACE_wrappers/TAO/docs/tutorials/Quoter/index.html 2002, Download 06.04.2004

[OTH04] Othman, Ossama; Schmidt, Douglas C.: *Optimizing Distributed System Performance via Adaptive Middleware Load Balancing*, http://www.cs.berkeley.edu/~bodik/om2001/papers/ossama.pdf, Download 18.06.2004

[PUD01] Puder, Arno; Römer, Kay: *Middleware für vereteilte Systeme*, 1. Auflage, dpunkt.verlag GmbH Heidelberg 2001

[PUDPRO01] Puder, Arno: *CORBA Product Profiles*, http://www.ap-c.org/corba/matrix, Download 07.09.2001

[PUDPRO04] Puder, Arno: *CORBA Product Profiles*, http://www.ap-c.org/corba/matrix, Download 18.05.2004

[RAY01] Ray, Erik T.: *Einführung in XML*, O'Reilly Verlag GmbH & Co. KG Köln 2001

[ROEM01] Roemer, Kay: *MICO - MICO Is CORBA*, http://opensource.ee.ethz.ch/compet-sites/[...], Download 06.09.2001

[SCHI98] Schildt, Herbert: *C++: The Complete Reference*, Third Edition. McGraw-Hill Companies Berkeley (USA) 1998

[SCHOE95] Schöning, Uwe: *Logik für Informatiker*, 4. Auflage, Spektrum Akademischer Verlag GmbH Heidelberg Berlin Oxford 1995

[SCHOE97] Schöning, Uwe: *Theoretische Informatik - kurzgefaßt*, Spektrum Akademischer Verlag GmbH Heidelberg Berlin 1997

[SCHR199] Schreiner, Rudolf: *TIS plug-gw as a CORBA firewall*, http://www.objectsecurity.com/whitepapers/corba/tis_plug_gw/cfwexp1.html, 1999, Download 23.10.2002

[SCHR299] Schreiner, Rudolf: *SOCKS as a CORBA firewall*, http://www.objectsecurity.com/whitepapers/corba/socks/socks_exp.html, 1999, Download 23.10.2002

[SELL00] Sellentin, Jürgen: *Datenversorgung komponentenbasierter Informationssysteme*, Springer Verlag Berlin Heidelberg 2000

[SIER02] Sierig, Peter: *Das Microsoft-Internet. .NET und was dranhängt*, http://www.heise.de/ct/02/04/086/, c't 2002, Download 08.04.2004

[SING94] Singhal, Mukesh; Shivaratri, Niranjan G.: *Advanced Concepts in Operating Systems*, McGraw-Hill, Inc. (USA) 1994

[STE04] Stein, Greg; Whitehead, Jim [Maintainer]: *Welcome to WebDAV Resources*, http://www.webdav.org/, Download 17.11.2004

[STEV92] Stevens, W. Richard: *Programmieren von UNIX-Netzen. Grundlagen, Programmierung, Anwendung*, Prentice-Hall International, Inc. London 1992

[STRO92] Stroustrup, Bjarne: *Die C++ Programmiersprache*, 2. Auflage. Addison-Wesley 1992

[TID02] Tidwell, Doug: *XSLT. XML-Dokumente transformieren*, 1. Auflage. O'Reilly Verlag GmbH & Co. KG Köln 2002

[TOR03] Torge, Wolfgang: *Geodäsie*, 2. Auflage, Walter de Gruyter GmbH & Co. KG Berlin 2003

[UZ04] Uni Zürich: *Fallstudie: DCE*, http://www.ifi.unizh.ch/~riedl/lectures/V-8-1.pdf, Download 06.04.2004

[VLI03] Vlist, Eric van der: *XML Schema. XML-Daten modellieren*, 1. Auflage. O'Reilly Verlag GmbH & Co. KG Köln 2003

[WAE04] Wächtler, Peter: *Kerberos. Eine Frage des Vertrauens*, http://www.linux-magazin.de/Artikel/ausgabe/1999/05/Kerberos/kerberos.html, Linux-Magazin 05/1999, Download 14.11.2004

[WALD95] Waldschmidt, Klaus (Hrsg.): *Parallelrechner. Architekturen - Systeme - Werkzeuge*, B. G. Teubner Stuttgart 1995

[WEB104] http://www.uni-paderborn.de/cs/ag-taentzer/internet/CGI_ODBC.ppt, Download 08.04.2004

[WEB204] http://www.uni-bayreuth.de/departments/math/~rbaier/lectures/ws2001-02/java_multimedia/details/node7.html, Download 08.04.2004

[WEB304] http://dict.die.net/xdr/, Download 13.04.2004

[WEB404] http://www.webopedia.com/TERM/s/socket.html, Download 08.04.2004

[WEB504] http://www.dpunkt.de/java/Die_Sprache_Java/Einleitung/11.html, Download 14.04.2004

[WEB602] http://www.mico.org/FrameDescription.html, Download 25.01.2002

[WFMC04] WfMC: *The Workflow Management Coalition*, http://www.wfmc.org, Download 05.10.2004

[WOJ03] Wojke, Guido: *Webservices als Multiagentensystem am Beispiel des UniCats Projekts. Studienarbeit*, Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe 2003

[XTRA04] Xtradyne Technologies AG: *The I-DBC - Xtradyne's IIOP Firewall*, http://www.xtradyne.com/products/i-dbc/i-dbc.htm, Download 11.10.2004

Index