FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Ein flexibler und effizienter Dateitransfer für UNICORE

Bastian Tweddell

FZJ-ZAM-IB-2006-01

Februar 2006

(letzte Änderung: 6.2.2006)

Hochschule Aachen
Standort Jülich

Fachbereich:
Angewandte Naturwissenschaften und Technik

Lehr- und Forschungsbereich:
Technomathematik

Ein flexibler und effizienter Dateitransfer für UNICORE

Diplomarbeit von Bastian Tweddell-Krumbügel

Jülich, Februar 2006

Die vorliegende Diplomarbeit wurde in Zusammenarbeit mit der Forschungszentrum Jülich GmbH, Zentralinstitut für Angewandte Mathematik (ZAM), angefertigt.

Diese Diplomarbeit wurde betreut von:

Referent: Prof. Dr. rer. nat. Volker Sander

Koreferent: Dr. Bernd Schuller

Diese Arbeit ist selbstständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Jülich, Februar 2006

Inhaltsverzeichnis

Kapitel 1

Einleitung

Schon zu Beginn der ersten Computernetzwerke war es von großer Bedeutung, Daten, meist in Form von Dateien, von einem Rechner über ein Netzwerk zu einem anderen Rechner zu übertragen. Im Laufe der Zeit haben sich Schritt für Schritt unterschiedliche Konzepte zu Dateitransfers entwickelt, welche jeweils auf festgelegte Anforderungen abgestimmt sind. Heutzutage gibt es einige weitverbreitete Lösungen, welche auch in den gängigen Betriebssystemen manifestiert sind. Dabei unterscheiden sie sich in elementaren Gesichtspunkten. Diese sind auf der einen Seite Effizienz und Datendurchsatz, während auf der anderen Seite Flexibilität und Sicherheit dagegenstehen.

Im Rahmen dieser Arbeit wird ein Programmsystem entwickelt, welches in erster Linie daraufhin ausgerichtet ist, eine flexible Programmbibliothek bereitzustellen. Flexibel bedeutet hier, dass die unterschiedlichen Bedürfnisse der Anwender gedeckt werden. Dazu zählt, dass zum einen die Verwendung unterschiedlicher Übertragungsprotokolle möglich sein soll, wodurch auch zum anderen Sicherheitsaspekte geboten werden können, wie Verschlüsselung und Authentifizierung der Kommunikationspartner.

Aus der Sicht eines Programmierers soll die entwickelte Bilbiothek einfache Anbindungen an die Umgebung haben, so dass diese ohne großen Aufwand in einem beliebigen Programmsystem zur Verwendung kommen kann. Zur Demonstration wird diese Bibliothek in einem schon bestehenden Grid-System zur Anwendung kommen. Dabei handelt es sich um die etablierte und umfangreiche Software namens UNICORE.

Die Namensgebung der neu entwickelten Bibliothek führte zu „*HPFT*“. Diese Abkürzung steht für

High **P**erformace **F**ile **T**ransfer

und fasst damit die wichtigsten Eigenschaften dieses Konzeptes zusammen.

Das Dokument wird zunächst auf die Grundlagen von Grid-Systemen und Dateitransfers eingehen, wobei UNICORE und schon existierende Dateitransferlösungen genauer unter die Lupe genommen werden. Im folgenden Kapitel wird auf die Entwicklungsgeschich-

te von *HPFT* eingegangen. Dort werden die Fragen zu dem Softwaredesign im Detail erläutert. Aufbauend auf diesen Erkenntnissen wird gezeigt, wie *HPFT* aufgrund seiner flexiblen Softwarearchitektur in einer bereits existierenden Umgebung zur Anwendung kommt.

Kapitel 2

Vertiefung des Themas

Das „Zentralinstitut für Angewandte Mathematik (ZAM)“, ansässig in der „Forschungszentrum Jülich GmbH“, ist verantwortlich für die Softwarewartung des UNICORE Klienten. Dies bot ideale Vorraussetzungen, diese Arbeit dort anzufertigen. Der ursprüngliche Anlass für das *HPFT*-Projekt war, einen effizienteren Dateitransfer für das *Grid-System* UNICORE zu entwickeln, da der bisherige aufgrund seiner verstrickten Architektur nur sehr geringen Datendurchsatz zulässt.

Die praktische Aufgabe kann in zwei Bereiche geteilt werden. Der erste Teil beschäftigt sich mit dem Design einer flexiblen Dateitransferlösung. Dieser soll transparent für den Benutzer sein und verschiedene Protokolle unterstützen. Anhand dieser Ergebnisse wurde eine Programmbibliothek implementiert, welche im zweiten Teil zur Demonstration in UNICORE zur Anwendung kommt, wodurch eine Verbesserung des Datendurchsatzes angestrebt wird.

2.1 Grid-Systeme

Der Begriff *Grid-System* umfasst einen relativ neuen Bereich der Informatik, welcher erst 1998 von Ian Foster und Carl Kesselman geprägt worden ist. In ihrer Veröffentlichung *„The Grid: Blueprint for a New Computing Infrastructure“* [3] definierten sie das *Grid* als eine Hard- und Softwareinfrastruktur, welches einen zuverlässigen, beständigen und kostengünstigen Zugang zu Hochleistungsrechnern bietet[1]. Vor dieser Zeit wurden die Bemühungen, verteilte Ressourcen über ein Netzwerk zu nutzen, als *Metacomputing* bezeichnet.

Obwohl *Grid-Systeme* nun mehr als 6 Jahre in der Entwicklung sind, ist man damit

[1] „A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.“ [3]

beschäftigt, den Begriff „*Grid*“ exakt zu definieren. Die Kernkomponenten, welche die Infrastruktur eines *Grids* ausmachen, bedarf es immer noch zu erforschen und festzulegen. *Grid-Systeme* haben sich mittlerweile von einer behutsam konfigurierten Infrastruktur, die nur eine begrenzte Anzahl von Anwendungen auf wenigen Hochleistungsrechnern unterstützten, zu einer nahtlosen und dynamischen Anwendungsumgebung entfaltet. Die heute gängige Definition „*Grid-System*“ wurde 2001 von Foster, Kesselman und Tuecke neu definiert und legt das *Grid* als eine koordinierte, gemeinsame Ressourcennutzung für Problemlösungen in dynamischen, multi-institutionellen, virtuellen Organisationen aus[2].

Darauf folgend erstellte Foster eine Prüfliste, anhand deren drei Punkte ein System als ein *Grid* identifiziert werden kann.

1. Es wird gefordert, dass es eine koordinierte Ressourcennutzung mehrerer Anwender gibt, welche aber nicht von einer zentralisierten Instanz verwaltet wird. Das heißt, die Anwender unterliegen auf jeder genutzten Ressource stets einer eigenständigen Administration.

2. Es wird gefordert, dass innerhalb eines *Grid-Systems* standardisierte und öffentliche Protokolle und Schnittstellen zur Anwendung kommen. Ist dies nicht der Fall, so liegt die Vermutung nahe, dass es sich hierbei mehr um ein anwendungsspezifisches System handelt.

3. Es werden nicht triviale Dienste von dem *Grid* gefordert, welche im Zusammenspiel durch die Koordination einen größeren Nutzen haben, als jeder Dienst für sich allein.

Grid-Systeme bei IBM [4]
IBM erweitert die Definition des *Grid-Systems* um den Aspekt der Wirtschaftlichkeit und der Nutzenmaximierung von verteilten Ressourcen. Es existieren bereits Lösungen zu *Grid-Systemen*, welche bestimmte Arten der wirschaftlichen Nutzung unterstützen. Darunter fallen die folgenden Punkte:

- Es steht mehr Rechenleistung für wirtschaftliche Anwendungen zur Verfügung
- Analysen sind kostengünstiger
- Analysen können genauer und öfter berechnet werden
- Ergebnisse sind schneller verfügbar
- Schneller Zugang zu sämtlichen Informationen, unabhängig von der Lage innerhalb der Organisation

[2] „A computational grid is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.“[3]

- Stoßkanten zwischen verschiedenen Hardware- und Softwarekomponenten werden gelöst

2.1.1 Klassifikation von Data/Compute Grids

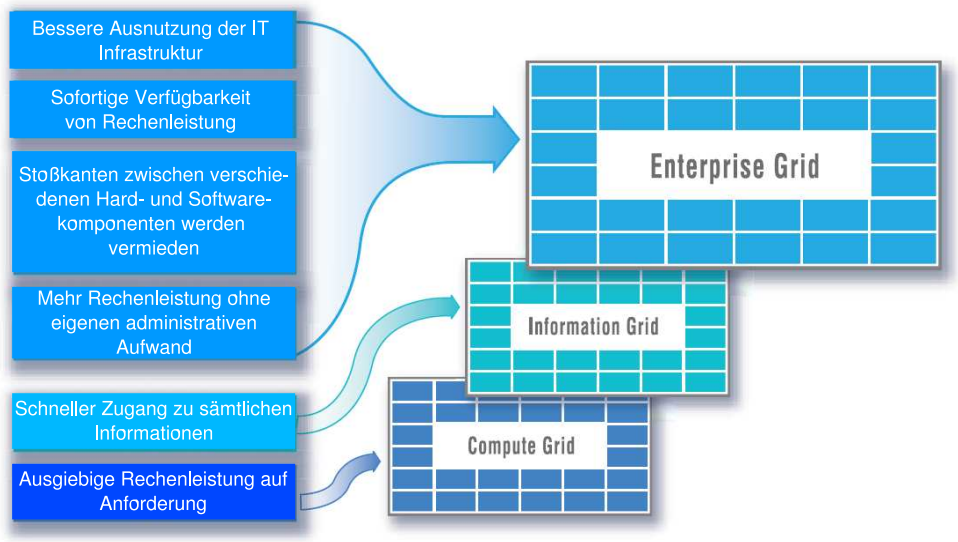

Abbildung 2.1: Klassifikation von Grid-Systemen

Wie in der Abbildung 2.1 zu sehen ist, lassen sich *Grid-Systeme* in drei Bereiche unterteilen.

1. Rechen-Grids bieten einen Zugriff auf verteilte Rechen-Ressourcen an. Diese sollen jederzeit zur vollen Verfügung stehen.

2. Informations-Grids hingegen bieten dem Benutzer einen Zugriff auf Informationen und Daten an.

3. Enterprise-Grids umfassen die beiden Bereiche um Rechen- und Informations-Grids. Diese Kombination ermöglicht es, innerhalb eines Unternehmens eine optimale Ressourcenaussnutzung zu erreichen.

2.1.2 UNICORE

UNICORE bezeichnet ein in Deutschland entwickeltes *Grid-System*, welches mittlerweile in Europa weit verbreitet genutzt wird. Dieser Abschnitt bezieht sich auf die Entste-

hungsgeschichte von UNICORE und lässt auch einen gewissen Einblick auf Implementierungsdetails zu. Dadurch werden essentielle Voraussetzungen in Bezug auf die Einbettung von *HPFT* in UNICORE vorgegeben.

UNICORE Einleitung

Bereits ein Jahr bevor Foster und Kesselman ihre Definition zu *Grid- Systemen* veröffentlichten, wurde 1997 vom Wissenschaftsrat in Deutschland eine Empfehlung herausgegeben, bis zu vier Supercomputer-Zentren in Deutschland zu errichten und mit Hochleistungsrechnern auszustatten. Somit war der Grundstein für Grid-Systeme auch in Deutschland gelegt. UNICORE war ein vom BMBF[3] finanziertes Projekt, welches nach 3 Jahren Entwicklungszeit mit seinen umfassenden Eigenschaften im Bereich Grid-Computing als weltweit etabliertes Softwaresystem angesehen und genutzt wird. Die UNICORE-Software wird aus Gründen der immer weitergehenden Forschung und Entwicklung unter Open Source Lizenzen frei zur Verfügung gestellt[4].

Das Hauptziel des Projektes UNICORE war es, einen nahtlosen, sicheren und intuitiven Zugang zu verteilten Ressourcen von deutschen Hochleistungssystemen zu entwickeln. Die Bedeutung der Abkürzung UNICORE

„**Uni**form **I**nterface to **Co**mputing **Re**sources“

fasst die umgesetzten Ziele des Projektes gut zusammen[5]. Diese sind im Einzelnen:

- UNICORE musste Stoßkanten verbergen, welche von unterschiedlichen Hardwarearchitekturen, Betriebssystemen, Dateisystemen und Sicherheitseinrichtungen herrühren.
- UNICORE musste eine geeignete Sicherheitsinfrastruktur verwenden. Diese beruht auf Verwendung von X.509 Zertifikaten, welche die Authentizität von Servern, Rechner, Software und Benutzern garantiert.
- UNICORE sollte von Wissenschaftlern und Ingenieuren gleichermaßen angewendet werden können, ohne zuvor fundierte Kenntnisse über die Hintergründe zu erlernen. Dafür existiert eine grafische Benutzeroberfläche, welche beim Erstellen von komplexen Jobs Unterstützung bietet.

Zu UNICORE gehört ein Umfang von mächtigen Kernfunktionen, die es dem Benutzer erlauben, auf einfache Art und Weise komplexe *Batch-Jobs* zu erstellen, verwalten und auf verschiedenen Systemen auszuführen. Ein *Batch-Job* beinhaltet in der Regel mehrere

[3] Bundesministerium für Bildung und Forschung
[4] `http://unicore.sourceforge.net`
[5] Einheitliche Schnittstelle zu Rechen- und Datenressourcen

Aufgaben, welche an ein Ressourcenverwaltungssystem übergeben werden. Dort wird die Abarbeitungsreihenfolge festgelegt. Der Begriff Job wird im Kapitel 2.1.3 genau erklärt und kann zunächst als Bearbeitungsauftrag an den Server gesehen werden.

Erstellen und Absenden von Jobs: Die grafische Oberfläche vom UNICORE-Klienten unterstützt den Anwender bei der Erstellung von komplexen und voneinander abhängigen Jobs, welche auf jedem UNICORE-System ohne eine Änderung zur Ausführung gebracht werden kann. Um Flexibilität zu gewährleisten, kann ein UNICORE-Job rekursiv weitere Jobs mit sich bringen.

Vor dem Absenden des erstellten Jobs wird von diesem auf der Klientseite eine abstrakte Abbildung (AJO[6]) in Form eines serialisierten Java Objektes erstellt. Das AJO wird via UPL[7] an das gewählte Gateway transferiert. Das an das Gateway übermittelte AJO kann nur auf einer sogenannten UNICORE Vsite[8] ausgeführt werden. Dazu analysiert das Gateway dieses AJO und leitet es an die entsprechende UNICORE Vsite weiter.

Job Management: Der UNICORE-Klient bietet dem Benutzer die volle Kontrolle über die Jobs und deren Daten. Die jeweiligen Status, welche mittels direkter Nachfrage abgerufen werden können, sind über verschiedene Farben gekennzeichnet. Das bringt eine intuitive Kontrolle über die verwalteten Jobs. Der Benutzer kann ohne weiteres erkennen, wo Fehler und wo keine Fehler aufgetreten sind.

Daten Management: Die UNICORE-Jobs enthalten Aufgaben, welche auf verschiedenen UNICORE-Systemen in von einander getrennten Computerzentren verteilt sein können. Die Ergebnisse (Output) eines Jobs können als notwendige Eingabe von einem Nachfolgejob benötigt werden. Es wird ein temporäres Verzeichnis namens *Uspace*[9] für jeden Job angelegt. Der Benutzer muss dafür im Voraus festlegen:

- welche Daten in den *Uspace* importiert werden müssen. Die Quellen können auf dem eigenen Klientrechner oder auf der UNICORE Site liegen.
- welche Daten nach Beendigung des Jobs exportiert werden sollen, um ein Löschen dieser zu verhindern.
- welche Daten zu einem anderen *Uspace* transferiert werden sollen.

[6] Abstract Job Object
[7] UNICORE Protocol Layer
[8] Virtuelle Site
[9] UNICORE space

Anwendungen: Die drei oben genannten Funktionen stellen das Grundgerüst zur effektiven Nutzung von verteilten Systemen. In der Forschung werden die verschiedesten Anwendungspakete zur Lösung bestimmter Problemstellungen benutzt. Über ein UNICORE Plugin können die geforderten Anwendungen auch über den UNICORE-Klient angesprochen werden. Dies unterstützt ein Angebot von nicht trivialen Diensten, wie es bei der Identifikation von *Grid-Systemen* (siehe Kapitel 2.1.1) erwähnt wurde und wirkt sich stark in der Flexibilität des Gesamtsystems aus.

Ressourcen Management: Das von UNICORE verwendete Management basiert auf einem vollständig dezentralisierten System. Die UNICORE Vsites werden von den lokalen Systemadministratoren gewartet, so dass diese Informationen über die vorhandenen Ressourcen publizieren. Es werden zunächst die Anforderungen eines Jobs mit den gebotenen Ressourcen verglichen und festgestellt, ob die Ausführung auf der gewählten UNICORE Vsite möglich ist.

Die UNICORE Architektur

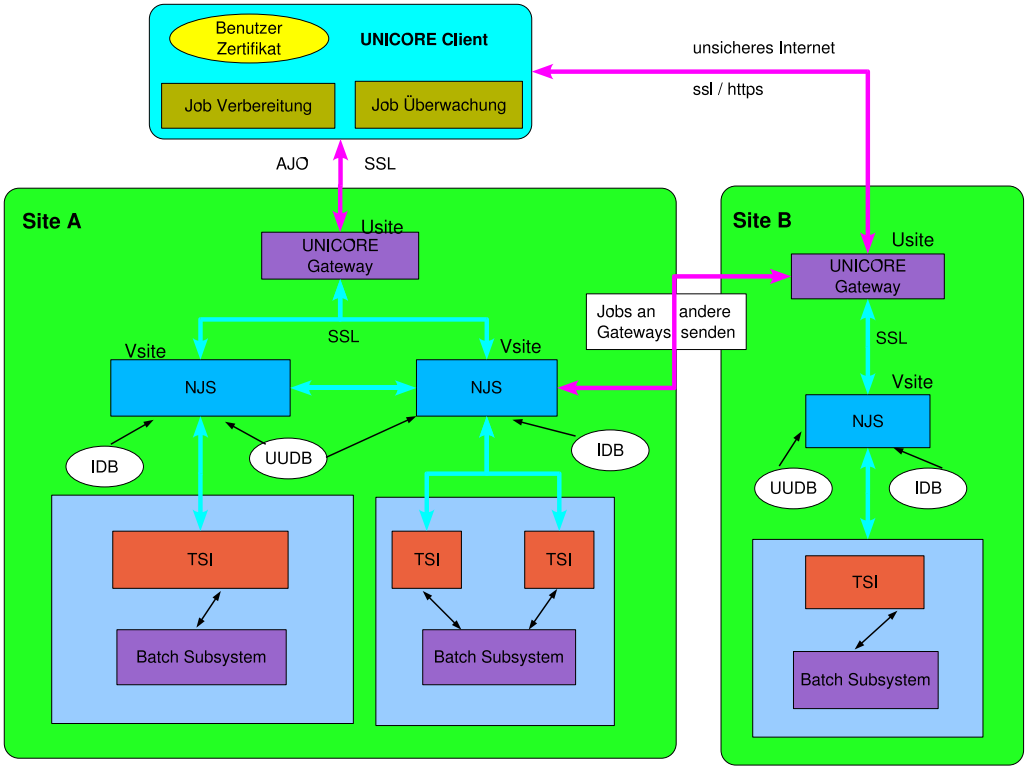

Abbildung 2.2: Die UNICORE Architektur

Die Softwarearchitektur von UNICORE basiert auf einem dreischichtigen- oder *Three-Tier*-Modell. Im Gegensatz zur zweischichtigen- bzw. *Two-Tier*-Architektur, bei der die Rechenkapazität weitestgehend auf die Klientrechner ausgelagert wird, um den Server zu entlasten, existiert bei der dreischichtigen Architektur noch eine zusätzliche Schicht. Diese wird als Logik-Schicht bezeichnet, welche die Datenverarbeitung vornimmt. Bei UNICORE existieren die folgenden drei Schichten:

1. UNICORE Klient
2. UNICORE Usite (Gateway)
3. UNICORE Vsite (NJS[10]/TSI[11])

Wie sich die einzelnen Schichten verhalten und welche Aufgaben diese haben, ist in den folgenden drei Paragraphen erklärt:

[10]Network Job Supervisor
[11]Target System Interface

Der UNICORE Klient wird von dem Benutzer auf einem Java-fähigen Rechner gestartet, der sich über ein Netzwerk mit einer UNICORE Usite (dem Gateway) verbindet. Dazu wird das x509-Zertifikat des Benutzers verwendet, um eine SSL[12]-Verbindung (siehe Kapitel 2.1.4) aufbauen zu können, worüber die einzelnen Aufgaben und Daten gesendet werden können.

Das UNICORE Gateway stellt den einzigen Zugangspunkt für den Klienten zu einer *Usite* dar. Es existiert nur eine Internetadresse und ein geöffneter Kommunikationsendpunkt (TCP-Port), was eine einfache Integration hinter einer Firewall ermöglicht. Das Gateway ist dafür verantwortlich, dass der vom UNICORE-Klient gesendete Job, in Form von abstrakten Daten, analysiert und zur weiteren Verarbeitung an ein gewähltes Endsystem übertragen wird.

Die UNICORE Vsite ist in die beiden Server „*Network Job Supervisor*“ und „*Target System Interface*“ aufgeteilt.

Der *Network Job Supervisor* (NJS) Server handhabt alle gesendeten UNICORE-Jobs. Anhand der *UNICORE User Data Base* (UUDB) wird nach einem passenden Benutzer-Loginnamen für das im Job mitgesendete Benutzerzertifikat gesucht, unter wessen Namen der Job auf dem Zielsystem ausgeführt wird. Der NJS Server extrahiert aus dem AJO einzelne Teiljobs, welche auf das Zielsystem (TSI) abgestimmt sind. Dabei werden auch die Abhängigkeiten der einzelnen Teilaufgaben untereinander berücksichtigt. Zusätzlich speichert der NJS-Server auch alle Job Status, die vom Klient explizit abgerufen werden können. Müssen einige Teiljobs auf andere Usites übertragen werden, so übernimmt der NJS Server die Rolle eines UNICORE-Klienten und delegiert diesen als eigenständigen Job weiter.

Der *Target System Interface* (TSI) Server ist die ausführende Instanz und empfängt die vom NJS Server erstellten Jobs. Diese werden an das lokale Batch-Verarbeitungssystem weiter geleitet, wo sie ausgeführt werden können. Enthält ein Job einen Dateitransfer, so werden diese vom TSI aus initiiert, da auch auf dem Zielsystem der Bedarf der Daten besteht.

[12] Secure Sockets Layer

2.1.3 Das UNICORE Job-Modell

Jeder Job, welcher vom Benutzer im UNICORE-Klient angefertigt wird, muss einer ausgewählten UNICORE Vsite zugewiesen werden, auf welcher alle dem Job zugehörigen Aufgaben bearbeitet werden. Es besteht jedoch die Möglichkeit, einem Job Unterjobs zuzuweisen, welche wiederum auf ihrer speziell zugewiesenen Vsite laufen. Diese Unterjobs werden als eigenständige Jobs behandelt und unterliegen demnach den gleichen Regeln wie ein normaler Job, das heißt, sie können auch wieder untergeordnete Jobs beinhalten.

Der Job wird auf dem Zielsystem unter den Rechten eines Benutzers ausgeführt. Dort wird ein temporäres Verzeichnis - der *Uspace*[13] - angelegt, in welchem die Daten für den gerade laufenden Job gespeichert werden. Alle zur Ausführung benötigten Dateien müssen demnach vor der Ausführung des Jobs in das *Uspace* importiert werden. Die Quelle dieser Datei kann sich einerseits schon auf dem Zielsystem - dem *Xspace*[14] - befinden oder auf dem lokalen Rechner - dem *Nspace*[15] - des Benutzers.

Da der *Uspace* nach der Ausführung des Jobs nicht mehr existiert, müssen die erzeugten Output-Dateien, welche zur weiteren Auswertung benötigt werden, bei der Erstellung des Jobs explizit als Export angegeben werden. Diese werden dann vor der Löschung des *Uspace* in einen weiteren temporären Datenraum verlegt, bis diese vom Benutzer abgeholt werden.

2.1.4 TLS / SSL / x509-Zertifikate

TLS/SSL [7] ist ein Protokoll überhalb der TCP Kommunikationsschicht, welche einen transparenten, sicheren Kommunikationskanal zwischen zwei Maschinen unterstützt, welches im RFC 2246 definiert ist. TLS/SSL steht für *Transport Layer Security / Secure Sockets Layer.* Dabei steht der Begriff TLS für eine Weiterentwicklung von der dritten SSL-Version. Der Ansatz bei TLS/SSL liegt in dem Verbindungsaufbau („SSL Handshake Protocol“), welches die Identifikation und Authentifizierung von Klient und Server vornehmen kann. Dabei kommen asymmetrische Verschlüsselungsverfahren und Kryptographie mit öffentlichen Schlüsseln zur Verwendung (X.509 Zertifikate). Desweiteren wird ein zu benutzender kryptographischer Algorithmus und ein symmetrischer Schlüssel ausgehandelt, welche während der Übertragung angewendet werden.

Das Protokoll zum Verbindungsaufbau ist in vier Schritte eingeteilt:

1. Der Klient sendet ein *client hello* zu dem Server, welcher darauf mit einem *server hello* antwortet.

[13] User Space
[14] UNIX Space
[15] Not UNICORE Space

2. (optional) Der Server sendet dem Klient auf Anforderung sein X.509v3 Zertifikat und stellt dem Klient somit seine Identität aus.

3. (optional) Zunächst kann der Klient das erhaltene Zertifikat vom Server authentifizieren. Gelingt dies nicht, kann die Verbindung abgebrochen werden. Verlangt der Server vom Klient auch ein Zertifikat, so wird dieses an dieser Stelle übermittelt.

4. Der letze Schritt des Handshake-Prozesses handelt sowohl den kryptographischen Algorithmus, als auch den Sitzungsschlüssel anhand der öffentlichen Schlüsseln, der Zertifikate und einem im Handshake benutzten Schlüssel aus. Dies ist ein symmetrischer Schlüssel und ermöglicht somit schnelle Ver- und Entschlüsselung.

SSL-Verschlüsselung wird heute vor allem mit HTTPS eingesetzt. Die meisten Webserver unterstützen TLS/SSL, viele auch SSLv2 und SSLv3 mit einer Vielzahl von Verschlüsselungsmethoden. Fast alle Browser und Server setzen jedoch bevorzugt TLS mit RSA- oder AES-Verschlüsselung ein.

2.1.5 Anwendungsgebiete für Dateitransfer

Wie schon im Vorhinein erwähnt, sind Dateitransfers unabdingbare Einrichtungen der heutigen vernetzten Computerwelt. Daraus folgernd werden ohne diesen auch keine *Grid-Systeme* existieren können. Zum Verwirklichen der Transfers kann bei der Entwicklung eines *Grid-Systems* auf bereits vorhandene Lösungen zurückgegriffen werden, welche im folgenden Kapitel 2.2 erwähnt sind. Existieren dahingegen infrastrukturelle Gegebenheiten oder weitere Anforderungen, so ist es auch möglich, wie im Beispiel UNICORE, einen eigens für diesen Bedarf abgestimmten Datenübertragungsmechanismus zu implementieren.

2.2 Dateitransfer

Das Schlagwort Dateitransfer umfasst einen sehr weiträumigen Bereich. Es bezeichnet oberflächlich betrachtet die Übertragung von Daten in Form von Dateien von einem Netzwerkendgerät über ein beliebiges Netzwerk zu einem Anderen. Für den Prozess ist es zwingend notwendig, dass die beiden Geräte das gleiche Dateitransferprotokoll beherrschen. Die Wahl, welches Protokoll verwendet werden soll, muss der Benutzer treffen. Dabei gibt es eine Vielzahl von Entscheidungskriterien, welche sich auf Datendurchsatz, Sicherheit, Verschlüsselung und Integrierbarkeit beziehen.

Als Aussicht kann vorweggegriffen werden, dass es möglich sein soll, alle schon existierenden Protokolle in *HPFT* zu integrieren.

2.2.1 Existierende Lösungen

Selbstverständlich existieren schon eine Vielzahl von Lösungen und Verfahrensweisen von Dateitransfers. Um einen guten Überblick über die gängigen Verfahren zu bekommen, werden in diesem Kapitel die wichtigsten Protokolle angesprochen.

FTP

FTP[16] ist das am weitest verbreitete Transferprotokoll, welches nach der Klient/Server-Architektur arbeitet. Der Server wartet und horcht auf einer festgelegten Adresse auf eine eingehende Verbindung. Der Klient kann einen Verbindungsaufbau zu dem Server anfordern. Besteht eine *FTP*-Verbindung, kann der Klient mehrere Dateioperationen durchführen. Die Wichtigsten sind das Transferieren, Löschen und Umbenennen von Dateien.

Der erste Standart des *FTP*-Protokolls wurde 1971 in der RFC[17] 114 veröffentlicht. Damals existierte noch kein TCP/IP-Netzwerkprotokoll, somit kam es über einige Zwischenschritte zu dem heute noch aktuellen *FTP* Protokoll vom Oktober 1985 (RFC 959).

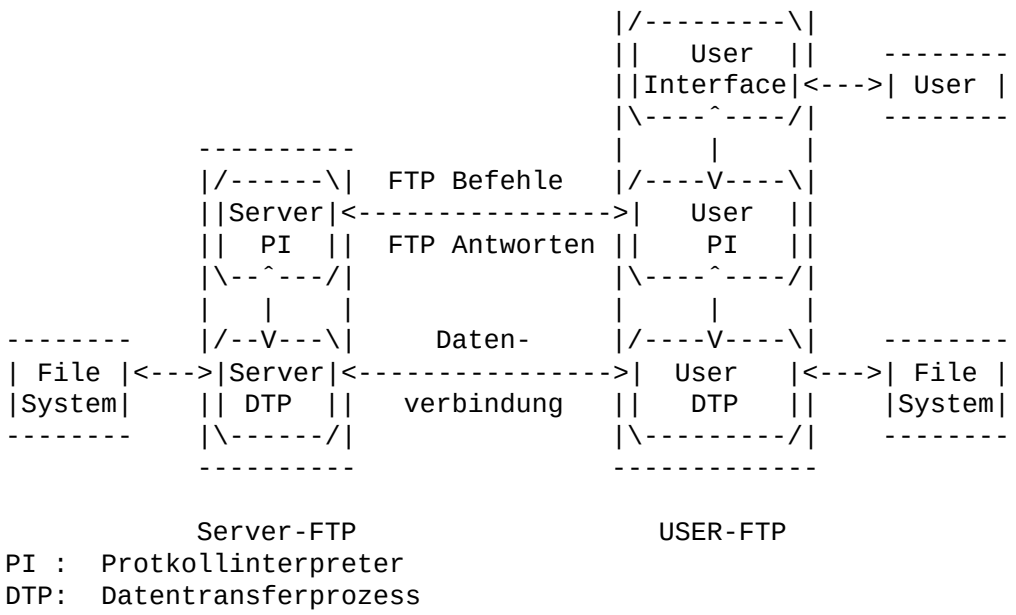

Abbildung 2.3: FTP Verbindungsdiagramm [6]

[16] File Transfer Protocol
[17] Request for Comment: `http://www.ietf.org/rfc.html`

FTP kommuniziert standardmässig auf zwei Ports (20 und 21) und läuft ausschließlich über das TCP-Netzwerkprotokoll. Der *FTP*-Server erwartet auf Port 21 eingehende Verdindungen vom *FTP*-Klient. Die Verbindung auf diesem Port bildet einen Kontrollkanal, über welchen lediglich die Befehle gesendet werden. Um den eigentlichen Inhalt der Datei zu übermitteln, wird eine weiterere parallele Verbindung hergestellt, welche den Datenkanal formt. Dabei gibt es zwei verschiedene Vorgehensweisen, wie der Datenkanal aufgebaut werden kann.

Beim „aktiven Modus“ horcht der Klient auf dem Port 20, welcher für den Datenkanal reserviert ist, und wartet auf eine eingehende Verbindung, die von der Serverseite aufgebaut wird. Ist dies aufgrund netzbasierter Restriktionen nicht möglich, so wird der Datenkanal im „passiven Modus“ aufgebaut. Dabei horcht der Server auf Port 20 auf eine weitere eingehende Verbindung vom Klient.

Das Trennen der beiden Kanäle wird als „Out of Band“-Steuerung bezeichnet, welches die Möglichkeit bietet, dass auch während der Datenübertragung die beiden Systeme miteinander kommunizieren können.

Beide Verfahren haben ihre Vor- und Nachteile. Der Klient kann für den Server nicht erreichbar sein, wenn der Klient zum Beispiel hinter einer Firewall oder in einem Netzwerk ist, welches NAT[18] benutzt. In dem Fall wird der „passive Modus“ verwendet. Der „aktive Modus“ hingegen beinhaltet eine Sicherheitsvorrichtung, welches zum Beispiel bei „spoofing“[19] ein Gegenmittel ist. Täuscht ein Hacker eine falsche Netzwerkidentität vor, so kann im „aktiven Modus“ keine Verbindung zu stande kommen, da der Server vergeblich versucht sich mit der falschen Netzwerkadresse zu verbinden.

Leider verfügt *FTP* über keine Sicherheitsmechanismen. Benutzername, Passwort und die Daten selbst werden unverschlüsselt über das Netz geschickt. Dies macht es Angreifern besonders leicht, an Benutzerzugänge und Daten zu gelangen.

Secure FTP

Secure FTP hat das gleiche Prinzip wie *FTP*. Es existiert ein Kontrollkanal auf Port 21. Diese Kommunikation wird jedoch über eine SSH-Verbindung[20] hergestellt und ist somit verschlüsselt. Diese Art der Verbindung stellt auch eine Authentifizierung der Kommunikationspartner sicher. Der Datenkanal läuft über einen beliebigen Port, bleibt aber weiterhin unverschlüsselt. Der Vorteil von *Secure FTP* liegt darin, dass die Benutzername - Passwort Kombinationen nicht mehr unverschlüsselt übers Netz geschickt werden.

[18] Network Adress Translation
[19] Vortäuschen einer anderen Netzwerkidentität von einem Angreifer
[20] Secure Shell

FTP über SSL (FTPS)

FTP über SSL ist ein gängiges Dateiübertragungsprotokoll auf Basis des FTP-Protokolls. Im Gegensatz zu *Secure FTP* wird sowohl der Datenkanal als auch der Kontrollkanal mittels TLS/SSL-Verfahren (siehe Kapitel 2.1.4) verschlüsselt und bietet somit einen besseren Schutz als *Secure FTP*, da Daten und Benutzername - Passwort Kombinationen nicht von Angreifern gelesen werden können.

SSH File Transfer Protocol (SFTP)

SFTP erweitert den Dateitransfer *scp*[21], welcher auf der Basis von *ssh* arbeitet. *scp* liefert als Sicherheitsmerkmale Vertraulichkeit, Integrität und Authentizität der über das Netzwerk gesendeten Daten. Als Erweiterung zu *scp* stehen *SFTP* auch Funktionen zur Dateimanipulation zur Verfügung, welche gleich den Befehlen bei FTP sind.

WebDAV

WebDAV[22] stellt einen Dienst zur Verfügung, bei welchem die Benutzer auf ihre Dateien in Form einer Online-Festplatte zugreifen können. Es stellt eine Erweiterung zu HTTP/1.1 dar, welche einige Einschränkungen aufhebt. HTTP/1.1 bietet lediglich ein Hoch- und Runterladen von einzelnen Dateien mittels den Befehlen HTTP-GET und HTTP-PUT an. Über *WebDAV* können hingegen auch ganze Verzeichnisse verschoben werden. Zudem ist auch eine Revisionskontrolle implementiert.

WebDAV bietet dem Benutzer einen sehr einfachen und intuitiven Zugriff auf Dateien an, welche auf einem Server liegen. Aufgrund des offenen Standards existieren auf allen gängigen Betriebssystemen nahtlose Implementierungen. In UNIX-Kernels und Windowssystemen existieren mittlerweile Module, um *WebDAV*-Ressourcen als logische Festplatte einzubinden.

SOAP

SOAP[23] bezeichnet ein Protokoll, mit welchem Daten über ein Netzwerk ausgetauscht werden können. Zusätzlich besteht die Möglichkeit, Prozeduren auf dem entfernten System auszuführen. *SOAP* basiert auf bereits bestehenden Protokollen und Standarts. XML wird zur Darstellung der Daten und Informationen genutzt. In der gängigen Form liegen TCP und HTTP zugrunde, um die Daten zu transferieren. Eine *SOAP*-Nachricht gliedert sich in zwei oder drei Teile. Die ersten beiden repräsentieren den Informationskopf und den Datenkontainer. Sollen binäre Daten übertragen werden, so kann als dritter Teil ein

[21] secure copy
[22] Web-based Distributed Authoring and Versioning
[23] Simple Object Access Protocol

Anhang zugefügt werden.

Ein Nachteil von *SOAP* liegt darin, dass bei kurzen Informationen die Nachrichten überaus stark aufgebläht werden, da der Informationskopf viele Metainformationen, wie Routing, Verschlüsselung oder Serienidentitätsnummern, enthalten kann. Das daraus resultierende Problem liegt nicht allein bei der beanspruchten Netzwerkbandbreite, sondern auch beim Erstellen und Analysieren solcher XML-*SOAP*-Nachrichten, welche die Kommunikation verlangsamen.

GridFTP

GridFTP ist ein von dem Global Grid Forum definiertes Protokoll, welches auf dem normalen FTP aufbaut. Die umgesetzten Ziele von *GridFTP* fokussieren sich auf die Sicherheit, Robustheit, Schnelligkeit und Effizienz des Dateitransfers. Dabei werden die Daten mittels sogenannter *Third Party Transfers* übertragen. Dies bedeutet, dass die Dateien zwischen zwei *GridFTP*-Servern transferiert werden. Der Klient baut zu zwei Servern jeweils einen Kontrollkanal auf und legt fest, welcher der beiden als Server und wer als Klient agiert. Die Sicherheit beruht in der Verwendung von x509 Zertifikaten, welche die Authentizitäten vom Klienten und den Servern sicherstellen.

Um eine hohe Datenübertragungsrate auch über ein WAN zu bekommen, geht *GridFTP* mehrere Methoden an. Zum einen wird anhand des *Bandwidth Delay Products* die TCP-Buffergrößen und somit die TCP-Fenstergröse für eine bestehende Verbindung angepasst. Bei einer zu kleinen Fenstergröße kann der der Datenfluß ins stocken geraten, da der Sender auf viele Bestätigungen von gesendeten Paketen warten muss. Bei zu großen Fenstern kann im Falle von einem Paketverlust ein erheblicher Mehraufwand im erneuten Senden des Paketes liegen. Desweiteren bietet *GridFTP* die Möglichkeit eines parallelen Datentransfers. Die zu übertragene Datei wird zerteilt und über mehrer TCP-Datenströme gesendet. Das bringt gleich zwei Vorteile mit sich. Die *Fairness* von TCP wird umgangen. Die TCP-*Fairness* gibt jeder Verbindung den gleichen Anteil der vorhandenen Bandbreite. Da *GridFTP* mehrere TCP-Verbindungen parallel öffnet, nimmt es für sich insgesamt mehr Bandbreite in Anspruch, als es von TCP eigentlich zusteht. Ein weiterer positiver Faktor ist, dass der gesamte Datentransfer stabilisiert wird, da bei einzelnen Paketverlusten lediglich eine Verbindung betroffen ist.

Wie sich Leistung und Durchsatz von *GridFTP* gegenüber UPL verhält wird im Abschnitt „GridFTP gegen UPL“ (siehe Kapitel 3.1.1) behandelt.

UPL

Obwohl *UPL*[24] ansich kein direkter Dateitransfer ist, wird es hier erwähnt, da es in UNICORE fest integriert ist. *UPL* definiert eine Struktur zum Austausch von Daten zwischen UNICORE-Server und -Klient über eine SSL-Verbindung. Diese Daten können in zwei Teile aufgeteilt sein. Der Erste ist zwingend notwendig und enthält das Abstrakte Job Objekt (AJO), welches eine Anforderung vom Klient oder eine Antwort vom Server ist. Der zweite Teil, welcher einfach an das AJO angehängt wird, ist optional und beinhaltet einen Datenstrom von Bytes. Dieser Datenstrom repräsentiert einen Dateitransfer. Somit ist bei *UPL* der Dateitransfer fest in der Datenarchitektur von *UPL* verkoppelt.

Dieses Kommunikationsdesign von UNICORE hat Vor- und Nachteile. Der große Vorteil besteht darin, dass nur eine SSL-Verbindung existiert. Dies macht die Integration des Gateways hinter einer Firewall sehr einfach, da nur ein Port geöffnet werden muss. Desweiteren wird über SSL eine sichere Verbindung aufgebaut, da nur authentifizierte Benutzer eine Verbindung öffnen dürfen.

Der Nachteil von *UPL* bezieht sich allein auf den Dateitransfer. Dieser schafft nur sehr geringen Datendurchsatz im Vergleich mit anderen Dateitransferprotokollen. Dabei werden die Daten in UPL Pakete zerlegt und komprimiert, bevor sie versendet werden. Im Kapitel 3.1.1 wird genauer auf die Effizienz von *UPL* eingegangen.

[24]UNICORE Protocol Layer

Kapitel 3

High Performance File Transfer (HPFT)

Das Programmsystem *„HPFT“* soll dem Benutzer einen Dienst bieten, der das Hoch- und Runterladens von Dateien über ein Netzwerk mittels unterschiedlicher Protokolle realisiert. Dazu werden im folgenden Kapitel Anforderungen gestellt, welche den Weg zu einer gut durchdachten Softwarearchitektur geben. Dies beinhaltet auch die Verwendung von schon bekannten Softwaremodellen, welche dem Entwickler des Systems während der Implementierung nur wenige Barrieren aufweisen. Desweiteren handelt es sich um die Wiederverwendung von schon existierenden Programmen, welche Dateitransfers als Grundlage anbieten (siehe Kapitel 2.2). Wie genau die einzelnen Programmteile von *HPFT* untereinander in Verbindung stehen, wird im Abschnitt Design (siehe Kapital 3.2) erläutert.

3.1 Ziele/Anforderungen von HPFT

Von *HPFT* wird gefordert, dass es eine flexible und effiziente Lösung eines Dateitransfers widerspiegelt, welche, dem Benutzer gegenüber transparent, verschiedene Protokolle unterstützt. Demnach soll der Benutzer lediglich die Wahl über das zu verwendende Protokoll und der Aktion des Hoch- bzw. Runterladens haben. Kurz ausgedrückt formen diese Anforderungen die Benutzerschnittstelle des *HPFT*-Klienten, welche für jedes zur Verfügung stehende Dateitransferprotokoll diese zwei Basisfunktionen bereitstellt.

Desweiteren ist es sehr wichtig für *HPFT*, dass es erweiterungsfähig bleibt, und dass Änderungen leicht vorzunehmen sind. Es müssen dazu unterschiedliche Dateitransferprotokolle innerhalb von *HPFT* zur Verfügung stehen. Dazu muss der Benutzer die Möglichkeit haben, dem Programm weitere Protokolle hinzufügen zu können, wodurch die verschiedenen Bedürfnisse der Benutzer auch in Zukunft abgedeckt werden. Darüber hinaus können auch neuartige und nichtgängige Protokolle im Nachhinein verwendet wer-

den.

Der wichtigste Punkt von *HPFT* ist seine Flexibiltät. Es wird verlangt, dass es sich in ein bestehendes Softwareprodukt einbinden lässt. Dabei kann es das Ziel sein, *HPFT* als Grundfunktion zur Dateiübertragung in einem Programm zu verwenden oder als alternativen Dateitransfer in einem schon existierenden Programm einzubinden. Dafür müssen der *HPFT*-Server und der *HPFT*-Klient einfache Schnittstellen zum Bedienen aufweisen. Im zweiten Teil dieser Arbeit (siehe Kapitel 4) wird *HPFT* als alternativer Dateitransfer im UNICORE Grid-System eingebettet.

3.1.1 GridFTP contra Unicore-UPL

Um die Performanz zwischen dem Dateitransfer via *UPL* mit der Performanz vom Dateitransfer via *GridFTP* zu vergleichen, wurde die Datenübertragungsrate der beiden Verfahren gemessen (siehe Kapitel 2.2.1 (GridFTP, UPL)). Diese Gegenüberstellung zeigt einen kontrastreichen Unterschied in der Leistung der beiden Systeme. *GridFTP* ist der Stand der Technik für Dateitransfers innerhalb Grid-Systemen und zeichnet sich durch seine hervorrangend hohe Datenübetragungsrate aus. *UPL* kann aufgrund seiner Verfahrensweise, wie es Dateien über ein Netzwerk transferiert, keine hohe Performanz erreichen. Die Daten für die Auswertung wurden der Quelle [5] (*GridFTP vs. UPL von Simone Lanzarini*) entnommen.

Zum Durchführen der Messung wurden zwei verschiedengroße Dateien erstellt, 5 MB und 1 GB. Dabei wurde darauf Rücksicht genommen, dass *UPL* einen komprimierten ZIP-Datenstrom verwendet, um zu vermeiden, dass große Datenmengen über das LAN-Netzwerk gesendet werden müssen. Daher wurden für *UPL* jeweils 2 Dateien erstellt. Die eine enthält binäre Daten, welche eine Kompressionsrate von nur ca. 1% zulässt, die andere hingegen enthält eine periodische Folge von Textzeichen und ist bis auf ca. 99% ihrer Größe komprimierbar.
Um direkt einen Einblick zu haben, wie sich eine verschiedene Anzahl von parallelen Datenströmen bei *GridFTP* auswirkt, wurde auch dieses berücksichtigt und mehrere Durchläufe mit bis zu 32 parallelen Datenströmen durchgeführt. Alle Tests wurden auf einem TCP/IP-LAN-Netzwerk mit einer maximalen Datenübetragungsrate von 100 MBit/sek durchgeführt.

Die Auswertungsdiagramme aus den Abbildungen 3.1 und 3.2 zeigen ganz eindeutig den signifikanten Unterschied zwischen *UPL* und *GridFTP*. *GridFTP* ist in Bezug auf die Datentransferrate im LAN-Umfeld *UPL* weit überlegen. Aufgrund der Eigenschaften von *GridFTP* ist davon auszugehen, dass der Unterschied in einem WAN-Umfeld ähnlich signifikant sein wird.

Die Datenübertragungsrate bei kleinen Dateien (5MB)mit *GridFTP* liegt bei ca 4.5 MB/sek. Es bringt hierbei einen zu beachtenden Nachteil, wenn überaus viele parallele Datenver-

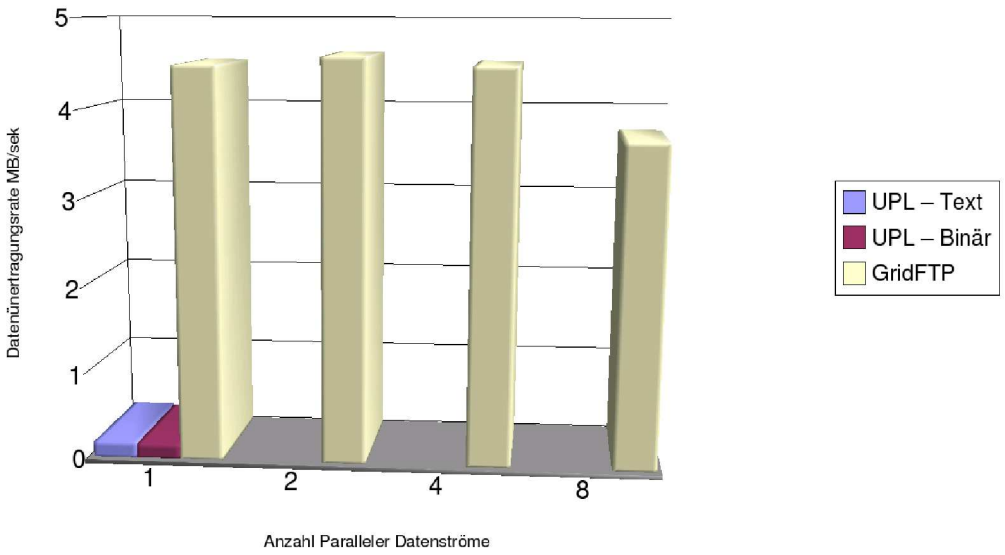

Abbildung 3.1: Vergleich *UPL - GridFTP* (Dateigröße 5 MB)

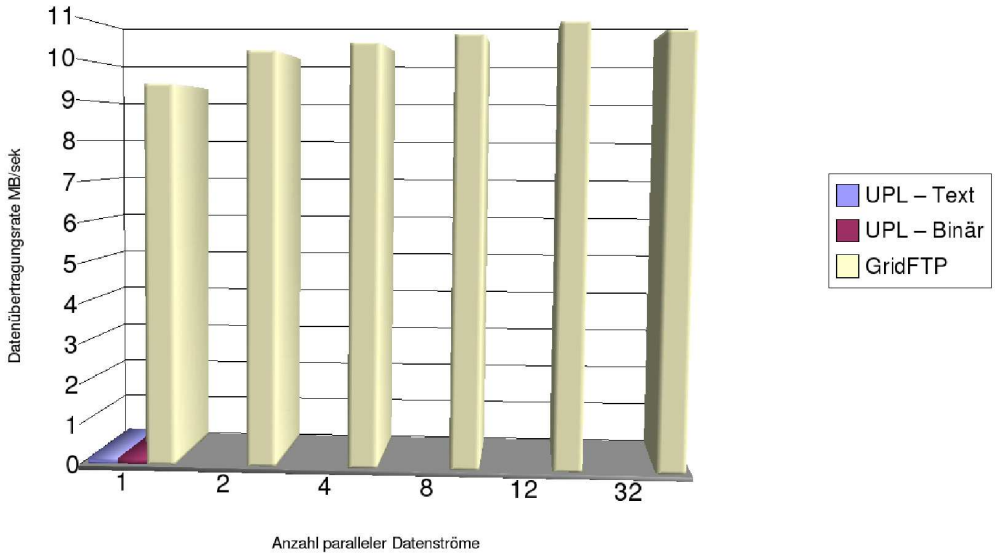

Abbildung 3.2: Vergleich *UPL - GridFTP* (Dateigröße 1 GB)

bindungen aufgebaut werden, da hierdurch auch die CPU-Last zum Zerlegen und Zusammenfügen der Dateien steigt. Ganz im Gegenteil bei großen Dateien (1GB). Dort bewirken mehrere Datenströme eine Steigerung des Datendurchsatzes bis zu 1 MB/sek, welcher in dem Fall von durchschnittlich 9.5 MB/sek auf 10.5 MB/sek gehoben werden kann. Dazu muss angemerkt werden, dass bei einem LAN-Netzwerk mit einer Bandbreite von 100 MBit/sek maximal 12.5 MB/sek übertragen werden können (inklusive Overhead-Daten) und somit nahezu die gesamte Bandbreite des Netzes von diesem Dateitransfer ausgenutzt wird.

Bei der Übertragung von großen Dateien kommt eine doppelt so hohe Datenübertragungsrate als bei kleinen Dateien zustande (4.5 und 10.5 MB/sek). Der Grund dafür liegt im Aufwand des Zerlegens und Zusammensetzens der Datei aus den einzelnen Datenströmen. Bei kleinen Dateien nimmt dieser Aufwand einen großen Anteil der Gesamtzeit der Übertragung ein, welcher sich bei großen Dateien nur noch marginal auswirkt.

UPL hingegen ist bei der Übertragung von Dateien mit einer Größe von 1 GB bis zu 95 mal langsamer als *GridFTP* und durchstößt während dem ganzen Versuchsablauf nicht die 200 KB/sek Marke. Das Bemerkenswerte ist, dass selbst bei den Text-Dateien, welche von *UPL* auf lediglich ca. einen Prozent der ursprünglichen Größe komprimiert werden, kein bedeutender Vorteil in Bezug auf die Übertragungsrate zu erkennen ist (vergl. Abbildung 3.1 UPL-Text mit UPL-Binär).

3.1.2 Warum ist UPL-Dateitransfer uneffizient?

Für UNICORE ist *UPL* die einzige Grundlage, Daten von einem Kommunikationspartner zum anderen zu senden. Die vom Benutzer erzeugten Jobs werden als ein ganzes Java-Objekt gesehen. Findet zu dem Job ein Dateitransfer statt, so wird der Dateiinhalt an den serialisierten Job angehängt (siehe Abbildung 3.3).
Zum Versenden des Datenstroms (Jobs und Dateiinhalte), wird dieser in einzelne Blöcke gleicher Größe zerteilt. Bevor diese an das Netzwerk übergeben werden, wird noch ein Komprimierungsverfahren (ZIP) angewendet, um die Länge des Datenstroms zu verringern (siehe Abbildung 3.4).
Das Verfahren, wie Dateien mit UPL transferiert werden, weist einige Nachteile auf. Bei UPL findet, wie erwähnt, eine eigene Paketierung der Daten statt. Dies erfordert einen Mehraufwand zum Zerteilen und Zusammen fügen dieser *UPL*-Pakete. Zudem kommt, dass ein gesendeter Job und somit auch der mitgesendete Dateitransfer über mehrere Schichten laufen muss. Vom Klient übers Gateway zum NJS und TSI. Auf allen Systemen werden jedes Mal die *UPL*-Pakete zusammengesetzt und wieder getrennt.

Desweiteren wird der Datenstrom beim Versenden mit dem ZIP-Verfahren komprimiert. Die Empfängerseite muss die komprimierten Daten wieder entpacken, bevor diese verwendet werden können. Dies führt auch zu einer erhöhten Nutzung der Rechenkapazität,

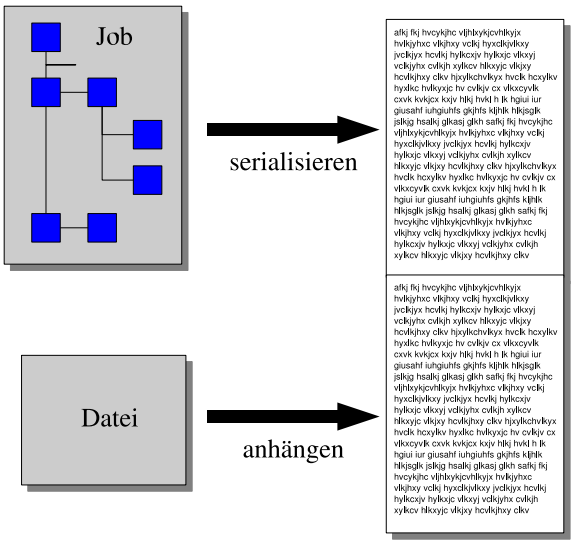

Abbildung 3.3: UPL Verfahren 1

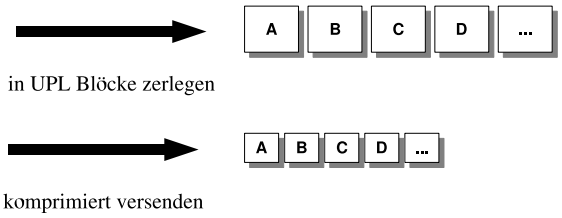

Abbildung 3.4: UPL Verfahren 2

welches den Transfer verlangsamt, zumal die Daten erst dann den Empfänger erreichen, wenn sowohl die Komprimierung als auch die Dekomprimierung abgeschlossen sind.

UPL ansich hat viele Vorteile, jedoch diese Verfahrensweise, mit welcher UNICORE Dateien über ein Netzwerk transportiert, ist nicht effektiv. Daher veranlasst dies, über einen alternativen Dateitransfer für UNICORE nachzudenken. Dazu werden zunächst Fragen zum Design dieses Dateitransfers gestellt.

3.2 Design von HPFT

Das Softwaredesign von *HPFT* muss sich nach den Anforderungen aus Kapitel 3.1 richten. Dazu wurde zunächst ein vereinfachtes Konzept (siehe Abbildung 3.5) entworfen, um eine Softwarestruktur zu entwickeln, welche am Besten diesen Bedingungen gerecht wird.

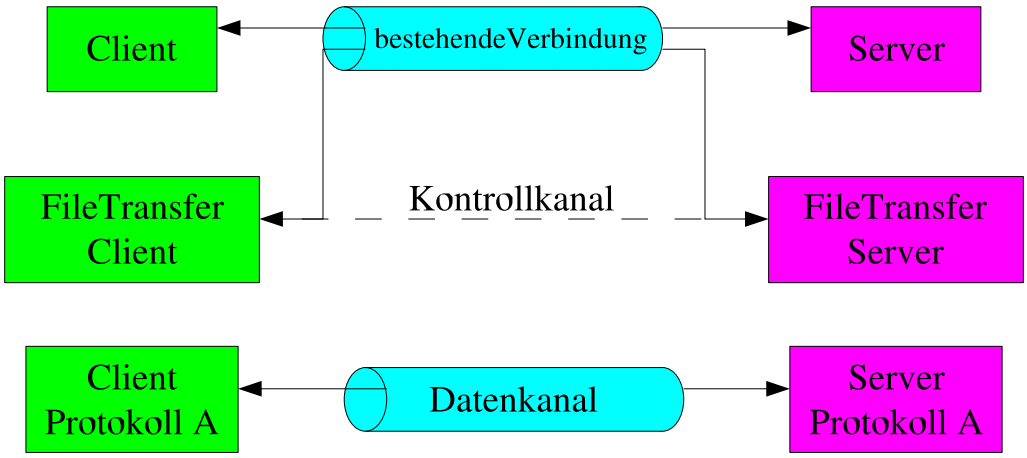

Abbildung 3.5: Vereinfachtes Konzept

Laut den Vorgaben wurde daraufhin gearbeitet, *HPFT* in einem vorhandenen System (z.B. UNICORE) zu verwenden. Dies bedeutet, dass zu dieser Verwendung schon eine Datenverbindung zwischen zwei Kommunikationspartnern existiert, worüber Daten ausgetauscht werden können. Es bot sich an, *HPFT* so zu strukturieren, dass mindestens zwei Kanäle zum Verarbeiten des Dateitransfers existieren. Als Vorbild wurde dazu FTP (siehe Kapitel 2.2.1) genommen und auf ähnliche Weise ein Kontrollkanal und ein Datenkanal definiert. Diese haben einerseits die Aufgabe, den Dateitransfer zu koordinieren, und andererseits die Daten zu übertragen. Dabei besteht die Möglichkeit, eine schon vorhandene Verbindung auch als Kontrollkanal zu verwenden. Dies hat den Vorteil, dass bei Bedarf und Möglichkeit nur noch die Datenverbindung geöffnet werden muss. Im Vergleich zum FTP-Verfahren wird die Datenverbindung im „passiven Modus“ aufgebaut. Es war nicht zu vereinbaren, den Klient auf eine eingehende Verbindung warten zu lassen, da dies in der Regel nicht möglich ist (z.B. durch Blocken innerhalb einer Firewall).

Abbildung 3.6 zeigt ein Klassendiagramm über die Struktur und den Benutzerschnittstellen von *HPFT* auf der Serverseite. Die Klientseite ist auf die gleiche Weise strukturiert und muss nicht explizit angegeben werden. Das Serverprogramm erstellt eine Instanz der Klasse *FileTransferServer* (FTS). Diese enthält die direkte Benutzerschnittstelle zum Anwender, mit deren Methoden (`startServer` und `stopServer`) die einzelnen

Servertypen gestartet und gestoppt werden können. Dazu kann zuvor mit einer Methode (`getPluginProtocols`) abgefragt werden, welche Plugins überhaupt zur Verfügung stehen.

Die wichtigste Methode (`processClientRequest`) wird als Anbindung an den Kontrollkanal benutzt. Die vom Server empfangene Kontrollkanalnachricht wird einfach an diese Methode übergeben. Diese bearbeitet die eingehende Anfrage und gibt eine entsprechende Nachricht im Kontrollkanalformat zurück, die auf direktem Wege über den Kanal zum Klient gesendet werden kann.

Da das Designziel von *HPFT* in der Flexibilität bestand, wurde ein *Factory*-Konzept verwendet, welches eine gewünschte Instanz eines Servers bzw. Klienten erstellt und an den Benutzer übergibt. Dies bedeutet, dass *HPFT* verschiedene Objekte für die unterstützten Protokolle bereithält. Damit eine einheitliche Benutzung dieser Protokollobjekte gewährleistet bleibt, existiert auf Server- und Klientseite jeweils eine Benutzerschnittstelle mit fest definierten Methoden, welche die Standardfunktionen für den Filetransfer definieren:

```
Klient:  public boolean getFile(String remote, File local);
         public boolean putFile(File local, String remote);
Server:  public boolean startService();
         public boolean stopService();
         public String grantUserAccess(String userinfo);
         public boolean denyUserAccess(String userinfo);
```

Die Benutzung solcher einheitlichen Schnittstellen bringt eine erhebliche Erleichterung zum Bewahren einer flexiblen Struktur. Dadurch kann mit einem Plugin-Konzept gearbeitet werden. Dies bedeutet insbesondere für das Programmsystem, dass es fortwährend erweitert werden kann. Es müssen lediglich weitere Plugin-Objekte erstellt werden, welche die gleiche Schnittstelle implementieren. Die Verwendung der einzelnen Plugins verbleibt dann im Aufrufen der angegeben Methoden der Schnittstelle, welche für jedes Protokoll-Plugin eigens implementiert werden müssen.

Zudem unterstützt *HPFT* eine Art der Benutzerverwaltung. Das Konzept läuft so ab, dass der Klient über den Kontrollkanal eine Anfrage an den Server sendet, mit welcher es um einen Zugriff auf einen laufenden Dateitransfer-Server bittet. Das Server-Plugin hat die Option, eine Autorisierung des Klient vorzunehmen und gibt eine entsprechende Antwort an den Klient zurück. Ist der Klient zugriffsberechtigt, wird das Server-Plugin (z.B. ein HTTP-Plugin) auf den Zugriff dieses Klienten eingerichtet. Dieser Zustand hält solange an, bis der Klient meldet, dass er mit dem Übertragen fertig ist. Danach wird der Zugriff von diesem Klient auf den Server wieder gesperrt.

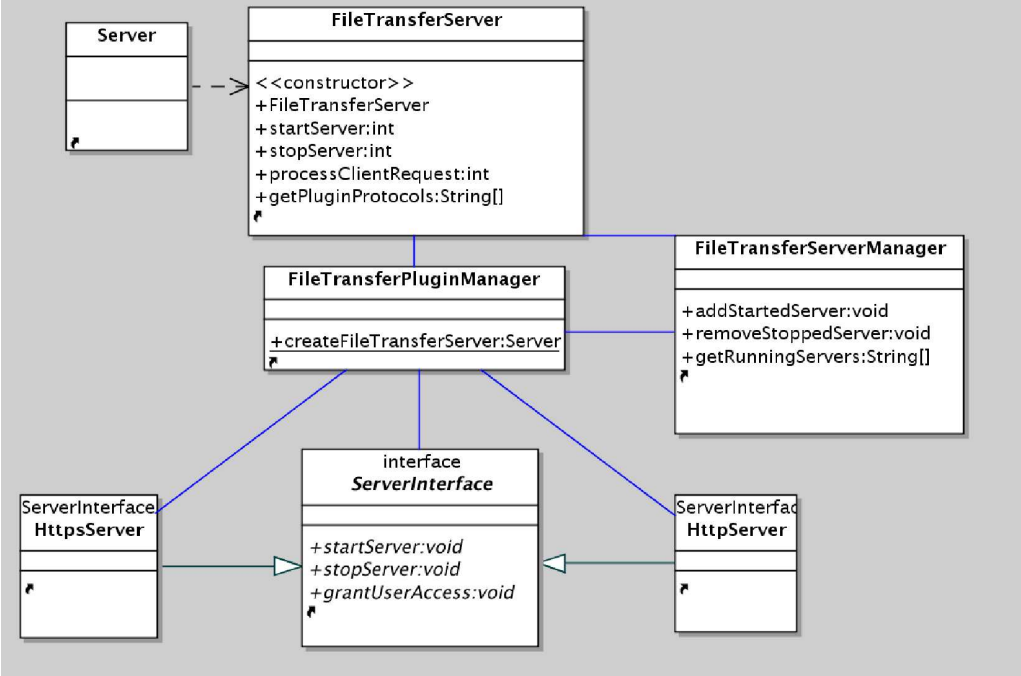

Abbildung 3.6: UML - Diagramm zur Festlegung der Schnittstellen (Server)

3.2.1 Design Kontrollkanal

Der Kontrollkanal hat, wie der Name bereits aussagt, die Aufgabe, den Dateitransfer zu koordinieren. Dazu werden für den Klient, welcher einen Dateitransfer anstoßen kann, verschiedene Nachrichtentypen zur Verfügung gestellt, mit welchen er Abfragen und Anforderungen an den Server stellen kann. Die Anforderung vom Klient werden hier als *Requests (REQ)* geführt und die Antworten, welche auf einen *Request* vom Server zurückgegeben werden, als *Answers (ANS).* Vergleiche dazu die Tabellen 3.1 und 3.2.

`REQ_CAPABILITIES` Abfrage der auf dem Server laufenden Protokolle.

`REQ_NEED_SERVER_PLS` Anforderung, einen Zugriff auf den im Protokoll spezifizierten Server zu erhalten.

`REQ_DONE_FILETRANSFER_THX` Mitteilung an den Server, dass alle Dateitransfers abgeschlossen sind.

Liste 3.1: Nachrichtentypen für den Klient

ANS_CAPABILITIES_LIST Enhält eine Liste mit den unterstützten Protokollen.

ANS_SERVER_IS_NOW_RUNNING_FOR_YOU Enthält den Namen des Protokolls und eine Info-Zeile mit den benötigten Informationen zum Verbinden an den Server.

ANS_OK_STOPPING_YOUR_SERVER_NOW Information, dass die Zugriffsrechte entfernt worden sind.

ANS_ERROR_CANT_START_REQUESTED_SERVER_SRY Fehlermeldung, dass die Zugriffsrechte nicht eingerichtet werden konnten.

ANS_ERROR_PROTOCOL_NOT_AVAILABLE Fehlermeldungen, dass das geforderte Protokoll auf dem Server nicht existiert.

Liste 3.2: Nachrichtentypen für den Server

Als Ansatz für die Strukturierung der Nachrichten wurden Zeichenketten (Character-Strings) gewählt. Diese sind leicht zu erstellen, zu analysieren und können ohne besonderen Kodierungsaufwand über Netzwerkverbindungen versendet werden. Dabei werden pro Nachricht mehrere Informationen zusammengefasst und durch ein Trennzeichen (senkrechter Strich |) voneinander getrennt. Für diesen Aufgabenbereich wurde ein Paket an Funktionen zusammengestellt.

Das Softwarepaket ControlChannel enthält drei wichtige Klassen. `ControlChannel`, `ClientRequestFactory` und `ServerAnswerFactory`. Die *Factories* stellen Methoden zur Erstellung der einzelnen Nachrichten bereit. Dazu werden je nach Nachrichtentyp Informationen verlangt, welche in die Nachricht übernommen werden. Die Klasse `ControlChannel` hingegen bietet Methoden an, die bei der Analyse der Nachrichten eine große Hilfe geben. Damit kann eine Nachricht auf syntaktische Korrekheit geprüft oder ihre Information extrahiert werden.

Es existiert eine Struktur, die den Aufbau der einzelnen Informationen in einer Nachricht festlegt. Der erste Teile einer Nachricht sagt immer aus, dass es sich um eine Kontrollkanal-Nachricht handelt (`HPFT Control Protocol v1.0`). Der zweite Teil legt den Typ der Nachricht fest. Vergleiche dazu die Tabellen 3.1 und 3.2. Danach kommen je nach Mitteilungstyp verschiedene Informationen, welche durch einen Prefix gekennzeichnet sind. `PROT=` legt das gewählte Prokoll fest und `INFO=` gibt die Zusatzinformation an, welche zum Beispiel die Zugriffsdaten zu einem Server enthalten können. Jede Nachricht wird mit einen Terminator beendet. Dieser ist ein eigenständiger Nachrichtenteil, welcher lediglich einen Punkt (.) enthählt.

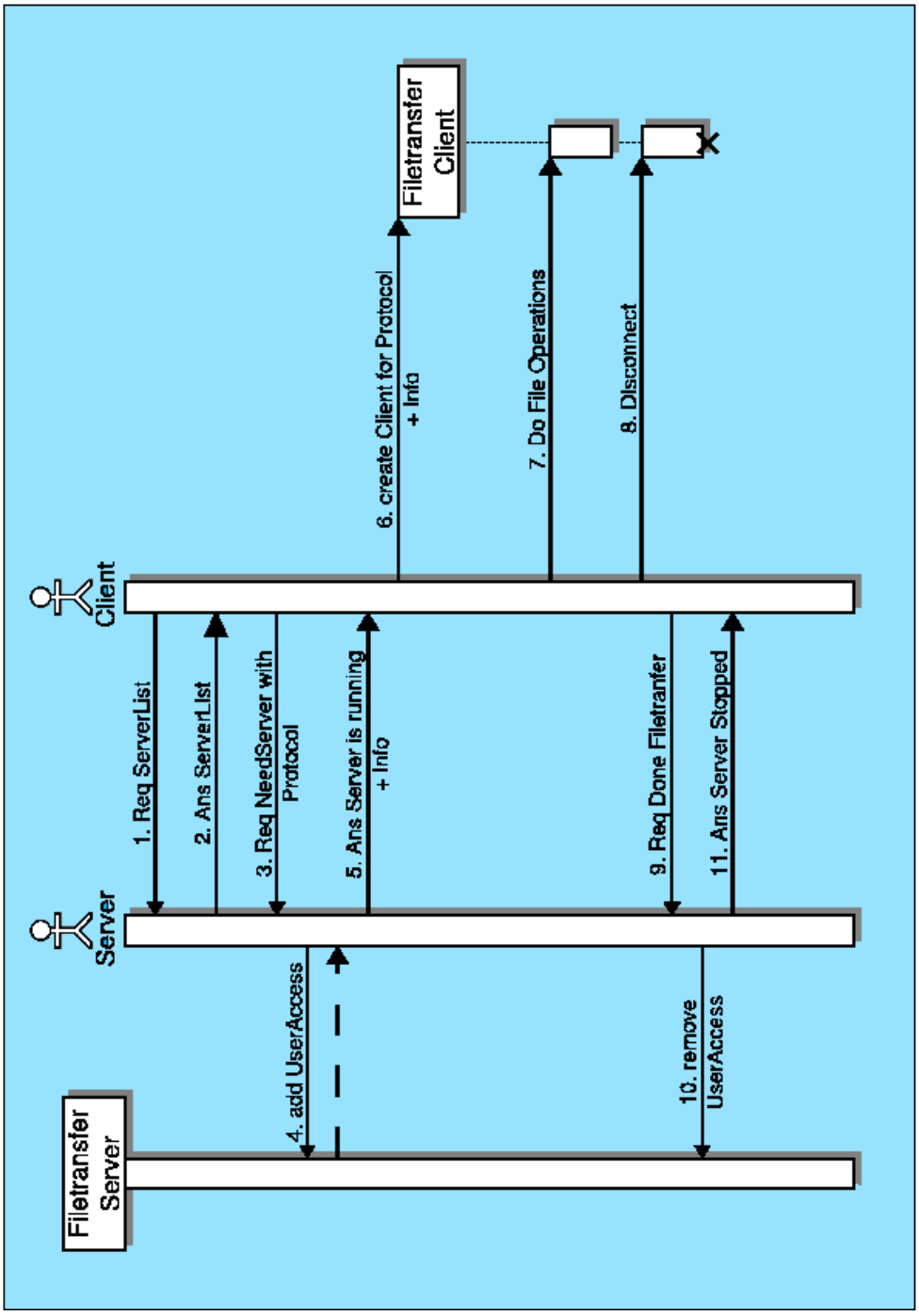

Abbildung 3.7: Ablauf Kontrollkanal

Das Diagramm in Abbildung 3.7 zeigt den Ablauf einer *HPFT*-Sitzung und listet insbesondere eine durchnummerierte Kommunikation über den Kontrollkanal auf. Desweiteren sind auch die Schritte zur Konfiguration von Server und Klient enthalten. Die folgende Kommunikation bezieht sich im Allgemeinen auf den Austausch von Nachrichten über den Kontrollkanal. Es werden auch die Nachrichten in ihrer rohen Form dargelegt.

1 Der Klient fordert beim Server eine Liste der verfügbaren Protokolle an (1).
```
HPFT Control Protocol v1.0|
6 REQ_CAPABILITIES|
.
```

2 Der Server gibt als Antwort die geforderte Liste der zur Verfügung stehenden Protokolle. Dies umfasst in diesem Beispiel lediglich http (2).
```
HPFT Control Protocol v1.0|
0 ANS_CAPABILITIES_LIST|
PROT=http|
.
```

3 Der Klient kann nun aus der Liste `http` auswählen und einen Zugriff beim Server anfordern (3).
```
HPFT Control Protocol v1.0|
7 REQ_NEED_SERVER_PLS|
PROT=http|
.
```

4, 5 Auf der Serverseite wird ein Zugriff vom anfragenden Klient auf den http-Server eingerichtet (4). Dem Klient wird danach eine Erfolgsmeldung mit den zugangsrelevanten Daten übermittelt (5).
```
HPFT Control Protocol v1.0|
1 ANS_SERVER_IS_NOW_RUNNING_FOR_YOU|
PROT=http|
INFO=host.domain.de:port:testuser:/home/server/pub/testuser:info|
.
```

6, 7, 8 Nach dem Erhalt der Zugangsdaten, wird auf der Klient-Seite mit diesen Informationen ein Dateitransfer-Klient erstellt (6). Über diesen kann der Klient seine Transfer-Operationen ausführen (7). Erfordert ein Protokoll ein Lösen der Verbindung, kann an dieser Stelle ein *disconnect* aufgerufen werden (8).

9 Der Klient meldet sich beim Server ab, indem er eine Meldung sendet (9).

```
HPFT Control Protocol v1.0|
8 REQ_DONE_FILETRANSFER_THX|
PROT=http|
INFO=testuser|
.
```

10, 11 Daraufhin werden die Zugangsrechte des Klienten auf dem http-Server wieder entfernt (10) und der Klient darüber informiert (11).

```
HPFT Control Protocol v1.0|
2 ANS_OK_STOPPING_YOUR_SERVER_NOW|
PROT=http|
.
```

3.2.2 Design Datenkanal

Der Datenkanal kann nicht genau spezifiziert werden, da die Datenübertragung bei *HPFT* durch unterschiedliche Verfahrensweisen realisiert wird. Es können unter anderem die im Kapitel 2.2.1 aufgeführten Dateitransferprotokolle eingebunden werden. Aber auch weiterführende und komplexere Lösungen sind möglich.

Als Standardplugin für *HPFT* wurde ein HTTP-Dateitransfer implementiert. Der Server wird mit dem `jetty HTTP1.1`-Server[1] realisiert. Als Klient dient der `httpclient` von `Jakarta Commons`[2]. Zum Trsanferieren der Dateien werden in dem Fall die HTTP-Methoden GET und PUT verwendet.

3.2.3 Erweiterbarkeit durch Plugins

Eine der elementaren Anforderung von *HPFT* war es, mit einem Programmsystem die verschiedenen Bedürfnisse von Benutzern abzudecken, welches die Verwendung von unterschiedlichen Protokollen mit sich bringt. Dies wurde mit der Einbindung eines Plugin-Systems realisiert und lässt somit die Möglichkeit offen, *HPFT* ständig zu erweitern.

Die Plugin-Schnittstelle wurde jeweils auf der Server- und Klientseite in dem Maße definiert, dass sie einfach und intuitiv ist, so dass in Zukunft ohne großen Aufwand weitere Plugins hinzugefügt werden können. Dabei bietet Java mit dem *reflect*-Paket eine umfassende Unterstützung. Es erlaubt einem Programm, zur Laufzeit auf dessen Namen und Eigenschaften zuzugreifen. Insbesondere kann so eine Methode erstellt und aufgerufen werden, von der nur der Name als String bekannt ist.

[1] http://jetty.mortbay.org
[2] http://jakarta.apache.org/commons/httpclient

Zum Aufruf von *HPFT* wird eine Konfigurationsdatei (.serverrc oder .clientrc) benötigt, welche einen Schlüssel mit dem Namen *PluginDirectory* enthält. In diesem angegebenen Verzeichnis muss eine jar-Datei[3] (ClientPlugin.jar oder ServerPlugin.jar) mit den Protokoll-Plugins liegen. Die Dateien enthalten die Java-Plugins, welche in den Paketen `server.protocols` bzw. `client.protocols` aufgeführt werden. Diese Plugins sind Java-Klassen mit dem Namen *PluginXXXXServer.class* bzw. *PluginXXXXClient.class.* XXXX steht für das jeweilige Protocol (siehe Auflistung in Tabelle 3.3).

```
$> cat .clientrc
#FileTransferClient
#Fri Jan 27 09:04:48 CET 2006
PluginDirectory=/home/user/HPFT/lib/
Version=0.1
Plugins=ClientPlugin.jar\:dummy, ClientPlugin.jar\:http,

$> jar tf ServerPlugin.jar
META-INF/
META-INF/MANIFEST.MF
server/
server/protocols/
server/protocols/PluginDummyServer.class
server/protocols/PluginHttpServer.class
```

Tabelle 3.3: Inhalte der Dateien .clientrc und ServerPlugin.class

Von den dort gefundenen Protokoll-Plugins können somit vom Server bzw. Klienten Instanzen erstellt und an den Benutzer übergeben werden, welcher diese dann mittels der Benutzerschnittstelle (siehe Kapitel 3.2) anwenden kann.

[3] Java Archive

Kapitel 4

Einbettung in Unicore 4.x

Die nun folgenden Kapitel werden die Einbettung von *HPFT* in UNICORE systematisch veranschaulichen. Dabei wird auf die Eigenschaften von UNICORE (insbesondere UPL) und den bewusst getroffenen Designentscheidungen von *HPFT* eingegangen und gezeigt, wie diese im Zusammenspiel funktionieren.

4.1 Einleitung

In Vorbereitung auf die Integration von *HPFT* wurden zunächst mehrere Testsysteme der UNICORE-Serverseite installiert. Dazu wurde das *Quickstart Bundle*[1] auf insgesamt drei Servern installiert. Die drei Servermaschinen befinden sich jeweils in einem anderen Netz mit unterschiedlichen Architekturen, wodurch im Kapitel 6.1 ein differenzierter Vergleiche gezogen werden kann (siehe Tabelle 6.1).

Serverhost	Beschreibung
localhost	UNICORE-Klient, Gateway, NJS und TSI laufen auf einer Maschine. (IBM Thinkpad, 2Ghz, 786 MB, SuSE Linux 9.3)
zfr094.zfr	Serverinstallation in einem 10Base-2 Netzwerk (10MBit/sek)
zam904.zam	Serverinstallation in einem 100Base-T Netzwerk (100MBit/sek)

Tabelle 4.1: Installation von UNICORE-Serverseiten

Die Grafik in der Abbildung 4.1 zeigt die Abhängigkeiten der einzelnen Komponenten einer UNICORE-Installation. Dabei ist die Kommunikation zwischen dem UNICORE-Klient und dem Gateway via UPL besonders hervorgehoben. Damit wird gezeigt, dass

[1] http://www.unicore.org/downloads.htm

lediglich eine Verbindung über ein Netzwerk hergestellt werden muss, wodurch eine optionale Firewall vor dem Gateway sehr einfach zu konfigurieren ist. Als Standardfall befinden sich das Gateway und der NJS-Server in einem Netzwerk, welches durch eine Firewall geschützt ist. Daher wurden diese in einem System zusammen gefasst. Da Gateway und NJS auch über UPL kommunizieren, besteht durchaus die Möglichkeit die beiden Systeme in getrennten Netzwerken zu betreiben.

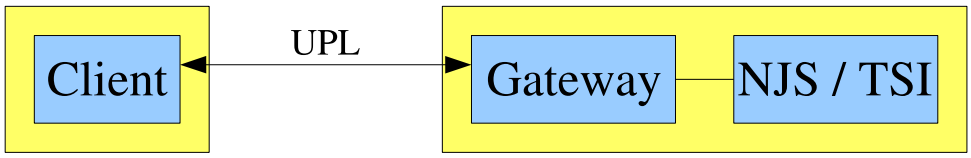

Abbildung 4.1: Aufbau von UNICORE

Desweiteren wird das Schema von UNICORE ohne der Integration von *HPFT* gezeigt. Alle Befehle und Daten werden ausnahmslos über das UPL-Protokoll gesendet. Nun soll der Dateitransfer über UPL im Folgenden durch den alternativen Dateitransfer *HPFT* (siehe Kapitel 3) ersetzt werden. Vor der Einbettung von *HPFT* in UNICORE wurde ein Konzept entwickelt, welches eine einfache und effektive Integration ermöglicht. Ein wichtiges Kriterium war, dass die Administratoren der Gateways und NJS-Server nur kleine Umstellungen an ihren Systemen vornehmen müssen. Die vorzunehmenden Änderungen dürfen am Klienten die Erstellung und Bearbeitung der Jobs für den Endbenutzer keineswegs modifizieren. Selbst unter der Verwendung von *HFPT* muss der Klient immer noch einfach und intuitiv zu handhaben sein. Darüber hinaus soll das neue Verfahren keine institutionellen Sicherheitsvorkehrungen, insbesondere die vorhandenen Firewalls, verletzen.

4.2 Designfragen zur Einbettung von HPFT in Unicore

Vor der Einbettung von *HPFT* und den Änderungen am Quellcode des UNICORE-Systems wurde eine adäquate Lösung gesucht, welche die eben genannten Anforderungen erfüllen kann. Die Grafik in Abbildung 4.2 zeigt, wie sich *HPFT* in das ursprüngliche Konzept von UNICORE einpasst. Der schwarze Rahmen im oberen Teil des Bildes umreißt die UNICORE-Komponenten, welche schon in der Abbildung 4.1 in gleicher Weise vorzufinden sind. Dabei fällt sofort auf, dass dieser nur an einer einzigen Stelle von neuen Komponenten durchbrochen wird. Nur an dieser werden die Änderungen an der UNICORE-Architektur vorgenommen, um den alternativen Dateitransfer einzubinden. Demnach wird der UNICORE-Klient in seiner Software derart abgeändert, dass er den

HPFT-Klient steuert, da von dort aus die Dateitransfers initiiert werden müssen. Die UNICORE-Serverseite hingegen verbleibt software-technisch im Originalzustand.

Der Kontroll- sowie der Datenkanal von *HPFT* werden nach dem Entwurf über das HTTP-Protokoll abgehandelt. Wird das Dateiübetragungs-Protokoll HTTP, HTTPS oder ähnliches verwendet, so muss neben UPL nur eine weitere Verbindung von der Klientseite zur Serverseite aufgebaut werden (siehe Kapitel 4.3), welche nur kleine Einschnitte in die Sicherheitsvorkehrungen darstellt.

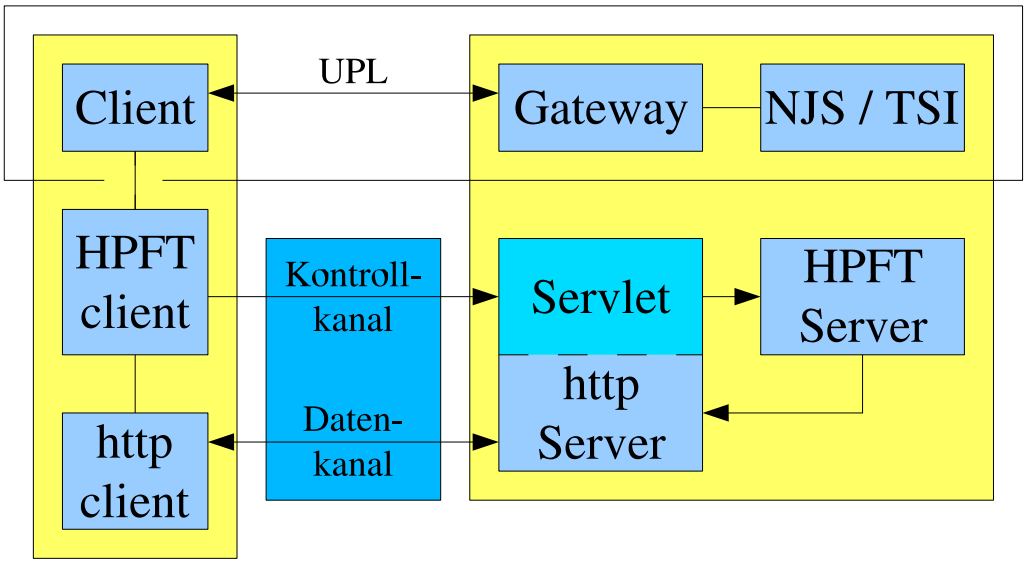

Abbildung 4.2: Einbettung von HPFT in UNICORE

Die Vorteile dieses Vorgehens liegen klar auf der Hand. Da nur der UNICORE-Klient einer Änderung an der Software unterworfen werden muss und die restlichen Komponenten unverändert bleiben, wird die Überarbeitung des UNICORE-Quellcodes stark eingeschränkt und erspart somit viel Zeit in der Entwicklung. Ein weiterer Pluspunkt ist, dass das ZAM[2] mit der Pflege und der weiteren Entwicklung des UNICORE-Klienten beauftragt ist. Somit können die hier eingebrachten Änderungen am Quellcode in den neuen Veröffentlichungen des Klienten direkt eingebunden werden, wodurch *HPFT* in der Zukunft stets in Verbindung mit UNICORE verwendet werden kann.

Diese Punkte zusammengefasst legen dar, dass die getroffenen Entscheidungen über die Einbindung von *HPFT* eine angemessene Lösung bieten. Es findet eine direkte und intuitive Integration des alternativen Dateitransfers statt. Auf der Klientseite wird *HPFT* über ein Plugin in die Software von UNICORE eingebunden, wohin gegen auf Serverseite eine eigenständige Instanz eines *HPFT*-Servers läuft, welcher mit einem speziellen HTTP-

[2]Zentralinstitut für Angewandte Mathematik - Forschungszentrum Jülich GmbH

Plugin zur Verwaltung des Kontrollkanals ausgestattet ist. Es muss erwähnt werden, dass zur Verwirklichung eines alternativen Dateitransfers neben UPL mindestens eine weitere Netzwerkverbindung geöffnet werden muss. Somit könnte sich hieraus ein Problem bei dem Betreiben des Dateitransfers entwickeln, wenn eine Institution keine Möglichkeit hat, einen weiteren Port ihrer Firewall zu öffnen. Dies sollte in der Regel aber keine großen Schwierigkeiten darstellen.

4.3 Gewählte Lösung im Detail

Wie schon erwähnt, wird lediglich der UNICORE-Klient durch ein Software-Plugin erweitert. Das bedeutet insbesondere, dass die Arbeitsweisen von Gateway und NJS nicht verändert werden. Die Möglichkeit der Einbindung von *HPFT* ergibt sich daraus, dass Dateitransfers in UNICORE als ein Datei-Import innerhalb eines Jobs behandelt werden. Diesen Datei-Import-Jobs wird die Quelle der zu importierenden Datei angegeben. Dabei gibt es zwei mögliche Quellen:

Nspace Die zu importierende Datei befindet sich auf dem lokalen PC des Benutzers und muss somit über das Netzwerk hochgeladen werden, bevor diese von einem nachfolgenden Job bearbeitet werden kann.

Xspace Die zu importierende Datei befindet sich bereits auf dem Zielsystem des Jobs und kann mit einem einfachen Kopierbefehl in das Uspace geholt werden.

Die Vorgehensweise bei der Einbettung von *HPFT* in den UNICORE-Klient besteht aus einer Änderung der abgeschickten Jobs (siehe Abbildung 4.3). Der UNICORE-Benutzer kann ohne weiteren Aufwand seinen Job erstellen und gibt bei einem Dateiimport wie gewohnt seine Dateien auf dem lokalen Rechner an. Jedoch kurz bevor der Job über UPL versendet wird, tritt das *HPFT*-Plugin in Kraft und wendet den erstellten Sendefilter an. Dieser überprüft zunächst, ob Dateiimporte vom Nspace des Benutzers erfolgen sollen. Ist dies der Fall, so versucht das Plugin die Dateien mittels *HPFT* auf das Zielsystem (das Xspace) zu übertragen. Erst nach einem erfolgreichen Hochladen der Dateien, wird der abgefangene Job in dem Rahmen bearbeitet, dass die Dateiimporte vom Nspace auf Dateiimporte vom Xspace umgeändert werden, sodass der Job keinen Dateitransfer über UPL mehr enthält. Dann gibt der Filter den geänderten Job zurück, welcher wie gewohnt über UPL versendet wird. Auf dem Zielsystem bewirkt dieser Job nur noch einen üblichen Kopierbefehl, welcher die transferierte Datei vom Xspace in das Nspace kopiert. Somit ist keine Änderung der serverseitigen UNICORE-Software erforderlich, da mit schon vorhandenen Funktionen gearbeitet werden kann.

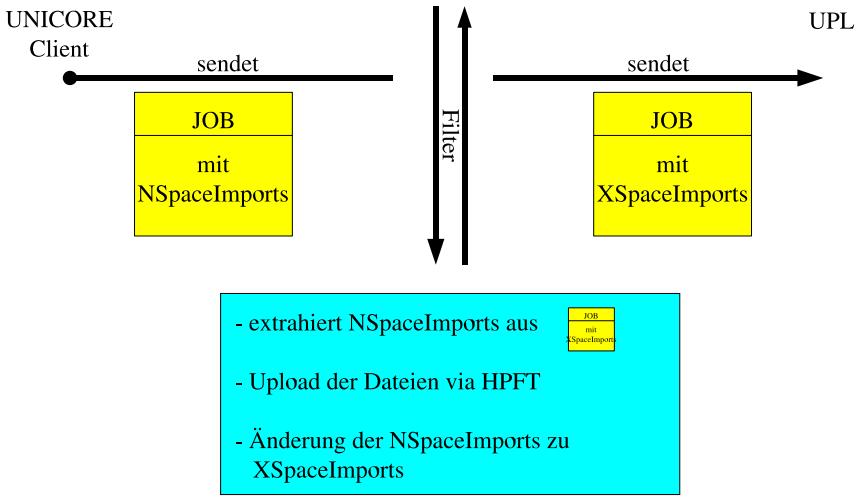

Abbildung 4.3: HPFT Plugin Sende-Filter

Eine Besonderheit befindet sich auch in der Konzeption des Kontrollkanals. Das HTTP-Plugin unterstützt in diesem Falle nicht nur den Datenkanal, sondern auch den Kontrollkanal. Dazu wird dem HTTP-Server ein kleines Java-Servlet[3] unter der Adresse `http://hpfthost:port/controlchannel` hinzugefügt. Die Kontrollkanalanfragen (vergl. Tabelle 3.1) werden an diese Adresse mit einer normalen HTTP-GET-Methode gesendet, welche unter dem Header `HPFT ControlMessage` die Anfrage vom Klient enthält. Das kleine Servlet extrahiert diesen Header und leitet diese Anfrage an den *HPFT*-Server weiter. Darauf hin erhält das Java-Servlet die vom Server generierte Antwort (vergl. Liste 3.2) und erstellt daraus eine Antwort auf die HTTP-GET-Methode. Die Kommunikation zwischen *HPFT*-Klient und -Server wird demnach über das Java-Servlet im HTTP-Server abgehandelt.

```
----------    HTTP-GET    -----------  processClientReq   ----------
|        | ------------> |         | ------------------> |        |
|  HPFT  |               |  HTTP   |                     |  HPFT  |
| CLIENT | <------------ | SERVER  | <================== | SERVER |
|        |   Antwort     |(SERVLET)|    Konfiguration    |        |
----------               |         |    und Rueckgabe    ----------
                         -----------    der Nachricht
```

Tabelle 4.2: Kontrollkanal über das Servlet

[3]Java-Programm, welches Anfragen über HTTP bearbeiten kann (siehe Anhang B)

4.4 Änderungen am UNICORE-Klient

Der UNICORE-Klient unterstützt bereits die Verwendung von Erweiterungs-Plugins. Ein solches Plugin wurde auch für HPFT entwickelt. Dieses enthält eine speziell für diese Aufgabe entworfene Schnittstelle (`ISubmitFilter`). Dafür mussten am Quellcode des UNICORE-Klienten die folgenden Java-Klassen erweitert werden:

`com.pallas.unicore.client.util.ExtensionPlugable.java:`

```
   public ISubmitFilter getSubmitFilter(JobContainer job)
      throws Exception {
         return null;
   }

   public interface ISubmitFilter {
      void filter(JobContainer job) throws Exception;
   }
```

`com.pallas.unicore.client.trees.JPATree.java:`

```
submit:
   ...
   for(int i=0;i<v.size();i++){
      ext=(ExtensionPlugable)v.get(i);
      try{
         filter=ext.getSubmitFilter(job);
         if(filter!=null)filter.filter(job);
         }catch(Exception filterEx){
    ...
```

Um ein Erweiterungs-Plugin zu erstellen, muss diese Plugin-Klasse von der Klasse `ExtensionPlugable` abgeleitet werden. Dadurch wird nun die Methode `getSubmitFilter` mitvererbt. In der vererbten Definition wird einfach der Wert `null` zurückgegeben, was bedeutet, dass dieses kein SubmitFilter enthält. Das `HPFT`-Plugin ist von der Klasse `ExtensionPlugable` abgeleitet, überlädt jedoch diese Methode, so dass beim Aufruf nicht `null` sondern die Instanz einer Klasse mit der Software-Schnittstelle des `ISubmitFilter` zurückgegeben wird. Diese enthält die Methode `filter`, mit welcher der Job vor jedem Absenden aufgerufen und verändert wird.

4.5 Änderungen am Unicore Gateway / NJS / TSI

Für *HPFT* muss selbstverständlich eine Zieladresse existieren, welche den Kommunikationsendpunkt vom Kontrollkanal angibt. Dies muss in der Datei *njs.idb*[4] als Information angegeben werden, welche von jeder Vsite an den UNICORE Klient übergeben wird. Diese Datei enthält unter Anderem einen Informationstyp (*TextInfoResource*), womit dem UNICORE Klient Informationen nach verschiedenen Schlüsseln mitgeteilt werden können. Die einzige Änderung an der UNICORE Serverseite liegt also in einer Zeile in der Datei *njs.idb*:

TextInfoResource [HPFT Servlet ControlChannel] Tag [Typ] Value [host:port]

Der Schlüssel [HPFT Servlet ControlChannel] zeigt unverkennbar an, dass mit der angegebenen Netzwerkadresse [host:port] das Ziel zum Kontrollkanal bekannt gemacht wird. Diese ist auch erforderlich, damit der *HPFT*-Klient eine Verbindung zum *HPFT*-Server aufbauen kann.

Wie in der Abbildung 4.2 zu erkennen ist, muss auf dem Zielsystem zusätzlich der *HPFT*-Server gestartet werden, welcher somit über die Angaben aus der *njs.idb* zu erreichen ist.

[4] NJS - Incarnation Database

Kapitel 5

Benutzeranleitung

5.1 HPFT-Dateitransfer

Die Anleitung soll verdeutlichen, wie der *HPFT*-Filetransfer allgemein verwendet werden kann. In den Grundzügen besteht es aus der Erstellung und der Analyse der Kontrollkanalnachrichten. Die Übertragung dieser Nachrichten ist kein direkter Bestandteil von *HPFT*. Es sei lediglich angemerkt, dass der mitgeliefert HTTP-Server mit einem Java-Servlet ausgerüstet ist, welches die Übertragung dieser Nachrichten unterstützen kann (siehe Kapitel 4.3).

Die Programmierschnittstelle (API[1]) von *HPFT* wird zum besseren Verständnis durch Beispiele erklärt. Der Klient möchte die lokale Datei

`/home/testuser/local/beispiel.txt`

in das Remote-Verzeichnis

`/home/testuser/remote`

hochladen. Dort soll die Datei den Namen `uploaded_file.txt` haben. Als Demonstration wird diese Datei wieder heruntergeladen und als

`/home/testuser/local/downloaded_file.txt`

gespeichert. Die in den folgenden Kapitel aufgelisteten Schritte auf Klient- und Serverseite zeigen den verwendeten Java-Quellcode. Bei erstellten Kontrollkanalnachtichten werden auch die erzeugten Strings in Anführungsstrichen angegeben.

[1] engl. application programming interface

5.1.1 Starten des Servers

Serverseite:

```
FTS fts = new FTS();
fts.startServer("http", "testserver:8080");
```

Ein Server wird mit der `startServer`-Methode gestartet. Diese erhält als erstes Argument den Protokollnamen und als zweites eine Infozeile, welche je nach Plugin variiert. In diesem Fall, wird das HTTP Plugin angesprochen, welches zwei Angaben fordert. Die Angabe von `testserver` ist lediglich ein String zum Identifizieren des Servers, da mehrere HTTP-Server gestartet und angesprochen werden können. Die Port-Nummer gibt an, auf welchem Port der HTTP-Server laufen soll.

5.1.2 Abfrage der Protokolle

Klientseite:

```
String cr = ClientRequestFactory.createCapabilitiesRequest();
```

```
> "HPFT Control Protocol v1.0|6 REQ_CAPABILITIES|."
```

Serverseite:

```
String sa = fts.processClientRequest(cr, null, null);
```

```
> "HPFT Control Protocol v1.0|0 ANS_CAPABILITIES_LIST|PROT=http|."
```

Bevor der Klient ein Dateitransferprotokoll anfordert, hat er die Möglichkeit, die zur Verfügung stehenden Protokolle abzufragen. Wie in der Antwort des Servers zu sehen ist, wird nur das HTTP-Protokoll angeboten.

5.1.3 Anforderung der Zugriffsrechte

Klientseite:

```
cr = ClientRequestFactory.createNeedServerRequest("http");
```

```
> "HPFT Control Protocol v1.0|7 REQ_NEED_SERVER_PLS|PROT=http|."
```

Serverseite:

```
sa = fts.processClientRequest(cr, "testuser", "/home/testuser/remote");
```

```
> "HPFT Control Protocol v1.0|1 ANS_SERVER_IS_NOW_RUNNING_FOR_YOU|
   PROT=http|
   INFO=zam637.zam.kfa-juelich.de:8080:testuser:
   /home/testuser/remote/testuser:reservedinfo|."
```

Klientseite:

```
String protInfo[] = ControlChannel.parse(sa);
protInfo[0]="http"
protInfo[1]="zam637.zam.kfa-juelich.de:8080:testuser:
            /home/testuser/remote/testuser:reservedinfo"
FTCClient httpClient = ftc.createClient(protInfo[0], protInfo[1]);
```

Zunächst kann der Klient die Informationen für seinen Zugriff auf den HTTP-Server anfordern. Bei der Zusammenstellung der Anforderung werden der Methode `processClientRequest` zusätzlich noch der Benutzername und das Root-Verzeichnis des Benutzers angegeben. Dabei legt der *HPFT*-Server in seinem HTTP-Server einen neuen Kontext an. Ein Kontext verbindet eine URL[2] mit einem lokalem Verzeichnis auf dem HTTP-Server. Zusätzlich können dem Kontext bestimmte HTTP-Methoden zugewiesen werden. In dem Fall sind es die HTTP-GET und HTTP-PUT Methoden zum Transferieren von Dateien. Die URL des Kontextes ist unter:
`http://zam637.zam.kfa-juelich.de:8080/testuser`
zu erreichen, worüber der Klient auf das für ihn erstellte Verzeichnis
`/home/testuser/remote/testuser`
zugreifen kann. Diese Informationen erhält der Klient in der Antwort des Servers, worauf er einen HTTP-Klient erstellen kann. Durch die `parse`-Methode werden dabei der Server-Antwort die entsprechenden Informationen entnommen.

Das Feld `reservedinfo` kann ein vom Server generiertes Passwort enthalten, welches der Klient für den temporären Zugriff auf den angeforderten HTTP-Server braucht.

5.1.4 Dateitransfer

Klientseite:

```
httpClient.putFile("/home/testuser/local/beispiel.txt",
                   "uploaded_file.txt");

httpClient.getFile("uploaded_file.txt",
                   "/home/testuser/local/downloaded_file.txt");
```

Die Methoden des Hoch- und Runterladens haben eine sehr einfache und intuitive Struktur. Es wird jeweils als erstes Argument die Quelldatei und als zweites die Zieldatei angegeben. Für die lokalen Dateien ist es empfehlenswert, absolute Pfadnamen zu verwenden.

[2] engl. Uniform Resource Locator (einheitlicher Ortsangeber für Ressourcen)

5.1.5 Beenden des Dateitransfers

Klientseite:

```
cr = ClientRequestFactory.createDoneFiletransfer("http",
        "/home/testuser/remote/testuser");

> "HPFT Control Protocol v1.0|8 REQ_DONE_FILETRANSFER_THX|PROT=http|
  INFO=/home/testuser/remote/testuser|."
```

Serverseite:

```
sa = fts.processClientRequest(cr, "testuser", "/home/testuser/remote");

> "HPFT Control Protocol v1.0|2 ANS_OK_STOPPING_YOUR_SERVER_NOW|
  PROT=http|."

fts.stopServer("http", "testserver:8080");
```

Nachdem der Klient seine Transfer-Operationen beendet hat, meldet er dies dem Server. Zum Erstellen der Kontrollnachricht muss das Protokoll und das Root-Verzeichnis angegeben werden, welches dem Klienten bei den Zugriffsrechten mitgeteilt worden ist. Daraus folgernd kann der Server die Zugriffrechte wieder entfernen. Bei Bedarf kann der HTTP-Server auch ganz gestoppt werden.

5.2 UNICORE-Einbettung

5.2.1 Auf der Klientseite

Der UNICORE-Klient lässt nur Erweiterungs-Plugins zu, die mit einem Zertifikat versehen sind, welche von einer festegelegten Certification Authority (CA) unterschrieben sind. Daher wird zur Einbettung in den UNICORE-Klienten ein Installations-Skript (build.xml) mitgeliefert. Vor dem Ausführen bedarf es zunächst, einige Werte anzupassen:

```
<property name="home" location="/home/tweddell/"/>
<property name="keystore"  value="${home}/.unicore/mykeystore"/>
<property name="passwort"  value="geheim"/>
<property name="alias"  value="keystore alias"/>
<property name="instdest"  location="${home}/uclient/build/lib"/>
```

home Das Home-Verzeichnis des Benutzers vom UNICORE-Klient.

instdest Das Verzeichnis, wo die Plugins des UNICORE-Klienten liegen.

keystore Die keystore-Datei des Benutzer vom UNICORE-Klient.

passwort Das Passwort der keystore-Datei.

alias Der Alias-Name des Eigentümers der keystore-Datei.

Das Passwort der keystore-Datei muss hier angegeben werden, damit bei der Installation die erzeugte Plugin-Jar-Datei mit dem Zertifikat vom Benutzer unterzeichnet werden kann.

Zur Ausführung des Skriptes werden die folgenden Dateien benötigt:

HPFTClient.jar Der *HPFT*-Klient (siehe Kapitel 5.1)

ClientPlugin.jar Die *HPFT*-Plugins für den Klient (siehe Kapitel 5.1) Da hier der *commons-httpclient*[3] verwendet wird, müssen auch die Dateien *commons-codec-1.3.jar, commons-logging.jar und commons-httpclient-3.0-rc4.jar* zur Verfügung stehen.

client.jar Der UNICORE-Klient.

ajo.jar Das UNICORE Abstract Job Object.

Mit dem Programm „*ant*“[4] kann das *HPFT*-Plugin kompiliert und installiert werden. Zum Installieren wird „*ant install*“ aufgerufen, wobei als erstes die Quellen kompiliert werden. Danach wird die Plugin-Jar-Datei für UNICORE erstellt und mit dem Zertifikat vom Benutzer unterschrieben. Diese Datei wird mit samt allen abhängigen Dateien ins Plugin-Verzeichnis des UNICORE-Klienten kopiert. Bei einem Neustart des Klienten oder neuem Einlesen der Plugins kann unter dem Menu-Eintrag „Help - Plugin Info“ festgestellt werden, ob sich ein Eintrag mit dem Namen „HPFT Plugin “ befindet.

Es wäre auch möglich, das Plugin von einer zentralen Stelle unterzeichnen zu lassen, somit fällt der Punkt des Kompilierens und der Erstellung der Plugin-Jar-Datei nicht mehr dem Benutzer zur Last, so dass dieser nur noch die Jar-Dateien in sein UNICORE-Verzeichnis kopieren muss.

Dem *HPFT*-Plugin muss zusätzlich noch mitgeteilt werden, wo es seine Plugins für den Dateitransfer findet. Dazu befindet sich in dem Verzeichnis `$(HOME)/.unicore/` eine Datei mit dem Namen `HPFTclientrc` mit dem Inhalt:

```
#FileTransferClient
#Wed Jan 18 16:16:38 CET 2006
PluginDirectory=/home/tweddell/unicore/client/build/lib/
Version=0.1
Plugins=ClientPlugin.jar\:dummy, ClientPlugin.jar\:http,
```

Der Eintrag `PluginDirectory` muss dementsprechend auf das Verzeichnis gesetzt werden, in welchem die Datei *ClientPlugin.jar* liegt. Beim Start des UNICORE-Klienten wird gleizeitig auch das *HPFT*-Plugin gestartet. Die gefundenen *HPFT*-Plugins werden in dem Eintrag `Plugins` aufgelistet.

[3] `http://jakarta.apache.org/commons`
[4] The Apache Ant Project

Verwendung von HPFT im UNICORE-Klient

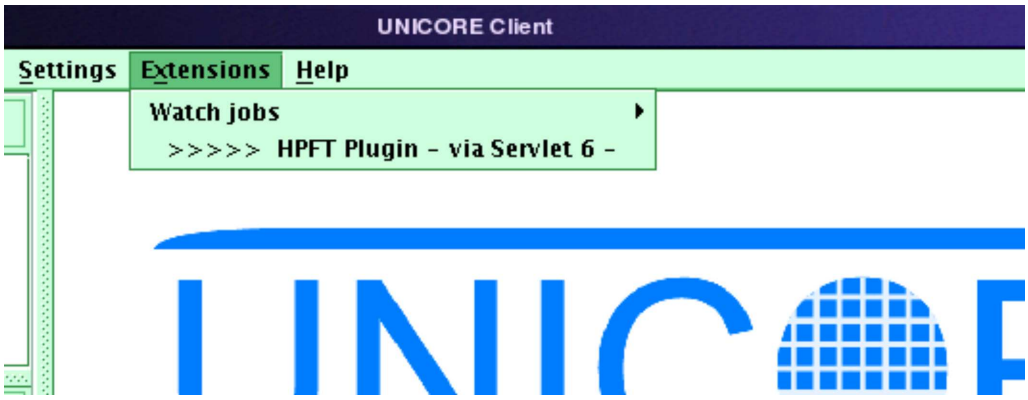

Abbildung 5.1: Menu-Eintrag des HPFT-Plugins

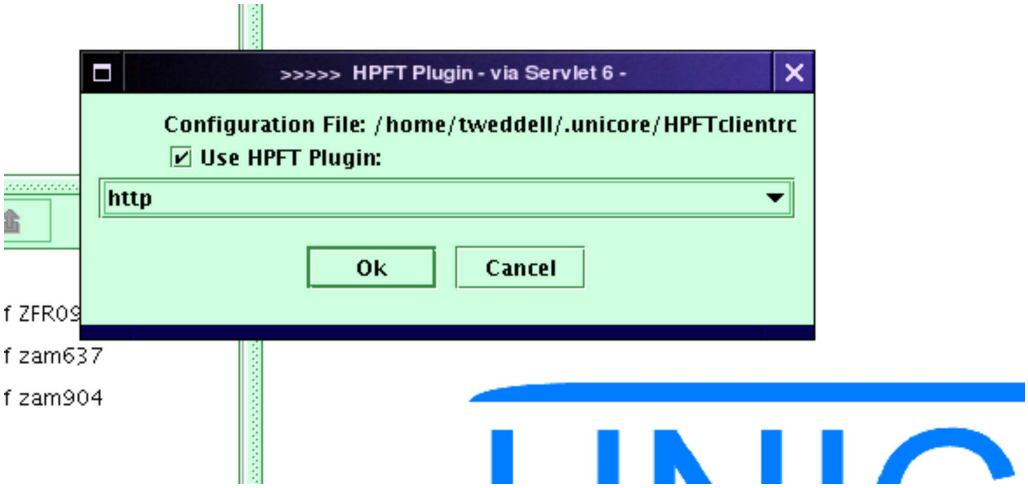

Abbildung 5.2: Einstellungen des HPFT Plugins

Nach dem Start des UNICORE-Klienten sollte das *HPFT*-Plugin in dem Menu „Help - Plugins“ aufgelistet sein. Desweiteren entsteht ein Eintrag im Menu „Extensions - HPFT Plugin“ (siehe Abbildung 5.1). Es wird ein kleines Konfigurationstool (siehe Abbildung 5.2) angeboten, welches Informationen über die verwendete Konfigurationsdatei *HPFTclientrc* und das gewählte Dateitransferprotokoll gibt. Als Option kann die Verwendung des alternativen Dateitransfers auch ganz abgeschaltet werden.

Desweiteren treten in der Handhabung des UNICORE-Klienten keine Änderungen auf. Jegliche erstellten Jobs können in gewohnter Weise abgesendet und empfangen werden.

5.2.2 Auf der Serverseite

Auf der UNICORE-Serverseite (TSI) muss der *HPFT*-Filetransfer-Server installiert werden, wobei das HTTP-Plugin, wie in Kapitel 4.3 beschrieben, die Aufgabe hat, den Kontrollkanal und den Datenkanal zu verwalten. Zum Kompilieren des Programms kann das *ant*-Skript ausgeführt werden. Die erzeugten Dateien der Distribution befinden sich danach im dem Verzeichnis: *./dist/lib*. Diese müssen alle auf das Zielsystem kopiert werden. Dort kann der *HPFT*-Server nun mit:

```
java -jar HPFTServer.jar <HPFTRootDir> <serverport>
```

gestartet werden. Das Verzeichnis `HPFTRootDir` legt fest, wo die temporären Dateien zum Hochladen zwischengespeichert werden. Als letztes muss in der Datei *njs.idb* die Information über den gestarteten Server in einer Zeile eingetragen werden:

```
TextinfoResource [HPFT Servlet ControlChannel] [Typ]
                                    Value [host:serverport]
```

Nach einem Neustart des NJS Servers kann der UNICORE-Klient auf diese Information zugreifen und einen Dateitransfer mit diesem durchführen.

Zur einfachen Benutzung wird auch eine Skript-Datei (*start.sh*) ausgeliefert:

```
#!/bin/sh


echo .
echo . Be sure to provide a TextInfoResource in your njs.idb file
echo . TextInfoResource [HPFT Servlet ControlChannel]   Tag [Typ] \
 Value [host:port]
echo .

export ROOTDIR=./HPFTROOT
export PORT=9178

java -jar HPFTServer.jar $ROOTDIR $PORT
```

In den lokalen Variablen *ROOTDIR* und *PORT* können die gewünschten Werte eingetragen werden. Somit braucht der *HPFT*-Server nur noch mit diesem Skript gestartet werden. Das Programm gibt während seiner Abarbeitung Informationen über die gestarteten Dateitransfers.

Kapitel 6

Fazit

6.1 Performanz

In Kapitel 4 wurde bereits beschrieben, dass insgesamt drei UNICORE Server-Systeme auf verschiedenen Netzwerkarchitekturen benutzt wurden, um *HPFT* in UNICORE zu integrieren und zu testen. Auf diesen drei Systemen werden auch die Tests für die Messung der Performanz durchgeführt.

Serverhost	Beschreibung
localhost	UNICORE-Klient, Gateway, NJS und TSI laufen auf einer Maschine. (IBM Thinkpad, 2Ghz, 786 MB, SuSE Linux 9.3)
zfr094.zfr	Serverinstallation in einem 10Base-2 Netzwerk (10MBit/sek)
zam904.zam	Serverinstallation in einem 100Base-T Netzwerk (100MBit/sek)

Tabelle 6.1: Installation von UNICORE-Serverseiten

Als Klientrechner wird das *IBM Thinkpad T30* genutzt. Dies hat den Hostnamen zam637.zam und befindet sich in dem gleichen 100Base-T Netzwerk wie der Rechner mit dem Namen zam904.zam.

Um zu einem aussagekräftigen Vergleich zwischen den erreichten Datentransferraten zu kommen, muss zwischen drei Kriterien unterschieden werden. Die Unterschiede resultieren insbesondere aus dem verwendeten Transfer-Protokoll, dem zugänglichen Netzwerk und den Eigenschaften der zu übertragenen Datei. Aus diesem Grund werden zwei Typen von Dateien in in den Grössen 1 KB, 500 KB und 10 MB mit einem Programm generiert.

Typ A Der Dateityp A wird lediglich durch das Zeichen '.' (ASCII[1]-Wert 46) aufgefüllt und ist somit sehr schnell und effizient zu komprimieren. Die Kompression mit

[1] American Standard Code for Information Interchange

GZIP der 10 MB großen ASCII-Datei dauert 0.28 Sekunden und erreicht eine Verringerung der Größe von über 99%.

Typ B Der Dateityp B wird mit einer zufälligen Folge von binären Werten aufgefüllt. Daher wird eine Komprimierung der Daten verhindert, wodurch das Verfahren auch stark verlangsamt wird. Die verwendete Datei ist nach dem 1.29 Sekunden langen Kompressionsverfahren mit GZIP sogar größer (Kompressionsrate ca -0.15%)

Insgesamt werden sechs verschiedene Dateien erstellt und in den unterschiedlichen Varianten über die Netzwerke transferiert. Daraus ergeben sich Mittelwerte, die in der unten stehenden Tabelle (siehe Tabelle 6.2) aufgeführt sind. Die Messungen der Dateitransfers wurden im ZAM[2] nach dem üblichen Feierabend durchgeführt, so dass von einer sehr geringen Nebenlast des LAN-Netzwerks ausgegangen werden kann[3]. Als *HPFT*-Dateiübertragungsprotokoll wird HTTP verwendet. Dies ist ein sehr weit verbreitetes und gängiges Protokoll in der heutigen Netzwerkwelt und stellt damit einen guten Vergleich dar.

	Dateigröße	1KB		500KB		10MB	
		ascii	binär	ascii	binär	ascii	binär
localhost	*UPL*	*18,3*	*13,8*	*145*	*141*	*192*	*191*
	HPFT	46	78	4571	828	7767	1316
100Mbit	*UPL*	*5,5*	*5,6*	*156*	*153*	*197*	*198*
	HPFT	56	93	4330	817	7276	1126
10Mbit	*UPL*	*5,7*	*5,5*	*151*	*148*	*194*	*192*
	HPFT	33	53	724	723	825	865

Angaben in KB / Sekunde

Tabelle 6.2: Datenübertragungsraten von UPL und HPFT

Die Tabelle 6.2 zeigt den gewünschten Erfolg der Verwendung von *HPFT* innerhalb von UNICORE. Ähnlich, den schon im Kapital 3.1.1 angegeben Resultaten über den Vergleich mit GridFTP, wirkt UPL in seiner Dateitransferrate nicht überzeugend. Auch hier lagen alle Dateitransferraten mittels UPL unterhalb der 200 KB/Sekunden Marke.

Eigentlich wurde vermutet, dass bei UPL die Übertragung von Dateien des Typs A markant schneller sind als die des Typs B, da die Komprimierung einen erheblichen Unterschied in der Menge der Daten bringt. Bei der Betrachtung eines Netzwerkmonitors während der Absendung eines UNICORE-Jobs mit einem eingebundenen Dateiimport (5 MB), kann die Netzwerkauslastung ermittelt werden. Die Abbildung 6.2 zeigt die Menge

[2] Zantralinstitut für Angewandte Mathematik

[3] Anmerkung: Während den Arbeitszeiten kamen die Messungen teilweise nur auf die Hälfte der in der Tabelle 6.2 angegebenen Werte. Dies ist ein Indikator für Überlastsituationen, auf denen TCP mit einer Reduktion der Übertragungsrate reagiert.

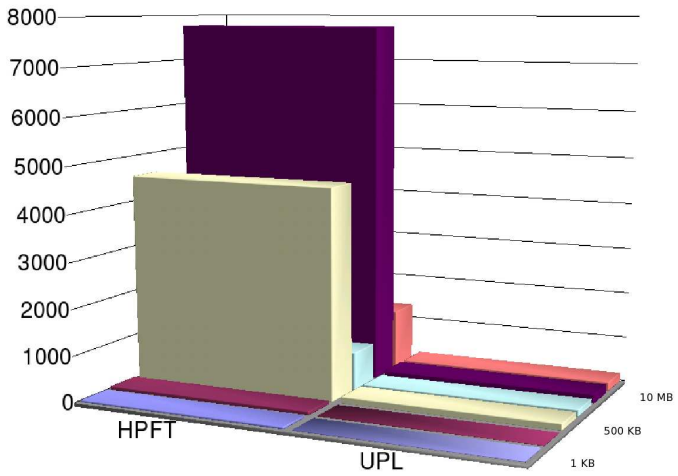

Abbildung 6.1: Datentransferraten via localhost

und die Geschwindigkeit der übers Netzwerk verschickten Daten bei dem Sendevorgang zweier Jobs. Die oberen beiden Diagramm zeigen das Hoch- und Runterladen über die lo-Schleife des UNICORE-Servers. Das bedeutet, dass der Datenaustausch zwischen den einzelnen Serverkomponenten (Gateway / NJS / TSI) gemessenwerden kann. Das dritte Diagramm zeigt die vom UNICORE-Klienten gesendeten Datenraten. Dabei zeigen die Höhen der Diagramme die tatsächliche Datenübertragungsrate an. Somit kommt der ZIP-komprimierte UPL-Datenstrom auf ca. 250 KB/Sekunde Datendurchsatz. Die Messungen weisen aber immer weniger als 200 KB/Sekunde auf. Das hängt damit zusammen, dass zum Abarbeiten des Sendvorganges eines Job Rechenzeit verbraucht wird, welches sich negativ auf den Datendurchsatz von UPL auswirkt.

Die senkrechten Striche grenzen den zeitlichen Verlauf des Sendevorganges eines Jobs ein, wie er vom Benutzer am UNICORE-Klienten verfolgt wird. Die Startmarkierung befindet sich zu dem Zeitpunkt, wenn der Sendebefehlt erfolgt, die Endmarkierung zeigt an, wann der UNICORE-Klient den Sendejob als erledigt meldet. Aus diesen Zeitwerten wird die Datenübertragungsrate bei UNICORE-UPL gemessen. Das Absenden eines Jobs geschieht einmal für eine Datei des Typs A (rechts) und des Typs B (links).Dabei ist klar zu erkennen, dass im Fall von Typ A zwischenzeitlich keine Netzwerklast auf dem Rechner vom UNICORE-Klient zu verzeichnen ist. Daraus folgernd ist die Gesamtzeit zum Versenden des Jobs wesentlich höher als die wirklich verbrauchte Sendezeit über das Netzwerk. Die Frage stellt sich, worauf der Klient wartet, nachdem die komprimierte Datei hochgeladen wurde. Dieses begründet sich in der Architektur von UNICORE. Wird eine Datei einem Job als Datenstrom angehängt, so wird dieser Datenstrom vom Gateway

direkt an das NJS weitergeleitet. Das NJS jedoch muss die Daten entpacken und analysieren, um diese an das TSI weiterleiten zu können. Der UPL-Transfer zwischen dem NJS und dem TSI allerdings findet ohne Komprimierung statt, wobei die vollen 5 MB Daten übers Netz gesendet werden.

```
Klient ----------> Gateway ----------> NJS ---------> TSI
       komprimiert         komptimiert     rohe Daten
```

Der UNICORE-Klient muss darauf warten bis die Datei auf dem TSI angekommen ist und erhält erst danach ein OK vom NJS. Daher macht es für die Performanz in diesem Fall keinen erheblichen Unterschied, ob die Datei gut oder schlecht komprimierbar ist. Da in der Regel der UNICORE-Klient über ein WAN mit dem Gateway verbunden ist, kann sich hier ein komprimierter Datenstrom positiv auswirken. Dieser Effekt kann in diesen Tests leider nicht kenntlich gemacht werden. Ein weiterer Vorteil liegt auch in dem niedrigen Transfervolumen auf der Klientseite. Es sollte an dieser Stelle überlegt werden, ob es sinnvoll wäre, dass der NJS die Daten auch wieder mit GZIP komprimert, bevor er diese an den TSI weiterleitet, da dies einen beträchtlichen Vorteil in der Übertragungsgeschwindigkeit bringen würde. Es wäre auch möglich, dass der UNICORE-Klient ein OK erhält, solbald die Datei vollständig auf dem NJS eingetroffen ist. Dies könnte jedoch zu Problemen führen, wenn der TSI Server abgeschaltet oder ausgefallen ist. Somit müsste der NJS über Verwaltungsstrategien nicht zustellbarer Jobs verfügen.

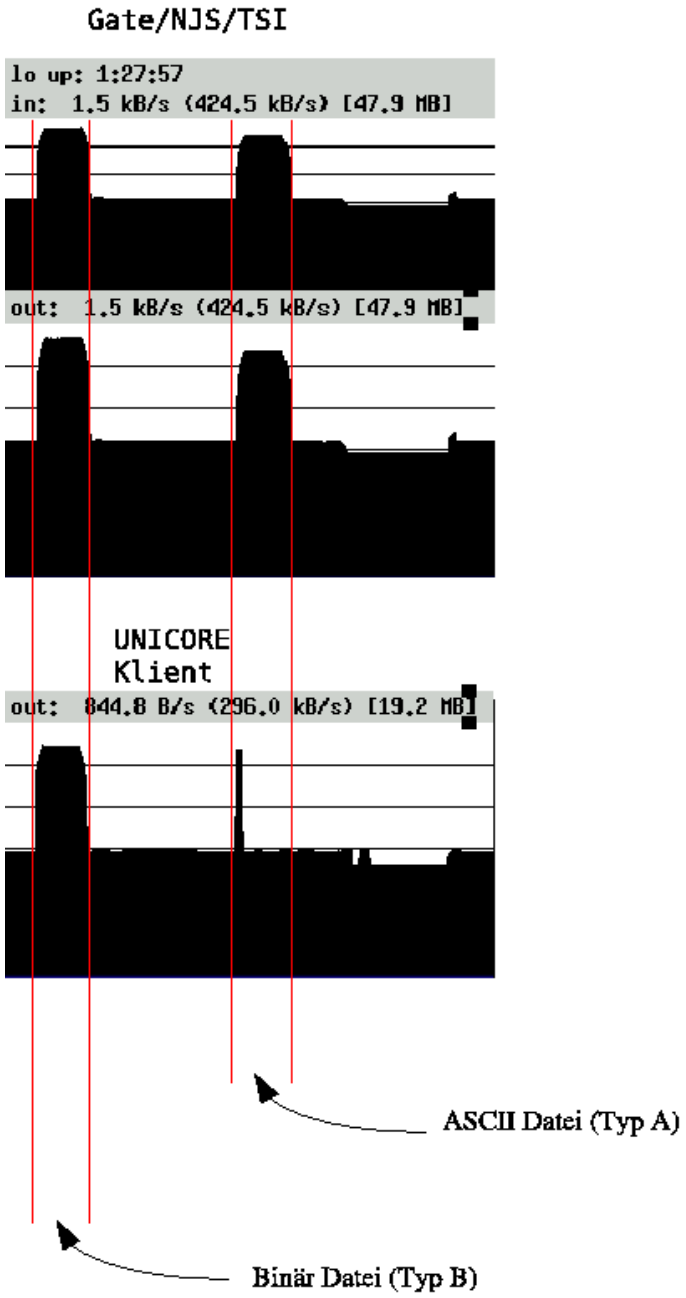

Abbildung 6.2: Netzauslastung von UPL (Screenshot von xnetload)

6.2 Zusammenfassung

HPFT hat gezeigt, wie es sich durch seine bewusst getroffenen Designentscheidungen in das UNICORE-Grid-System integrieren kann und stellt somit seine Eigenschaften unter Beweis. Es wurde auch, wie gewünscht, eine Verbesserung des Datentransfers innerhalb von UNICORE erreicht. Diese bezieht sich derzeit nur auf das Hochladen von Dateien vom UNICORE-Klienten zum TSI. In einer Erweiterung kann auch das Herunterladen von Dateien implementiert werden. Dieses birgt jedoch einen weit größeren Aufwand, da nicht direkt mit schon vorhandenen Funktionen von UNICORE gearbeitet werden kann, wie es beim Hochladen der Fall war. Es wird davon ausgegangen, dass zu diesem Zweck der umfangreiche und teilweise unübersichtliche Code der UNICORE-Server Komponenten angepasst oder verändert werden müssen.

Aus dieser erfolgreichen Einbindung von *HPFT* in ein bestehendes System, kann schlussgefolgert werden, dass *HPFT* auch in anderen Systemen zur Anwendung kommen kann. Die einfache Handhabung des Kontrollkanals sollte keine Probleme bei eine Integrierung bringen. Zudem besteht dem Benutzer die Möglichkeit weitere Dateitransferprotokolle in *HPFT* einzubinden, die auf seine Bedürfnisse hin abgestimt werden können.

Anhang A

Glossar

AES	Der Advanced Encryption Standard (AES) ist ein symmetrisches Kryptosystem, das als Nachfolger für DES bzw. 3DES im Oktober 2000 vom National Institute of Standards and Technology (NIST) als Standard bekannt gegeben wurde.
AJO	Das UNICORE Abstract Job Objekt steht für ein vom UNICORE Klienten erstellter Job zum Versenden über UPL
ANT	Ant (englisch für Ameise) ist ein in Java geschriebenes Werkzeug zum automatisierten Erzeugen von Programmen aus Quelltext.
API	Eine Programmierschnittstelle ist die Schnittstelle, die von einem Betriebssystem oder von einem anderen Softwaresystem weiteren Programmen zur Verfügung gestellt wird.
BMBF	Bundesministerium für Bildung und Forschung
CA	Eine Zertifizierungsstelle (engl. Certificate Authority, kurz CA) ist eine Organisation, die digitale Zertifikate herausgibt.
CPU	Der Hauptprozessor, englisch Central Processing Unit (CPU), im allgemeinen Sprachgebrauch oft auch nur als Prozessor bezeichnet, ist der Teil eines Computers, der alle anderen Bestandteile steuert.
Firewall	Eine Firewall (engl. Feuerschutzwand) ist ein System aus Software- und Hardwarekomponenten, das den Zugriff zwischen verschiedenen Rechnernetzen beschränkt, um ein Sicherheitskonzept umzusetzen.
FTP	engl. File Transfer Protocol (siehe Kapitel 2.2.1)
FTPS	FTP über SSL (siehe Kapitel 2.2.1)
Gateway	Das UNICORE Gateway (siehe Kapitel 2.1.2)
HPFT	engl. High Performance Filetransfer
IBM	International Business Machines ist eines der ältesten IT- Unternehmen.

IP	Das Internet Protocol ist ein in Computernetzen weit verbreitetes Netzwerkprotokoll.
JAR	Bei einer JAR-Datei (kurz für Java-Archiv) handelt sich um eine ZIP-Datei, die zusätzliche Metadaten in einer immer vorhandenen Datei „META-INF/MANIFEST.MF“ enthält. JARs werden vor allem zur Verteilung von Java-Libraries und -Programmen eingesetzt.
JAVA	Java ist eine objektorientierte Programmiersprache und als solche ein eingetragenes Warenzeichen der Firma Sun Microsystems. Sie ist eine Komponente der Java-Technologie.
NAT	engl. Network Address Translation ist in Computernetzen ein Verfahren, um eine IP-Adresse in einem Datenpaket durch eine andere zu ersetzen. Häufig wird dies benutzt, um private IP-Adressen auf öffentliche IP-Adressen abzubilden.
NJS	UNICORE Network Job Supervisor (siehe Kapitel 2.1.2)
Nspace	Not UNICORE Space (siehe Kapitel 2.1.3)
Overhead	Mehraufwand in Form von Hilfs- oder Verwaltungsdaten im Rahmen der EDV
Port	Ports sind Adresskomponenten, die in Netzwerkprotokollen eingesetzt werden, um Datenpakete den richtigen Diensten (Protokollen) zuzuordnen. Dieses Konzept ist z.B. in TCP implementiert.
RFC	Die Requests for Comments (zu deutsch etwa Bitten um Kommentare) sind eine Reihe von technischen und organisatorischen Dokumenten des RFC-Editor zum Internet, die am 7. April 1969 begonnen wurden.
RSA	Das RSA-Kryptosystem ist ein asymmetrisches Kryptosystem, d.h., es verwendet verschiedene Schlüssel zum Ver- und Entschlüsseln.
SFTP	SSH File Transfer Protocol (siehe Kapitel 2.2.1)
SOAP	Simple Object Access Protocol ist ein Protokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können. (siehe Kapitel 2.2.1)
spoofing	engl. Manipulation, Verschleierung. Bezeichent verschiedene Täuschungsversuche in Computernetzwerken zur Verschleierung der eigenen Identität.
SSH	Secure shell ist sowohl ein Programm als auch ein Netzwerkprotokoll, mit dessen Hilfe man sich zum Beispiel über das Internet auf einem entfernten Computer einloggen und dort Programme ausführen kann.

SSL	Secure Sockets Layer ist ein Verschlüsselungsprotokoll für Datenübertragungen im Internet.
TCP	Das Transmission Control Protocol ist eine Vereinbarung (Protokoll) darüber, auf welche Art und Weise Daten zwischen Computern ausgetauscht werden sollen.
TSI	UNICORE Target System Interface (siehe Kapitel 2.1.2)
TLS	Transport Layer Security ist ein Verschlüsselungsprotokoll für Datenübertragungen im Internet.
UNICORE	Das (engl.) Uniform Interface to Computing Resources ist ein in Deutschland entwickeltes Grid-System.
UPL	UNICORE Protocol Layer (siehe Kapitel 2.1.2)
Uspace	UNICORE User Space (siehe Kapitel 2.1.3)
Vsite	UNICORE Virtual Site (siehe Kapitel 2.1.2)
WebDAV	engl. Web-based Distributed Authoring and Versioning ist ein offener Standard zur Bereitstellung von Dateien im Internet.
XML	Die Extensible Markup Language ist ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur, der vom World Wide Web Consortium (W3C) definiert wird.
Xspace	UNCORE UNIX Space (siehe Kapitel 2.1.3)
Zertifikat	Zertifikate bestätigen die Zugehörigkeit eines kryptografischen Schlüssels zu einer Person/Firma/Institution oder einer Maschine/Software. Dadurch können Authentizität, Vertraulichkeit und Integrität von Daten gegenüber Dritten garantiert werden.
ZIP	Das ZIP-Dateiformat ist ein offenes Format zur komprimierten Archivierung von Dateien.

Anhang B

Java Servlet

Aus der Quelle „Java Servlets“ [10]:
Das englische Wort `Servlet` wird nicht übersetzt. Es handelt sich hierbei um eine Wortkreation aus den Begriffen „Server“ und „Applet“, also serverseitiges Applet und somit Servlet.

Als Servlets werden Java-Klassen bezeichnet, deren Instanzen innerhalb eines Webservers Anfragen von Klienten entgegen nehmen und beantworten. Solche Klassen müssen immer die Schnittstelle `javax.servlet.Servlet` oder eine davon abgeleitete (normalerweise `javax.servlet.http.HttpServlet` implementieren. Der Inhalt der Antworten kann dabei dynamisch, also im Moment der Anfrage, erstellt werden und muss nicht bereits statisch (etwa in Form einer HTML-Seite) für den Webserver verfügbar sein. Servlets stellen somit das Java-Pendant zu CGI-Skripten oder anderen Konzepten, mit denen dynamisch Web-Inhalte erstellt werden können (z.B. PHP), dar.

Bei Verwendung der Servlet-Spezifikation und einer entsprechenden Webcontainer-Umgebung (zum Beispiel Jakarta Tomcat) besteht die Implementierung einer dynamischen Webseite in folgendem: Es wird eine von javax.servlet.http.HttpServlet abgeleitete Klasse erstellt. Die beiden Methoden `doGet` und `doPost` der Superklasse werden überschrieben um die beiden wichtigsten HTTP-Methoden `GET` und `POST` verarbeiten zu können.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Ralf Ratering
Unicore Pro - Plugin Programmer's Guide
http://unicore.sf.net/docs/plugin_programmers_guide.pdf
(Dez 2005)

[2] Dietmar Erwin (2003)
UNICORE Plus Final Report - Uniform Interface to Computing Resources
ISBN 3-00-011592-7
http://www.unicore.org/documents/UNICOREPlus-Final-Report.pdf
(Dez 2005)

[3] Ian Foster, Carl Kesselman (1998)
The Grid: Blueprint for a New Computing Infrastructure
Morgan Kaufmann Publishers; 1st edition (ISBN 1-558-60-4758)

[4] John Palfreyman (PDF)
IBM - The Basic of Grid
http://searchoracle.techtarget.com/searchOracle/
downloads/foster_ch2.pdf (Dez 2005)

[5] Simone Lanzarini (PDF)
GridFTP vs UPL
http://www.summit.unicore.org/2005/Simone_Lanzarini.pdf
(Dez 2005)

[6] IETF - RFC (1985)
FILE TRANSFER PROTOCOL (FTP)
http://www.ietf.org/rfc/rfc0959.txt

[7] de.wikipedia.org
Secure Sockets Layer
http://de.wikipedia.org/wiki/Secure_Sockets_Layer

[8] de.wikipedia.org
Secure FTP
http://de.wikipedia.org/wiki/Secure_FTP

[9] de.wikipedia.org
FTP über SSL
http://de.wikipedia.org/wiki/FTP_over_SSL

[10] de.wikipedia.org
Java Servlet
http://de.wikipedia.org/wiki/Servlet