INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Managementsystem zur verlässlichen Speicherung VO-kritischer Daten im Grid

Jens Hellwig

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Michael Schiffers

Abgabetermin: 14. August 2007

INSTITUT FÜR INFORMATIK

DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Managementsystem zur verlässlichen Speicherung VO-kritischer Daten im Grid

Jens Hellwig

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Michael Schiffers

Abgabetermin: 14. August 2007

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 14. August 2007

. . . .

(Unterschrift des Kandidaten)

Zusammenfassung

Es wird ein System vollständig entworfen, das Dateien innerhalb von virtuellen Organisationen im Grid verteilt und überwacht. Es orientiert sich an der Open Grid Service Architecture (OGSA) und verwendet offene und standardisierte Protokolle des Globus Toolkit. Funktionalität, die von den Diensten des Globus Toolkit angeboten wird, wird nicht neu implementiert.

Das System garantiert die Verfügbarkeit von wichtigen Daten, auch wenn Datenspeicher unzuverlässig sind oder gar ausfallen. Erreicht wird dies durch ein spezielles Verfahren des Erasure Coding, bei dem Daten schon mit geringer Redundanz rekonstruierbar sind und über eine Vielzahl von Speicherressourcen verteilt werden. Auf Ausfall oder Hinzutreten von Datenspeichern reagiert das System mit einer automatischen Neuverteilung der Daten zur Wiederherstellung der ursprünglichen Redundanz. Derselbe Mechanismus kommt dynamischen virtuellen Organisationen zugute, deren Nutzerzahl und Ressourcen sich häufig ändern. Eine weitere Folge der Kodierung und Datenverteilung ist die Datensicherheit, da Angreifer den Inhalt von Dokumenten nicht entschlüsseln können, so lange sie nicht mindestens so viele verschiedene Datenspeicher kontrollieren, wie Datenblöcke zur Rekonstruktion einer gesamten Datei nötig sind.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In vielen Bereichen der Information Technology beobachtet man in den letzten Jahren Konvergenzbewegungen. Dazu gehört die Abkehr von proprietären Lösungen zugunsten von unabhängigen und offenen Standards. Man möchte die Zahl der verschiedenen Protokolle, Formate, Schnittstellen usw. niedrig halten, verlangt aber dafür größere Flexibilität und Vielseitigkeit. Im Bereich des Distributed Computing ist dabei das Grid Computing ein wichtiges Konzept.

1.1 Grid Computing

Oft wird Grid Computing mit Cluster Computing verwechselt. Zwar ist verteiltes Rechnen, vor allem im wissenschaftlichen Bereich, eines der Haupteinsatzgebiete. Unter Grid Computing versteht man aber nicht die verteilte Anwendung selbst, sondern die Software, die diese Anwendung global verwaltet und koordiniert. Die Managementsysteme für Cluster funktionieren nur, wenn alle Rechenknoten mit der gleichen Hard- und Software konfiguriert sind. Grid Computing dagegen ermöglicht ein einheitliches Management trotz unterschiedlicher Betriebssysteme und Rechnerarchitekturen. Vor allem aber ermöglicht es, die Koordination von Ressourcen organisationsübergreifend und in globalem Maßstab durchzuführen. Dabei geht es nicht nur um die Nutzung von Rechenleistung, sondern auch um Datenspeicher und prinzipiell jede beliebige Ressourcenart. Die Konvergenz besteht darin, dass alle Ressourcen im Grid die gleichen Management- und Zugriffsprotokolle implementieren, d.h. jeder Dienst im Grid lässt sich unabhängig von Betriebssystem, Architektur, Konfiguration und Art des Dienstes auf die gleiche Art und Weise ansprechen. Jeder neue Dienst kann mit den vorhandenen Werkzeugen verwaltet werden.

Der Mehraufwand für die Entwicklung einer geeigneten Infrastruktur lohnt sich. Im Endeffekt wird Arbeit eingespart, weil Anwendungen besser verwaltet und ihre Komponenten wiederverwendet werden können.

1.1.1 Definition

Ian Foster definiert Grid Computing in seinem Artikel „What is the Grid? A Three Point Checklist“[IF02] anhand von drei Eigenschaften:

„A Grid is a system that:

1. *coordinates resources that are not subject to centralized control ...* (A Grid integrates and coordinates resources and users that live within different control domains—for example, the user's desktop vs. central computing; different administrative units of the same company; or different companies; and addresses the issues of security, policy, payment, membership, and so forth that arise in these settings. Otherwise, we are dealing with a local management system.)

2. *... using standard, open, general-purpose protocols and interfaces ...* (A Grid is built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. As I discuss further below, it is important that these protocols and interfaces be standard and open. Otherwise, we are dealing with an application- specific system.)

3. *... to deliver nontrivial qualities of service.* (A Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, and security, and/or co-allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.)“

1.1.2 OGSA

Wenn man vom Grid spricht, meint man oft die Open Grid Services Architecture. Sie verfeinert die Definition von Ian Foster, indem sie genaue Aussagen darüber macht, welche Dienste, Protokolle und Schnittstellen nötig sind und welche Aufgaben sie erfüllen sollten. Man kann die OGSA als einen abstrakten Entwurf für eine Grid Middleware ansehen, der beschreibt, welche grundlegenden Funktionen im Grid erfüllt werden sollten.

Die OGSA beschreibt das Grid anhand von Capabilities [OGSA06]. Jede Capability ist ein funktioneller Bereich und umfasst bestimmte Dienste sowie die Schnittstellen und Protokolle, mit denen sie kommunizieren.

Es werden folgende Capabilities beschrieben:

- **Infrastructure Services.** Die OGSA ist als Service Oriented Architecture (SOA) konzipiert. Die Dienste in der OGSA sind als Web Services zu implementieren. Das allein macht aber noch keine Grid Software aus. Auf der Infrastrukturebene müssen auch alle Mechanismen definiert werden, die sich auf alle Dienste auswirken. Dazu gehören die Namensgebung, Sicherheitsmechanismen, und weitere.
- **Execution Management Services.** Hier geht es um das Management und Scheduling von rechenintensiven Anwendungen, die im Grid ausgeführt werden
- **Data Services.** Die OGSA gibt eine Übersicht über Typen von Datenressourcen. Dazu gehören einfache Dateien, aber auch spezielle Typen, wie Streams, Datenbanken, Kataloge, abgeleitete Daten und Dienste, die Daten erzeugen.
- **Resource Management Services.** Es gibt drei Typen von Management: Management der physikalischen und logischen Ressourcen, d.i. Management der Hardware (z.B. Host neu starten, VLAN eines Switches einstellen), Management der OGSA Grid Ressourcen, d.s. Managementfunktionen, die von den Grid Diensten selbst zur Verfügung gestellt werden (z.B. Monitoring von Jobs), Management der OGSA Grid Infrastruktur, d.s. Managementfunktionen, die von der Infrastruktur vorgegeben sind und allen Grid Diensten gemeinsam sind.
- **Security Services.** Das Grid soll organisationsübergreifend funktionieren. Es ist unerlässlich, Mechanismen und Dienste zur Authentifizierung und Autorisierung zu definieren, die es erlauben, Rechte temporär an andere abzugeben (Delegation) und Policies auf unterschiedlichen Ebenen durchzusetzen.
- **Self Management Services.** Systemkomponenten können ihre Konfiguration selbst anpassen, um Fehlersituationen zu beseitigen, die Performanz zu steigern und den Administrationsaufwand zu reduzieren.
- **Information Services.** Zum Ressourcenmanagement gehört das Sammeln von Daten in den Bereichen Monitoring, Accounting und Indexierung von Anwendungen. Das Ziel der Information Services, ist es, diese Daten zu vereinheitlichen und zu verarbeiten.

1.1.3 Ressourcen

In dieser Arbeit ist eine Ressource eine Entität, mit der ein Nutzer in Verbindung treten kann und die eine bestimmte Aufgabe erfüllt. Meist stellt man sich unter einer Ressource einen Computer mit Internetanschluss vor. Ressourcen können auch zusammengesetzt sein. Zum Beispiel gibt es Grid Dienste, die ganze Cluster als einzelne Ressourcen darstellen. Umgekehrt können auch mehrere Ressourcen innerhalb eines Geräts existieren (siehe Kapitel 3.3). Wichtig ist, dass Ressourcen über einen Grid Dienst erreicht werden können und eine global eindeutige Adresse besitzen. Ressourcen sind immer an reale Geräte gebunden. Der Besitzer wird als Ressourcenanbieter bezeichnet. Er hat immer die Kontrolle über seine Ressourcen und kann Policies gegenüber Benutzern definieren und durchsetzen.

1.1.4 Virtuelle Organisationen

Das Ziel, organisationsübergreifend zu sein, wirft vor allem Probleme bei der Autorisierung auf. Eine Ressource muss entscheiden können, ob sie einem Nutzer Zugang gewähren oder verweigern soll. Ohne zentrales Benutzermanagement muss jeder Ressourcenanbieter für jeden einzelnen Nutzer Rechte vergeben. Es ist

jedoch für den Nutzer aufwändig, bei jedem Anbieter einen Zugang zu beantragen. Außerdem ist es nicht akzeptabel, dass böswillige Nutzer erst mit beträchtlicher Verzögerung und ziemlichem Aufwand von allen Ressourcen ausgeschlossen werden können.

Deshalb verfolgt man das Konzept der virtuellen Organisationen (VO). Eine VO besteht aus Nutzern mehrerer realer Organisationen (RO). Jeder Nutzer kann innerhalb der VO bestimmten Gruppen zugeordnet werden, Rollen einnehmen und Zugriffsrechte erhalten. Die Organisation wird virtuell genannt, da ihre Struktur an keine reale Organisation gebunden ist. Die Zuordnung zu Gruppen, die Vergabe von Rollen und Zugriffsrechten geschieht aus der VO heraus. Das heißt, die VO bestimmt, wer ihre Mitglieder sind, und was sie dürfen.

Anbieter können jetzt Policies gegenüber der VO's als Ganze durchsetzen. Die Ressource braucht nicht mehr jeden einzelnen Nutzer zu kennen, sondern trifft ihre Autorisierungsentscheidung mit Hilfe der VO. Das ist insofern bemerkenswert, als die Kontrolle über die Ressource vollständig beim Anbieter bleibt, die VO aber trotzdem Nutzer hinzufügen und löschen kann, ohne mit dem Anbieter in Kontakt zu treten.

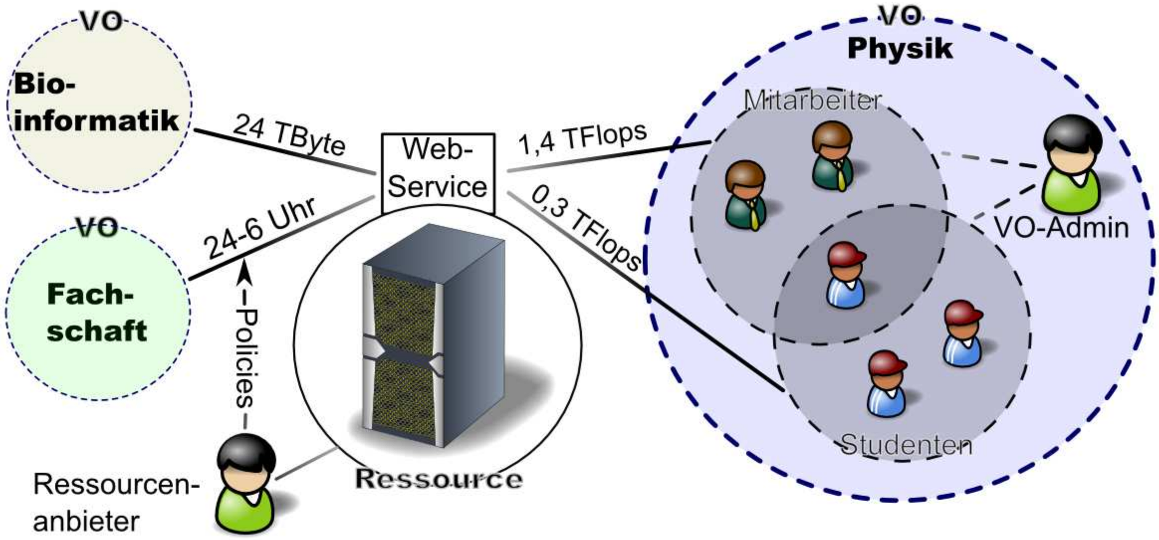

Abbildung 1.1: Beziehung zwischen Ressource und VO

Abbildung 1.1 zeigt beispielhaft das Zusammenspiel zwischen Ressource und VO. Der Ressourcenanbieter legt Policies gegenüber Gruppen innerhalb der VO fest, aber nicht gegenüber einzelner Mitglieder.

1.1.5 Daten in virtuellen Organisationen

Wie jede reale Organisation besitzen auch virtuelle Organisationen vertrauliche Daten, deren Verlust oder Weitergabe sehr negative Auswirkungen hätte. Im Folgenden werde ich diese Daten VO-kritisch nennen. Die Auswahl der Ressourcen zur Datenspeicherung muss in diesem Fall mit besonderer Sorgfalt getroffen werden. Es werden sowohl hohe Anforderungen an die Zuverlässigkeit der Ressource als auch an die Ehrlichkeit und Fachkundigkeit des Administrators der Ressource gestellt.

Erreicht wird diese Zuverlässigkeit bisher meist durch den Einsatz von speziellen professionellen Speicherlösungen (z.B. Storage Area Networks), die als einzelne Ressourcen im Grid freigegeben werden. Dadurch beschränkt man sich auf einige wenige, dafür aber sehr leistungsfähige und leider auch teuere Datenspeicher.

Dieses Vorgehen steht zum Teil im Widerspruch zu den Zielen des Grid Computing. Die Kosten und der Wartungsaufwand sind hoch. Viele vorhandene Ressourcen können nicht verwendet werden, da sie die Anforderungen an die Vertraulichkeit und die Zuverlässigkeit nicht erfüllen.

Ein besonderes Problem der Dateiverwaltung in virtuellen Organisationen ist das Management der Zugriffsrechte auf Dateien und Verzeichnisse. Die VO legt fest, welche Dateizugriffsrechte ihre Mitglieder haben.

Letztlich ist es aber die Ressource, die diese Rechte durchsetzen muss. Ist sie fehlerhaft konfiguriert oder unterstützt sie das VO-Management nicht, können Zugriffsrechte nicht richtig vergeben werden. Im schlimmsten Fall können alle Nutzer einer VO auf alle Daten zugreifen.

Im Folgenden werde ich den Begriff des Datenspeichers in bestimmter Weise verwenden. Als Datenspeicher bezeichne ich Ressourcen, die über einen eigenen physikalischen Speicherplatz verfügen (im Allgemeinen Festplatten) und deren Funktion es ist, Dateien zu speichern.

1.2 Ziel der Arbeit

Das Ziel der Arbeit ist der Entwurf eines OGSA-kompatiblen Systems zur verlässlichen Speicherung von Dateien auf unzuverlässigen Datenspeichern. Das System soll garantieren, dass VO-kritische Daten innerhalb der VO bleiben. Sie sollen weder verloren gehen, noch sollen Personen außerhalb der VO die Daten lesen können. Insbesondere soll das System selbstständig auf Änderungen der VO reagieren. Daten sollen automatisch kopiert, verschoben oder wiederhergestellt werden, falls Datenspeicher ausfallen oder entfernt werden. Datenspeicher sollen automatisch genutzt werden, wenn sie verfügbar werden. Dadurch können dynamische virtuelle Organisationen unterstützt werden, deren Nutzer und Ressourcen sich ständig ändern.

Der Nutzen für die VO ist sowohl eine Minimierung des Verwaltungsaufwandes als auch eine Minimierung der Kosten. Die Flexibilität steigt, da keine Notwendigkeit besteht, Dateien von Hand zu kopieren, und der Nutzer kann unabhängig von der eigentlichen Datenverteilung arbeiten.

Die Machbarkeit und die Architektur werden durch eine prototypische Implementierung überprüft.

1.3 Anforderungen

Im Einzelnen sind folgende Anforderungen an das System vorgegeben:

1. **OGSA.** Das System orientiert sich an der Open Grid Services Architecture. Auf die Security Services wird besonderer Wert gelegt.

2. **Virtuelle Organisationen.** Der Gesamtspeicher präsentiert sich einer VO gegenüber als eine einzige virtuelle Verzeichnisstruktur. Der Benutzer wird mit Hilfe eines VO-Management Dienstes autorisiert und nutzt den Speicher auf die gleiche Weise, wie eine einzelne Ressource.

3. **Sicherheit.** Es soll erreicht werden, dass möglichst nur Mitglieder der VO Zugang zu den Inhalten der Dateien erhalten. Kompromittierte Datenspeicher dürfen dabei nicht die Sicherheit und Funktion des Systems gefährden. Dazu ist eine Datenverteilung nötig, die es auf Ressourcenebene verhindert, Dateien zu entschlüsseln. Zugriffsrechte sollten ausschließlich über ein VO-Managementsystem vergeben werden. (Bemerkung: Der Ressourcenanbieter selbst wird hier als unzuverlässig angesehen und deshalb in seinen Rechten eingeschränkt. Er kann Policies nur gegenüber der VO als Ganze festlegen.)

4. **Zuverlässigkeit.** Die Zuverlässigkeit soll auch dann gewährleistet sein, wenn Datenspeicher ausfallen, beeinträchtigt sind oder entfernt werden. Dafür müssen die Daten redundant gehalten werden. Durch den Einsatz von Algorithmen zur Fehlerkorrektur kann der Mehrverbrauch an Speicherplatz niedrig gehalten werden. Wenn Datenspeicher ausfallen, muss die Redundanz durch die Nutzung weiterer Datenspeicher automatisch wiederhergestellt werden. Datenspeicher, die neu in die VO eingefügt werden, müssen automatisch genutzt werden.

5. **Geringer Verwaltungsaufwand.** Das Hinzufügen und Entfernen von Datenspeichern sollte möglichst geringen Aufwand verursachen. Das System muss eine Strategie zur Verteilung der Daten besitzen, mit der es ohne menschlichen Eingriff auf neue, entfernte oder beeinträchtige Datenspeicher reagieren kann. Es wird nicht gefordert, dass die Datenspeicher VO-Management Dienste unterstützen. Das System stellt wenig Anforderungen an die Schnittstelle zu den Datenspeichern.

2 Verwandte Projekte und weiteres Vorgehen

Unter den vielen Systemen, die zur Speicherung von Daten dienen, wurden die folgenden ausgewählt und kurz beschrieben, weil sie beispielhaft für Konzepte sind, die aus der Sicht der Anforderungen an diese Arbeit interessant sind. Dabei sind die Einsatzbereiche der Projekte durchaus verschieden und nicht alle bewegen sich im Rahmen des Grid Computing.

2.1 Globus Toolkit 4

Das Globus Toolkit 4 [GT4] ist eine weit verbreitete Implementierung der OGSA. Wie wir noch sehen werden, wird sich diese Arbeit stark daran anlehnen (siehe Kapitel 3.4). Globus Toolkit enthält natürlich auch Mittel zur Datenverwaltung. Die Funktionalität ist aber auf mehrere Dienste aufgeteilt. Der Anwender nutzt je nach Aufgabe einen oder mehrere dieser Dienste.

Folgende Dienste sind im Rahmen dieser Arbeit interessant:

- **CAS** (Community Authorization Service). VO-Management Dienst
- **GridFTP** (Schnittstelle zu Datenspeichern). Übertragung von Dateien, Anlegen von Verzeichnissen.
- **RLS** (Replica Location Service). Verzeichnis zur Abbildung logischer Dateinamen auf physikalische Dateinamen, welches ermöglicht, alle Dateien einer VO zu einer Verzeichnisstruktur zusammenzufassen.
- **DRS** (Data Replication Service). Repliziert eine vordefinierte Liste von Dateien.

Mit Hilfe dieser Dienste lassen sich die Dateien einer VO speichern und übertragen. Die Verwaltung der Dateizugriffsrechte kann über den CAS erfolgen. Zuverlässigkeit und ein zentrales Verzeichnis lassen sich über den RLS realisieren. Allerdings ist der Benutzer selbst für die Replikation seiner Dateien, die Registrierung im RLS und für die Konsistenz der Replikate verantwortlich. Die Replikation VO-kritischer Daten bedeutet im Allgemeinen, dass für jede Datei der VO mindestens eine Sicherungskopie angelegt wird. Die Größe des benötigten Speicherplatzes beträgt das Doppelte der gesamten Datenmenge. Bei vollständiger Replikation können einzelne Ausfälle zugelassen werden. Die Datenspeicher müssen vertrauenswürdig sein, da immer vollständige und unverschlüsselte Dateien gespeichert werden. Der Ressourcenanbieter hat die Möglichkeit, sich über die Policies der VO hinwegzusetzen und beliebigen Nutzern Zugriff auf einzelne Dateien zu gewähren. Neue Datenspeicher werden erst dann genutzt, wenn Benutzer ihre Dateien darauf ablegen.

Die oben aufgestellten Anforderungen werden wie folgt erfüllt:

1. **OGSA.** Das Globus Toolkit 4 wurde explizit anhand der OGSA entwickelt.
2. **Virtuelle Organisationen.** Der CAS erfüllt die Anforderungen an die Vergabe von Zugriffsrechten. Eine zentrale Verzeichnisstruktur kann über den RLS gegeben werden.
3. **Sicherheit.** Kann nur durch sichere Datenspeicher erreicht werden.
4. **Zuverlässigkeit.** Wie wir noch sehen werden, gibt es effizientere Verfahren als vollständige Replikation.
5. **Geringer Verwaltungsaufwand.** Die Dienste des Globus Toolkit sind gut aufeinander abgestimmt. Allerdings muss jeder Datenspeicher für das VO-Management konfiguriert werden.

2.2 GPFS

Das General Parallel File System von IBM [GPFS06] ist ein Dateisystem für AIX 5L und Linux Cluster. Es wird auf Grund der guten Leistungsfähigkeit und Skalierbarkeit für viele Supercomputer eingesetzt. Als Cluster-Dateisystem macht GPFS keinen Unterschied zwischen Client und Server. Jeder Knoten, der auf Dateien zugreifen kann, ist selber auch ein Speicherknoten. Interessant sind hier vor allem zwei Features. GPFS kann in neueren Versionen mehrere Cluster über globale Entfernungen verbinden. Außerdem besitzt es einen automatischen Fail Over. Ausgefallene Knoten werden automatisch kompensiert und müssen nicht sofort ersetzt werden.

GPFS ist nach der obigen Definition aber nicht zum Grid Computing zu rechnen. Es ist eine proprietäre Software, die nur für eine begrenzte Zahl von Betriebssystemen und Prozessortypen verfügbar ist. Die Administration kann nur zentral erfolgen. Die eingesetzten Sicherheitsmechanismen sind nicht kompatibel mit Grid Sicherheitsmechanismen.

Die oben aufgestellten Anforderungen werden wie folgt erfüllt:

1. **OGSA.** GPFS ist laut Definiton keine Grid Software.
2. **Virtuelle Organisationen.** Zentrales Benutzermanagement verhindert den Einsatz von VO-Management.
3. **Sicherheit.** Nur wenige Personen haben die Kontrolle über das gesamte System. Die Sicherheit ist sehr hoch.
4. **Zuverlässigkeit.** Der automatische Fail Over sorgt für Zuverlässigkeit.
5. **Geringer Verwaltungsaufwand.** Die Hardware für Speicherknoten kann nicht frei gewählt werden. Datenspeicher können nicht nach Belieben hinzugefügt und entfernt werden.

Insbesondere die Anforderung, Speicherressourcen auf einfache Weise hinzuzufügen, ist nicht erfüllt. Aufgrund des Hauptanwendungsbereiches als Cluster-Dateisystem, erscheint es nicht sinnvoll, GPFS in dieser Arbeit zugrunde zu legen und um die beschriebenen Anforderungen zu erweitern.

2.3 GridBlocks DISK

GridBlocks DISK [PIT06] ist ein System zur Speicherung von Dateien. Haupteinsatzbereich ist die zuverlässige Speicherung von Dateien unter Verwendung von unsicheren Speicherressourcen. Ausgefallene Ressourcen können durch den Einsatz eines fehlerkorrigierenden Erasure Codes (siehe Kapitel 3.5) kompensiert werden. Der Mehrverbrauch an Speicherplatz kann im Vergleich zur vollständigen Replikation gering gehalten werden.

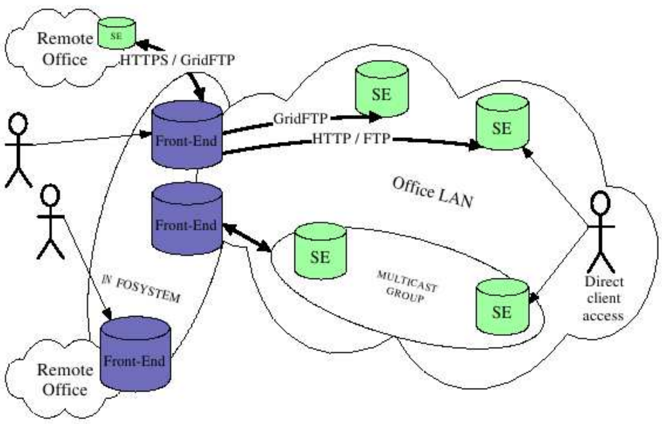

Abbildung 2.1: Architektur von GridBlocks DISK [PIT06] S. 3

Abbildung 2.1 zeigt die Architektur von GridBlocks DISK. Die Datenspeicher werden Storage Elements (SE) genannt. Benutzer greifen über Front Ends auf die Dateien zu. Die Front Ends enthalten einen Metadata Service, der Informationen über alle Dateien verwaltet. Über den Metadata Service kann ein Front End feststellen, welche Storage Elements an der Speicherung einer bestimmten Datei beteiligt sind. Wenn ein Benutzer auf eine Datei zugreift, fordert das Front End die Bestandteile der Datei von den Storage Elements an, setzt diese mit Hilfe des Erasure Codes zusammen und überträgt sie an den Client.

Die Front Ends betreiben einen HTTP-Server über den man auf alle Dateien zugreifen kann. Das kann entweder über einen Web Browser oder über WebDAV geschehen. Den Benutzern bleibt verborgen auf welchen Storage Elements ihre Daten tatsächlich liegen.

Die Schnittstelle zu den Storage Elements ist erweiterbar. Die aktuelle Version unterstützt HTTP, FTP und GridFTP.

Die oben aufgestellten Anforderungen werden wie folgt erfüllt:

1. **OGSA.** Wie der Name schon sagt gibt es eine gewisse Nähe zum Grid Computing. Die aktuelle Version setzt die Konzepte des Grid Computing noch nicht um.
2. **Virtuelle Organisationen.** Es ist anzunehmen, dass zukünftige Versionen virtuelle Organisationen unterstützen werden.
3. **Sicherheit.** Aus der Dokumentation geht nicht hervor, ob die Kenntnis einzelner Dateibestandteile Rückschlüsse auf den Inhalt der Datei geben.
4. **Zuverlässigkeit.** Erasure Codes garantieren gute Ausfallsicherheit.
5. **Geringer Verwaltungsaufwand.** Das Hinzufügen von Storage Elements ist dank der Freiheit des Übertragungsprotokolls sehr gut möglich.

2.4 SRB

Der Storage Resource Broker (SRB) [SRB07] wird meist als globales verteiltes Dateisystem genutzt. Es besteht dabei die Möglichkeit, den SRB und Speicherressourcen mit unterschiedlichen Protokollen anzubinden. Diese Heterogenität wird durch den SRB verborgen, indem der SRB alle Ressourcen zu einer logischen Verzeichnisstruktur zusammenfasst. Dabei können sich Dateien innerhalb eines Verzeichnisses auch auf unterschiedlichen Speicherressourcen befinden. Zusätzlich verfügt er über die Fähigkeit, Metadaten und detaillierte Zugriffsrechte zu den Dateien zu speichern und zu verwalten. Für den Nutzer gibt es grafische Benutzeroberflächen. Für Programmierer gibt es Bibliotheken zum Einbinden in eigene Anwendungen. Um Ausfallsicherheit zu erreichen, kann der Nutzer Dateien vollständig replizieren. Um Sicherheit gegenüber nicht vertrauenswürdigen Speicherressourcen zu erlangen, können Dateien verschlüsselt werden. Der SRB kann so konfiguriert werden, dass die Zugriffskontrolle kompatibel zur Globus Grid Security Infrastrukture ist. Somit erfüllt der SRB die Anforderungen des Grid Computing.

Abbildung 2.2 zeigt die Architektur des SRB. Der SRB-Server vermittelt zwischen Client und Speicherressourcen. Der MCAT ist eine Datenbank und dient der Verwaltung von Metadaten und der Abbildung von logischen in physikalische Dateinamen.

Die oben aufgestellten Anforderungen werden wie folgt erfüllt:

1. **OGSA.** Der SRB orientiert sich nicht an der OGSA, kann aber als Grid Software angesehen werden.
2. **Virtuelle Organisationen.** Virtuelle Organisationen werden nicht unterstützt.
3. **Sicherheit.** Benutzer können ihre Dateien verschlüsselt ablegen.
4. **Zuverlässigkeit.** Der SRB unterstützt Replikation von Dateien. Der Benutzer muss Replikate selbst anlegen und im Falle eines Verlustes wiederherstellen.
5. **Geringer Verwaltungsaufwand.** Das Hinzufügen von Datenressourcen ist dank der Freiheit des Übertragungsprotokolls sehr gut möglich.

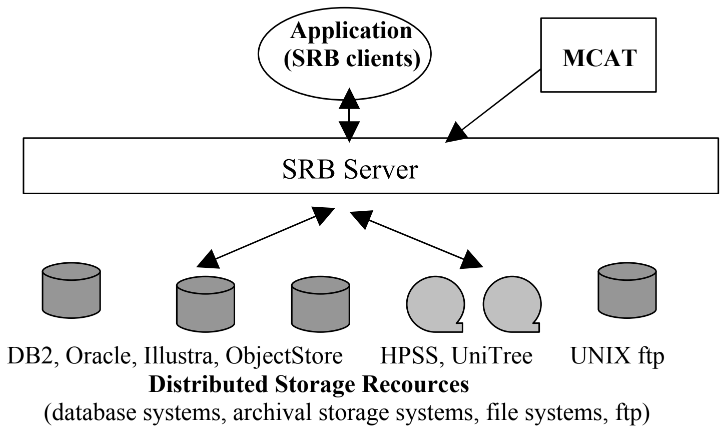

Abbildung 2.2: Architektur des SRB [SRB07]

2.5 Vorgehen

Keines der genannten Projekte erfüllt alle Anforderungen von sich aus. Sowohl der SRB als auch Gridblocks DISK sind erweiterbar und könnten als Grundlage dieser Arbeit dienen. Der SRB ist gut im Bereich des Grid Computing einzusetzen, GridBlocks DISK bietet besonders gute Zuverlässigkeit. Beide unterstützen kein VO-Management.

Ich habe mich jedoch dagegen entschieden, eines der beiden Systeme zu erweitern, da ich einen allgemeingültigen Entwurf erstellen wollte. Ein besonderer Schwerpunkt meiner Arbeit liegt auf der Berücksichtigung von virtuellen Organisationen. Die Verwendung von Software zum VO-Management ist ein wesentlicher Teil der Arbeit.

Um möglichst allgemein und kompatibel zur OGSA zu bleiben, aber dennoch klare Schnittstellen definieren zu können, habe ich mich entschieden, ein System zu entwerfen, das vollständig auf dem Globus Toolkit 4 aufsetzt. Das nächste Kapitel beschreibt die Dienste, die dafür nötig sind. Kapitel 4 bis 6 beschreiben den Prozess der Softwareentwicklung und damit die Architektur und Funktionsweise des Systems.

3 Grundlagen

Bevor der Entwurf beschrieben werden kann, bedarf es eines Überblicks über den Rahmen, in dem sich die Arbeit bewegt. Dazu gehört die Beschreibung der verwendeten Konzepte und der Software.

3.1 Public Key Cryptography

Die Public Key Cryptography basiert auf asymmetrischen Verschlüsselungsverfahren. Jeder Teilnehmer besitzt einen öffentlichen und einen privaten Schlüssel. Der private Schlüssel bleibt immer bei dem Besitzer, der öffentliche Schlüssel wird weitergegeben.

Nachrichten, die mit dem privaten Schlüssel verschlüsselt wurden, können mit dem öffentlichen Schlüssel entschlüsselt werden. Nachrichten, die mit dem öffentlichen Schlüssel verschlüsselt wurden, lassen sich nur mit dem privaten Schlüssel entschlüsseln.

Mit Hilfe der Public Key Cryptography lassen sich zwei Ziele erreichen:

- **Vertraulichkeit.** Nachrichten, die mit dem öffentlichen Schlüssel einer Person verschlüsselt wurden, lassen sich nur von ihr entschlüsseln.

- **Integrität.** Nachrichten, für die sichergestellt werden soll, dass sie nicht verändert wurden, können signiert werden. Eine Signatur wird erzeugt, indem ein Hashwert der gesamten Nachricht gebildet wird und mit dem privaten Schlüssel des Senders verschlüsselt wird. Der verschlüsselte Hashwert wird hinten an die Nachricht angehängt. Der Empfänger entschlüsselt den Hashwert und vergleicht ihn mit der Nachricht. Wenn der Hashwert stimmt, kann die Nachricht nur vom Besitzer des privaten Schlüssel stammen.

3.1.1 Zertifikate

Mit Hilfe von Signaturen können Zertifikate erstellt werden. Ein Zertifikat ist ein Dokument, das garantiert, dass ein öffentlicher Schlüssel zu einem bestimmten Benutzer gehört. Es beinhaltet Informationen zu einer Person und den öffentlichen Schlüssel der Person. Das Zertifikat ist mit einem speziellen privaten Schlüssel signiert. Dieser Schlüssel gehört zu einer besonders vertrauenswürdigen Organisation, deren öffentlicher Schlüssel bekannt ist. Man nennt diese Organisationen *Certificate Authorities*. Jeder Benutzer, der im Grid arbeiten will, benötigt als erstes ein Zertifikat. Im Allgemeinen muss er seine Identität durch persönliches Erscheinen und Ausweisen (zum Beispiel durch seinen Reisepass) gegenüber der Certificate Authority beweisen. Das Zertifikat ist sozusagen das digitale Äquivalent zum Personalausweis. Es besagt, dass die Certificate Authority garantiert, dass der Benutzer tatsächlich der ist, für den er sich ausgibt, und bestätigt seinen öffentlichen Schlüssel.

Mit Hilfe von Zertifikaten kann die Identität eines Senders auch dann sichergestellt werden, wenn sein öffentlicher Schlüssel zunächst nicht bekannt ist. Der Sender signiert seine Nachrichten wie zuvor mit seinem privaten Schlüssel. Zusätzlich schickt er dem Empfänger sein Zertifikat. Der Empfänger kennt den öffentlichen Schlüssel der Certificate Authority und stellt die Gültigkeit des Zertifikats fest. Dadurch hat er den öffentlichen Schlüssel des Senders erhalten und kann sicher sein, dass die Nachrichten unverändert vom Sender stammen. Die Verifikation der Identität anhand von Zertifikaten nennt man *Authentifizierung*.

3.1.2 Delegation

Zertifikate müssen nicht unbedingt von einer Certificate Authority signiert werden. Proxy Zertifikate sind von Benutzern signiert. Anhand von Proxy Zertifikaten können Rechte delegiert werden. Das lässt sich am besten anhand eines Beispiels verdeutlichen: Angenommen Bob möchte eine Aktivität im Namen von Alice durchführen. Er generiert dafür einen privaten und einen öffentlichen Schlüssel. Den öffentlichen Schlüssel überträgt er an Alice. Sie erstellt daraus ein Proxy Zertifikat, das aussagt, dass der Besitzer des Proxy Zertifikats in ihrem Namen handeln darf. Sie signiert das Proxy Zertifikat mit ihrem eigenen privaten Schlüssel. Bob kann das Proxy Zertifikat benutzen, weil er den zugehörigen privaten Schlüssel besitzt. Im Allgemeinen haben Proxy Zertifikate begrenzte Lebensdauer. Das heißt Bob kann nur für begrenzte Zeit im Namen von Alice arbeiten.

3.1.3 Single Sign On

In der Praxis sind private Schlüssel meist durch ein Passwort gesichert und sollten nicht in unverschlüsselter Form gespeichert werden. Die Folge ist, dass jeder Verbindungsaufbau mit Hilfe des Benutzerzertifikats die Eingabe des Passwortes erfordert. So wie man Rechte an andere delegieren kann, kann man auch Rechte an sich selbst delegieren. Das mag zunächst seltsam erscheinen. Wegen der begrenzten Lebensdauer des Proxy-Zertifikats kann aber die Speicherung eines ungesicherten privaten Schlüssels in Kauf genommen werden. Der Benutzer meldet sich genau einmal mit seinem Passwort an. Alle nachfolgenden Aktivitäten verwenden das dabei generierte Proxy-Zertifikat.

3.2 Web Services

Das Grid Computing wird oft als Service Oriented Archtecture (SOA) angesehen. Verteilte Anwendungen bestehen aus mehreren Diensten, die miteinander kommunizieren. Im Grid ist es wichtig, offene Standarts für diese Schnittstellen einzuführen. Web Services stellen eine Möglichkeit dar, alle Schnittstellen auf eine einheitliche Basis zu stellen.

Web Services sind nachrichtenorientiert. Nachrichten bestehen aus speziellen XML-Dokumenten, die SOAP-Nachrichten genannt werden. Im Normalfall werden die Nachrichten über das HTTP-Protokoll übertragen. (Andere Protokolle wie z.B. smtp sind möglich, aber nicht gebräuchlich.) SOAP implementiert das Remote Procedure Call Pattern (RPC). Eine SOAP Engine übernimmt die Serialisierung und Deserialisierung von Objekten in Form von XML basierten SOAP-Nachrichten. Für den Entwickler präsentiert sich die Schnittstelle in Form von Client und Server Stubs. Das sind Objekte, die wie lokale Objekte angesprochen werden, deren Methoden aber entfernt ausgeführt werden. Wegen der Transformation in XML und zurück sind Web Services nicht an eine bestimmte Programmiersprache oder Architektur gebunden.

Die besondere Eigenschaft von Web Services ist ihre Fähigkeit, sich selbst zu beschreiben. Die Funktionalität eines Dienstes wird anhand der Web Services Definiton Language (WSDL) definiert. Ein WSDL-Dokument beschreibt die Nachrichten, die ein Dienst empfangen und senden kann, und die Methoden, die er zur Verfügung stellt. Ein Client kann einen Dienst nach seiner WSDL-Definition fragen und erkennen, ob der Dienst die gewünschten Methoden implementiert.

Die Kombination aller Softwarekomponenten, die zum Betrieb von Web Services gebraucht werden, nenn man Web Services Container. Er besteht aus der SOAP Engine einem Application Server und einem HTTP Server. Der Web Services Container kann mehrere Web Services verschiedener Art verwalten und ausführen.

3.3 WSRF

Web Services machen zwar Aussagen über die Art eines Dienstes, aber nicht über den Zustand. Aufeinander folgende Aufrufe durch den Client haben immer das gleiche Ergebnis, weil der Web Service keine Informationen über den vorherigen Aufruf speichert. Das Web Service Resource Framework (WSRF) erweitert Web Services um Zustandsinformationen.

Da Web Services selbst immer als zustandslos angesehen werden, führt man den Begriff der WS-Resource ein. In diesem Sinne ist eine WS-Resource ein Zustand eines Web Service. Der Web Service und die WS-Resource sind getrennte Einheiten. Das heißt, ein Web Service kann Zugang zu mehreren WS-Resources geben. Das geschieht anhand von speziellen Adressen, die Endpoint Reference (EPR) genannt werden. Clients können die EPR nutzen um auf bestimmte WS-Resources zuzugreifen.

Der Zustand einer WS-Resource wird in Form von WS-Resource Properties gespeichert, die im WSDL Dokument definiert werden. Es gibt Methoden, mit denen der Client die WS-Resource Properties auslesen und verändern kann.

WS-Notification ist ein mit WSRF eng verwandter Standard und wird in dieser Arbeit später noch benötigt. Es gibt Anwendungen, die fortlaufend Informationen über den Zustand benötigen. Eine einfache Lösung wäre es, die WS-Resource Properties in kurzen Zeitabständen immer wieder abzufragen. Mit Hilfe der WS-Notification kann unnötiger Netzverkehr verhindert werden, indem der Dienst den Client erst bei Änderung seines Zustandes informiert. Das geht jedoch nur, wenn der Client selbst einen WSRF Dienst hat, über den er WS-Notifications empfangen kann. Beide Programme müssen sowohl über eine Server als auch über eine Client Schnittstelle verfügen.

3.4 Globus Toolkit

Das Globus Toolkit enthält alle Werkzeuge und Dienste, um Grid Systeme zu entwickeln und zu betreiben. Es implementiert nicht nur die Capabilities der OGSA, sondern beinhaltet auch Libraries und Skripts, die den Programmierer bei der Erstellung, Compilierung und dem Deployment eigener Grid Software unterstützen. Es gibt Dienste zum Monitoring, Execution Management, Datenmanagement und VO-Management. Die umfangreichen Libraries enthalten Funktionen für Sicherheitsmechanismen, die Bereitstellung oder den Zugriff auf Dienste, Input und Output in oder aus Datenströmen, um nur einige zu nennen.

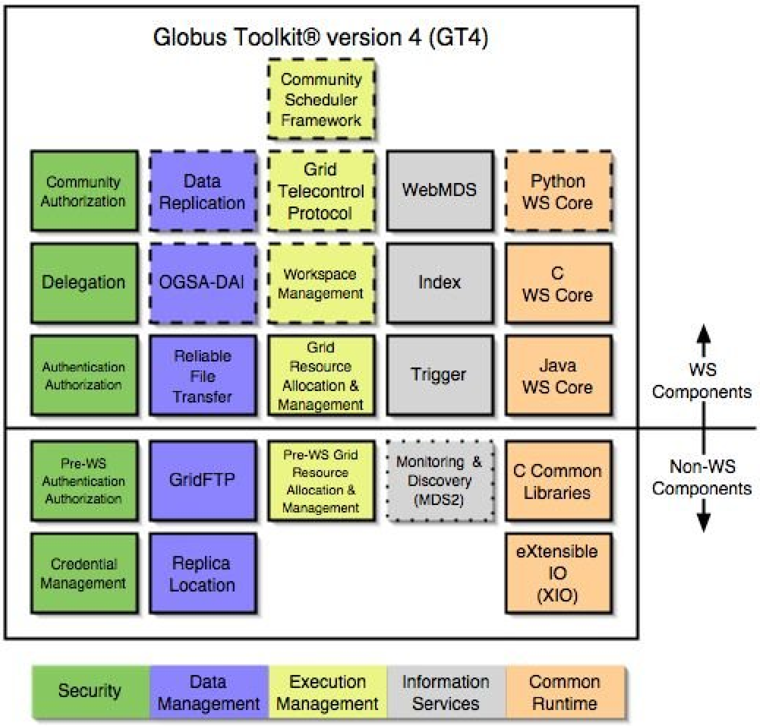

Abbildung 3.1: Komponenten des Globus Toolkit [GT4]

3.4.1 Überblick

Das Globus Toolkit besteht im wesentlichen aus fünf Bereichen, zu denen jeweils Libraries und Dienste gehören. Abbildung 3.1 gibt einen Überblick über die wesentlichen Komponenten des Globus Toolkit.

Die folgenden Abschnitte beschreiben die Bereiche *Common Runtime*, *Security*, *Date Management* und *Information Services*. Der Bereich des *Execution Management* ist für diese Arbeit nicht relevant.

3.4.2 Common Runtime

Die *Common Runtime* ist gewissermaßen das Framework des Globus Toolkit. Sie bildet die Grundlage für alle Dienste, die im Globus Toolkit enthalten sind, und für alle Dienste, die für das Globus Toolkit entwickelt werden.

Sie umfasst:

- Web Services Container für Dienste, die in Java, C oder Python entwickelt werden.
- Web Services Clients für Java, C und Python.
- Libraries, die den Programmierer bei typischen Aufgaben der Sofwareentwicklung im Grid unterstützen.
- Tools, mit deren Hilfe Dienste auf möglichst einfache Art und Weise erstellt und in laufende Grid-Systeme eingebunden werden können.

3.4.3 Grid Security Infrastructure

Die Grid Security Infrastructure (GSI) nimmt eine Sonderstellung im Globus Toolkit ein. Alle Dienste, die im Rahmen des Globus Toolkit betrieben werden, sind verpflichtet, die GSI zu unterstützen. Dadurch übernimmt die GSI eine zentrale Aufgabe und alle Komponenten sind von ihr abhängig. Ihre Hauptaufgaben sind die Überprüfung der Identität (Authentifizierung), die Sicherung der Kommunikation (Integrität, Vertraulichkeit) und die Zugriffskontrolle (Autorisierung).

Authentifizierung. Die Grid Security Infrastructure basiert auf Public Key Cryptography. Sie unterstützt die Authentifizierung mit Hilfe von X.509 Zertifikaten und sowohl den Single Sign On als auch Delegation.

Sicherung der Kommunikation. Die Grid Security Infrastructure unterscheidet zwischen Message Level Security und Transport Level Security. Transport Level Security wird durch TLS erreicht und sichert die gesamte Kommunikation. Sie ist etwas performanter als die Message Level Security aber nicht so flexibel. Die Message Level Security verschlüsselt nur die SOAP Messages, nicht aber den HTTP Verkehr. Nur die Message Level Security unterstützt die Delegation von Rechten.

Autorisierung. Die Entscheidung, ob ein Benutzer eine bestimmte Operation durchführen darf, wird durch den Server gefällt. Der Ressourcenanbieter legt fest, wie diese Autorisierung durchgeführt wird. In Grid-Systemen ohne VO-Management besitzt jede Ressource eine Liste der berechtigten Benutzer. Die Server des Globus Toolkit speichern diese Liste in der *gridmap*-Datei. Sie befindet sich in */etc/grid-security/grid-mapfile*. In großen Grid-Umgebungen ist es schwierig, die *gridmap*-Dateien auf allen Ressourcen aktuell zu halten. Man verwendet daher VO-Management-Systeme.

3.4.4 CAS

Der Community Authorisation Service ist ein VO-Management-Dienst. Er erlaubt es Benutzergruppen, Service Typen und Objekte zu definieren. Die Ressourcen können dann anhand dieser VO-Informationen Policies durchsetzen. Wenn eine Ressource zu einer VO hinzugefügt wird, handelt die VO mit dem Ressourcenanbieter Policies aus. Diese beziehen sich nicht auf einzelne Benutzer, sondern auf die Benutzergruppen in der Datenbank des CAS. Neue Benutzer müssen nicht in der Ressource registriert werden. Änderungen an der Struktur

der VO, z.B. neue Benutzergruppen, müssen jedoch den Ressourcen mitgeteilt werden, um entsprechende Policies zu setzen.

Jede VO benötigt ein Zertifikat, das die VO repräsentiert. Eine VO verwendet dieses Zertifikat um einen CAS Server zu betreiben.

Ressourcenanbieter, die der VO Rechte einräumen wollen, tragen das Zertifikat der VO als vertrauenswürdig ein. Der Ressourcenanbieter legt zunächst also Policies gegenüber der VO als Ganzes fest und ordnet die ganze VO einem einzigen lokalen Benutzer zu.

Die Datenbank des CAS enthält Informationen über detaillierte Zugriffsrechte der VO-Mitglieder. Darin werden nicht nur Rechte auf Dienste als ganze, sondern auf einzelne Objekte innerhalb der Ressourcen verwaltet. Dadurch ist es insbesondere möglich, Dateien und Ordner als Objekte im CAS einzutragen und Dateizugriffsrechte zu vergeben. Natürlich verwaltet der CAS auch die Zugriffsrechte auf sich selbst. Dadurch kann man Anwendern die Möglichkeit geben, ihrerseits Rechte zu vergeben. So kann der Besitzer einer Datei selbst festlegen, welche Benutzer und Gruppen innerhalb der VO darauf zugreifen dürfen.

Will ein Benutzer auf eine Ressource zugreifen, die vom CAS verwaltet wird, macht er zunächst eine Anfrage an den CAS. Falls in der Datenbank die nötigen Zugriffsrechte verzeichnet sind, erstellt der CAS ein spezielles Proxy-Zertifikat. Dieses funktioniert genauso wie ein normales Proxy-Zertifikat. Es enthält aber zusätzlich Informationen über die Zugriffsrechte, die der Benutzer von der VO erhalten hat. Die Vertrauenswürdigkeit dieser Informationen wird dadurch sichergestellt, dass das Zertifikat vom CAS signiert ist.

Anhand der CAS-Informationen im Proxy-Zertifikat und eigener Policies kann die Ressource entscheiden, ob dem Benutzer Zugriff gewährt wird, oder nicht. Die VO kann an ihre Mitglieder detaillierte Zugriffsrechte vergeben. Trotzdem bleibt die Kontrolle letztlich vollständig beim Ressourcenanbieter.

3.4.5 Information Services

Eine VO benötigt nicht nur Informationen über ihre Mitglieder, wie sie der CAS verwalten kann, sie muss auch wissen, welche Ressourcen sie verwenden kann. Das Monitoring and Discovery System (MDS) gehört zu den Information Services des Globus Toolkit und hat die Aufgabe, Ressourcen zu entdecken und zu überwachen. Es gibt Auskunft über die Ressourcen einer virtuellen Organisation.

Das MDS besteht aus drei Diensten zur Überwachung von Ressourcen (ein vierter ist in Entwicklung).

Index Service. Der Index Service gibt Auskunft über die WS-Resource Properties (vgl. WSRF) von Ressourcen, die im Index registriert sind. Ein Index Service kann wiederum in einem anderen Index registriert sein. Dadurch entsteht eine Hierarchie. Der oberste Index enthält Informationen über alle Ressourcen einer virtuellen Organisation. Man kann zum Beispiel den freien Speicherplatz von Datenressourcen als WS-Resource Property darstellen. Mitglieder einer VO können über den Index Service herausfinden, welche Datenressourcen es gibt und über wieviel Speicherplatz sie verfügen. Der Index Service unterstützt WS-Notifications. Dienste können über Änderungen der Ressourcen einer VO informiert werden und darauf reagieren.

Trigger Service. Der Trigger Service sammelt die gleichen Informationen wie der Index Service. Seine Aufgabe ist es, auf Bedingungen, die in einer Konfigurationsdatei festgelegt werden, zu reagieren und bestimmte Aktionen auszuführen. Zum Beispiel könnte er dem VO-Administrator eine E-Mail schicken, wenn der Speicherplatz knapp wird.

3.4.6 GridFTP

Das Grid File Transfer Protocol dient, wie der Name sagt, zur Dateiübertragung. Da es besonders auf Geschwindigkeit ankommt, ist GridFTP nicht als Web Service implementiert, sondern basiert auf einer Erweiterung des bekannten FTP-Protokolls. Die Entscheidung für das FTP-Protokoll wurde vor allem auf Grund der großen Verbreitung und der guten Erweiterbarkeit getroffen. So ist GridFTP auf und abwärts kompatibel zu FTP. Wie FTP trennt GridFTP die Übertragung in Kontroll- und Datenkanal. Über GridFTP kann man jedoch mehrere Datenkanäle öffnen und Datenkanäle können auch zwischen zwei Servern aufgebaut werden (siehe unten).

GridFTP beinhaltet gegenüber FTP unter anderem folgende Erweiterungen:

- **GSI.** Wie alle anderen Dienste arbeitet GridFTP mit der Grid Security Infrastructure zusammen. Auf Grund der Kompatibilität zu FTP kann die Authentisierung zwar mit Benutzername und Passwort stattfinden. Davon wird jedoch dringend abgeraten. Normalerweise werden die Zertifikate der GSI verwendet. Die Kommunikation kann durch Transport Level Security gesichert werden. Standardmäßig geschieht dies jedoch aus Performanzgründen nur auf dem Kontrollkanal.

- **CAS.** Die Globus Implementierung des GridFTP Servers besitzt ein Modul zur Unterstützung des CAS. Die VO muss die Verwendung des CAS mit dem Ressourcenanbieter aushandeln, da das Modul standardmäßig nicht aktiviert ist.

- **Third-party transfers.** Ein Client braucht selbst nicht am Datentransfer beteiligt zu sein. Es kommt häufig vor, dass ein Client zwischen zwei GridFTP Servern vermittelt, indem er zu beiden einen Kontrollkanal aufbaut. Der Datenkanal wird zwischen den Servern aufgebaut.

- **Partial File Access.** Es kann auf beliebige Teile einer Datei zugegriffen werden, indem Offset und Länge des gewünschten Blocks angegeben werden.

- **Reliability/Restart.** Abgebrochene Transfers können fortgesetzt werden.

- **Data Channel Reuse.** Datenkanäle können für nachfolgende Befehle offen gehalten werden, wenn die Quelle, Ziel und das Zertifikat gleich bleiben.

- **Parallel Transfers.** Performanzsteigerung durch mehrere gleichzeitige TCP-Streams zwischen zwei Hosts.

3.5 Erasure Coding

In der Einleitung wurde erwähnt, dass Gridblocks DISK (Kapitel 2.3) Erasure Coding verwendet, um Zuverlässigkeit ohne vollständige Replikation zu gewährleisten. Erasure Coding ist ein Verfahren der Fehlerkorrektur. Genauer gesagt, handelt es sich um eine Gruppe von Algorithmen aus dem Bereich Forward Error Correction (FEC).

FEC Codes dienen zur Sicherung von Datenübertragung oder Datenspeicherung. Sie berücksichtigen dabei die Eigenschaften des entsprechenden Mediums. Die meisten unzuverlässigen Medien verursachen einzelne Bitfehler oder kurze Ausfälle. Etablierte FEC Algorithmen wie z.B. der Reed-Solomon Code sind gut dafür geeignet, kurze fehlerhafte Bitfolgen zu korrigieren. Durch Umordnung der Daten kann man den Reed-Solomon Code auch einsetzen, wenn große zusammenhängende Blöcke ausfallen. Es gibt jedoch Codes, die hier besser geeignet sind.

Die Erasure Codes sichern einen Kommunikationskanal gegen Auslöschung von ganzen Nachrichten. Ein Erasure Code transformiert eine Anzahl von n ursprünglichen Blöcken in eine Anzahl von m kodierten Blöcken (der gleichen Größe). Sie besitzen die erstaunliche Eigenschaft die ursprünglichen Blöcke aus beliebigen n dieser m kodierten Blöcke wiederherzustellen. Es können also beliebige $m - n$ Blöcke ausgelöscht werden, ohne dass die Nachricht zerstört wird. Die Anzahl m der kodierten Blöcke ist frei wählbar und wird je nach Anwendung festgelegt. Typischerweise wird etwa der anderthalbfache Wert von n verwendet. Dadurch vergrößert sich die Gesamtmenge der Daten um 50%. Ich werde den Begriff der Redundanz verwenden um die prozentuale Menge der kodierten Daten im Vergleich zu den Originaldaten anzugeben. Obiges Beispiel ergibt eine Redundanz von 150%. Fällt die Redundanz unter 100% können die Originaldaten nicht mehr rekonstruiert werden. Ein Wert von 0% bedeutet, dass überhaupt keine Daten vorhanden sind.

Der bekannteste und auch einfachste Erasure Code kommt bei der Konfiguration von Festplatten als Raid 5 zum Einsatz. Dem interessierten Leser ist Raid sicherlich geläufig. Ich möchte hier jedoch zum weiteren Verständnis kurz auf die Codierung eingehen.

Raid 5 benutzt Paritätsbits als FEC Code. Zu jeweils n Bits wird ein Paritätsbit berechnet:

$$p = a_1 \otimes a_2 \otimes \cdots \otimes a_n$$

Besitzt man n dieser $n+1$ Bits, kann man die ursprünglichen n Bits rekonstruieren. Ist z.B. Bit i ausgefallen, kann es folgendermaßen berechnet werden:

$$a_i = p \otimes a_1 \otimes \cdots \otimes a_{i-1} \otimes a_{i+1} \otimes \cdots \otimes a_n$$

Raid 5 verwendet *Block Striping with Distributed Parity*. Damit soll ausgedrückt werden, dass die Daten zunächst in Blöcke von einigen KByte Größe aufgeteilt werden. Anschließend wird zu jeweils n Blöcken ein Paritätsblock berechnet. Zur Steigerung der Schreibgeschwindigkeit wird dieser Paritätsblock nicht immer auf dieselbe Festplatte geschrieben, sondern im Wechsel über alle Laufwerke hinweg.

3.6 Linearer Block Code über einem endlichen Körper

Raid 5 gehört zur Klasse der Linearen Block Codes. Ihnen ist gemeinsam, dass sie durch eine Generatormatrix über einem Vektorraum erzeugt werden. Die verwendeten Vektorräume sind oft komplex (siehe [JAK04]). Ich möchte hier einen relativ einfachen selbstentwickelten Code vorstellen, zu dessen Verständnis Grundlagen der linearen Algebra ausreichen.

3.6.1 Idee

Wenn man eine Matrix $A \in \mathbb{R}^{m \times n}$ mit einem Vektor $\vec{x} \in \mathbb{R}^n$ multipliziert, erhält man einen Vektor $\vec{y} \in \mathbb{R}^m$. Wenn $m > n$ ist, dann hat $\vec{y}$ eine höhere Dimension als $\vec{x}$. Wenn man Vektoren als Speicher für Zahlen ansieht, dann kann man sagen, $\vec{y}$ speichert mehr Zahlen als $\vec{x}$.

Wenn die Matrix A spezielle Bedingungen erfüllt, dann ist diese Transformation umkehrbar. Dazu ist nicht der vollständige Vektor $\vec{y}$ nötig, sondern es reicht bereits eine Untermenge der Skalare, aus denen der Vektor besteht. Im Folgenden wird beschrieben, unter welchen Bedingungen die Rücktransformation geschehen kann, und wie man daraus einen Erasure Code entwickeln kann.

Sei $A \in \mathbb{R}^{m \times n}$ eine Matrix mit mehr Zeilen- als Spaltenvektoren ($m > n$). Ferner fordern wir, dass beliebige n Zeilenvektoren aus A linear unabhängig sind.

Genauer gesagt:

Sei $f(i)$ eine Folge der Länge n, in der alle Werte zwischen 1 und m liegen und keine Zahl doppelt vorkommt. Das heißt:

$$für\ alle\ i \in \{1,..,n\}\ gilt: \quad f(i) \in \{1,..,m\}\ und\ f(j) \neq f(i)\ gdw.\ j \neq i \tag{3.1}$$

Angenommen, die quadratische Matrix $B \in \mathbb{R}^{n \times n}$ wird durch n der m Zeilenvektoren aus A gebildet. Dann sind die Zeilenvektoren von B laut Forderung linear unabhängig und B ist invertierbar.

$$\begin{matrix} B \in \mathbb{R}^{n \times n} \\ B_{i,j} = A_{f(i),j} \end{matrix} \Rightarrow\ es\ existiert\ B^{-1}\ und\ es\ gilt:\ B * B^{-1} = \begin{pmatrix} 10\cdots0 \\ 01\cdots0 \\ \vdots\vdots\ddots\vdots \\ 00\cdots1 \end{pmatrix} \tag{3.2}$$

Wir haben jetzt die Möglichkeit, aus der Transformation $\vec{x}$ nach $\vec{y}$ die Rücktransformation abzuleiten.

$$y_i = \sum_{j=1}^{n} A_{i,j} * x_j \tag{3.3}$$

Durch Ersetzung von i durch $f(i)$ erhält man:

$$y_{f(i)} = \sum_{j=1}^{n} A_{f(i),j} * x_j \tag{3.4}$$

Das entspricht laut Definition von B:

$$y_{f(i)} = \sum_{j=1}^{n} B_{i,j} * x_j \tag{3.5}$$

Da B invertierbar ist, lässt sich die Gleichung umkehren:

$$x_i = \sum_{j=1}^{n} B_{i,j}^{-1} * y_{g(j)} \tag{3.6}$$

Das heißt, aus dem Vektor $\vec{y}$ können beliebige Skalare gelöscht werden. So lange der entstehende Vektor mindestens die Dimension n hat, kann trotzdem die Rücktransformation nach $\vec{x}$ stattfinden. Für die Bildung der Matrix zur Rücktransformation B sind genau die jenigen Zeilen der Kodiermatrix A nötig, die den verbleibenden Skalaren aus $\vec{y}$ entsprechen. Die Indizes der verbleibenden Skalare müssen also bekannt sein.

Die Skalare der Vektoren $\vec{x}$ und $\vec{y}$ können als Nachrichten angesehen werden. Es handelt sich also um einen Erasure Code (mit sehr kurzen Nachrichten). Dieser Ansatz hat jedoch einen Fehler: Die Nachrichten bestehen aus reellen Zahlen.

Invertiert man eine Matrix anhand des Gauß-Verfahrens, kommen die Grundrechenarten (Addition, Subtraktion, Multiplikation, Division) zum Einsatz. Es handelt sich dabei um die sogenannte Körperarithmetik. Ein Körper ist eine Algebraische Struktur in der die vier Grundrechenarten und die gewohnten Rechenregeln gelten. Die bekanntesten Körper sind auf den unendlichen Mengen der rationalen und reellen Zahlen definiert. Computer können Zahlen eines unendlichen Wertebereichs jedoch nicht darstellen (sonst würden sie einen unendlich großen Arbeitsspeicher benötigen). Fließkommazahlen sind nur eine Annäherung und führen hier zu Rundungsfehlern. Das kommt daher, dass sich Matrizen nicht bruchfrei invertieren lassen. Die Brüche lassen sich nicht genau speichern. Bei der Rücktransformation erhält man Werte, die geringfügig, aber signifikant abweichen.

3.6.2 Restklassenring modulo einer Primzahl

Will man eine Matrix verlustfrei invertieren, muss man in einem endlichen Körper rechnen. Endliche Körper lassen sich auf Grund des begrenzten Wertebereichs maschinell und ohne Rundungsfehler verarbeiten. Im Folgenden stelle ich einen endlichen Körper vor und zeige, wie man ihn als Erasure Code einsetzen kann.

Ein Restklassenring modulo einer Zahl p ist eine Struktur, in der Zahlen nur die Werte zwischen 0 und $p-1$ annehmen können. Die Addition, Subtraktion und Multiplikation geschieht analog zur gewohnten Ganzzahlarithmetik. Wenn eine Operation zum Verlassen des erlaubten Wertebereich führen würde, wird das Ergebnis modulo der Zahl p genommen.

$$\begin{aligned} a + b &:= (a + b)\, mod\, p \\ a - b &:= (a - b)\, mod\, p \\ a * b &:= (a * b)\, mod\, p \end{aligned} \tag{3.7}$$

Genau dann, wenn p eine Primzahl ist, kann die Division definiert werden und der Restklassenring ist ein Körper. Der kleine fermatsche Satz sagt aus, dass im Restklassenring modulo der Primzahl p: $a^{p-1} mod\, p = 1$.

Daraus folgt, dass es für jede Zahl ausser 0 ein multiplikativ Inverses gibt und der Restklassenring ein endlicher Körper ist:

$$\begin{aligned} &Sei\ \ a \neq 0\, und\, a' = a^{p-2}\ dann\ gilt: \\ &a * a' = 1 \end{aligned} \tag{3.8}$$

Die Division kann also geschrieben werden als:

$$a/b = a * b' = a * b^{p-2} \tag{3.9}$$

Für das Gauß-Verfahren bedeutet das, dass die Inverse nur aus ganzen Zahlen im Wertebereich zwischen 0 und $p-1$ bestehen kann. Es kann keine Rundungsfehler geben.

Bis jetzt können wir nur Vektoren transformieren und rücktransformieren. Wir möchten jedoch Blöcke aus beliebigen Daten kodieren. Jeder Datenblock lässt sich als eine Liste von Zahlen im Wertebereich einer beliebigen Primzahl darstellen. Aus n Datenblöcken erhält man n Zahlenlisten. Diese kann man als Liste von Vektoren der Dimension n ansehen. Jedes Skalar eines Vektors kommt aus einer anderen Liste. Wenn man alle Ursprungsvektoren aus der Liste mit der Kodiermatrix multipliziert, erhält man eine Liste mit Vektoren der Dimension m die man als m Listen von Zahlen ansehen kann. Daraus erhält man m Datenblöcke zur Weiterverarbeitung. Die ursprünglichen n Datenblöcke können aus n der m berechneten Datenblöcke rekonstruiert werden.

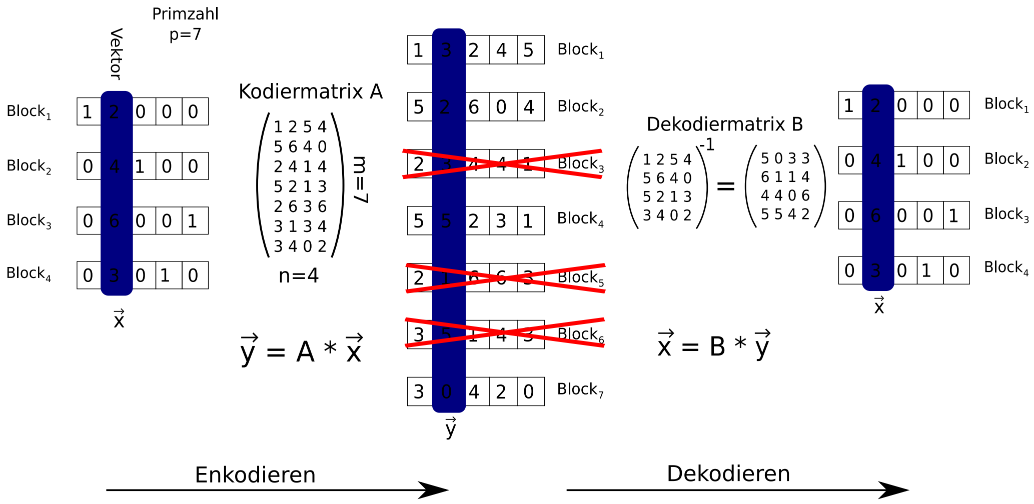

Abbildung 3.2: Kodierung von Blöcken

Abbildung 3.2 zeigt exemplarisch den Kodier- und Dekodiervorgang von Blöcken. In diesem Beispiel sind nach dem Enkodieren die Blöcke *3, 5 und 6* ausgefallen. Die Dekodiermatrix ist die Inverse der Zeilen *1, 2, 4 und 7* aus der Kodiermatrix.

3.6.3 Anwendung

Allein auf Grund der Fähigkeit, Blöcke fester Länge zu kodieren und zu dekodieren, können wir noch keine Dateien variabler Länge auf Datenspeicher verteilen. Um dieses Ziel zu erreichen, werden die kodierten Blöcke gruppiert. Eine Datei, die in $n*q$ Blöcke aufgeteilt wurde (q hängt dabei von der Dateilänge ab), wird in $m*q$ Blöcke transformiert. Diese werden anhand ihres Index gruppiert. Somit ergeben sich m Blockgruppen, die jeweils aus q Blöcken bestehen. Jede Blockgruppe erhält einen Dateinamen und den Index ihrer Blöcke. Um die komplette Datei wiederherzustellen, benötigt man genau n Blockgruppen.

Durch Weglassen einzelner Zeilen der Kodiermatrix vor dem Kodieren können bei Bedarf auch weniger als m Blockgruppen erstellt werden.

4 Anforderungsanalyse

Die Anforderungsanalyse verfeinert die Anforderungen an das Gesamtsystem, die bereits in der Einleitung formuliert wurden.

4.1 Vorgehensweise

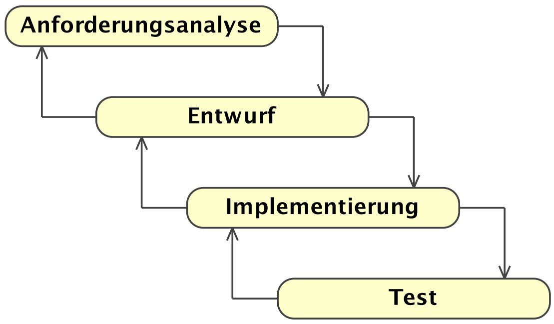

Abbildung 4.1: Wasserfallmodell

Die folgenden Kapitel (Anforderungsanalyse, Entwurf, Implementierung, Leistungsbewertung) folgen dem erweiterten Wasserfallmodell (Abbildung 4.1). Sie dienen nicht nur zur Dokumentation der Arbeit, sondern sind selbst Teil des Vorgehens. Jedes Kapitel stellt einen Schritt in der Softwareentwicklung dar. Die Schritte werden jedoch nicht als abgeschlossen betrachtet. Rückschritte sind erlaubt und durchaus erwünscht. So hat jeder Schritt nicht nur Einfluss auf den folgenden, sondern auch auf den vorausgehenden.

Die folgenden Kapitel beschreiben diesen Prozess. Anhand von Beispielen wird zunächst die grobe Architektur festgelegt. Das heißt, es werden alle Komponenten des Systems identifiziert und es wird festgelegt, welche Aufgaben sie übernehmen. Der Entwurf verfeinert dann noch einmal die Anforderungsanalyse. Er legt fest, wie die an die einzelnen Komponenten gestellten Anforderungen erfüllt werden können.

4.2 Use Case Analyse

Die Use Case Analyse ([UML04] S.175) dient dazu, das externe Verhalten eines Systems aus der Sicht der Nutzer zu beschreiben. Ein Use Case ist ein Anwendungsfall in Gestalt eines typischen Szenarios, das bei der Nutzung des Systems vorkommt.

An jedem dieser Szenarien sind sowohl Personen, als auch Dienste beteiligt. Die UML fasst alle Kommunikationspartner der Use Cases als Aktoren zusammen. In diesem Sinne werden Personen und Dienste durch dasselbe Symbol dargestellt. Ich halte es jedoch für transparenter, unterschiedliche Symbole zu verwenden. So werden Personen als Strichmännchen und Dienste als Rechtecke gezeichnet.

Bei der Use Case Analyse werden zunächst die Anwendungsfälle und die beteiligten Aktoren möglichst vollständig identifiziert und in Beziehung zueinander gesetzt. Die Use Case Analyse gibt Antwort auf die Frage: „Was soll das gesamte System leisten, woraus besteht es und in welcher Beziehung zueinander stehen die Komponenten?“

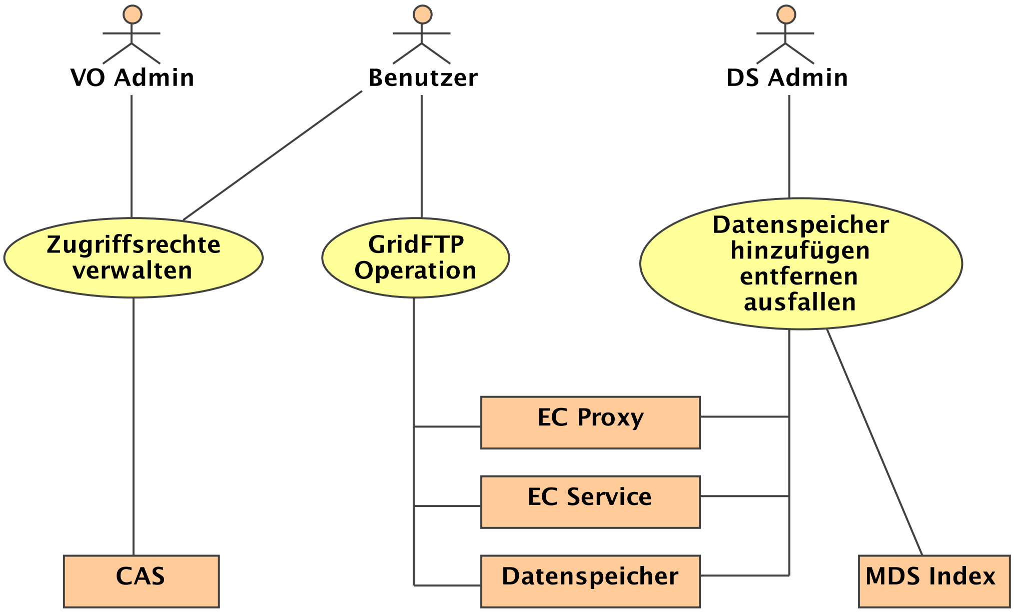

Abbildung 4.2: Use Cases

Das Use Case Diagramm 4.2 zeigt drei Use Cases, *Zugriffsrechte verwalten*, *GridFTP Operation* und *Datenspeicher hinzufügen/entfernen/ausfallen*.

Der *VO Admin* ist sozusagen der Chef der virtuellen Organisation. Er soll die Zugriffsrechte auf alle Dateien und Verzeichnisse der VO verwalten. Der *Benutzer* ist daran interessiert, auf Daten in der VO zuzugreifen, seine Dateien im System zu speichern und vielleicht anderen Benutzergruppen aus der VO Zugriff auf seine Daten zu gewähren. Der *DS Admin* tritt der VO gegenüber als Ressourcenanbieter auf. Als Administrator seiner Datenspeicher kann er diese zur VO hinzufügen oder entfernen.

Das System besteht aus fünf Komponenten, von denen drei bereits als Dienste im Globus Toolkit enthalten sind. Die *Datenspeicher* sind Ressourcen, die ihren Speicherplatz der VO über GridFTP zur Verfügung stellen. Die Anzahl der Datenspeicher einer VO kann sehr verschieden sein und reicht von einigen wenigen bis zu Tausenden Ressourcen.

Für die Überwachung der Datenspeicher wird der MDS Index Service verwendet. Er gibt Auskunft über alle Ressourcen, die zu einer VO gehören. Leider ist GridFTP kein Web Service. Deshalb werden die Datenspeicher nicht im MDS Index Service registriert. Außerdem gibt es keine GridFTP Befehle, die Auskunft über den freien oder belegten Speicherplatz geben. Es muss eine Möglichkeit geschaffen werden, die Datenspeicher trotzdem zu registrieren. Deshalb schlage ich vor, einen einfachen WSRF-Dienst zu implementieren, den ich *SE Properties Service* nenne (siehe Kapitel 5.2.2). Seine Aufgabe ist es, Auskunft über die Eigenschaften von Datenspeichern zu geben.

Die Benutzerverwaltung wird mit dem CAS Service durchgeführt.

Die Anforderungen (siehe Kaptiel 1.3), die nicht durch Dienste des Globus Toolkit erfüllt werden können, werden von den drei Diensten EC Proxy, EC Service und SE Properties Service erfüllt. Sie sind Teil des Entwurfs und müssen spezifiziert und implementiert werden. Die Namensgebung leitet sich aus der Verwendung des Erasure Coding (siehe Kapitel 3.5) ab.

Ausgehend von den Use Cases werde ich die Aufgaben der Dienste beschreiben und die jeweiligen Anforderungen ableiten.

4.2.1 Zugriffsrechte verwalten

In Kapitel 1.3 wurde bereits die Anforderung formuliert, dass alle Dateien der virtuellen Organisation in einer einzigen (virtuellen) Verzeichnisstruktur darzustellen sind. Die Verwaltung der Zugriffsrechte geschieht zentral über den CAS Service. Die virtuelle Organisation sollte mindestens einen Administrator (*VO Admin*) ernennen, der alle Rechte für den CAS besitzt und somit beliebigen Benutzern und Gruppen Zugriffsrechte auf beliebige Dateien geben kann. Im Allgemeinen ist es auch sinnvoll, Benutzern die Möglichkeit zu geben, auf den CAS zuzugreifen und Rechte auf bestimmte Ordner und Dateien selbst zu verwalten. Zu beachten ist, dass die Zugriffsrechte ausschließlich über den CAS verwaltet werden. Alle anderen Schnittstellen, die möglicherweise vorhanden sind, bleiben wirkungslos.

4.2.2 GridFTP Operation

Im auf Globus basierten Grid ist GridFTP das überwiegende Datentransferprotokoll. Ich habe mich deshalb entschieden, GridFTP auch für die Kommunikation mit dem Client und zu den Datenspeichern zu verwenden.

Das System präsentiert sich dem Benutzer gegenüber als GridFTP Server. Die Dateien sollen aber so verteilt werden, dass sie sicher gespeichert sind. Deshalb gibt es keine direkte Verbindung vom Client zu den Datenspeichern. Das System funktioniert wie ein Proxy, der selbst keine Daten speichert, sondern sie mit Hilfe des linearen Block Codes (siehe Kapitel 3.6) in Blockgruppen zerlegt und auf die Datenspeicher verteilt.

Will der Benutzer eine Datei herunterladen, so muss erst festgestellt werden, welche Blockgruppen sich auf welchen Datenspeichern befinden. Dafür betreibt die VO eine zentrale Datenbank, in der zu jeder Datei Verteilungsparameter gespeichert sind. Sie enthalten eine Liste von Adressen zu den einzelnen Blockgruppen einer Datei. Durch Herunterladen ausreichend vieler Blockgruppen kann das System eine Datei wiederherstellen und zum Client übertragen.

4.2.3 Ausfall, Hinzufügen oder Entfernen eines Datenspeichers

Fällt ein Datenspeicher aus oder wird er vom Ressourcenanbieter hinzugefügt oder entfernt, steigt oder sinkt der Gesamtspeicherplatz im System. Daraus folgt meist eine Umverteilung der Daten. Der Umverteilungsprozess sollte automatisch geschehen. Dafür muss das System auch ohne Interaktion mit dem Benutzer Blockgruppen verschieben und erstellen können. Es muss geeignete Verteilungsstrategien geben, um die Datenspeicher so effizient wie möglich zu nutzen.

4.3 Architektur

Abbildung 4.3 zeigt alle Komponenten des Systems und ihre Beziehungen untereinander. Jede gerichtete Assoziation entspricht einer Client- Server- Beziehung. Zu jeder der Beziehungen gehört eine Schnittstelle, die das jeweilige Übertragungsprotokoll beschreibt. Zum Beispiel nimmt der EC Proxy gegenüber dem EC Service sowohl eine Client- als auch Server-Rolle ein, wobei beide Rollen unabhängig voneinander vorkommen können. Das heißt, je nach Richtung können unterschiedliche Schnittstellen eingesetzt werden.

Die gestrichelte Linie stellt eine Vertrauensbeziehungen dar. Die Kommunikation ist in diesem Fall indirekt und erfolgt über die CAS-Erweiterungen im Zertifikat des Benutzers.

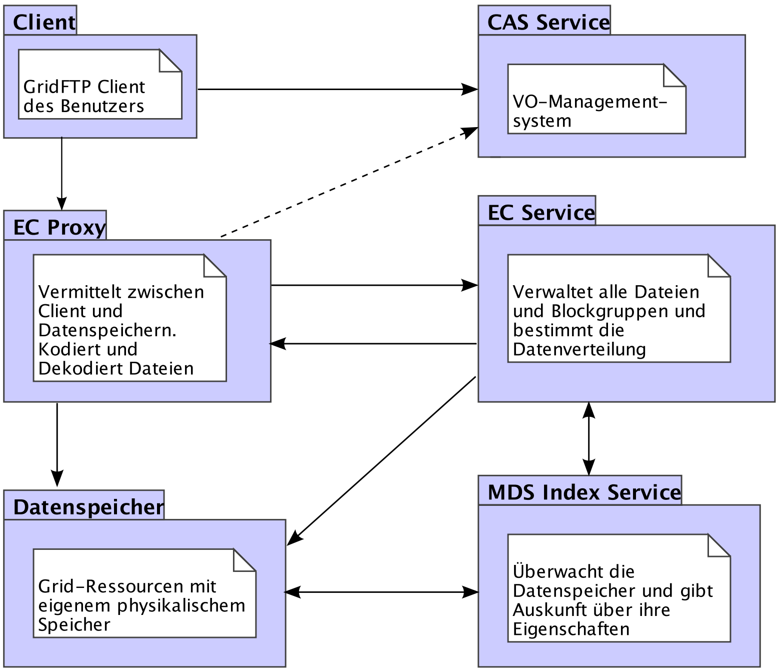

Abbildung 4.3: Architektur

4.3.1 CAS

Für jede VO gibt es genau einen CAS-Server. Damit Benutzer jederzeit autorisiert werden können, sollte dieser möglichst immer erreichbar sein. Der CAS kann prinzipiell detaillierte Zugriffsrechte für jede Art von Ressource verwalten. Um Dateien und Verzeichnisse zu verwalten, muss der Service Typ für GridFTP eingetragen sein und Dateien und Verzeichnisse als Objektgruppen eingetragen werden.

4.3.2 Client

Als Mitglied in der VO benötigt der Benutzer Client-Software für GridFTP und CAS. Bevor der Nutzer die Ressourcen der VO nutzen kann, lässt er sich mit dem Befehl *cas-proxy-init* ein Zertifikat mit CAS Erweiterung ausstellen. Mit diesem Zertifikat greift er über den EC Proxy auf die VO-kritischen Daten zu.

4.3.3 EC Proxy

Der EC Proxy ist die Schnittstelle zum Client. Seine Hauptaufgaben sind die Enkodierung und Dekodierung der Dateien und die Speicherung der Blockgruppen auf den Datenspeichern. Dadurch, dass der EC Proxy Daten parallel zu oder von den Datenspeichern übertragen kann, steigt der Gesamtdurchsatz. Deshalb sollte sich der EC Proxy im lokalen Netz des Client befinden. Die virtuelle Organisation sollte mehrere EC Proxies nutzen können, so dass möglichst alle ihrer Mitglieder einen schnellen Zugang erhalten. Idealerweise betreibt jede beteiligte reale Organisation einen oder mehrere EC Proxies.

Da es für die Datenspeicher nicht erforderlich sein soll, das VO-Management zu unterstützen, kann der EC Proxy sich nicht mit dem Proxy-Zertifikat des Benutzers ausweisen. Im Allgemeinen kennt der Datenspeicher den Benutzer nicht. Stattdessen schlage ich vor, den EC Proxy mit einem eigenen Zertifikat auszustatten. Man kann den Konfigurationsaufwand weiter einschränken, wenn alle EC Proxies und der EC Service dasselbe Zertifikat verwenden. Das bedeutet, dass die VO als Ganze über eine einzige Identität verfügt. Anbieter von Datenspeichern können diese freigeben, indem sie den Zugriff mit dem VO-Zertifikat erlauben. Auf diese

Weise können beide Dienste auch ohne Interaktion mit dem Benutzer auf die Datenspeicher zugreifen. Bemerkung in Kapitel 8.1.1 wird ein anderes Verfahren vorgestellt.)

4.3.4 EC Service

Die EC Proxies verwalten selbst keine Informationen über die Datenverteilung. Eine zentrale Lösung ist hier einfacher zu entwerfen und zu implementieren. Deshalb hat eine VO genau einen EC Service, der die Verteilungsparameter speichert.

Zur Speicherung der Verteilungsparameter und Verzeichnisstruktur benötigt der EC Service lokalen Speicher. Da diese Daten strukturiert sind, und oft nach speziellen Einträgen gesucht wird, bietet es sich an, die Daten in einer SQL Datenbank abzulegen.

Die im System gespeicherten Dateien müssen konsistent gehalten werden. Der EC Service muss verhindern, dass zwei Clients gleichzeitig Dateien ändern. Außerdem müssen veraltete Versionen von Blockgruppen automatisch gelöscht werden.

Neben dieser passiven Aufgabe besitzt der EC Service auch die aktive Aufgabe, den Umverteilungsvorgang automatisch, also ohne Nutzerbeteiligung, zu starten und zu koordinieren. Er greift dafür auf die EC Proxies und die Datenspeicher zu, um Blockgruppen zu verschieben, zu löschen oder neu zu erstellen.

Der EC Service muss über eine Verteilungsstrategie verfügen, die festlegt unter welchen Umständen und wie die Verteilung der Blockgruppen durchgeführt wird. Dazu benötigt er auf jeden Fall die Information, welche Datenspeicher zur VO gehören. Wünschenswert wären auch detaillierte Informationen über jeden einzelnen Datenspeicher, wie zum Beispiel Gesamtkapazität und verfügbarer Speicherplatz. Diese Informationen bekommt er als WS-Resource Properties vom MDS Index Service. Über WS-Notifications kann der EC Service über alle Änderungen an den Datenspeichern benachrichtigt werden.

4.3.5 Datenspeicher

Ein Datenspeicher ist einer Ressource, die physikalischen Speicherplatz zugänglich macht. Was als Datenspeicher gelten kann, reicht von einem einzelnen Arbeitsplatzrechner mit Festplatte bis zu Supercomputern mit Cluster-Dateisystem. Wichtig ist nur, dass der Speicherplatz real ist und über einen GridFTP Server erreicht werden kann.

Typischerweise nutzt eine virtuelle Organisation sehr viele Datenspeicher aus unterschiedlichen realen Organisationen. Es sollte deshalb möglichst einfach sein, diese einzurichten und in die VO einzubinden. Dazu gehört, dass möglichst wenige Anforderungen gestellt werden.

Vor allem die Anforderungen an die Zuverlässigkeit sind gering. Es muss jedoch verhindert werden, dass viele Datenspeicher gleichzeitig ausfallen. Die Ausfallwahrscheinlichkeit muss bekannt sein und es muss Mindestanforderungen geben (siehe Kapitel 7.3). Die VO muss also die Dienstgüte der Datenspeicher anhand von Service Level Agreements festlegen. Insbesondere das geplante Entfernen von vielen Datenspeichern darf nicht auf einmal und ohne Vorwarnung geschehen, sondern muss vorher angekündigt werden.

4.3.6 MDS Index Service

Der MDS Index Service verwaltet alle Ressourcen einer VO. Die EC Property Services registrieren die Datenspeicher der VO und schicken WS-Notifications falls sich die Eigenschaften ändern. Der MDS Index Service fasst alle diese Informationen zusammen und informiert den EC Service über alle Änderungen der Datenspeicher.

4.4 Anforderungen an die einzelnen Komponenten

Die drei Dienste EC Proxy, EC Service und SE Properties Service müssen zusammen die Anforderungen an das Gesamtsystem erfüllen. An jeden der drei Dienste sind spezielle Anforderungen zu stellen, damit das gesamte System funktioniert.

4.4.1 EC Proxy

Der EC Proxy führt die Verteilung der Dateien so durch, dass die Anforderungen an die Sicherheit und Zuverlässigkeit erfüllt werden können. Er muss auch die Policies der VO gegenüber dem Client durchsetzen.

Anforderungen:

- GridFTP Server.
- GridFTP Client Library für Zugriff auf Datenspeicher.
- Web Services Client für Zugriff auf EC Service.
- Setzt die Policies der VO anhand der CAS Erweiterungen durch.
- Authentifiziert sich mit dem Zertifikat der VO.
- Kodierung mit einem Erasure Code, der gegen Ausfall und Kompromittierung schützt.

4.4.2 EC Service

Der EC Service wird in Java implementiert und läuft im Web Services Container des Globus Toolkit. Er ist ein Grid Dienst im Sinne der OGSA. Seine Verteilungsstrategie orientiert sich an der Struktur der virtuellen Organisation. Er stellt die Zuverlässigkeit des Systems sicher, indem er Datenspeicher überwacht und gegebenenfalls darauf reagiert.

Anforderungen:

- Dienst für GT4 Web Services Container (Java WS Core).
- Kommunikation mit MDS Index Service anhand WS-Notifications.
- Überwachung der Datenspeicher einer VO.
- Implementierung einer geeigneten Verteilungsstrategie.
- Synchronisation der Zugriffe (Lese und Schreibsperren auf Dateien).
- Überwachung der Konsistenz von Blockgruppen.
- Speicherung von Verwaltungsinformationen in einer Datenbank (JDBC).
- Zuordnung zwischen Dateien und Blockgruppen.

4.4.3 EC Properties Service

GridFTP erfüllt die Anforderungen der OGSA nicht. Damit Datenspeicher trotzdem von der VO überwacht werden können, muss der EC Properties Service Auskunft über die Eigenschaften von Datenspeichern machen.

Der EC Properties Service wird als Web Service ausgeführt. Zusätzlich zum GridFTP-Server müssen der Web Services Container und natürlich eine Java Virtual Machine installiert werden. Das ist nicht immer möglich oder erwünscht. Deshalb kann ein SE Properties Service mehrere Datenspeicher registrieren, indem das *SE-ResourceProperties* Objekt mehrere *StorageElement* Objekte enthält.

Weil nicht von jedem Datenspeicher gefordert werden kann, zusätzlich zu GridFTP einen Web Services Container zu betreiben, kann der EC Properties Service auch mehr als einen Datenspeicher registrieren. So kann eine VO auch Datenspeicher nutzen, die selbst nicht über einen EC Properties Service verfügen.

Anforderungen

- Dienst für GT4 Web Services Container (Java WS Core).
- Informationen für einen oder mehr Datenspeicher als WS-Resource Properties.
- Resource Properties die Aussagen über Gesamtspeicherplatz, freiem Speicherplatz und URL von Datenspeichern machen,

5 Entwurf

Ziel des Entwurfes ist eine möglichst präzise Beschreibung des gesamten Systems und seiner Komponenten. Er soll zeigen, wie die Komponenten funktionieren. Dazu gehören sowohl die Schnittstellen zwischen den Komponenten, als auch die innere Struktur der Komponenten, des EC Proxy und des EC Service.

5.1 Kontrollfluss der Use Cases

Um die Schnittstellen zu bestimmen, betrachten wir noch einmal die Use Cases auf einem detaillierteren Niveau. Interessant sind dabei besonders die Nachrichten, die sich die Komponenten schicken. Aus ihnen kann man die Kommunikationsbeziehungen ableiten. Die Schnittstelle zwischen EC Proxy und EC Service wird auf diese Weise entworfen. Interessant sind auch die Besonderheiten bei der Nutzung vorhandener Schnittstellen. So wird weder zwischen EC Proxy und Datenspeichern, noch zwischen EC Service und Datenspeichern der volle Befehlssatz des GridFTP Protokolls verwendet.

5.1.1 Erweiterung der Use Cases

Um den Ablauf und die Kommunikationsbeziehungen zu analysieren, werden die Use Cases jeweils durch ein Sequenzdiagramm ([UML04] S. 323) repräsentiert. Dabei ist es wünschenswert, dass ein Use Case keine oder nur wenige Verzweigungen aufweist. Die Use Cases aus der Anforderungsanalyse weisen diese Eigenschaft leider nicht auf. So gehören zum Beispiel das Beschreiben einer Datei und das Wechseln eines Verzeichnisses beide zum Use Case *GridFTP Operation*. Sie haben aber eine stark voneinander abweichende Struktur.

Deshalb wird das Use Case-Diagramm aus der Anforderungsanalyse erweitert (Abbildung 4.2). Weist ein Use Case mehrere Abläufe auf, so wird jeder Ablauf durch einen eigenen Use Case dargestellt und mittels $\langle\langle extend\rangle\rangle$ Beziehung in den ursprünglichen Use Case eingefügt.

Diese Erweiterungen wurden nicht bereits in Abbildung 4.2 aufgenommen, weil es Designentscheidungen gibt, die nicht zwingend aus den Anforderungen folgen und deshalb erst hier im Entwurf getroffen werden können. Sie betreffen die Funktionsweise des EC Service und werden im folgenden beschrieben.

Flache Verzeichnisstruktur auf den Datenspeichern. Die Blockgruppen sind auf den Datenspeichern als Dateien abgelegt. Als solche erhalten sie natürlich auch Dateinamen. Es ist nicht nötig, dass sich die Dateinamen der Blockgruppen von den Dateinamen der zugehörigen Dateien ableiten. Sinnvoller ist es, zufällige Namen zu wählen und alle Blockgruppen auf den Datenspeichern in einem einzigen Verzeichnis aufzulisten. Der Vorteil dieses Verfahrens ist, dass Verzeichnisse angelegt und Dateien verschoben werden können, ohne auf die Datenspeicher zuzugreifen. Verzeichnisse existieren nur in der Datenbank des EC Service. Zur Umbenennung oder Verschiebung von Dateien ist keine Umbenennung der Blockgruppen nötig. Außerdem wird ein Angreifer, der einen Datenspeicher kontrolliert, daran gehindert, Dateinamen und Verzeichnisnamen auszuspähen.

Istzustand und Sollzustand Das Hinzufügen oder Entfernen eines Datenspeichers kann eine zeitaufwendige Operation nach sich ziehen (Stunden bis Tage), während der die Redundanz von Dateien wieder hergestellt wird. Es ist wahrscheinlich, dass mehrere solcher Abläufe gleichzeitig stattfinden. Wenn z.B. ein Datenspeicher nur kurzzeitig ausfällt, kann es passieren, dass der Use Case *Datenspeicher hinzufügen/entfernen/ausfallen* für zwei Datenspeicher gleichzeitig eintritt. Auf folgende Weise wird dies verhindert: Alle Use Cases, die Änderungen an der Datenverteilung nach sich ziehen, führen diese Änderungen nicht selbst aus, sondern schreiben den Änderungswunsch in die Datenbank des EC Service, der dann später darauf reagiert. Es gibt einen Istzustand, der die aktuelle Datenverteilung repräsentiert, und einen Sollzustand, der die gewünschte

Datenverteilung beinhaltet. Use Cases ändern nur den Sollzustand und können sofort ausgeführt werden. Es kommt nicht zu Konflikten wegen paralleler Ausführung.

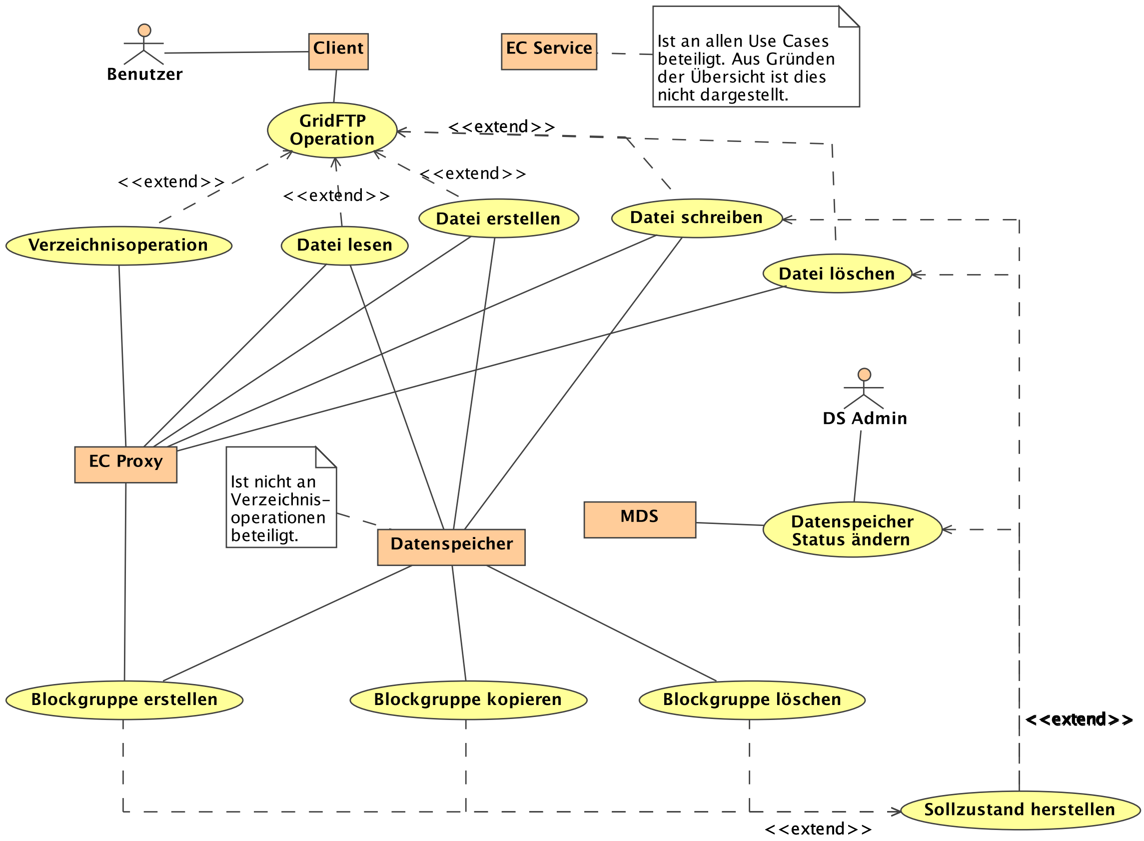

Abbildung 5.1: Use Cases erweitert

Vergleicht man das Use Case-Diagramm aus der Anforderungsanalyse mit dem erweiterten Diagramm in Abbildung 4.2, sieht man, dass der Use Case *GridFTP Operation* jetzt in fünf Use Cases aufteilt worden ist: *Verzeichnisoperation*, *Datei lesen*, *Datei erstellen*, *Datei löschen* und *Datei schreiben*.

Der Use Case *Datenspeicher hinzufügen/entfernen/ausfallen* wurde umbenannt in *Datenspeicher Status ändern*. Der Grund dafür wird später klar (siehe Kapitel 5.1.8).

Use Cases, die Änderungen am Sollzustand vornehmen, werden genau dann durch den Use Case *Sollzustand herstellen* erweitert, wenn dieser nicht bereits läuft. *Sollzustand herstellen* verwendet wiederum *Blockgruppe erstellen*, *Blockgruppe kopieren* und *Blockgruppe löschen*, um seine Aufgabe zu erfüllen.

Der Use Case *Zugriffsrechte verwalten* wurde hier weggelassen, da daran außer dem CAS-Service keine andere Komponente beteiligt ist und da der CAS-Service bereits vollständig im Rahmen des Globus Toolkit implementiert ist

Jetzt können wir die Abläufe der einzelnen Use Cases betrachten.

5.1.2 GridFTP Operation

Dieser Use Case liegt immer dann vor, wenn der Client einen GridFTP-Befehl an den EC Proxy sendet. Streng genommen müsste man jeden GridFTP-Befehl einzeln beschreiben. Hier geht es aber weniger um eine vollständige Beschreibung des GridFTP-Protokolls, als um den Entwurf des EC Proxy und darum, wie die Anforderungen aus Kapitel 4 erfüllt werden können.

Ich fasse daher ähnliche GridFTP-Befehle jeweils zu einem Use Case zusammen. Je nach Befehl wird *GridFTP Operation* durch einen der fünf folgenden Use Cases aufgerufen: *Verzeichnisoperation*, *Datei lesen*, *Datei erstellen*, *Datei löschen*, *Datei schreiben*.

5.1.3 Verzeichnisoperation

Als Verzeichnisoperationen fasse ich alle GridFTP-Befehle zusammen, die höchstens Änderungen an der Datenbank des EC Service, nicht aber an der Datenverteilung an sich machen. Hier werden also alle GridFTP-Befehle behandelt, an denen die Datenspeicher nicht beteiligt sind.

GridFTP Befehle:

- CWD: Verzeichnis wechseln
- LIST, MDTM, NLST, PWD, SIZE: Verzeichnis- oder Datei-Informationen abrufen.
- MKD: Verzeichnis erstellen.
- RMD: Verzeichnis löschen.
- RNFR,RNTO: Datei oder Verzeichnis umbenennen bzw. verschieben.

Vorbedingung: Mindestens ein EC Proxy läuft und kann auf den EC Service zugreifen.

Nachbedingung: Befehl vollständig ausgeführt oder Fehlercode zurückgegeben.

Primärszenario:

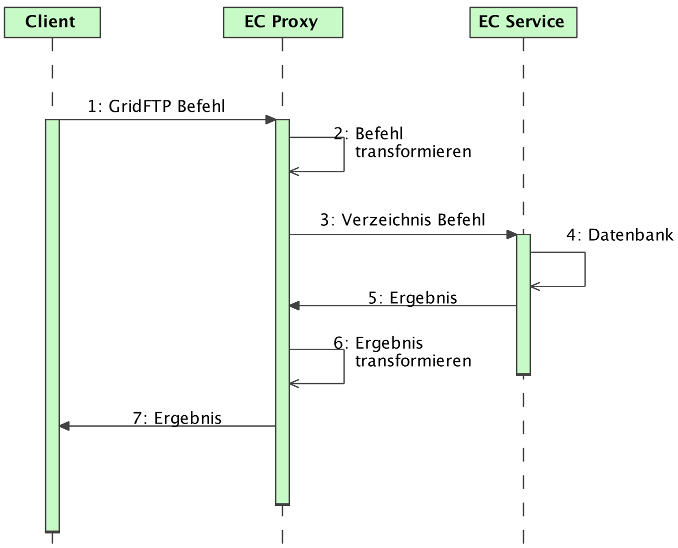

Abbildung 5.2: Verzeichnisoperation

Sekundärszenarien:

- Zugriff verweigert. Benutzer fehlt Autorisierung durch CAS.
- Datei oder Verzeichnis nicht gefunden.
- EC Proxy fällt aus. Das ist nicht kritisch, da die Operation vom EC Service auch ohne EC Proxy beendet werden kann.
- EC Service fällt aus. Die Datenbank wird durch Transaktionen auch im Falle eines Ausfalls konsistent gehalten.

5.1.4 Datei lesen

Eine Datei soll zum Client übertragen werden. Dazu muss bekannt sein, auf welchen Datenspeichern die Blockgruppen liegen. Zunächst müssen die Verteilungsparameter vom EC Service abgefragt werden. Anschließend verbindet sich der EC Proxy mit ausreichend vielen Datenspeichern und startet den Übertragungsvorgang. Sobald von jedem Datenspeicher jeweils ein Block übertragen wurde, kann der EC Proxy die ersten Blöcke dekodieren und zum Client übertragen. Falls während der Übertragung Datenspeicher ausfallen oder zu langsam sind, können weitere Datenspeicher eingesetzt werden, sofern sie das im GridFTP-Protokoll definierte *partial get* (siehe [GFD03]) unterstützen. Es ist sinnvoll, von Anfang an ein oder zwei zusätzliche Datenspeicher zu nutzen. Dadurch werden zwar insgesamt mehr Daten übertragen, aber ein einzelner langsamer Datenspeicher kann nicht die ganze Übertragung beeinträchtigen. Während der gesamten Übertragung darf die Datei weder gelöscht, noch von einem anderen Benutzer beschrieben werden. Deshalb wird über den EC Service eine Lesesperre auf die Datei gesetzt und am Ende wieder entfernt.

GridFTP Befehle:

- RETR, ERET: Datei empfangen.

Vorbedingung: EC Proxy läuft und kann sowohl auf den EC-Service als auch auf genügend viele Datenspeicher zugreifen.

Nachbedingung: Datei vollständig übertragen oder Fehlercode zurückgegeben.

Primärszenario:

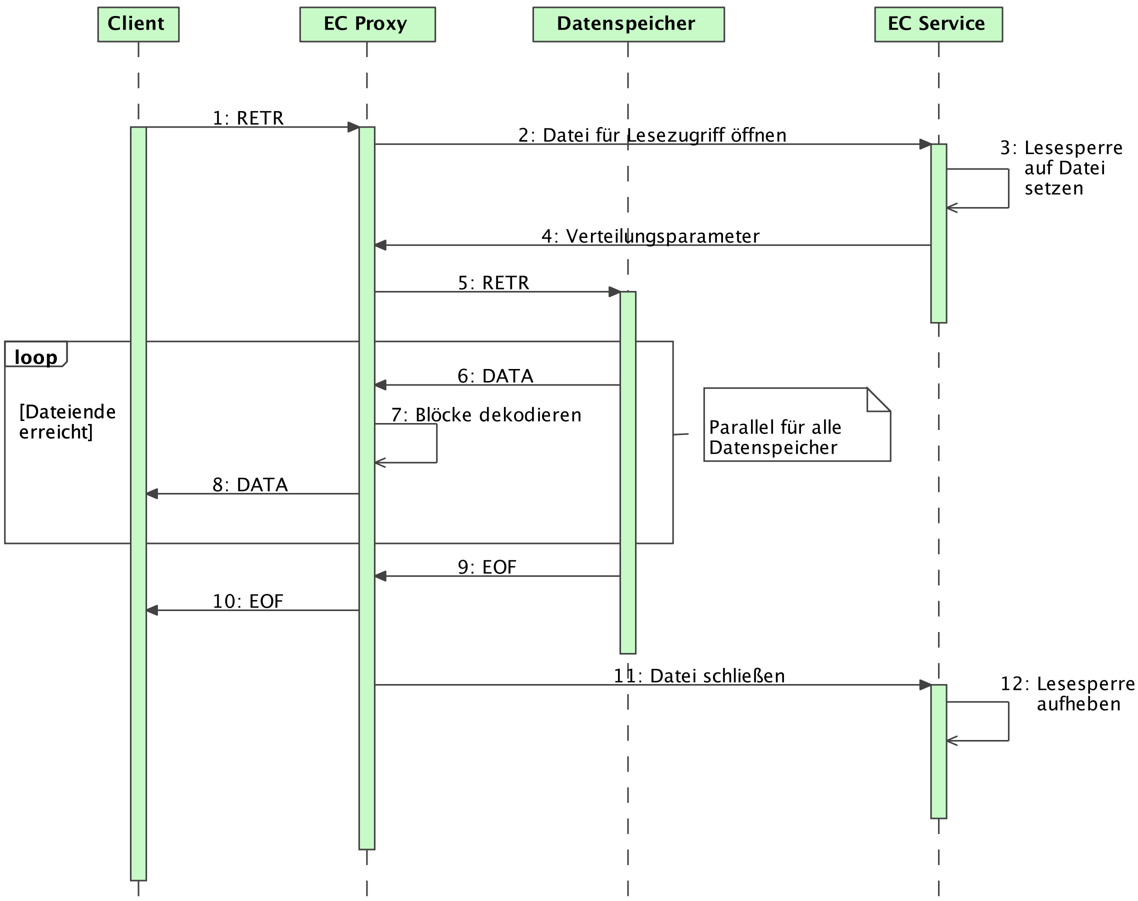

Abbildung 5.3: Datei lesen

Sekundärszenarien:

- Partial Get. Statt RETR wird ERET verwendet.
- Zugriff verweigert.
- Datei oder Verzeichnis nicht gefunden.
- Zugriff steht in Konflikt mit einer bestehenden Schreibsperre.
- EC Proxy fällt aus. Lesesperre muss nach einem Timeout automatisch aufgehoben werden.
- Datenspeicher fällt aus. Zusätzliches Datenspeicher zur Übertragung hinzufügen und EC Service über den Ausfall informieren.
- Ein Datenspeicher ist wesentlich langsamer als andere Datenspeicher. Der EC Proxy informiert den EC Service.
- EC Service fällt aus. Beim Neustart des EC Service werden alle Sperren auf Dateien aufgehoben. Es kann zu Sperrenverletzungen kommen. Dieses Verhalten kann wegen seiner Seltenheit in Kauf genommen werden.
- Redundanz der Datei unter 100%. Datei kann nicht gelesen werden. Fehlercode zurückgeben. EC Service informieren.

5.1.5 Datei erstellen

Eine Datei, die noch nicht vorhanden ist, soll auf die Datenspeicher verteilt werden. Sie muss im EC Service registriert werden und es müssen Verteilungsparameter erstellt werden. Anschließend kann der EC Proxy die Datei kodieren und ihre Blockgruppen zu den Datenspeichern übertragen. Die Anzahl der Blockgruppen hängt von der gewünschten Redundanz ab. Im Allgemeinen werden mehr Blöcke erstellt, als die Originaldatei besitzt. Das Schreiben ist wegen der erhöhten Datenmenge etwas langsamer als das Lesen. Eine Datei gilt erst dann als vorhanden, wenn sie vollständig übertragen ist. Vorher ist sie im EC Service als temporär eingetragen. Es kann nur dann zum Konflikt kommen, wenn ein anderer Client zufällig zur gleichen Zeit eine Datei mit gleichem Namen erstellt.

GridFTP Befehle:

- STOR: Datei senden.

Vorbedingung: EC Proxy läuft und kann sowohl auf den EC-Service als auch auf genügend viele Datenspeicher zugreifen.

Nachbedingung: Datei im System gespeichert und im EC Service registriert oder - im Fehlerfall - von allen Datenspeichern wieder gelöscht.

Primärszenario:

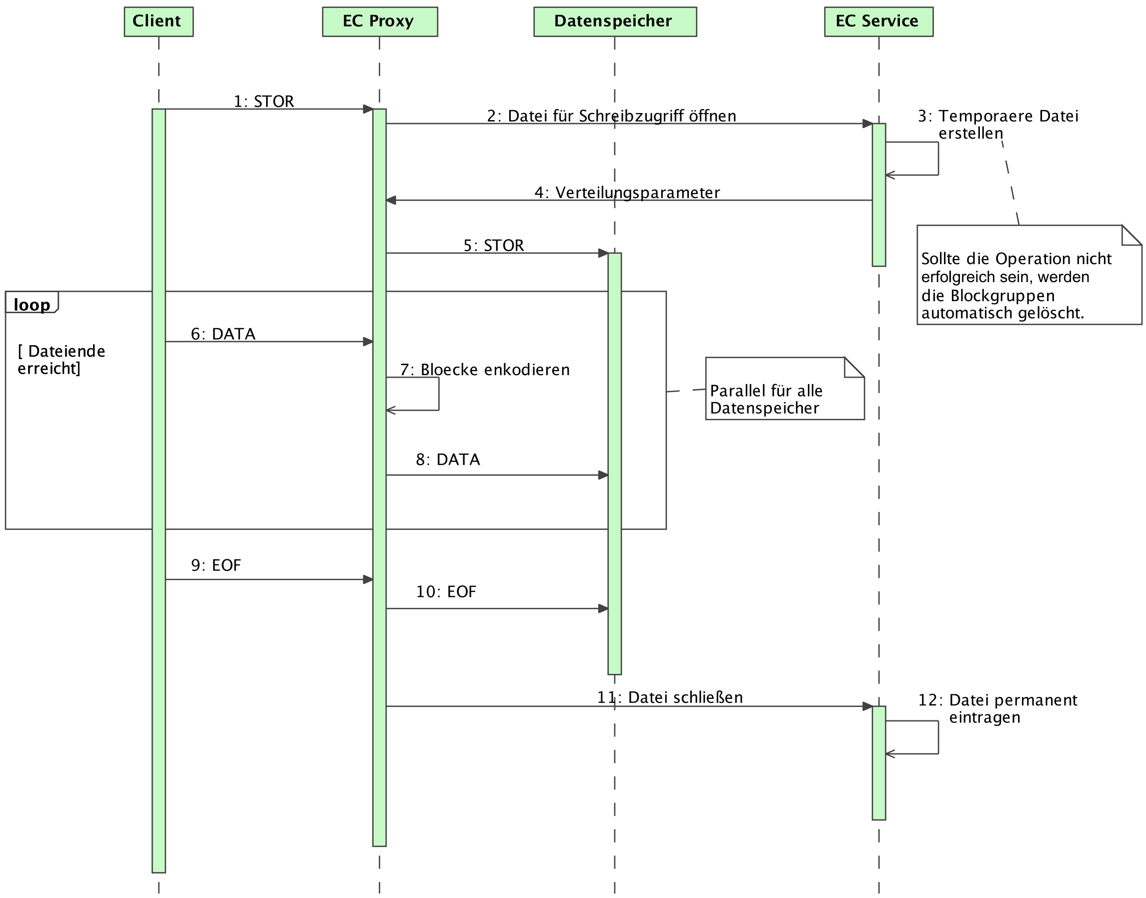

Abbildung 5.4: Datei erstellen

Sekundärszenarien:

- Zugriff verweigert.
- Verzeichnis nicht gefunden.
- Zugriff steht in Konflikt mit einer bestehenden Schreib- oder Lesesperre.
- Datenspeicher hat keinen freien Speicherplatz. Der EC Proxy informiert den EC Service. Der Datenspeicher kann weiter verwendet werden, aber es sollten keine weiteren Schreibversuche unternommen werden.
- Ein Datenspeicher ist wesentlich langsamer als andere Datenspeicher. Der EC Proxy informiert den EC Service.
- EC Proxy fällt aus. Nach einem Timeout hebt der EC Service die Schreibsperre auf und entfernt alle übrig gebliebenen Blockgruppen von den Datenspeicher.
- Datenspeicher fällt aus. Der EC Proxy informiert den EC Service über den Ausfall. Falls die Redundanz unter 100% absinkt, muss die Übertragung abgebrochen werden.
- EC Service fällt aus. Datei kann zwar übertragen, aber nicht im EC Service permanent eingetragen werden. Der EC Service geht beim Neustart davon aus, dass die Datei vollständig übertragen wurde, und trägt sie permanent ein. Falls EC Proxy und EC Service gleichzeitig ausfallen, wird die Datei inkonsistent.

5.1.6 Datei schreiben

Der Client schreibt in eine bereits vorhandene Datei. Es ist zu unterscheiden, ob die Datei komplett oder nur teilweise geschrieben wird.

Wenn die komplette Datei geschrieben wird, ist es am einfachsten, sie neu anzulegen und die Blockgruppen erst nach erfolgreicher Übertragung zu löschen. Im Fehlerfall ist die ursprüngliche Version der Datei noch vorhanden und kann weiter verwendet werden. Es reicht, die Use Cases *Datei erstellen* und *Datei löschen* nacheinander auszuführen.

Partial Put. Eine GridFTP-Erweiterung lässt das partielle Beschreiben einer Datei zu. Im Allgemeinen orientiert sich der Client nicht an Blockgrenzen. Dass heißt, die ersten und letzten Datenblöcke werden nicht vollständig übertragen. Es müssen Blöcke von den Datenspeichern geholt werden, um die unvollständigen Blöcke zu ergänzen und zu kodieren. Blockgruppen auf ausgefallenen Datenspeichern können nicht aktualisiert werden. Sie müssen gelöscht werden, sobald die Datenspeicher wieder erscheinen.

GridFTP Befehle:

- ESTO: Datei senden (erweiterte Version).

Vorbedingung: EC Proxy läuft und kann sowohl auf den EC-Service, als auch auf genügend viele Datenspeicher zugreifen.

Nachbedingung: Neue Version vollständig übertragen.

Primärszenario:

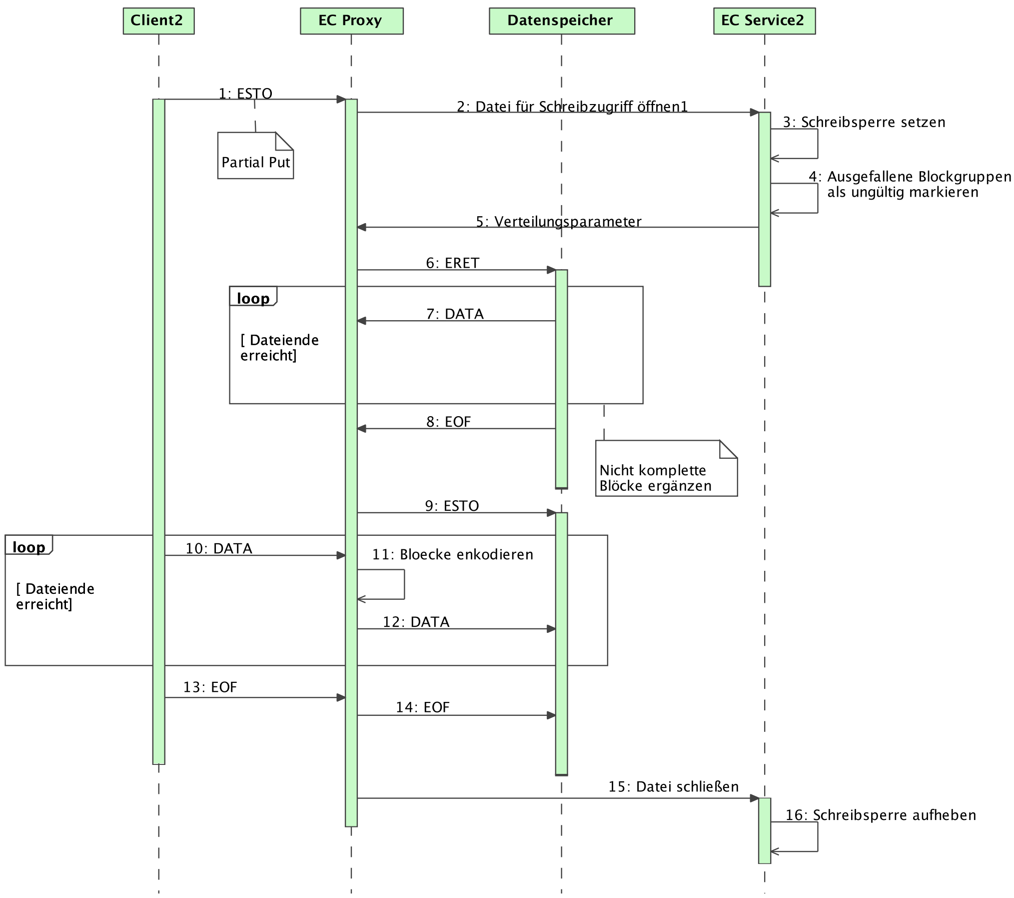

Abbildung 5.5: Datei schreiben

Sekundärszenarien:

- Zugriff verweigert.
- Datei oder Verzeichnis nicht gefunden.
- Zugriff steht in Konflikt mit einer bestehenden Schreib- oder Lesesperre.
- Ein Datenspeicher ist wesentlich langsamer als andere Datenspeicher. Der EC Proxy informiert den EC Service.
- EC Proxy oder Client fällt aus. Beim Partial Put lässt sich nicht vermeiden, dass die Datei inkonsistent wird.
- EC Service fällt aus. Bei Neustart wird die Schreibsperre aufgehoben.

5.1.7 Datei löschen

Soll eine Datei entfernt werden, müssen alle Blockgruppen von den Datenspeichern gelöscht werden und die Datei muss aus der Datenbank des EC Service entfernt werden. Blockgruppen, die sich auf vorübergehend ausgefallenen Datenspeichern befinden, können erst gelöscht werden, wenn die entsprechenden Datenspeicher wieder verfügbar werden. Der Vorgang kann deshalb Stunden bis Tage dauern und muss asynchron ausgeführt werden. Die Aufgabe wird dem EC Service übergeben, der den Vorgang erst dann beendet, wenn alle Blockgruppen von allen Datenspeichern gelöscht sind. Ausgefallene Datenspeicher bleiben im EC Service registriert. Falls sie wieder verfügbar werden, kann der EC Service alle Änderungen, die während des Ausfalls gemacht wurden, ausführen.

GridFTP Befehle:

- DELE: Datei löschen.

Vorbedingung: EC Proxy läuft und kann auf den EC Service zugreifen, Datenspeicher sind nicht nötig.

Nachbedingung: Der Sollzustand wurde geändert und der Use Case *Sollzustand herstellen* wird ausgeführt.

Primärszenario:

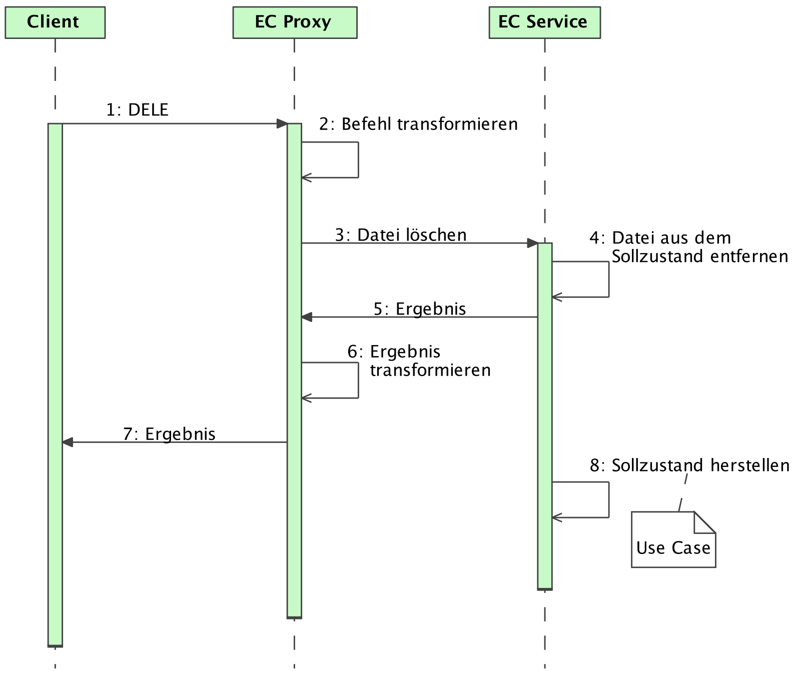

Abbildung 5.6: Datei löschen

Sekundärszenarien:

- Zugriff verweigert.
- Datei oder Verzeichnis nicht gefunden.
- Zugriff steht in Konflikt mit einer bestehenden Schreib- oder Lesesperre.
- EC Proxy fällt aus. Dies ist nicht kritisch, da die Operation vom EC Service erfolgreich beendet werden kann.
- EC Service fällt aus. Die Datenbank wird durch Transaktionen auch im Falle eines Ausfalls konsistent gehalten.

5.1.8 Datenspeicher-Status ändern

Aus der Analyse der vorangegangenen Use Cases geht hervor, dass Datenspeicher nicht nur hinzugefügt oder entfernt werden. Es kann auch zu Fehlverhalten kommen, das mehr oder weniger gravierend ist. Zum Beispiel können volle und inperformante Datenspeicher mit Einschränkungen weiter genutzt werden.

Ausgefallene Datenspeicher sollten weiterhin bekannt sein, damit die Blockgruppen bei Wiedererscheinen noch richtig zugeordnet werden können.

Fehler können auch unbemerkt entstehen. Wenn funktionierende Datenspeicher fehlerhaft werden und im MDS registriert bleiben, kann der EC Service diese Situation nicht erkennen. Es gibt keine andere Möglichkeit, als regelmäßige Tests aller Datenspeicher durchzuführen und zu überprüfen, ob noch alle Blockgruppen vorhanden sind.

Zu beachten ist, dass es Statusänderungen gibt, an denen kein Benutzer beteiligt ist. Der EC Service hat durch die Verwendung des VO-Zertifikats die Fähigkeit, vollkommen autonom zu arbeiten und die Datenverteilung zu ändern. Das System kann sich selbst an Änderungen der VO anpassen.

Die konkreten Ereignisse, die zum Statuswechsel führen, werden in (Kapitel 5.4.3) besprochen. Sie werden von einer der drei folgenden Komponenten ausgelöst:

- *EC Proxy.* Der EC Proxy meldet ein Fehlverhalten oder Ausfall eines Datenspeichers.
- *SE Properties Service.* Der MDS Index Service leitet eine Änderungen in den Eigenschaften eines Datenspeichers weiter, oder meldet das Erscheinen oder Verschwinden.
- *EC Service.* Der EC Service sorgt intern für einen Statuswechsel. Er wird entweder durch das Ergebnis eines Tests oder durch den Ablauf eines Timeouts ausgelöst.

Vorbedingung: EC Service läuft.

Nachbedingung: Status einer Datenressource geändert. Sollzustand neu berechnet. Sollzustand wird hergestellt.

Primärszenario:

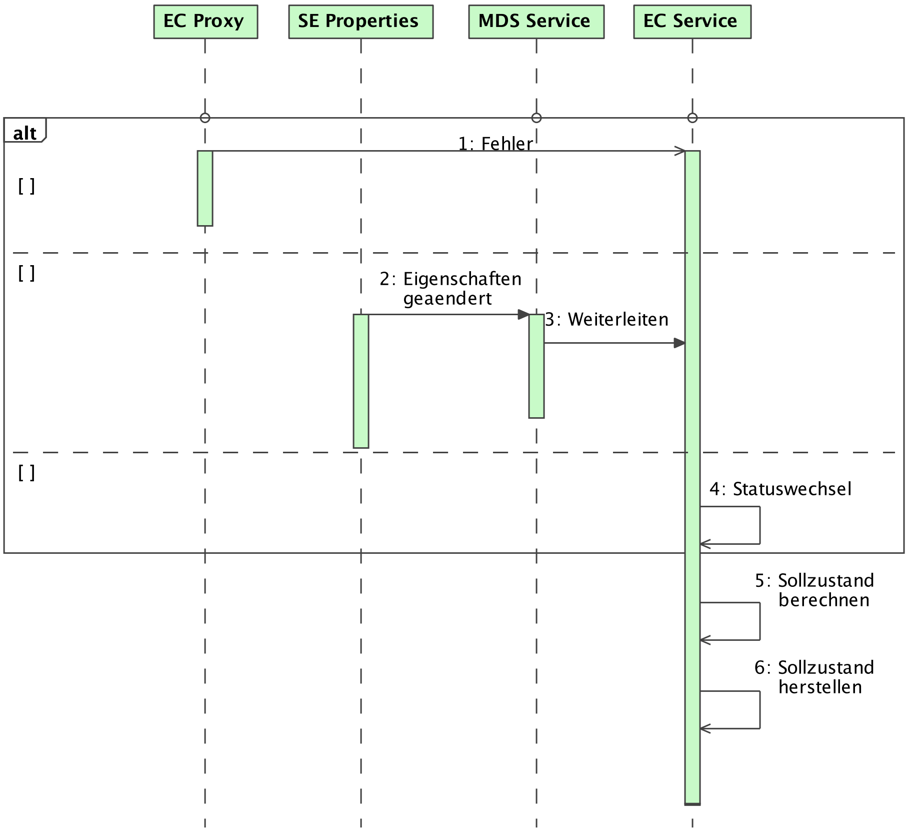

Abbildung 5.7: Status einer Datenressource ändern

Sekundärszenarien:

- Gesamtspeicherplatz reicht nicht aus. VO Administrator informieren. Verminderte Redundanz, solange kein Speicher verfügbar.
- EC Service fällt aus. Ereignis wird nach Neustart wieder auftreten.
- MDS Service fällt aus. Ohne MDS können keine neuen Datenspeicher eingefügt werden, die vorhandenen bleiben aber erhalten.

5.1.9 Sollzustand herstellen

Zieht einer der oben genannten Use Cases Änderungen an der gewünschten Datenverteilung nach sich, so müssen diese Änderungen ausgeführt werden. Es ist wahrscheinlich, dass der Sollzustand mehrere Änderungen in kurzem Abstand erfährt. Dadurch kommt es vor, dass der Sollzustand geändert wird, während er noch hergestellt wird. Es ist nicht sinnvoll weitere Instanzen des Use Cases zu starten. Der bereits laufende Use Case bezieht den neuen Sollzustand mit ein und wird erst beendet, wenn der Istzustand mit dem neuen Sollzustand übereinstimmt.

Vorbedingung: EC Service läuft.

Nachbedingung: Sollzustand = Istzustand.

Primärszenario:

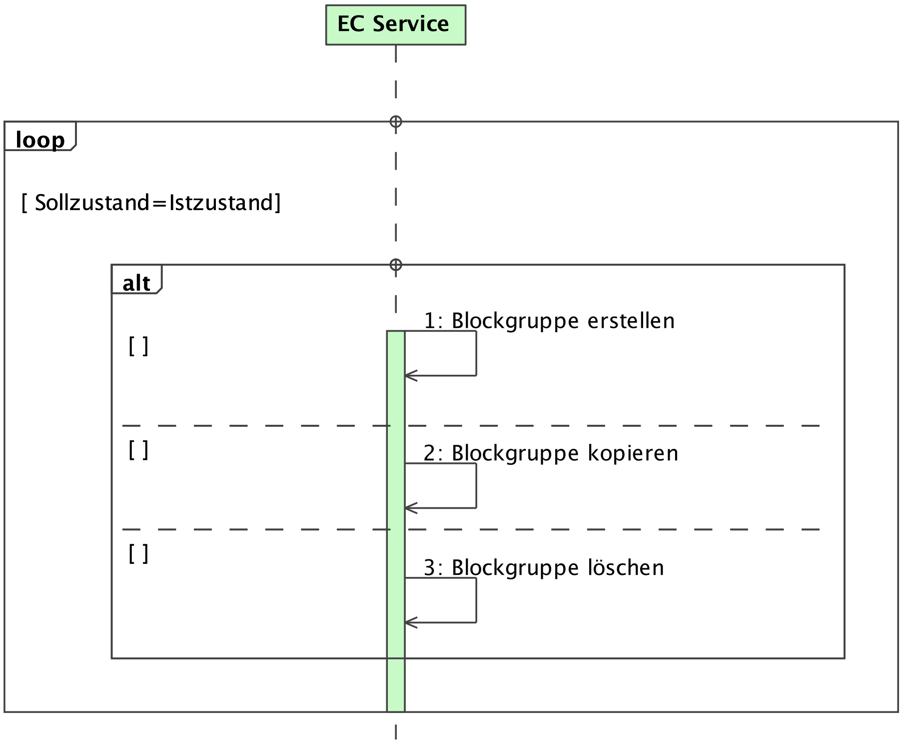

Abbildung 5.8: Sollzustand herstellen

Sekundärszenarien:

- Fehler in Datenressource. Status der Ressource als fehlerhaft markieren und Sollzustand neu berechnen.
- Sollzustand wird während der Ausführung geändert. Falls Operationen, die gerade im Gange sind, mit dem neuen Sollzustand in Konflikt stehen, entsprechende Operationen abbrechen.
- Client greift auf Datei zu, die gerade bearbeitet wird. Aktuelle Operation abbrechen und die Aktion des Clients ausführen.
- EC Service fällt aus. Operationen, die zum Zeitpunkt des Ausfalls im Gange waren, müssen erneut ausgeführt werden. Der Istzustand darf zu keinem Zeitpunkt inkonsistent sein.

5.1.10 Blockgruppe löschen

Ein Blockgruppe wird gelöscht, wenn die Redundanz einer Datei höher ist, als gefordert, oder wenn die Datei insgesamt gelöscht wird.

GridFTP Befehle:

- DELE: Datei löschen.

Vorbedingung: Datenressource läuft. EC Service läuft.

Nachbedingung: Blockgruppe gelöscht.

Primärszenario:

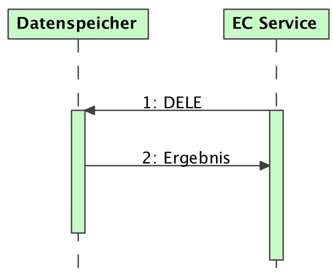

Abbildung 5.9: Blockgruppe löschen

Sekundärszenarien:

- Zugriff verweigert. Datenspeicher im EC Service als ausgefallen eintragen.
- Datei oder Verzeichnis nicht gefunden. Ignorieren. Blockgruppe wurde schon gelöscht.
- EC Service fällt aus. Der Use Case *Sollzustand herstellen* wird nach dem Neustart ausgeführt. *Datei löschen* wird schließlich erneut ausgeführt.

5.1.11 Blockgruppe kopieren

Wenn eine Blockgruppe von einer Datenressource auf eine andere verschoben werden soll, wird sie zunächst kopiert und dann gelöscht. So kann sichergestellt werden, dass die Redundanz auch im Fehlerfall nicht abnimmt. Die Datenübertragung wird vom EC Service initiiert und findet als Third Party Transfer direkt zwischen beiden Datenspeichern statt.

GridFTP Befehle:

- RETR: Datei empfangen.
- STOR: Datei senden.

Vorbedingung: Beide Datenressourcen unterstützen den Third Party Transfer. EC Service läuft.

Nachbedingung: Blockgruppe kopiert.

Primärszenario:

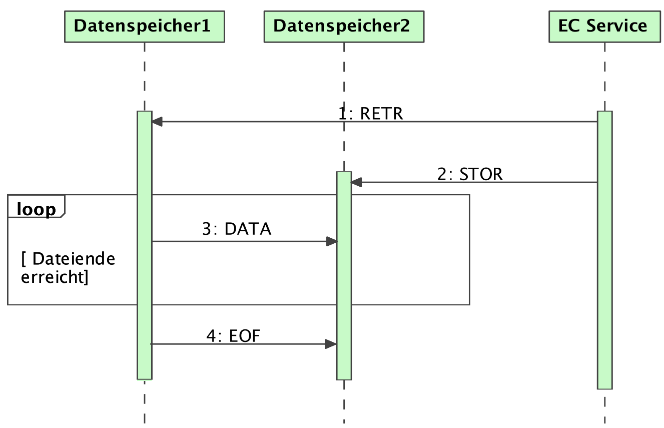

Abbildung 5.10: Blockgruppe kopieren

Sekundärszenarien:

- Datei oder Verzeichnis nicht gefunden. Kann nur vorkommen, wenn eine Blockgruppe lokal gelöscht wurde. Status des Datenspeichers auf *unzuverlässig* ändern.
- Datenspeicher fällt aus. Status des Datenspeichers ändern.
- EC Service fällt aus. Der Use Case *Sollzustand herstellen* wird nach dem Neustart ausgeführt. Der Use Case *Blockgruppe kopieren* wird schließlich erneut ausgeführt.

5.1.12 Blockgruppe erstellen

Nach dem Ausfall eines Datenspeichers muss eine Blockgruppe neu erstellt werden. Es ist sinnvoll, eine Blockgruppe mit einem Index zu erstellen, der noch nicht vorgekommen ist. So kann verhindert werden, dass eine Blockgruppe doppelt vorkommt, falls der Datenspeicher wieder verfügbar wird.

Zum Erstellen der Blockgruppe ist die komplette Datei nötig. Es lässt sich nicht vermeiden, die gesamte Datei zu dekodieren und den gewünschten Teil daraus zu enkodieren. Die einzige Komponente, die dazu in der Lage ist, ist der EC Proxy. Durch eine spezielle Endung zum normalen Dateinamen wird er beauftragt, eine bestimmte Blockgruppe anstatt der vollständigen Datei zu erstellen.

GridFTP Befehle:

- RETR: Datei empfangen.
- STOR: Datei senden.

Vorbedingung: EC Proxy läuft. Die Ziel-Datenressource unterstützt den Third Party Transfer. EC Service läuft.

Nachbedingung: Blockgruppe erstellt und auf der Ziel-Datenressource gespeichert.

Primärszenario:

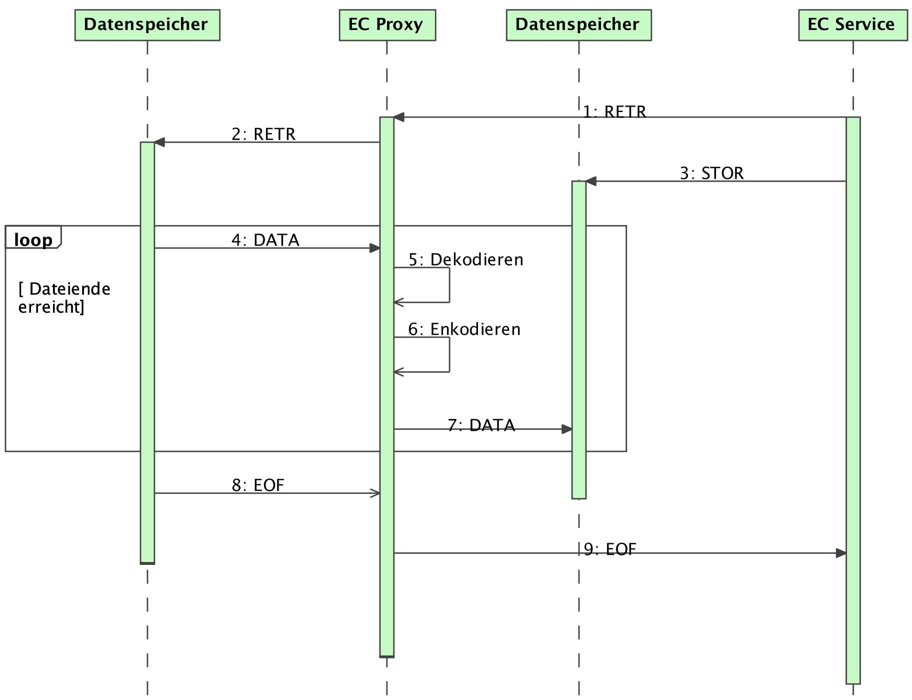

Abbildung 5.11: Blockgruppe erstellen

Sekundärszenarien:

- Zugriff verweigert. Status des Datenspeichers in *ausgefallen* ändern.
- Datei oder Verzeichnis nicht gefunden. Kann nur vorkommen, wenn eine Blockgruppe lokal gelöscht wurde. Status des Datenspeichers in *unzuverlässig* ändern.
- EC Proxy fällt aus. Anderen Proxy wählen und noch einmal probieren.
- Datenspeicher fällt aus. Zusätzlichen Datenspeicher zur Übertragung hinzufügen.
- EC Service fällt aus. Der Use Case *Sollzustand herstellen* wird nach dem Neustart ausgeführt. Der Use Case *Blockgruppe erstellen* wird schließlich erneut ausgeführt.
- Redundanz der Datei unter 100%. Datei kann nicht gelesen werden. Fehlercode zurückgeben.

5.2 Schnittstellen

In den Use Cases läßt sich erkennen, welche Nachrichten sich die Komponenten schicken. Anhand dieser Nachrichten können jetzt Schnittstellen definiert werden.

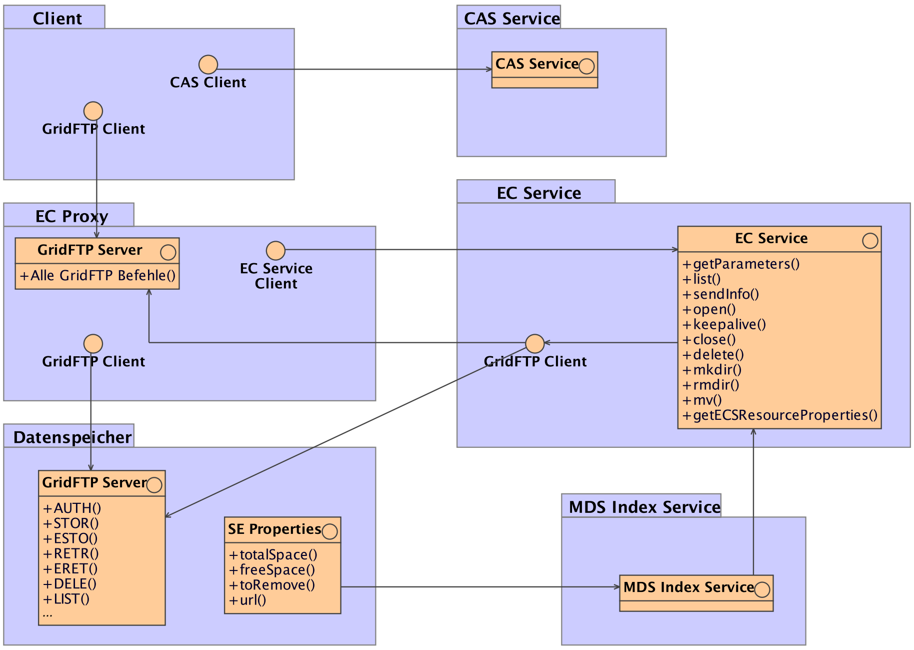

Abbildung 5.12: Schnittstellen zwischen den Komponenten

Abbildung 5.12 zeigt noch einmal die Komponenten des Systems und ihre Schnittstellen. Jede Schnittstelle besitzt eine Client- und eine Serverseite. Die Clients sind hier als Kreise dargestellt. Sie greifen auf die Schnittstellen der Server zu, die hier als Rechtecke dargestellt sind. Die Schnittstellen der Globus Toolkit eigenen Dienste (*CAS, MDS, GridFTP*) sind hier nicht vollständig dargestellt (siehe [CAS07], [IND07] und [GFD03]).

Für den Zugriff auf die Datenspeicher ist nur ein begrenzter Teil der Befehle des GridFTP-Protokolls nötig. EC Proxy und EC Service verwenden folgende Befehle: *AUTH*, *STOR, ESTO*, *RETR, ERET*, *DELE*, *LIST*

Die Schnittstellen des EC Service und des SE Properties Service müssen noch genau beschrieben werden. Abbildung 5.12 zeigt nur die Namen der Operationen. Die folgenden zwei Abschnitte beschreiben die Schnittstellen inklusive aller Datenstrukturen.

5.2.1 Schnittstelle des EC Service

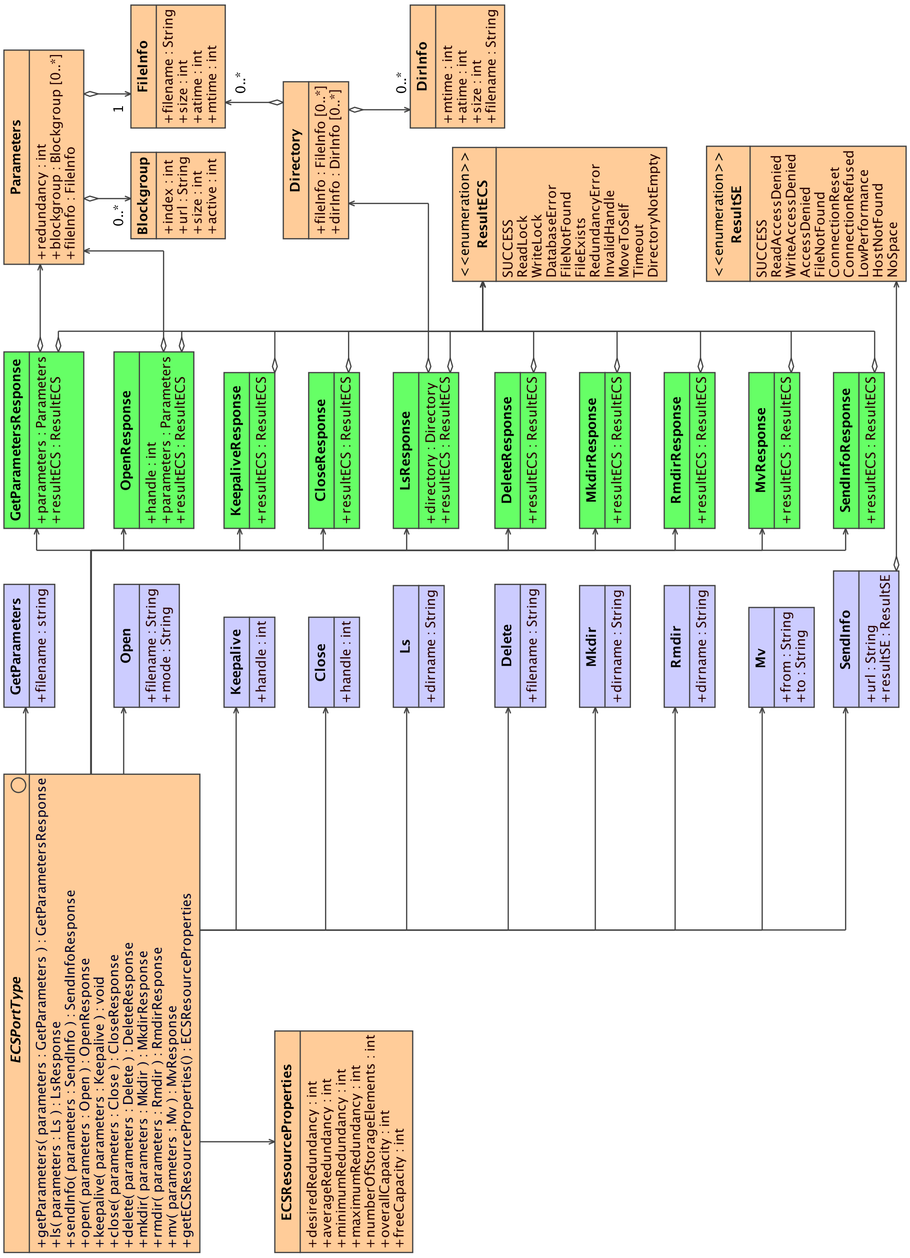

Abbildung 5.13: Schnittstelle des EC Service

Wenn man Grid-Dienste für das Globus Toolkit entwirft, erstellt man zunächst eine Schnittstellenbeschreibung mit Hilfe der WSDL. Die WSDL ist nicht an bestimmte Programmiersprachen gebunden. Sie ist aber sehr ähnlich zu den Klassendefinitionen, wie sie in der objektorientierten Programmierung vorkommen. Deshalb habe ich mich entschieden, die Schnittstelle des EC Service anhand eines Klassendiagramms zu erläutern. Das Globus Toolkit enthält ein Werkzeug, das aus einer WSDL-Schnittstellenbeschreibung Java-Klassen generiert (Java Stubs). Abbildung 5.13 zeigt die Klassen, so wie sie in der späteren Implementierung des EC Service vorkommen.

Die Schnittstelle zum EC Service ist durch ECSPortType gegeben. Wer genau hinsieht, erkennt, dass es sich um ein Interface handelt. Bei der Implementierung des EC Service müssen genau die Methoden dieses Interface implementiert werden.

Alle anderen Klassen in Abbildung 5.13 beschreiben die Informationen, die in den Nachrichten zwischen EC Proxy und EC Service übertragen werden. Sie werden vollständig vom Globus Toolkit generiert.

Bei der Definition von Grid-Diensten ist es üblich, für jede Nachricht jeweils einen Eingabe- und einen Ausgabetyp zu definieren. Die Java Stubs enthalten für jeden dieser Typen eine eigene Klasse. Die Namen der Eingabe- und Ausgabeklassen leiten sich vom Namen der entsprechenden Operation ab. Zur Übersichtlichkeit sind die Klassen der Eingabeparameter blau und die der Ausgabeparameter grün dargestellt. Von der Möglichkeit, einfache Typen direkt als Ein- oder Ausgabewerte zu nutzen, wurde hier abgesehen. Dadurch werden alle Nachrichten einheitlich behandelt. Erweiterungen am WSDL-Dokument können mit wenigen Änderungen am Programmcode durchgeführt werden.

Hier ist eine kurze Übersicht der Operationen von ECSPortType:

- *getParameters*: Verteilungsparameter einer Datei abrufen.
- *ls*: Inhalt eines Verzeichnisses auflisten.
- *sendInfo*: Der EC Proxy sendet Informationen über das Fehlverhalten eines Datenspeichers.
- *open*: Datei öffnen (Lese- oder Schreibzugriff).
- *keepalive*: Dateioperation aufrecht erhalten.
- *close*: Datei schließen.
- *delete*: Datei löschen.
- *mkdir*: Verzeichnis erstellen.
- *rmdir*: Verzeichnis löschen.
- *mv*: Datei oder Verzeichnis verschieben.
- *getECSResourceProperties*: Managementinformationen über den EC Service.

Das WSDL Dokument des EC Service befindet sich im Anhang A.

5.2.2 Schnittstelle des SE Properties Service

Der SE Properties Service implementiert genau eine Operation *getSEResourceProperties*. Sie gibt ein Objekt des Typs *SEResourceProperties* zurück, das Objekte vom Typ *StorageElement* enthält. Die *StorageElement* Objekte beschreiben jeweils einen Datenspeicher mittels folgender Informationen:

- *totalSpace*: Gesamtgröße des Speichers, den die VO nutzen darf.
- *freeSpace*: Freier Speicherplatz für die VO.
- *toRemove*: Wird vom Ressourcenanbieter von 0 auf 1 gesetzt, bevor der Datenspeicher entfernt wird.
- *url*: Addresse des Datenspeichers und Name des Verzeichnis, in dem die VO ihre Daten ablegen soll.

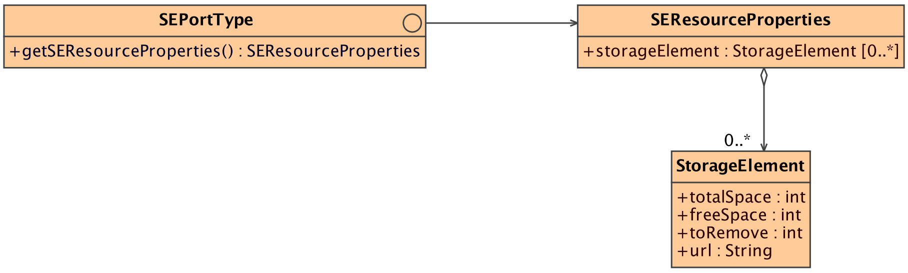

Abbildung 5.14: Schnittstelle des SE Properties Service

Über unterschiedliche Implementierungen des EC Properties Service können folgende Szenarien abgedeckt werden:

- Datenspeicher führt EC Properties Service aus. Dieser Fall ist zu bevorzugen. Der Datenspeicher gibt nur über sich selbst Auskunft.
- Reale Organisation führt einen EC Properties Service für alle ihre Datenspeicher aus. Die reale Organisation bestimmt Mechanismen, mit denen Informationen über die Datenspeicher gesammelt werden. Der EC Properties Service veröffentlicht die gesammelten Informationen.
- Die virtuelle Organisation betreibt einen EC Properties Service. Die VO kann alle Datenspeicher, die nicht bereits registriert sind, manuell eintragen.

Das WSDL Dokument des EC Service befindet sich im Anhang B.

5.3 Entwurf des EC Proxy

5.3.1 Architektur

Der EC Proxy basiert auf der Globus Implementierung des GridFTP-Servers. Sein modulares Konzept erlaubt es, das Modul für den Dateizugriff auszutauschen. Normalerweise ist das Data Storage Interface (DSI) für den Zugriff auf lokale Speichermedien gedacht. Es ist aber möglich, ein DSI zu schreiben, das stattdessen per GridFTP-Client auf die Datenspeicher zugreift. Der GridFTP-Server des Globus Toolkit liegt als Ansi-C Quelltext vor, verwendet aber durchaus objektorientierte Ansätze. Es macht hier durchaus Sinn, mit einem Klassendiagramm zu arbeiten. Die Klassen in Abbildung 5.15 sind als Kollektionen von Funktionen zu sehen. Funktionen, die in derselben Quelltextdatei definiert sind und einem ähnlichen Zweck dienen, werden hier als Klassen dargestellt.

Abbildung 5.15 zeigt ein vereinfachtes Klassendiagramm für den EC Proxy. Funktionen und Attribute, die im Rahmen dieses Entwurfs nicht interessant sind, wurden hier weggelassen. Von der Schnittstelle zum EC Service sind hier nur ECSPortType und seine Methoden gezeigt.

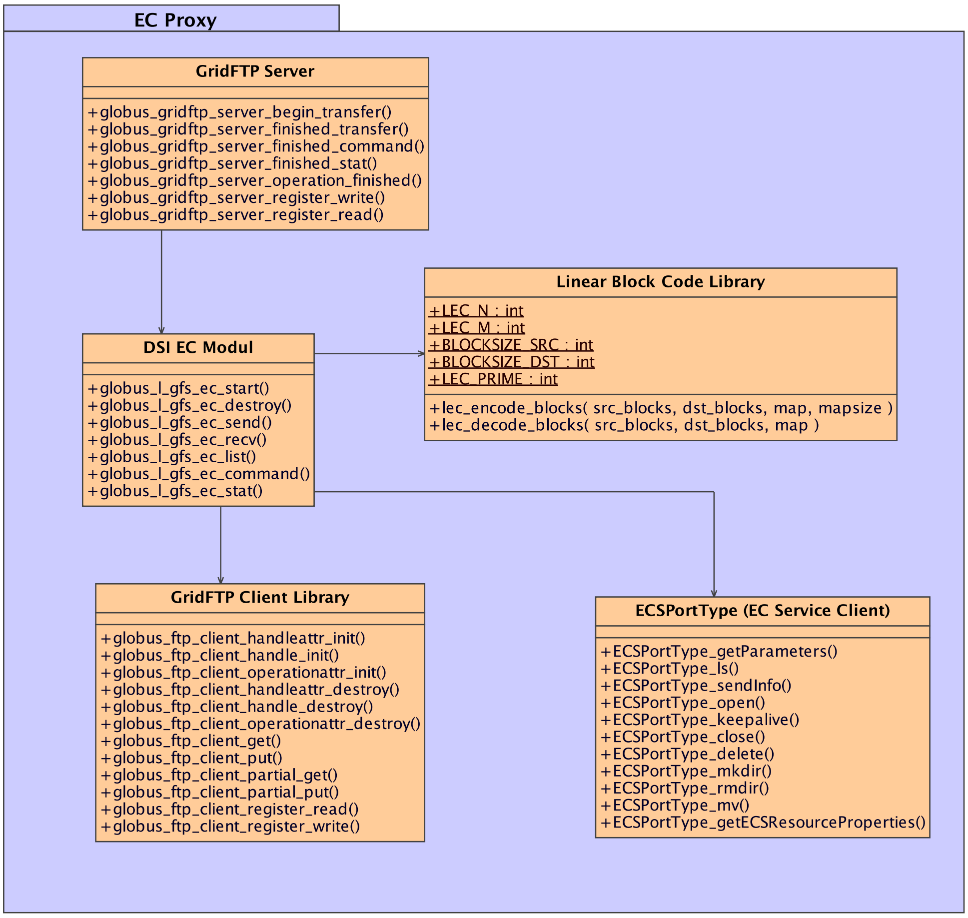

Abbildung 5.15: Architektur des EC Proxy

5.3.2 GridFTP Server

Die Header Datei *grid_ftp_server.h* deklariert alle Funktionen, die benötigt werden um GridFTP Server Module zu schreiben. Das DSI-Modul nutzt die folgenden Funktionen, um den GridFTP-Server über den Zustand von Datenübertragungen zu informieren und um Puffer zu senden oder zu empfangen.

globus_gridftp_server_begin_transfer(). Das DSI-Modul informiert den GridFTP Server über den Beginn einer Datenübertragung.

globus_gridftp_server_finished_transfer(). Das DSI-Modul informiert den GridFTP-Server, dass die Datenübertragung beendet wurde. Falls sie nicht erfolgreich war, kann der Fehler angegeben werden.

globus_gridftp_server_finished_command(). Die Funktion signalisiert die Ausführung von *globus_l_gfs_ec_command()*.

globus_gridftp_server_finished_stat(). Die Funktion signalisiert die Ausführung von *globus_l_gfs_ec_stat()*.

globus_gridftp_server_operation_finished(). Die Funktion signalisiert die Ausführung von *globus_l_gfs_ec_start()*.

globus_gridftp_server_register_read(). Die Funktion empfängt Daten vom GridFTP-Server, d.h. das DSI-Modul übergibt dem GridFTP-Server einen leeren Puffer, der dann mit Daten, die vom Client empfangen werden, gefüllt wird.

globus_gridftp_server_register_write(). Die Funktion sendet Daten an den GridFTP-Server, d.h.das DSI Modul übergibt dem GridFTP Server einen Puffer, der mit Daten gefüllt ist, die dann an den Client gesendet werden.

Bemerkung: Die beiden Funktionen *globus_gridftp_server_register_read()* und *globus_gridftp_server_register_write()* sind nicht-blockierende Funktionen. Sie registrieren den Lese- oder Schreibvorgang im Framework des GridFTP-Servers und kehren sofort zurück. Das Framework führt den Vorgang im Hintergrund in einem anderen Thread durch. Hier im Entwurf reicht es, sich die Funktionen als blockierend vorzustellen. Auf die parallele Ausführung wird in Kapitel 6.1.2 eingegangen.

5.3.3 Linear Block Code Library

Die Library zur Kodierung der Blöcke besitzt vier öffentliche Attribute und zwei öffentliche Funktionen. Die spätere Implementierung wird auf bestimmte Werte der Attribute angewiesen sein. Deshalb sind diese hier als Konstanten definiert.

LEC_N, LEC_M. *LEC_N und LEC_M* geben die Größe der Kodiermatrix an. Aus *LEC_N* Blöcken werden durch Anwendung der Matrix maximal *LEC_M* Blöcke.

BLOCKSIZE_SRC, BLOCKSIZE_DST. Durch die Transformation der binären Datendarstellung zur Darstellung in einem Restklassenring geht Speicherplatz verloren (siehe Kapitel 6.1.1). Blöcke nehmen auf den Datenspeichern geringfügig mehr Speicherplatz ein. *BLOCKSIZE_SRC* ist die Größe eines Blocks in der ursprünglichen Datei. *BLOCKSIZE_DST* ist die Größe eines Blocks innerhalb einer Blockgruppe auf einem Datenspeicher.

lec_encode_blocks(src_blocks, dst_blocks, map, mapsize). Blöcke enkodieren. Es können auch weniger als *LEC_M Blöcke* erstellt werden. Die Argumente der Funktion bedeuten das Folgende:

- *src_blocks:* zu kodierende Blöcke,
- *dst_blocks:* zu erstellende Blöcke.
- *map:* Liste der zu erstellenden Blöcke. Jede Zahl dieser Liste bestimmt den Index des entsprechenden Blocks.
- *mapsize:* Länge der Liste bzw. Anzahl der zu erstellenden Blöcke.

lec_decode_blocks(src_blocks,dst_blocks, map). Blöcke dekodieren. Von den maximal *LEC_M* kodierten Blöcken werden genau *LEC_N* Blöcke benötigt. Die Argumente der Funktion bedeuten folgendes:

- *src_blocks:* Zu dekodierende Blöcke.
- *dst_blocks:* Zu erstellende Blöcke.
- **map:** Liste der vorhandenen Blöcke. Gibt zu jedem Element aus **src_blocks** den Index des Blocks an.

5.3.4 ECSPortType

Mit dem Globus Toolkit kann man natürlich auch C Stubs aus WSDL-Dokumenten erstellen. Der EC Proxy nutzt Client Stubs, die aus der WSDL-Definition der Schnittstelle des EC Service generiert werden. Der Programmierer kann die Client Stubs auf die gleiche Art und Weise verwenden wie lokale Strukturen und Funktionen. Er braucht sich nicht um die Details der Netzwerkübertragung zu kümmern. Um Kollisionen bei den Funktionsnamen zu vermeiden, wird allen Funktionen der Name der Schnittstelle vorangestellt.

5.3.5 GridFTP Client Library

Der EC Proxy nutzt die GridFTP Client Library des Globus Toolkit [ALL07], um auf die Datenspeicher zuzugreifen. Aufgrund der beschränkten Nutzung des GridFTP Protokolls werden neben den Funktionen zur Initialisierung nur Funktionen zum Empfangen und Senden von Dateien aufgerufen.

globus_ftp_client_handleattr_init, globus_ftp_client_handle_init(), globus_ftp_client_operationattr_init(), globus_ftp_client_handleattr_destroy, globus_ftp_client_handle_destroy(), globus_operationattr_destroy(). Initialisierung und Freigabe von Strukturen für die Verwaltung von GridFTP-Verbindungen.

globus_ftp_client_get, globus_ftp_client_partial_get(). Vollständiges bzw. teilweises Empfangen einer Datei.

globus_ftp_client_put, globus_ftp_client_partial_put(). Vollständiges bzw. teilweises Senden einer Datei.

globus_ftp_client_register_read(). Diese Funktion füllt einen Puffer mit Daten, die von einem GridFTP-Server empfangen werden.

globus_ftp_client_register_write(). Diese Funktion überträgt den Inhalt eines Puffers an einen GridFTP Server.

Bemerkung: *globus_ftp_client_register_read()* und *globus_ftp_client_register_write()* sind nicht-blockierende Funktionen, werden aber in diesem Entwurf als blockierend angesehen (siehe oben).

5.3.6 DSI EC Modul

Der GridFTP-Server des Globus Toolkit greift über ein Data Storage Modul auf das Dateisystem zu, indem er Funktionen des Moduls aufruft. Durch vollständige Neuimplementierung dieser Funktionen wird der GridFTP-Server zum EC Proxy.

globus_l_gfs_ec_start(). Der GridFTP-Server ruft diese Funktion immer dann auf, wenn eine neue Sitzung begonnen wird. Das DSI-Modul kann hier eine eigene Datenstruktur initialisieren, die allen nachfolgenden Funktionen als Parameter übergeben wird. Auf diese Weise erhält das DSI einen Zustand.

Das DSI EC Modul speichert in dieser Struktur:

- Informationen über die Datei, die gerade bearbeitet wird,
- Verteilungsparameter,
- Informationen über die Verbindung zum EC Service,
- Zustand der Datenübertragung,
- Informationen über bestehende Verbindungen zu Datenspeichern,
- für jeden beteiligten Datenspeicher und den Client einen Puffer für die Zwischenspeicherung von Blöcken.

Bei der Initialisierung dieser Struktur werden die nötigen Speicherbereiche und der Client für den EC Service initialisiert.

globus_l_gfs_ec_destroy(). Bei Beenden einer Sitzung wird der Zustand des DSI aufgehoben, d.h.die Struktur, die zu Beginn angelegt wurde, wird wieder freigegeben. Alle Verbindungen, die eventuell noch zu Datenspeichern oder zum EC Service bestehen, werden getrennt.

globus_l_gfs_ec_send(). Diese Funktion implementiert den Use Case *Datei lesen*. Sie wird aufgerufen, wenn der GridFTP-Server eine Datei sendet. Im Allgemeinen wird sie durch den Client ausgelöst, der eine Datei empfangen will.

Der folgender Pseudocode zeigt, wie die Übertragung stattfindet.

Listing 5.1: globus_l_gfs_ec_send()

```
openResponse = ECSPortType_open(filename,"r");
globus_gridftp_server_begin_transfer();
for (all blockgroups) {
  globus_ftp_client_get(blockgroup->url);
}
while (!end of file) {
  for (all storage_elements) {
    globus_ftp_client_register_read(storage_element_buffer);
  }
  if (enough blocks in storage_element_buffers) {
    lec_decode_blocks(storage_element_buffers, client_buffer, blockgroup_indices);
  }
  if (data in client_buffer) {
```

```
    globus_gridftp_server_register_write(client_buffer)
  }
  if (timeout passed) {
    ECSPortType_keepalive(openResponse->handle);
  }
}
globus_gridftp_server_finished_transfer();
ECSPortType_close(openResponse->handle);
```

Bemerkung: Die sequentielle Implementierung aus Listing 5.1 ist zwar möglich aber nicht leistungsfähig. Der GridFTP-Server unterstützt parallele Verarbeitung. In Kapitel 6.1.4 wird darauf eingegangen.

globus_l_gfs_ec_recv(). Diese Funktion implementiert die Use Cases *Datei schreiben* und *Datei erstellen*. Sie wird aufgerufen, wenn der GridFTP-Server eine Datei empfängt. Im Allgemeinen wird sie durch den Client ausgelöst, der eine Datei senden will.

Der folgender Pseudocode zeigt, wie die Übertragung stattfindet.

Listing 5.2: globus_l_gfs_ec_recv()

```
openResponse = ECSPortType_open(filename,"c");
globus_gridftp_server_begin_transfer();
for (all blockgroups) {
  globus_ftp_client_put(blockgroup->url);
}
while (!end of file) {
  globus_gridftp_server_register_read(client_buffer)
  if (enough blocks in client_buffer) {
    lec_encode_blocks(client_buffer, storage_element_buffers, blockgroup_indices);
  }
  for (all storage_elements) {
    if (date in storage_element_buffer) {
      globus_ftp_client_register_read(storage_element_buffer);
    }
  }
  if (timeout passed) {
    ECSPortType_keepalive(openResponse->handle);
  }
}
globus_gridftp_server_finished_transfer();
ECSPortType_close(openResponse->handle);
```

Bemerkung: *Partial Put* ist hier nicht berücksichtigt.

globus_l_gfs_ec_list(). Verzeichnisse können viele Dateien und Unterverzeichnisse enthalten. Der Inhalt des Verzeichnis wird auf die gleiche Weise übertragen wie der Inhalt von Dateien.

Listing 5.3: globus_l_gfs_ec_list()

```
lsResponse=ECSPortType_ls(directory_name);
globus_gridftp_server_begin_transfer();
for (all lsResponse->fileInfo) {
    globus_gridftp_server_register_write(fileInfo)
}
for (all lsResponse->dirInfo) {
    globus_gridftp_server_register_write(dirInfo)
}
globus_gridftp_server_finished_transfer();
```

globus_l_gfs_ec_command(). Diese Funktion fasst mehrere GridFTP Befehle zusammen.

Listing 5.4: globus_l_gfs_ec_command()

```
switch(cmd_info->command) {
  case GLOBUS_GFS_CMD_MKD:
    ECSPortType_mkdir(cmd_info->pathname);
    break;
  case GLOBUS_GFS_CMD_RMD:
    ECSPortType_rmdir(cmd_info->pathname);
    break;
  case GLOBUS_GFS_CMD_DELE:
    ECSPortType_delete(cmd_info->pathname);
    break;
  case GLOBUS_GFS_CMD_RNTO:
    ECSPortType_mv(cmd_info->rnfr_pathname,cmd_info->pathname);
    break;
}
```

globus_l_gfs_ec_stat(). Mit dieser Funktion ruft der GridFTP Server Dateiinformationen ab wie zum Beispiel das Datum der letzten Änderung oder die Zugriffsrechte. Die Funktion orientiert sich dabei am POSIX-Standard. Das DSI EC-Modul greift nicht auf ein Dateisystem zu und Zugriffsrechte werden über den CAS vergeben. Deshalb werden die entsprechende Felder mit Standardwerten belegt.

5.4 Entwurf des EC Service

5.4.1 Architektur

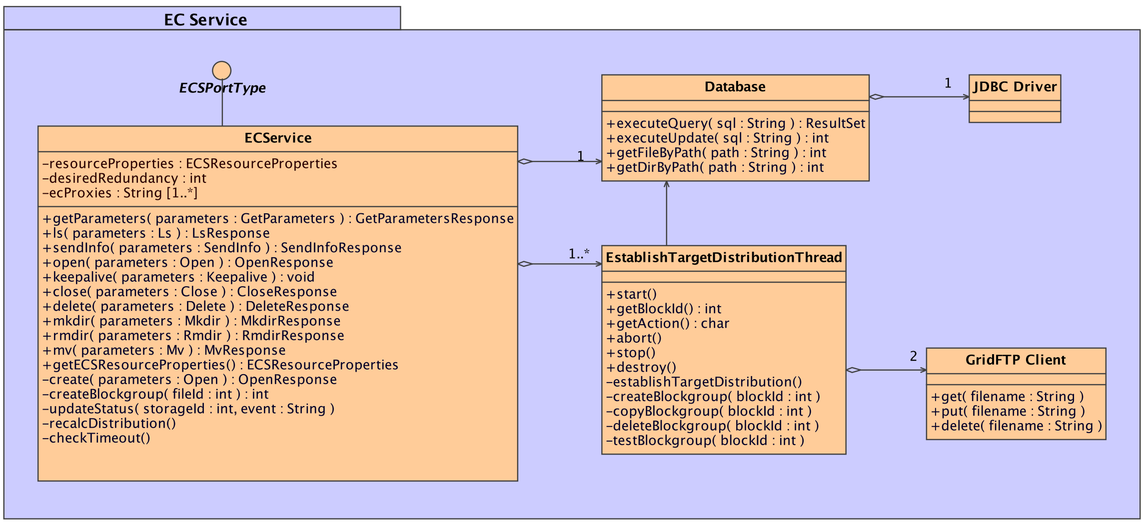

Abbildung 5.16: Architektur des EC Service

Wie man Abbildung 5.16 entnehmen kann, ist der EC Service selbst als Klasse entworfen, die das Interface *ECSPortType* implementiert. Das heißt, die Klasse *ECService* beinhaltet die Funktionen, die in der Schnittstellenbeschreibung definiert wurden. Zusätzlich gibt es einige private Methoden.

Der Service speichert die Informationen über die Datenverteilung in einer SQL Datenbank, die über JDBC angebunden wird. Sie wird in der Klasse *Database* gekapselt. Wichtige SQL-Queries, die häufig vorkommen, werden als Methoden der *Database*-Klasse implementiert. Man kann aber auch direkt SQL-Queries und SQL-Updates ausführen lassen.

Da das System auch ansprechbar sein muss, während der Use Case *Sollzustand herstellen* ausgeführt wird, gibt es eine eigene Klasse *EstablishTargetDistributionThread*, die in einer oder mehreren Instanzen im Hintergrund, jeweils in einem eigenen Thread, läuft. Jeder dieser Threads kann bis zu zwei GridFTP-Server kontaktieren und Blockgruppen kopieren, erstellen und löschen. Die Datenübertragung wird im Allgemeinen von GridFTP-Server zu GridFTP-Server als *Third Party Transfer* durchgeführt.

Bevor ich die einzelnen Klassen und ihre Methoden beschreiben kann, sind noch ein paar allgemeine Betrachtungen notwendig.

5.4.2 Datenbankschema

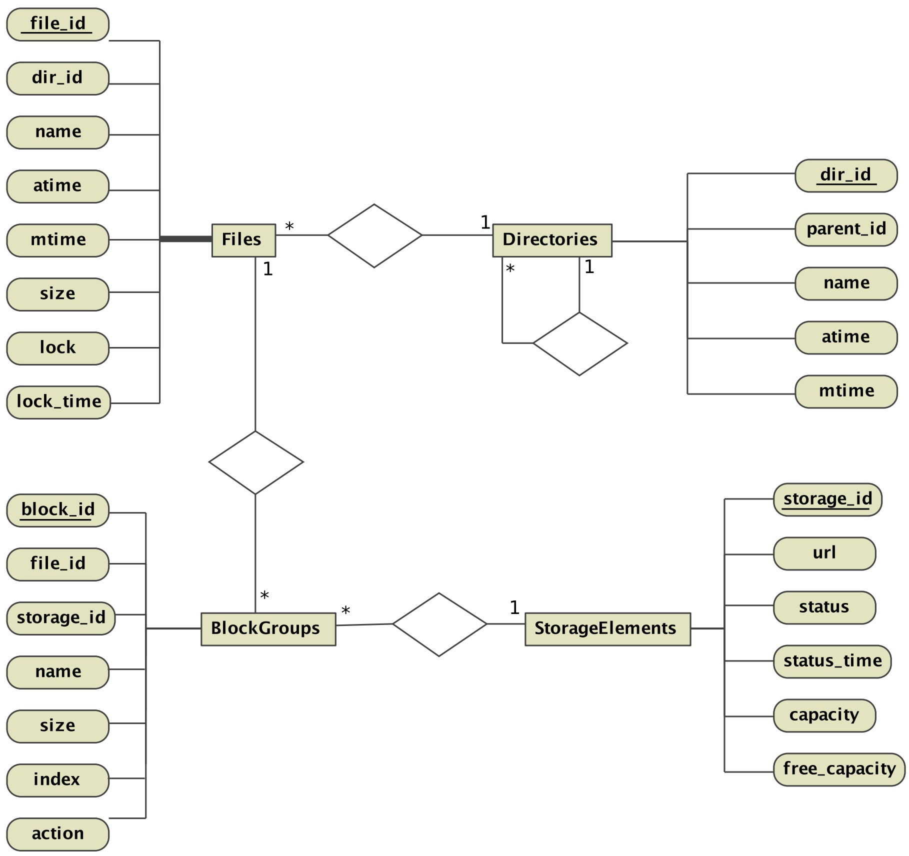

Abbildung 5.17: Datenbankschema des EC Service

Abbildung 5.17 zeigt das Datenbankschema als Entity-Relationship-Diagramm. Es besteht aus vier Tabellen: *Directories*, *Files*, *BlockGroups*, *StorageElements*. Jede Tabelle enthält als erstes Attribut einen Identifikator, der automatisch für jeden neuen Eintrag vergeben wird und Primary Key ist. Relationen zwischen den Tabellen werden über diesen Identifikator hergestellt.

Directories-Tabelle

Die Verzeichnisstruktur ist ein Baum, in dem jedes Verzeichnis einen Knoten bildet. Die Abbildung des Baumes im relationalen Modell beruht darauf, dass jeder Kindknoten in einer Relation zu seinem Elternknoten steht. Die Tabelle enthält ein Attribut, mit dem der Elternknoten referenziert wird. Insgesamt enthält die Tabelle *Directories* die folgenden Attribute:

- *dir_id:int.* Identifikator.

- *parent_id:int* referenziert den Elternknoten. Ein Verzeichnis beinhaltet alle Unterverzeichnisse, deren *parent_id* gleich seiner *dir_id* ist. Ein Verzeichnis dessen *parent_id* gleich 0 ist, befindet sich im *root* Verzeichnis.
- *name:string.* Name des Verzeichnisses ohne Pfadangabe. Der komplette Pfad setzt sich aus allen Namen der übergeordneten Verzeichnisse zusammen.
- *atime:int.* Zeit des letzten Zugriffs.
- *mtime:int.* Zeit der letzten Änderung.

Ein Verzeichnis darf nicht mehrere Unterverzeichnisse mit demselben Namen enthalten. Deshalb sollte *(parent_id,name)* Schlüsseleigenschaften besitzen. Verzeichnisse können inklusive Inhalt verschoben werden, indem die *parent_id* geändert wird. Um Zyklen zu vermeiden, ist darauf zu achten, dass Verzeichnisse nicht in eines ihrer Unterverzeichnisse verschoben werden.

Files-Tabelle

Jeder Ordner kann Dateien aufnehmen. Die Tabelle *Files* hat die folgenden Attribute:

- <u>*file_id:int.*</u> Identifikator.
- *dir_id:int.* Ordner, in dem die Datei liegt.
- *name:string.* Name ohne Pfadangabe.
- *atime:int.* Zeit des letzten Zugriffs.
- *mtime:int.* Zeit der letzten Änderung.
- *size:int.* Dateigröße in Byte.
- *lock:char.* Zugriffssperre. Mögliche Werte:
"n" Keine Sperre.
"r" Lesesperre (read). Ein Benutzer liest momentan aus der Datei.
"w" Schreibsperre (write). Ein Benutzer schreibt in die Datei.
"c" Erstellsperre (create). Datei wird gerade angelegt.
- *lock_time.* Zeitpunkt des Anlegens oder des letzten Keep-alive der Sperre. Reagiert der EC Proxy nicht mehr, kann die Sperre nach einem Timeout aufgehoben werden.

BlockGroups-Tabelle

Jede Datei besteht aus bis zu *LEC_M* Blockgruppen. Die Tabelle *BlockGroups* hat die folgenden Attribute:

- <u>*block_id:int.*</u> Identifikator.
- *file_id:int.* Datei, zu der diese Blockgruppe gehört.
- *storage_id:int.* Datenspeicher, auf dem diese Blockgruppe gespeichert ist.
- *name:string.* Dateiname auf dem Datenspeicher.
- *size:int.* Dateigröße auf dem Datenspeicher.
- *index:int.* Zeile der Kodiermatrix, mit der diese Blockgruppe erstellt wurde.
- *action:char.* Geplante Aktion für die Blockgruppe. Der Istzustand setzt sich aus allen Blockgruppen zusammen, für die keine Aktion geplant ist. Der Sollzustand entsteht, wenn alle Aktionen ausgeführt wurden. Mögliche Aktionen:
"n" Keine Aktion, Blockgruppe existiert und soll bestehen bleiben.
"d" Blockgruppe existiert, soll aber gelöscht werden.
"c" Blockgruppe existiert nicht und soll erstellt werden.
"m" Blockgruppe existiert, soll aber auf einen anderen Datenspeicher verschoben werden.

"t" Datenspeicher testen. Keine reale Blockgruppe (ungültige *file_id*). Dient zur Signalisierung, dass ein Datenspeicher getestet werden soll, indem eine Testdatei hochgeladen, heruntergeladen und wieder gelöscht wird.

StorageElements-Tabelle

Zusätzlich zum MDS Index-Dienst werden die Datenspeicher in einer Tabelle der Datenbank verwaltet. Dadurch kann innerhalb der Datenbank eine Relation zwischen Blockgruppen und Datenspeichern geschaffen werden. Vorübergehend ausgefallene oder fehlerhafte Datenspeicher bleiben in der Datenbank bestehen. Die Tabelle *StorageElements* hat die folgenden Attribute:

- *storage_id:int.* Identifikator.
- *url:string.* Adresse des Datenspeichers inklusive des Verzeichnisses, in dem alle Blockgruppen gespeichert werden sollen.
- *status:char.* Status des Datenspeichers (siehe unten). Mögliche Werte:
  "n": Normal (OK).
  "b": Ausgelastet (Busy).
  "u": Unzuverlässig (Unreliable).
  "d": Ausgefallen (Down).
  "r": Soll entfernt werden (ToBeRemoved).
- *status_time:int.* Zeitpunkt der letzten Statusaktualisierung. Wird für Statusänderungen gebraucht, die durch ein Timeout verursacht werden.
- *capacity:int.* Gesamter Speicherplatz.
- *free_capacity:int.* Freier Speicherplatz.

5.4.3 Verteilungsstrategie

Wie wir gesehen haben, gibt es einige Ereignisse, die sich auf die Verteilung der Blockgruppen auswirken. Das System muss auf Fehlverhalten von Datenspeichern, auf Ausfälle und auf neu angelegte oder gelöschte Dateien reagieren. Zusätzlich wirkt sich der freie Speicherplatz auf die Datenverteilung aus. Dieser Abschnitt umreißt eine mögliche Strategie für diese Ereignisse.

Status von Datenspeichern

Wie bereits in Kapitel 5.1.8 erwähnt, wird Datenspeichern ein Status zugewiesen, der Genaueres über die aktuelle Verfügbarkeit aussagt. Die Ereignisse, die zu Statuswechseln führen, und die Werte, die der Status annehmen kann, werden am besten anhand eines Zustandsdiagramms dargestellt.

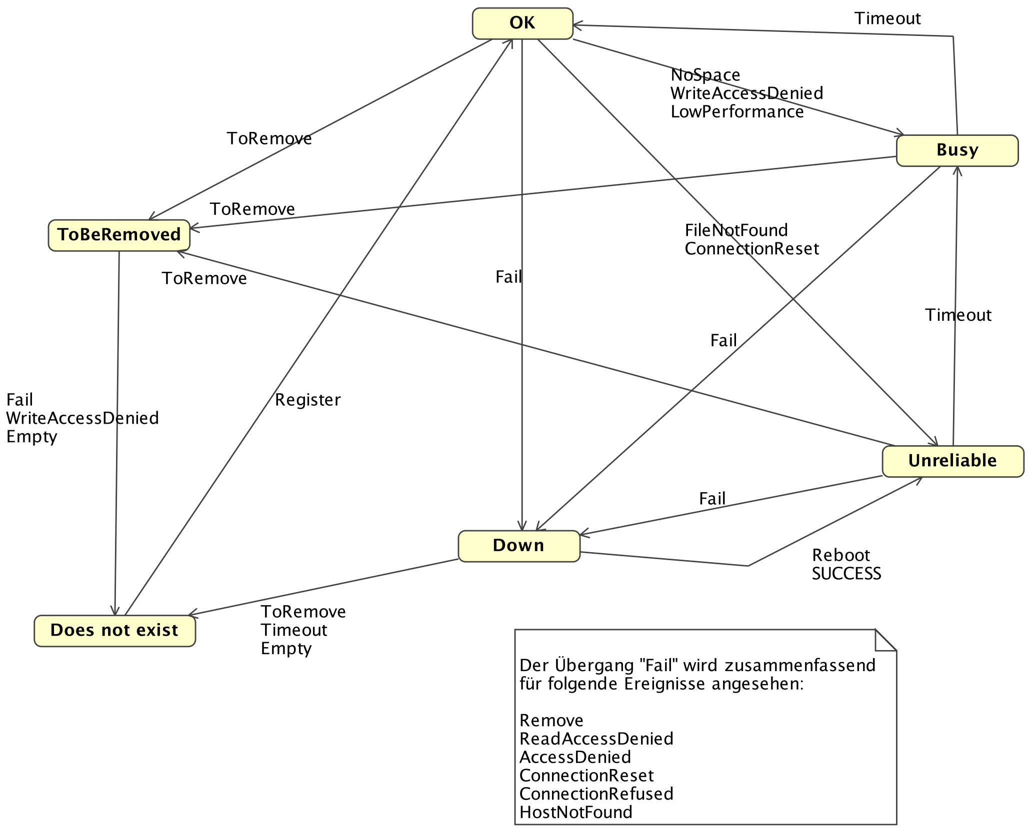

Abbildung 5.18: Status von Datenspeichern

Abbildung 5.18 zeigt die sechs Zustände, die ein Datenspeicher annehmen kann, und die Übergänge, die zu einem Statuswechsel führen:

- *Does not exist:* Datenspeicher ist nicht registriert und kommt in der Datenbank nicht vor. Blockgruppen, die sich eventuell von einer früheren Teilnahme noch auf ihm befinden sind unbrauchbar, da sie nicht mehr in der Datenbank vorhanden sind.
- *OK:* Datenspeicher funktioniert ordnungsgemäß und wird verwendet.
- *Busy:* Datenspeicher ist vorübergehend beeinträchtigt. Er wird weiterhin benutzt, neue Blockgruppen werden aber nicht angelegt.
- *Unreliable:* Es gibt eine erhöhte Ausfallwahrscheinlichkeit. Der Datenspeicher wird zwar weiterhin verwendet, die Redundanz der zugehörigen Datei wird aber vorsichtshalber erhöht.
- *ToBeRemoved:* Der Anbieter eines Datenspeichers kann diesen absichtlich als zu entfernen markieren. Alle Blockgruppen werden automatisch auf andere Datenspeicher verschoben. Der Anbieter kann den Datenspeicher gefahrlos entfernen, sobald sich keine Dateien der VO mehr auf ihm befinden.
- *Down:* Wenn ein Datenspeicher einen schwerwiegenden Fehler verursacht hat oder aus dem MDS-Index entfernt wurde, gilt er als ausgefallen. Falls der Datenspeicher wieder erscheint, gilt er zunächst als unzuverlässig, da mit weiteren Ausfällen gerechnet werden muss.

Ereignisse, die zu einem Statuswechsel führen

Es gibt vier Situationen, die zu einem Statuswechsel führen können:

- Das MDS meldet das Erscheinen, Ausfallen oder Ändern eines Datenspeichers.

- Der EC Proxy meldet ein Fehlverhalten eines Datenspeichers. Er ruft dazu die Operation *sendInfo()* des EC Service auf.
- Beim direkten Zugriff auf einen Datenspeicher tritt ein Fehler auf.
- Der Status wird aufgrund eines Timeouts verändert.

Folgende Ereignisse können auftreten:

- *Register*. Neuer Datenspeicher.
- *Reboot*. Bereits eingetragener Datenspeicher wird erneut registriert.
- *Remove*. Datenspeicher aus dem MDS entfernt.
- *ToRemove*. Datenspeicher soll entfernt werden.
- *NoSpace*. Kein freier Speicherplatz.
- *Empty*. Keine Daten der VO vorhanden.
- *ReadAccessDenied*. Lesezugriff verweigert.
- *WriteAccessDenied*. Schreibzugriff verweigert.
- *AccessDenied*. Verbindungsaufbau verweigert.
- *FileNotFound*. Datei nicht gefunden.
- *ConnectionReset*. Verbindung wurde unterbrochen.
- *ConnectionRefused*. TCP Port für GridFTP gesperrt
- *LowPerformance*. Datenübertragung ist viel langsamer als zu anderen Datenspeichern.
- *HostNotFound*. Datenspeicher ist nicht im DNS verzeichnet.
- *Timeout*. Statuswechsel aufgrund von verstrichener Zeit seit dem letzten Statuswechsel.
- *SUCCESS*. Ein Test war erfolgreich.

Neue Blockgruppen

Damit der Gesamtspeicherplatz möglichst vollständig genutzt werden kann, wird jede neue Blockgruppe auf dem Datenspeicher mit dem größten freien Speicherplatz angelegt. Es sei denn, der Datenspeicher enthält schon eine andere Blockgruppe derselben Datei. Dann wird der nächste Datenspeicher verwendet, usw. Auf diese Weise enthält kein Datenspeicher mehr als eine Blockgruppe pro Datei. Daraus folgt aber auch, dass es eine Mindestanzahl von Datenspeichern geben muss. Es muss vermieden werden, dass Blockgruppen mehrfach vorkommen. Deshalb wird für jede neue Blockgruppe ein neuer Index verwendet, der unter den Blockgruppen der Datei noch nicht vorkommt. Dabei werden natürlich auch ausgefallene Datenspeicher miteinbezogen.

Sollzustand berechnen

Der Sollzustand wird neu berechnet, indem alle Dateien überprüft werden. Für jede Datei wird der aktuelle Istzustand und der Sollzustand kontrolliert und möglicherweise angepasst.

Folgende Situationen können dabei auftreten:

- Niedrige Redundanz. Die Anzahl verfügbarer Blockgruppen ist kleiner als erwünscht. Neue Blockgruppen müssen (nach oben genanntem Verfahren) angelegt werden.
- Erhöhte Redundanz. Die Anzahl verfügbarer Blockgruppen ist größer als erwünscht. Diese Situation kommt durch das Wiedererscheinen zuvor ausgefallener Datenspeicher zustande. Falls die Redundanz stark erhöht ist, sollten Blockgruppen gelöscht werden.

- Unsinniger Sollzustand. Es macht keinen Sinn, zu versuchen Blockgruppen auf Datenspeichern abzulegen, die sich nicht im Zustand *OK* befinden. Der Sollzustand muss in diesem Fall korrigiert werden.
- Datenspeicher soll entfernt werden. Im Sollzustand muss eingetragen werden, dass die Blockgruppen verschoben werden sollen.

5.4.4 Database-Klasse

In den folgenden Abschnitten werden die Klassen des EC Service beschrieben (siehe Abbildung 5.16).

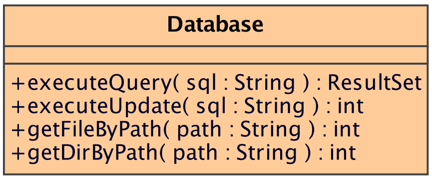

Abbildung 5.19: Database-Klasse

Die *Database* Klasse (Abbildung 5.19) besteht aus vier Methoden für den Zugriff auf die Datenbank.

Database.executeQuery(sql:String):ResultSet. Eine SQL Abfrage ausführen.

Database.executeUpdate(sql:string):int. Ein SQL Update durchführen.

Database.getFileByPath(path:String):int. Dateien mit gleichem Namen dürfen in unterschiedlichen Verzeichnissen vorkommen. Diese Methode berücksichtigt alle Verzeichnisse und Unterverzeichnisse und gibt die *file_id* der Datei zurück, die durch den Dateinamen inklusive Pfadangabe gegeben ist.

Database.getDirByPath(path:String):int. Diese Methode gibt die *dir_id* zurück, die durch den Verzeichnisnamen inklusive Pfadangabe *path* gegeben ist.

5.4.5 EstablishTargetDistributionThread-Klasse

Die Threads, die den Sollzustand herstellen, befinden sich alle in einer Schleife, in der jeweils eine Blockgruppe bearbeitet wird. Jeder Thread sucht sich eine Blockgruppe aus dem Sollzustand, führt die entsprechende Aktion aus und wiederholt diesen Schritt für die nächste Blockgruppe. Die Schleife läuft so lange, bis der Server heruntergefahren wird. Ein Thread kann aber angehalten werden. Dies geschieht entweder explizit durch den Befehl *stop()*, oder automatisch, wenn der Sollzustand hergestellt ist.

Je mehr Instanzen aktiv sind, desto mehr Transfers können gleichzeitig ausgeführt werden. Es ist jedoch darauf zu achten, dass der EC Service nicht das gesamte Netzwerk überlastet. Die optimale Anzahl sollte später in praktischen Tests ermittelt werden.

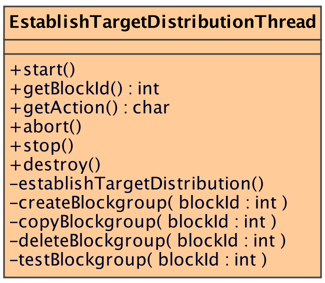

Abbildung 5.20: EstablishTargetDistributionThread-Klasse

Es folgt die Beschreibung der Methoden aus Abbildung 5.20

EstablishTargetDistributionThread.start(). Thread starten, falls er nicht schon läuft. Wird immer dann aufgerufen, wenn der Sollzustand geändert wurde.

EstablishTargetDistributionThread.getBlockId():int. Identifikator der Blockgruppe, die gerade bearbeitet wird.

EstablishTargetDistributionThread.getAction():char. Aktion, die gerade ausgeführt wird. (siehe Datenbankschema)

EstablishTargetDistributionThread.abort(). Falls der Benutzer auf eine Datei zugreifen will, deren Blockgruppen gerade bearbeitet werden, kann es zum Konflikt kommen. Der Benutzer soll Vorrang bekommen. Deshalb kann die aktuelle Aktion abgebrochen werden. Der Hintergrund-Thread springt dann zur nächsten Blockgruppe, die zu einer anderen Datei gehört.

EstablishTargetDistributionThread.stop(). Aktuelle Aktion abbrechen und Thread so lange anhalten, bis wieder *start()* aufgerufen wird.

EstablishTargetDistributionThread.destroy(). Aktuelle Aktion abbrechen und Thread beenden. Wird erst beim Herunterfahren des Servers aufgerufen.

EstablishTargetDistributionThread.establishTargetDistribution(). Endlosschleife zur Herstellung des Sollzustandes. Folgender Pseudocode verdeutlicht die Funktionsweise.

Listing 5.5: establishTargetDistribution()

```
while (program running)
{
  // get information about one blockgroup, that satisfies following properties
  // blockgroup is part of target distribution
  // status of the corresponding storage element is normal
  // file is not locked
  database.executeQuery(SELECT *
    FROM Blockgroups Files StorageElements
    WHERE action!='n' and status='n' and lock='n');

  if (blockgroup found)
  {
    switch (blockgroup.action)
    {
      case "c": createBlockgroup();
      case "m": copyBlockgroup(); deleteBlockgroup();
      case "d": deleteBlockgroup();
      case "t": testBlockgroup(); // originally not part of use cases
    }
    if (success)
    {
       // mark as beeing part of current distribution
      database.executeUpdate(UPDATE Blockgroups
         SET action='n' WHERE block_id=blockgroup);
    }
    else // error or action aborted
    {
      // try again later
    }
  }
  if ( (no blockgroup found) // target distribution established
     || (stop() called)
  {
    suspendThread();
  }
}
```

EstablishTargetDistributionThread.createBlockgroup(blockId:int). Implementierung des Use Case *Blockgruppe erstellen*.

Listing 5.6: createBlockgroup(blockId:int)

```
database.executeQuery(SELECT * FROM Blockgroups Files WHERE block_id=blockId);
gridFTPClient.get(ecproxy + file.name +blockgroup.index);
gridFTPClient.put(blockgroup.url);
```

EstablishTargetDistributionThread.copyBlockgroup(blockId:int). Implementierung des Use Case *Blockgruppe kopieren*.

Listing 5.7: copyBlockgroup(blockId:int)

```
database.executeQuery(SELECT * FROM Blockgroups WHERE block_id=blockId);
gridFTPClient.get(blockgroup.from);
gridFTPClient.put(blockgroup.to);
```

EstablishTargetDistributionThread.deleteBlockgroup(blockId:int). Implementierung des Use Case *Blockgruppe löschen*.

Listing 5.8: deleteBlockgroup(blockId:int)

```
database.executeQuery(SELECT * FROM Blockgroups WHERE block_id=blockId);
gridFTPClient.delete(blockgroup.name);
```

EstablishTargetDistributionThread.testBlockgroup(blockId:int). Datenspeicher müssen regelmäßig getestet werden. Die Tests sollen natürlich auch im Hintergrund stattfinden. Durch eine spezielle Art von Blockgruppe, die keine Zugehörigkeit zu einer Datei hat, wird der *EstablichTargetDistributionThread* angewiesen einen Datenspeicher zu testen. Er überprüft anhand einer kleinen Probedatei, ob Blockgruppen zu und von dem Datenspeicher übertragen werden können. Außerdem überprüft er, ob noch alle Blockgruppen vorhanden sind.

Listing 5.9: testBlockgroup(blockId:int)

```
database.executeQuery(SELECT * FROM Blockgroups WHERE block_id=blockId);
// test for connectivity and access rights
gridFTPClient.put(testfile);
gridFTPClient.get(testfile);
if (! test successfull)
{
  updateStatus(blockgroup.storage_id, error);
}
// test for missing blockgroups
database.executeQuery(SELECT * FROM Blockgroups, StorageElements
     WHERE block_id=blockId);
gridFTPClient.list(url);
if (lists not identical)
  updateStatus(blockgroup.storage_id, FileNotFound);
```

5.4.6 ECService-Klasse

Die *ECService*-Klasse implementiert die Schnittstelle und enthält die gesamte Logik des EC Service.

Neben den Resource Properties, die sich dynamisch verändern, gibt es auch Parameter, die manuell in einer Konfigurationsdatei festgelegt werden.

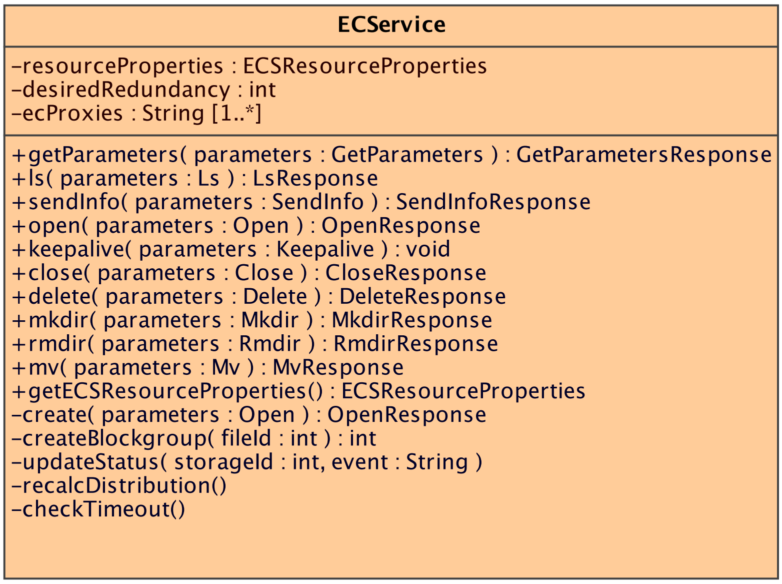

Abbildung 5.21: ECService-Klasse

ECService.desiredRedundancy. Um die Redundanz von Dateien wiederherzustellen, muss der EC Service wissen, wie viele Blockgruppen pro Datei gewünscht sind. Der Wert *desiredRedundancy* gibt die gewünschte Redundanz in Prozent an. Je höher dieser Wert ist, desto sicherer sind die VO-kritischen Daten und desto mehr Speicherplatz wird benötigt. Die VO legt diesen Wert anhand ihrer Anforderungen an das System fest.

ECService.ec_proxies. Der EC Service muss über eine Liste von EC Proxies verfügen. VO-kritische Daten passieren die EC Proxies unverschlüsselt. Deshalb halte ich eine automatische Registrierung über das MDS für nicht sicher genug und habe ich mich für eine manuelle Eintragung entschieden.

Im folgenden werden die Methoden der ECService-Klasse beschrieben.

ECService.getParameters(parameters:GetParameters):GetParametersResponse. Die Verteilungsparameter einer Datei setzen sich aus einem Eintrag in der Tabelle *Files* und aller zugehörigen Einträge der Tabelle *BlockGroups* zusammen. Blockgruppen auf Datenspeichern, die ausgefallen sind, werden nicht berücksichtigt. Blockgruppen auf unbeeinträchtigten Datenspeichern werden zuerst genannt.

Listing 5.10: getParameters(parameters:GetParameters):GetParametersResponse

```
file=database.getFileByPath(parameters.filename);
database.executeQuery(SELECT * FROM Files WHERE file_id=file.file_id);
// select all blockgroups of the file, and sort according to status
database.executeQuery(SELECT * FROM BlockGroups StorageElements
     WHERE file_id=file and action='n' and status!='d' sort by status);
```

ECService.ls(parameters:Ls):LsResponse. Auflistung aller Dateien und Unterverzeichnisse im angegebenen Verzeichnis.

Listing 5.11: ls(parameters:Ls):LsResponse

```
directory = database.getDirByPath(parameters.dirname)
database.executeQuery(SELECT * FROM Files WHERE dir_id=directory);
database.executeQuery(SELECT * FROM Directories WHERE parent_id=directory);
```

ECService.sendInfo(parameters:Sendinfo):SendInfoResponse. Benachrichtigung über ein Fehlverhalten eines Datenspeichers. Im Allgemeinen hat dies einen Statuswechsel des entsprechenden Danspeichers zur Folge.

Listing 5.12: sendInfo(parameters:Sendinfo):SendInfoResponse

```
database.executeQuery(SELECT storage_id
    FROM StorageElements
    WHERE url=parameters.url);
updateStatus(storage_id,parameters.resultSE);
```

ECService.open(paramaters:Open):OpenResponse. Datei öffnen und Handle zurückgeben. Eine Datei kann entweder zum Lesen (*"r"*) oder zum Schreiben (*"w"*) geöffnet werden. Falls die Operation erfolgreich ist, wird die *file_id* aus der Tabelle *Files* als Handle zurückgegeben und das Attribut *lock* auf den gewünschten Wert gesetzt.

Listing 5.13: open(paramaters:Open):OpenResponse

```
file=database.getFileByPath(parameters.filename);
if (file)
{
  if (lock compatible with parameters.mode || file not locked)
  {
    file.lock+=parameters.mode;
    database.executeUpdate(UPDATE Files
       SET lock=file.lock lock_time=now() atime=now()
       WHERE file_id=file.file_id);
    return file.file_id;
  }
}
// file not found, maybe we can create one
if (parameters.mode="w")
{
  return create(parameters.filename);
}
return FileNotFound;
```

ECService.keepalive(parameters:Keepalive):KeepaliveResponse. Wenn der EC Proxy ausfällt, während noch eine Sperre gesetzt ist, muss diese freigegeben werden. Der EC Service hat keine Möglichkeit zu überprüfen, ob der EC Service die Dateioperation noch ausführt. Deshalb schickt der EC Proxy während der Übertragung regelmäßig Keep-alive-Nachrichten, die dem EC Service mitteilen, dass der Transfer noch stattfindet.

Listing 5.14: keepalive(parameters:Keepalive):KeepaliveResponse

```
database.executeUpdate(UPDATE Files SET lock_time=now()
        WHERE file_id=parameters.handle);
```

ECService.close(parameters:Close):CloseResponse. Datei schließen und Sperre aufheben.

Listing 5.15: close(parameters:Close):CloseResponse

```
database.executeUpdate(UPDATE Files SET lock='n' WHERE file_id=parameters.handle);
```

ECService.delete(parameters:Delete):DeleteResponse. Das Löschen einer Datei geschieht asynchron. Der Dateieintrag der Tabelle *Files* kann sofort gelöscht werden. Die Blockgruppen werden aber erst nach und nach gelöscht, indem als geplante AKtion *Löschen* eingetragen wird.

Listing 5.16: delete(parameters:Delete):DeleteResponse

```
file=database.getFileByPath(parameters.filename);
// mark blockgroups for removal
database.executeUpdate(UPDATE Blockgroups SET action='d', file_id=0
    WHERE file_id=file.file_id);
// remove file entry
database.executeUpdate(DELETE FROM Files WHERE file_id=file.file_id);
// start removal of blockgroups
```

```
forall (establishTargetDistribution)
{
  establishTargetDistribution.start();
}
```

ECService.mkdir(parameters:Mkdir):MkdirResponse. Verzeichnis erstellen.

Listing 5.17: mkdir(parameters:Mkdir):MkdirResponse

```
String dirname=plain name without path;
String parentpath=full path of parent directory
// request parent directory from database
parent=database.getDirByPath(parentname);
database.executeUpdate(INSERT INTO Directories (parent_id, name, atime, mtime)
   VALUES (parentID,dirname,now(),now()));
```

ECService.rmdir(parameters:Rmdir):RmdirResponse. Verzeichnis löschen.

Listing 5.18: rmdir(parameters:Rmdir):RmdirResponse

```
directory=database.getDirByPath(parameters.dirname)
// make shure directory is empty
database.execureQuery(SELECT * FROM Files WHERE dir_id=directory.dir_id);
database.execureQuery(SELECT * FROM Directories WHERE parent_id=directory.dir_id);
if (no results)
{
  database.executeUpdate(DELETE FROM Directories WHERE dir_id=directory);
}
else // recursive deletion of directories not supported
{
  return DirectroyNotEmpty;
}
```

ECService.mv(parameters:Mv):MvResponse. Datei oder Verzeichnis verschieben.

Listing 5.19: mv(parameters:Mv):MvResponse

```
from_directory=database.getDirByPath(parameters.from);
from_file=database.getFileByPath(parameters.from);
to_name=plain name of parameters.to;
to_directory=database.getDirByPath(parent directory of parameters.to);
if (database.getDirByPath(parameters.to)
  || database.getFileByPath(parameters.to)
{
  // cannot move and overwrite at once
  return FileExists;
}
if (from_directory)
{
  if (move to subdirectory of itself)
  {    return MoveToSelf;  }
  database.executeUpdate(UPDATE Directories
      SET name=to_name, parent_id=to_directory.dir_id
      WHERE dir_id=from_directory.dir_id);
}
if (from_file)
{
  database.executeUpdate(UPDATE Files
     SET name=to_name, dir_id=to_directory.dir_id
     WHERE file_id=from_file.file_id);
}
```

ECService.getECSResourceProperties():ECSResourceProperties. WSRF Resource Properties auslesen.

Listing 5.20: getECSResourceProperties():ECSResourceProperties

```
return resourceProperties
```

ECService.create(filename:String):OpenResponse. Neue Datei in der Datenbank erstellen. Erstellt einen neuen Eintrag in der Tabelle *Files* und für jede Blockgruppe einen neuen Eintrag in der Tabelle *BlockGroups*. Der Dateieintrag bekommt eine Erstellen-Sperre. Erst nachdem alle Blockgruppen vom EC Proxy gespeichert wurden, wird die Sperre aufgehoben.

Listing 5.21: create(filename:String):OpenResponse

```
// add new file entry
directory=database.getDirByPath(directory of filename);
database.executeUpdate(INSERT
   INTO Files (dir_id, name, atime, mtime, size, lock, lock_time)
   VALUES (directory.dir_id,filename,now(),now(),0,'c',now() ));

// create blockgroups
for (i=0; i<n * desiredRedundancy; i++)
{
  createBlockgroup(file_id);
}
return OpenResponse(file_id, blockgroups, SUCCESS);
```

ECService.createBlockgroup(fileId:int):int. Neue Blockgruppe erstellen.

Listing 5.22: createBlockgroup(fileId:int):int

```
// get the storage elements with the most free space
database.executeQuery(SELECT * FROM StorageElements SORT BY free_capacity);
// look for a storage element to create the block group
for (i=0; i < number of storage elements; i++)
{
  // test if a blockgroup for the file already exists on the storage element i
  database.executeQuery(SELECT * FROM StorageElements, BlockGroups
      WHERE file_id=fileId and storage_id=element[i].storage_id);
  if (!result)
  {
    // create a block group with a random name
    database.executeUpdate(INSERT
        INTO Blockgroups (file_id, storage_id, name, size, index, action)
        VALUES (fileId, element[i].storage_id, random(), 0, new index, 'n'));
    return block_id;
  }
}
```

ECService.updateStatus(strorage_id:int,event:String) Diese Methode implementiert das Zustandsdiagramm aus Abbildung 5.18.

Listing 5.23: updateStatus(strorage_id:int,event:String)

```
switch (event)
{
  case "ReadAccessDenied","AccessDenied","ConnectionRefused",
         "HostNotFound","Remove":
    // fatal error
    if (status=="r")
    { // storage element is to be removed. it will not reapear.
      delete from database;
    } else { // mark storage element as down
```

```
      status = 'd';
    }
    break;
  case "NoSpace","LowPerformance","WriteAccessDenied"
    // minor error
    // set state to busy
    if (status=="n") status="b";
    break;
  case "FileNotFound","ConnectionReset"
    // blockgroup has been deleted by a local process
    // mark storage element as unreliable
    if (status=="n" || status=="b") status="u";
    break;
  case "ToRemove"
    // planed removal
    if (status=="d") delete from database;
    else status="r";
    break;
  case "Register"
    // new storage element
    status="n";
    break;
  case "Reboot"
    // existing storage element reapeared
    status="u"
    break;
  case "Empty"
    // only relevant for storage elements to be removed
    if (status=="r") delete from database;
    break;
  case "Timeout"
    if (status=="d") delete from database;
    if (status=="b") status="n";
    if (status=="u") status="b";
    break;
}
```

ECService.recalcDistribution(). Einen neuen Sollzustand berechnen.

Listing 5.24: recalcDistribution()

```
for (each file in database)
{
  database.executeQuery(SELECT * FROM BlockGroups, StorageElements
       WHERE file_id=file);
  foreach(blockgroup)
  {
    if ((action=="c" || action=="m") && (status!=OK))
    {
      // blockgroups can only be created on working resources
      // abort creation by deleting blockgroup
      database.executeUpdate(DELETE FROM Blockgroups
           WHERE block_id=blockgroup.block_id);
    }
    if ((action=="n") && (status==Remove))
    {
      // storage element will be removed
      // add blockgroup to be copied
      createBlockgroup(blockgroup.fileId, blockgroup.index, 'm');
    }
  }
```

```
  // check redundancy
  // count all block groups that are stored on working storage elements
  // plus block groups that will be created
  database.executeQuery(SELECT COUNT(*)
        FROM BlockGroups, StorageElements
        WHERE file_id=file
        and (action='n' or action='c')
        and (status="n" or status="b") );

  // if the redundancy is much higher than desired
  while (result >> desiredRedundancy)
  {
    delete blockgroup;
    result--;
  }

  // if the redundancy is lower than desired
  while (result < desiredRedundancy)
  {
     createBlockgroup(file);
     result++;
  }
}
```

ECService.checkTimeout(). Datenbank nach Timeout Ereignissen durchsuchen und entsprechende Aktionen durchführen oder planen. Diese Methode wird periodisch aufgerufen.

Folgende Aktionen können durch einen Timeout in Gang gesetzt werden:

- **Statuswechsel.** Auf Grund eines Timeouts steht ein Statuswechsel an.
  Dafür wird *updatestatus(storage_id,"Timeout")* aufgerufen.
- **Aufhebung von Sperren.** Nach einigen Minuten Inaktivität des EC Proxy werden bestehende Sperren auf Dateien aufgehoben.
- **Tests durchführen.** Spezielle Test Blockgruppe für Datenspeicher erstellen und in Sollzustand eintragen.

6 Implementierung

Die prototypische Implementierung soll zeigen, dass der Entwurf anwendbar ist. Sie enthält den EC Proxy und den EC Service und implementiert den Use Case *Datei lesen*. Ein Test mit einer verteilten Datei überprüft die Funktion des Erasure Codes und das Zusammenspiel zwischen EC Proxy und EC Service.

6.1 EC Proxy

Im Entwurf wurde die Architektur des EC Proxy als Klassendiagramm dargestellt (siehe Abbildung 5.15). Bei näherer Betrachtung der Implementierung ist das objektorientierte Modell jedoch nicht zutreffend. Im Folgenden werden statt Klassendiagrammen kurze Ausschnitte aus dem Quelltext angegeben.

6.1.1 Linear Block Coding Library

Neben den beiden Funktionen zur Kodierung und Dekodierung von Blöcken, die bereits im Entwurf der Klasse *Linear Block Code Library* dargestellt wurden, benötigt die Library noch einige Funktionen zum Rechnen mit Matrizen im Restklassenring. Folgender C-Header definiert die Funktionen der Linear Block Coding Library.

Listing 6.1: linearencode.h

```c
#define LEC_BLOCKSIZE_SRC 255 // original block size in Bytes
#define LEC_BLOCKSIZE_DST 256 // size of an encoded block in Bytes
#define LEC_N 16 // number of columns in matrix. number of source blocks
#define LEC_M 32 // number of rows in matrix. maximum number of encoded blocks
#define LEC_PRIME 257 // prime number used in calculations

// functions to encode and decode blocks
void lec_encode_blocks(uint8** src_blocks, uint8** dst_blocks,
      int* map, int mapsize);
void lec_decode_blocks(uint8** src_blocks, uint8** dst_blocks, int* map);

/**********************************************************************/

// this library uses integers of 8 bit and 32 bit length.
// following definitions must be changed for architectures other than x86
typedef unsigned int uint32;
typedef unsigned char uint8;

// basic calculations with numbers in the finite field
uint32 lec_add(uint32 a, uint32 b);
uint32 lec_sub(uint32 a, uint32 b);
uint32 lec_mult(uint32 a, uint32 b);
uint32 lec_div(uint32 a, uint32 b);

// structure used for representing matrices
typedef struct {
  uint32** x; // [rows][cols]
  int rows;
  int cols;
} LECMatrix;
```

```
// the encoding matrix (see linearencode.c)
extern LECMatrix lecMatrix;

// functions for handling matrices
// construct an identity matrix
void lec_matrix_ident(LECMatrix* a, int rows, int cols);
// allocate a matrix
void lec_matrix_new(LECMatrix* a, int rows, int cols);
// copy a matrix to a new one
void lec_matrix_copy(LECMatrix* a, LECMatrix* b);
// deallocate a matrix
void lec_matrix_free(LECMatrix* a);
// invert matrix (a must be square)
void lec_matrix_invert(LECMatrix* a, LECMatrix* b);
// multiply a matrix by a vector (w = a*v)
void lec_matrix_mult_vector(LECMatrix* a, uint32* v, uint32* w);
// invert a submatrix of the decoding matrix given by map
void lec_get_decode_matrix(LECMatrix* result, int* map);
```

Wie man in der Header-Datei sieht, wurden konkrete Werte für die Blockgrößen, die Größe der Matrix und der Primzahl angegeben. Die Kodiermatrix befindet sich in *linearencode.c* und ist ebenfalls eine fest vorgegebene Konstante. In diesem Kapitel sind vor allem die Gründe für die Wahl der Konstanten interessant und wie die Konzepte aus Kapitel 3.6 angewendet werden. Die beiden Funktionen zur Kodierung und Dekodierung sind konzeptionell sehr ähnlich, zeigen aber auch Unterschiede. Das Herzstück beider Funktionen ist die Multiplikation einer Liste von Vektoren mit einer Matrix.

Listing 6.2: lecencodeblocks

```
uint32 src_vector[LEC_BLOCKSIZE_SRC][LEC_N];
uint32 dst_vector[LEC_BLOCKSIZE_SRC][LEC_M];
LECMatrix matrix;
int i,j;
...
// encode vectors
for (j=0; j<LEC_BLOCKSIZE_SRC; j++) {
  lec_matrix_mult_vector(&matrix, src_vector[j], dst_vector[j]);
}
...
```

Bei der Wahl der Primzahl gibt es ein Problem, dass erst bei der Implementierung offensichtlich wurde. Die Transformation von Blöcken beliebiger Daten in Listen von Vektoren und umgekehrt ist nicht trivial und vor allem nicht ohne zusätzlichen Speicherplatz durchzuführen, weil der Wertebereich nicht mit gewöhnlichen Festkommatypen übereinstimmt.

Direkte Transformation

Man betrachtet jeden Block als Liste von 32 Bit Festkommazahlen (*uint32*). Als Primzahl wählt man $p = 2^{32} + 1$. Dadurch kann man LEC_N Blöcke der Länge $LEC\_BLOCKSIZE\_SRC$ Bytes in eine Liste von $LEC\_BLOCKSIZE\_SRC/4$ Vektoren der Dimension LEC_N umwandeln. Nach Multiplikation mit der Kodiermatrix erhält man $LEC\_BLOCKSIZE\_SRC/4$ Vektoren der Dimension LEC_M. Will man diese wieder in Blöcke umwandeln tritt aber ein Problem auf. Die erstellten Vektoren können auch Skalare mit dem Wert 2^{32} beinhalten. Dieser Wert lässt sich jedoch nicht als 32 Bit Zahl speichern. Die Speicherung dieses zusätzlichen Bits führt zu einem Mehrverbrauch an Speicherplatz von ca. 3%.

Basistransformation

So, wie sich Zahlen als Binärzahlen, Dezimalzahlen und Hexadezimalzahlen darstellen lassen, lassen sie sich auch zu einer beliebigen Basis transformieren. Man kann einen Block als eine große Binärzahl, der Länge

l ansehen ($l = 8 * LEC\_BLOCKSIZE\_SRC$). Durch Transformation auf die Basis p erhält man eine Liste von Ziffern, die Werte im Bereich $\{0, 1, \cdots, p-1\}$ annehmen. Aus ähnlichen Überlegungen wie oben angestellt werden die kodierten Blöcke bei diesem Verfahren einige Bit größer. Der Mehrverbrauch ist zwar äußerst gering, der Rechenaufwand für die Basistransformation ist aber unverhältnismäßig hoch.

Lösung

Durch geschickte Wahl der Blockgrößen und der Primzahl kann eine direkte Transformation angewendet werden, ohne großen Mehrverbrauch zu verursachen. Analog zur direkten Transformation werden Blöcke in 255 Bytes zerlegt. Als Primzahl wird 257 gewählt. Nach der Matrixmultiplikation erhält man eine Liste von 255 Vektoren. Die Skalare einer bestimmten Zeile aller Vektoren ergeben eine Liste von 255 Zahlen mit dem Wertebereich $\{0, 1, \cdots, 256\}$. Die Zahl 256 ist die einzige, die sich nicht in einem Byte speichern lässt. Es gibt aber mindestens eine Zahl x, die in der Liste nicht vorkommt, aber kleiner als 256 ist. Wenn man in der Liste jede 256 durch x ersetzt, kann die Liste als 255 Bytes gespeichert werden. Zur Rücktransformation ist die Kenntnis von x nötig, sie wird deshalb hinten an den kodierten Block angehängt, der somit die Länge 256 erhält. Der Mehrverbrauch beträgt ca. $0,4\%$.

Caching der Matrix

Um Blöcke zu dekodieren, muss ein Teil der Kodiermatrix invertiert werden. Verglichen mit der Dekodierung selbst ist die Inversion sehr zeitaufwändig. Je nach verfügbaren Blöcken setzt sich die Dekodiermatrix aus einer Kombination von verschiedenen Zeilenvektoren der Kodiermatrix zusammen. Es ist nicht möglich, alle Kombinationen bereits beim Programmstart zu erstellen. Es ist aber sehr wahrscheinlich, dass aufeinanderfolgende Aufrufe von *lec_decode_blocks* dieselbe Dekodiermatrix benötigen. *lec_decode_blocks* speichert die zuletzt benutzte Matrix in einer statischen Variable und kann sie wiederverwenden.

Größe der Kodiermatrix

Die Wahl der Größe der Kodiermatrix hat im wesentlichen praktische Gründe. Die Performanz der Kodierung und Dekodierung hängen hauptsächlich von der Anzahl der Spalten ab. Bei der Multiplikation einer Matrix mit einem Vektor werden $n * m$ Multiplikationen und Additionen durchgeführt. Für jedes kodierte Byte müssen deshalb n Multiplikationen und Additionen ausgeführt werden. n sollte dennoch nicht zu klein gewählt werden, da sonst die Flexibilität bei der Datenverteilung eingeschränkt wird. Die Werte von $n = 16$ und $m = 32$ stellen einen Kompromiss zwischen Datenverteilung und Performanz dar. Der Inhalt der Matrix wurde zufällig erzeugt. Ein ca. 2-stündiger Test aller Kombinationen aus n Zeilenvektoren hat gezeigt, dass diese linear unabhängig sind. Ein größerer Wert für m wäre zwar wünschenswert gewesen, hätte die Laufzeit des Tests aber um Größenordnungen verlängert.

6.1.2 Asynchronous Event Handling

Die Komponenten des Globus Toolkit, die in C implementiert sind, verwenden ein gemeinsames ereignisgesteuertes Programmiermodell [ASY07]. Funktionen, von denen angenommen wird, dass sie längere Zeit für die Ausführung brauchen, können im Hintergrund abgearbeitet werden.

Programme, die das Event Handling unterstützen, befinden sich in einer Schleife, in der Ereignisse abgearbeitet werden. Funktionen, die im Hintergrund laufen soll, wird als Callback-Funktion implementiert und registriert. Das Programm kann weiter ausgeführt werden, während die Warteschlange vom Event Handling abgearbeitet wird. Das Globus Toolkit besitzt eine Single Threaded Version und eine Multi Threaded Version. In der Single Threaded Version wird zwischen Programm und Event Handling hin und her gesprungen. In der Multi Threaded Version werden Ereignisse parallel zum Programm ausgeführt.

Die Funktion *globus_callback_register_oneshot* fügt ein Ereignis zur Warteschlange hinzu.

Listing 6.3: linearencode.h

```
globus_callback_register_oneshot(
    globus_callback_handle_t *          callback_handle,
    const globus_reltime_t *            delay_time,
    globus_callback_func_t              callback_func,
    void *                              callback_user_args);
```

Der Entwickler implementiert eine Funktion vom Typ *globus_callback_func_t* und registriert diese mit *globus_callback_register_oneshot*. Nachdem *delay_time* Zeit abgelaufen ist, wird das Ereignis ausgelöst und die Funktion ausgeführt. Im Allgemeinen registrieren Callback-Funktionen gegen Ende der Ausführung weitere Callbacks. So kann zum Beispiel signalisiert werden, dass eine Funktion erfolgreich ausgeführt wurde.

Die GridFTP Client Library ist ein gutes Beispiel für die Verwendung des asynchronen Modells.

GridFTP Client Library

Folgendes Listing zeigt Ausschnitte aus einem Clientprogramm, das mit dem Source Code des Globus Toolkit mitgeliefert wird. Es dient dazu die Client Library zu testen, indem eine Datei heruntergeladen wird und auf der Standartausgabe ausgegeben wird.

Listing 6.4: globusftpclientgettest.c

```
static
void
data_cb(
    void *          user_arg,
    globus_ftp_client_handle_t *    handle,
    globus_object_t *        err,
    globus_byte_t *        buffer,
    globus_size_t        length,
    globus_off_t        offset,
    globus_bool_t        eof)
{
    fwrite(buffer, 1, length, stdout);
    if(!eof)
    {
  globus_ftp_client_register_read(handle,
          buffer,
          SIZE,
          data_cb,
          0);
    }
}
...
int main(int argc,
   char *argv[])
{
...
    globus_byte_t        buffer[SIZE];
    globus_size_t        buffer_length = sizeof(buffer);
...
    done = GLOBUS_FALSE;
    result = globus_ftp_client_get(&handle,
           src,
           &attr,
           GLOBUS_NULL,
           done_cb,
           0);
...
```

```
    globus_ftp_client_register_read(
        &handle,
        buffer,
        buffer_length,
        data_cb,
        0);
    }
...
}
```

Der Befehl *globus_ftp_client_get* startet einen Dateitransfer. (Die Initialisierung der Strukturen, die dafür nötig sind, wird hier nicht gezeigt.) Danach folgt ein Aufruf der Funktion *globus_ftp_client_register_read*. Ihr wird ein Puffer (*buffer*) übergeben, der gefüllt werden soll. *globus_ftp_client_register_read* registriert eine Funktion, die Daten aus dem TCP-Stream eines GridFTP Datenkanals liest und in einem Puffer speichert. Wenn diese ausgeführt wurde, dass heißt wenn sich Daten im Puffer befinden, wird die Callback-Funktion *data_cb* registriert. *data_cb* gibt den empfangenen Puffer aus und registriert so lange weitere Lesevorgänge, bis das Dateiende erreicht ist.

6.1.3 ECSPortType (C WS Core)

Die C-Bindings für den EC Proxy werden mit dem Befehl *globus-wsrf-cgen* aus der WSDL-Beschreibung des EC Service generiert. Dabei entstehen 158 Quellcode-Dateien mit insgesamt über 80.000 Zeilen. Die große Menge an Quellcode ist kein Zeichen für schlechten oder inperformanten Code des Globus Toolkit. Es ist durchaus normal, dass generierter Code mehr Zeilen enthält, als handgeschriebener. Das wird klar, wenn man die Funktionen zur Serialisierung und Deserialisierung betrachtet. Jeder Typ, der in der WSDL beschrieben wird, muss in eine Soap Message und zurück transformiert werden können. Anstatt Funktionen zu schreiben, die beliebige Datentypen serialisieren können, haben sich die Entwickler entschieden, Templates zu schreiben, die für jeden Typ verwendet werden. Dadurch verschiebt sich viel vom Quellcode, der zum Framework gehört, in das Anwendungsprogramm. Die Serialisierungsfunktionen der unterschiedlichen Typen sind deshalb zu großen Teilen identisch. Das ist jedoch kein Nachteil, da niemals Änderungen am generierten Code durchgeführt werden. Für den Entwickler einer Client Anwendung sind nur die Header-Dateien interessant, in denen die Typen und Funktionen deklariert sind, die von der Anwendung aufgerufen werden. Im folgenden wird am Beispiel der Operation *getParameters* gezeigt, wie die Client Stubs verwendet werden.

Die Operationen des EC Service werden in der Datei *ECSService_client.h* deklariert. Jede Operation kann auf mehrere Arten aufgerufen werden:

- *ECSPortType_getParameters*: Blockierender Aufruf anhand einer URL.
- *ECSPortType_getParameters_epr*: Blockierender Aufruf anhand einer End Point Reference.
- *ECSPortType_getParameters_register*: Nicht blockierender Aufruf (siehe Abbildung 5.13). Nach erfolgter Operation wird eine Callback Funktion aufgerufen.
- *ECSPortType_getParameters_epr_register*: Nicht blockierender Aufruf anhand einer End Point Reference.

Für den EC Proxy reicht die blockierende Implementierung aus. Folgender Quelltext zeigt die Definition des Client Stub für die Methode *getParameters* des Interface *ECSPortType*.

Listing 6.5: Ausschnitt aus ECSServiceclient.h

```
...
globus_result_t
ECSPortType_getParameters(
    ECSService_client_handle_t handle,
    const char * endpoint,
    const getParametersType * getParameters,
    getParametersResponseType * * getParametersResponse,
    ECSPortType_getParameters_fault_t * fault_type,
```

```
    xsd_any * * fault);
...
```

An die Typennamen aus der WSDL-Beschreibung wird in der C-Implementierung das Wort ”Type” angehängt. Folgendes Beispiel zeigt einen Ausschnitt aus der WSDL und den zugehörigen generierten C Code:

Listing 6.6: Ausschnitt aus ECS.wsdl

```
  <xsd:element name="getParametersResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:parameters"/>
        <xsd:element ref="tns:resultECS"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Listing 6.7: Ausschnitt aus getParametersResponseType.h

```
/**
 * C structure for the getParametersResponseType complex type
 * @ingroup getParametersResponseType
 */
struct getParametersResponseType_s
{
    /** parametersType subelement */
    parametersType parameters;
    /** resultECSType subelement */
    resultECSType resultECS;
};
```

Für den Abruf der Verteilungsparameter benötigt der EC Proxy die Typen, die in folgenden Header Dateien deklariert sind:

Listing 6.8: Ausschnitt aus dem DSI Modul des EC Proxy

```
#include "globus_wsrf_core_tools.h"
#include "ECSService_client.h"
#include "getParametersType.h"
#include "getParametersResponseType.h"
#include "parametersType.h"
#include "fileInfoType.h"
#include "blockgroupType.h"
#include "resultECSType.h"
...
```

6.1.4 DSI Modul

Das DSI Modul implementiert den Use Case *Datei lesen*. Der Quelltext befindet sich in der Datei *globus_gridftp_server_ec.c*. Ohne den Use Case *Datei schreiben* ist es nicht nötig, Sperren auf Dateien zu setzen. Die prototypische Implementierung verwendet die Operation *getParameters* an Stelle der Operation *open*. Die Schritte 3, 11 und 12 aus Abbildung 5.3 sind nicht implementiert.

Nachdem sich der Client mit dem EC Proxy verbunden und eine Datei angefordert hat, wird die Funktion *globus_l_gfs_ec_send* aufgerufen. Sie startet den Transfer, implementiert also die Schritte 1 bis 5 des Use Case. Folgender Ausschnitt zeigt die relevanten Zeilen:

Listing 6.9: Auschnitt aus globus_l_gfs_ec_send

```
static void globus_l_gfs_ec_send(
  globus_gfs_operation_t        op,
  globus_gfs_transfer_info_t *    transfer_info,
  void *            user_arg)
{
  ...
  // set the filename for wich the distribution parameters will be retrieved
  getParameters.filename=transfer_info->pathname;

  // URL of EC Service service should not be hard coded
  result = ECSPortType_getParameters(
    ec_handle->ecs_handle,            // ECSService_client_handle_t
    "http://globus-client1:8080/wsrf/services/ECS",  // const char * endpoint
    &getParameters,            // const getParametersType *
    &getParametersResponse,           // getParametersResponseType * *
    &getParameters_fault_type,        // ECSPortType_getParameters_fault_t *
    &fault);            // xsd_any * * fault
  ...
  // connect to all storage elements
  int i;
  for (i=0; i<LEC_M; i++) {
  ...
        result = globus_ftp_client_get(
          &ec_handle->storage_element_a[i]->handle,
          ec_handle->storage_element_a[i]->url,
          &ec_handle->storage_element_a[i]->operation_attr,
          GLOBUS_NULL,
          globus_l_gfs_ec_se_done_cb,
          ec_handle);
        }
  ...
  }
  // start the actual transfer
  globus_l_gfs_ec_send_dispatch(ec_handle);
  return;
}
```

Bemerkenswert ist, dass *globus_l_gfs_ec_send* den Transfer nur startet, jedoch nicht durchführt. Hier ist also eine Abweichung zum Entwurf aus Kapitel 5.3.6 zu erkennen.Die Schritte 6 bis 8 aus Abbildung 5.3 sind mit Hilfe des asynchronen Modells implementiert. Der EC Proxy muss gleichzeitig Daten von mehreren Datenspeichern empfangen, Blöcke dekodieren und Daten zum Client übertragen. Alle diese Transfers müssen synchronisiert werden. Zum Beispiel muss die Übertragung zum Client gebremst werden, wenn es keine dekodierten Blöcke gibt. Auch das Empfangen ist nicht unbegrenzt möglich, da der EC Proxy nicht über beliebig viel Arbeitsspeicher verfügt. Wenn die Empfangspuffer voll sind, muss die Übertragung von den Datenspeichern verlangsamt werden (irgendwie muss das DSI Modul Einfluss auf die TCP Window Size haben).

Das DSI Modul arbeitet mit Puffern fester Größe. Es handelt sich um kontinuierliche Speicherbereiche von der Größe weniger Megabyte. Es existiert ein Schreibpuffer für die Übertragung zum GridFTP Client des Benutzers und für jeden Datenspeicher jeweils ein Lesepuffer. Die Puffer werden nicht auf einmal vollständig gefüllt. Sie besitzen einen leeren und einen belegten Bereich. Diese Bereiche werden als kontinuierlich angesehen. Fragmentierung ist in dieser Implementierung nicht erlaubt.

An der Übertragung sind im wesentlichen folgende Funktionen beteiligt:

globus_ftp_client_register_read Lesevorgang registrieren. Der GridFTP Client füllt den angegebenen Puffer mit Daten von einem Datenspeicher

globus_gridftp_server_register_write Schreibvorgang registrieren. Der GridFTP Server sendet den angegebenen Puffer zum Client.

globus_l_gfs_ec_se_read_cb Wird vom Framework aufgerufen, wenn ein Lesevorgang beendet wurde. Der belegte Bereich des entsprechenden Lesepuffers wird um die Anzahl der gelesenen Bytes vergrößert.

globus_l_gfs_ec_server_write_cb Wird vom Framework aufgerufen, wenn ein Schreibvorgang beendet wurde. Der entsprechende Bereich des Schreibpuffers ist nicht mehr nötig. Der freie Bereich wird vergrößert.

globus_l_gfs_ec_decode_blocks Überprüft alle Lesepuffer und dekodiert Blöcke falls möglich.

globus_l_gfs_ec_send_dispatch Herzstück des Algorithmus. Hier wird die Übertragung koordiniert und synchronisiert.

Folgendes Aktivitätsdiagramm zeigt den Ablauf der Übertragung.

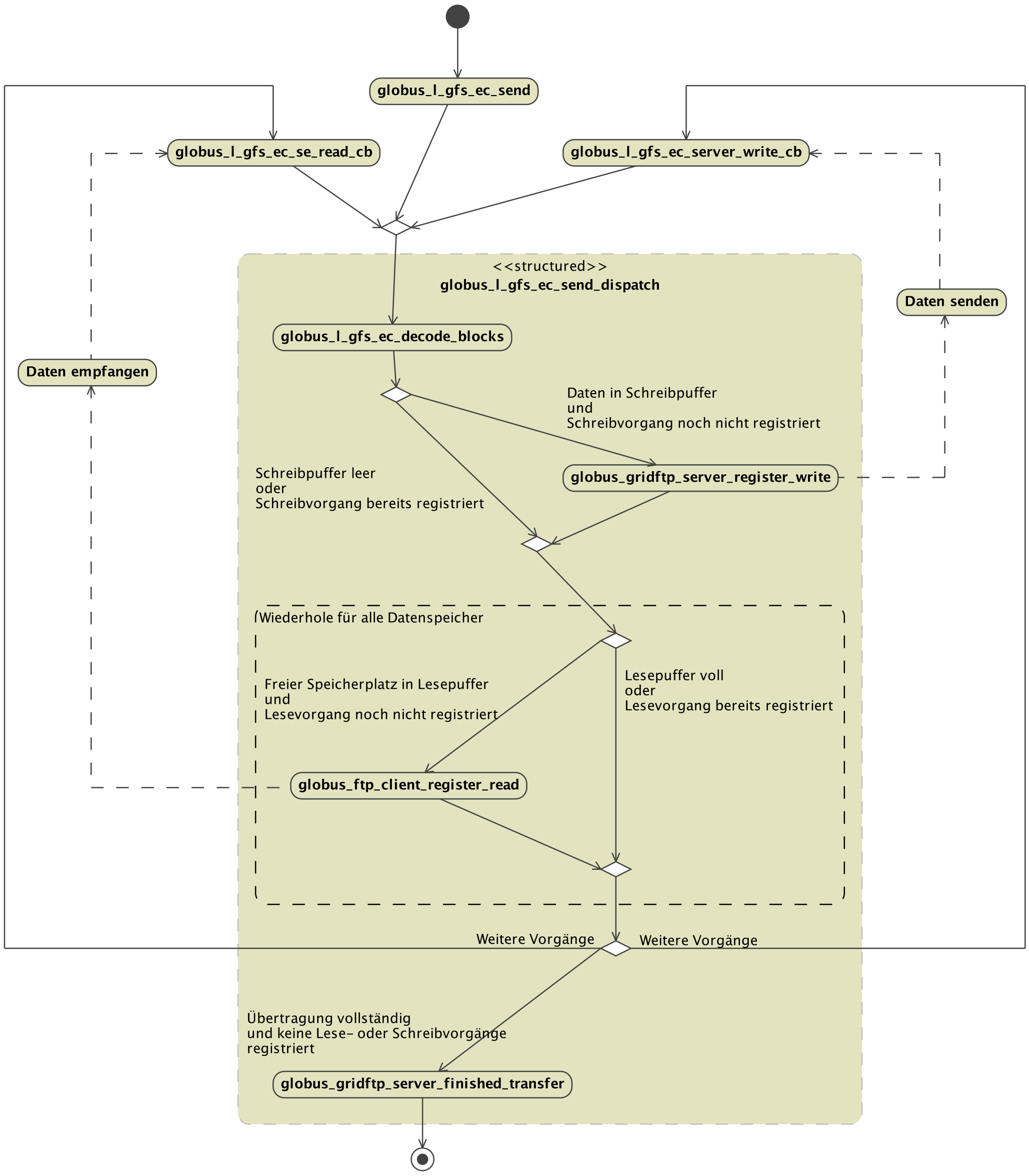

Abbildung 6.1: Ablauf von Datei lesen

Wie man in Abbildung 6.1 sieht, finden die meisten Aktivitäten innerhalb der Funktion *globus_l_gfs_ec_send_dispatch* statt. Der Transfer wird am leben gehalten, indem *globus_l_gfs_ec_send_dispatch* immer wieder aufgerufen wird. Das passiert zu Beginn ein einziges mal über *globus_l_gfs_ec_send* und danach über die beiden Callback Funktionen *globus_l_gfs_ec_se_read_cb* und *globus_l_gfs_ec_server_write_cb*.

Die beiden Aktionen *Daten empfangen* und *Daten senden* werden vom Globus Framework im Hintergrund parallel zur Programmabarbeitung durchgeführt. Die wirklichen Namen der entsprechenden Funktionen sind nicht angegeben, da der Programmierer nicht wissen muss, wie der Datenkanal von GridFTP angesprochen wird. Die gestrichelten Linien sollen andeuten, dass die Aktionen nicht sofort ausgeführt werden, sondern als Ereignisse in die Warteschlange des Event Handler eingefügt werden.

Der Transfer kommt deshalb nicht zum Erliegen, weil immer mindestens ein Lese- oder Schreibvorgang registriert wird, oder bereits registriert ist. Das Ende der Übertragung ist genau dann erreicht, wenn keine weiteren Vorgänge registriert sind. Das kann auf Grund vollständiger und erfolgreicher Übertragung oder auch durch einen Fehler erfolgen. In beiden Fällen wird die Übertragung durch den Aufruf von *globus_gridftp_server_finished_transfer* beendet.

Folgenden Beispiele verdeutlichen den Ablauf:

Start Zu Beginn sind alle Puffer leer. Es können keine Blöcke dekodiert werden und auch keine Schreibvorgänge registriert werden. Es werden aber Lesevorgänge für alle Datenspeicher registriert. Danach wird *globus_l_gfs_ec_send_dispatch* beendet, und das DSI Modul wartet auf ein Ereignis. Für jeden empfangenen Puffer von einem Datenspeicher wird *globus_l_gfs_ec_se_read_cb* aufgerufen. Solange nicht genug Daten vorhanden sind um die ersten Blöcke zu dekodieren, wird für jeden empfangenen Puffer ein neuer Lesevorgang für die nächsten Daten registriert. Erst wenn alle Datenspeicher mindestens einen Block übertragen haben, können die Blöcke dekodiert werden, und zum Senden registriert werden.

Langsame Übertragung von einem Datenspeicher. Angenommen die Aktion *Daten empfangen* dauert bei einem einzelnen Datenspeicher sehr lange. In dieser Zeit werden wiederholt Daten von anderen Datenspeichern empfangen. Die Funktion *globus_l_gfs_ec_send_dispatch* wird oft ausgeführt. Es können jedoch keine Blöcke dekodiert werden, weil dafür der Puffer des langsamen Datenspeichers nötig ist. Folglich werden die Puffer der restlichen Datenspeicher nicht mehr geleert. Wenn sie keine Daten mehr aufnehmen können, werden auch keine neuen Lesevorgänge registriert und die gesamte Übertragung wird verlangsamt. (Deshalb sieht der Entwurf vor, von Anfang an zusätzliche Datenspeicher mit einzubeziehen.)

Langsamer Client Angenommen der Client kann die Daten nicht schnell genug empfangen. Das führt dazu, dass die Aktion *Daten senden* längere Zeit braucht. In dieser Zeit werden Daten von den Datenspeichern empfangen und dekodiert bis der Schreibpuffer voll ist. Daraufhin können keine Blöcke mehr dekodiert werden und die Lesepuffer füllen sich auch. Es können keine Lesevorgänge mehr registriert werden und die Übertragung wird verlangsamt.

Der Vorteil dieser Implementierung ist, dass durch die Verwendung großer Puffer gewisse Schwankungen in der Datenübertragung abgefangen werden können.

Folgendes Listing zeigt die prototypische Implementierung des Aktivitätsdiagramm (Abbildung 6.1). Aus Gründen der Übersichtlichkeit wurde sämtlicher Code für Fehlerbehandlung und Threadsynchronisation weggelassen.

Listing 6.10: globus_l_gfs_ec_send_dispatch und Callback Funktionen

```
/*
 * called whenever a buffer has been transfered to the client
 */
static
void
globus_l_gfs_ec_server_write_cb(
  globus_gfs_operation_t   op,
  globus_result_t           result,
  globus_byte_t *           buffer,
  globus_size_t             nbytes,
  void *                user_arg)
{
  globus_l_gfs_ec_handle_t *     ec_handle;
  ec_handle = globus_l_gfs_ec_handle_get(user_arg,op);

  // reduce the buffer size by the number of bytes sent
```

```
  ec_handle->buffer_pos = (ec_handle->buffer_pos + nbytes) % BUFFER_SIZE_CL_BYTES;
  ec_handle->buffer_size -= nbytes;
  ec_handle->pending_writes--;

  // carry on with the transfer
  globus_l_gfs_ec_send_dispatch(ec_handle);
}

/*
 * called whenever a buffer has been read from a storage element
 */
globus_l_gfs_ec_se_read_cb(
  void *          user_arg,
  globus_ftp_client_handle_t *    handle,
  globus_object_t *      err,
  globus_byte_t *        buffer,
  globus_size_t        length,
  globus_off_t        offset,
  globus_bool_t        eof)
{
  globus_l_gfs_ec_handle_t *     ec_handle;
  globus_l_gfs_ec_storage_element_t *  storage_element;
  ec_handle = globus_l_gfs_ec_handle_get(user_arg,GLOBUS_NULL);

  // find out wich storage element this is (what block is stored on it)
  storage_element=ec_handle->storage_element_a[find_se(ec_handle,handle)];

  // reduce pending_reads
  storage_element->pending_reads--;
  // increase buffer size by the number of bytes received
  storage_element->buffer_size += length;

  globus_l_gfs_ec_send_dispatch(ec_handle);
}

globus_result_t
globus_l_gfs_ec_send_dispatch(globus_l_gfs_ec_handle_t* ec_handle) {
  globus_result_t result=GLOBUS_SUCCESS;

  // try to decode blocks
  globus_l_gfs_ec_decode_blocks(ec_handle);

  // if there is data in the write buffer, and no pending writes,
  // schedule for writing
  if (ec_handle->buffer_size > 0 && ec_handle->pending_writes==0) {
    // enqueue for writing to data channel
    // calculate the buffer size.
    int size=min(ec_handle->buffer_size, ec_handle->server_blocksize);
    int pos=ec_handle->buffer_pos % BUFFER_SIZE_CL_BYTES;
    // since this is a ring buffer, truncate at edge
    if ( pos + size > BUFFER_SIZE_CL_BYTES) {
      size = BUFFER_SIZE_CL_BYTES - pos;
    }

    result = globus_gridftp_server_register_write(
      ec_handle->op,
      (globus_byte_t*)ec_handle->buffer + pos,
      size,
      ec_handle->buffer_pos,
      -1,
      globus_l_gfs_ec_server_write_cb,
```

```
      ec_handle);
    ec_handle->pending_writes++;
  }

  globus_l_gfs_ec_storage_element_t* storage_element;

  // for all storage elements
  // if there is room in the read buffer, and no pending reads,
  // schedule for reading
  int i,j;
  j=0;
  for (i=0; i<LEC_M; i++) {
    storage_element = ec_handle->storage_element_a[i];
    if (storage_element) {
      if (  (storage_element->buffer_size < BUFFER_SIZE_SE_BYTES)
        && ( storage_element->pending_reads == 0 ) )
      {
        // enqueue for reading from storage element
        // calculate the buffer size.
        int size=LEC_BLOCKSIZE_DST;
        // relative position to the storage_element buffer
        // insert the data to read at the end of the buffer
        int pos=(storage_element->buffer_pos + storage_element->buffer_size) %
                    BUFFER_SIZE_SE_BYTES;
        // since this is a ring buffer, truncate at edge
        if ( pos + size > BUFFER_SIZE_SE_BYTES) {
          size = BUFFER_SIZE_SE_BYTES - pos;
        }
        result = globus_ftp_client_register_read(
          &storage_element->handle,
          (globus_byte_t*)storage_element->buffer + pos,
          size,
          globus_l_gfs_ec_se_read_cb,
          ec_handle);
        storage_element->pending_reads++;
      }
    }
  }

  // the code for
  // -concurrency
  // -error handling
  // -end of transfer
  // has been ommited (see globus_gridftp_server_ec.c)
  return result;
}
```

6.1.5 Deployment

Das DSI-Modul wird vom GridFTP zur Laufzeit als Shared Library geladen. Damit sie binär kompatibel sind, müssen GridFTP-Server und DSI-Modul mit den gleichen Einstellungen compiliert werden. Im Verzeichnis *src/ecproxy* befinden sich die Dateien *Makefile* und *makefile_header*. *makefile_header* muss auf die lokale Installation angepasst werden. Daraufhin kann man das DSI Modul mit *make* erstellt werden. Die Shared Library erhält je nach Architektur und Compiler einen anderen Namen. Das DSI Modul des vorliegenden Prototyps befindet sich in *globus_gridfto_server_ec_gcc32dbg.so*. Die Installation besteht darin, die Shared Library in das *lib* Verzeichnis des Globus Toolkit zu kopieren. Betreibt man mehrere Hosts mit der gleichen Architektur und der gleichen Version des Globus Toolkit, muss das DSI Modul nur einmal compiliert werden. Danach reicht es aus die Datei *globus_gridfto_server_ec_gcc32dbg.so* auf alle Hosts zu kopieren.

6.2 EC Service

Die Implementierung des EC Service bestätigt die Architektur indem gezeigt wird, dass der EC Proxy mit dem EC Service kommunizieren kann. Es wurde nur die Methode *getParameters* implementiert. Die Klassen *Database* und *EstablishTargetDistributionThread* wurden nicht implementiert. Da es keine Datenbankanbindung gibt, werden die Verteilungsparameter im Quelltext festgelegt. Folgende Implementierung der Methode *getParameters* erstellt die Verteilungsparameter für einen Testlauf des Use Case *Datei lesen*

Listing 6.11: getParameters

```java
public synchronized GetParametersResponse getParameters(GetParameters parameters)
   throws RemoteException
{
  Blockgroup[] blockgroup=new Blockgroup[32];
  String base_url="gsiftp://globus-client";
  for (int i=0; i<32; i++) {
    blockgroup[i]=new Blockgroup(1,18432,base_url +
          (new Integer(i % 4).toString()) + "/tmp/block" +
          new Integer(i).toString());
  }
  FileInfo fileInfo=new FileInfo(0,"test",0,292097);
  Parameters fileParameters=new Parameters(blockgroup,fileInfo,32);
  return new GetParametersResponse(fileParameters,ResultECS.SUCCESS);
}
```

Bevor der EC Service compiliert werden kann, müssen Deployment Parameter angegeben werden. Sie geben an, wie der Dienst in einen Web Services Container eingefügt werden kann. Es gibt zwei Dokumente, die das Deployment beschreiben.

6.2.1 WSDD

Der Web Services Deployment Descriptor (WSDD) enthält Informationen über den Dienst, die vom Web Services Container gebraucht werden. Er gibt an, wie der Dienst heißen soll und wo sich Programmcode und WSDL befinden. Folgendes Listing zeigt den WSDD des EC Service

Listing 6.12: deploy-server.wsdd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <service name="ECS" provider="Handler" use="literal" style="document">
    <parameter name="className" value="de.jenshellwig.services.ecs.impl.ECS"/>
    <wsdlFile>share/schema/ECS/ECS_service.wsdl</wsdlFile>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="scope" value="Application"/>
    <parameter name="loadOnStartup" value="true"/>
  </service>
</deployment>
```

$\langle service \rangle$ beschreibt die Deployment Parameter. Der Name des Service wird hier auf ECS gesetzt. Das heißt, er wird später unter: ”https://server-name:8443/wsrf/services/ECS“ erreichbar sein.

$\langle parametername ='' \ className'' \rangle$ ist der Name der Klasse des Web Service (inklusive Namespace).

$\langle wsdlFile \rangle$ gibt Namen und Ort des WSDL Dokumentes an.

6.2.2 JNDI

WSDL und WSDD beschreiben die Schnittstelle und das Deployment von Web Services. Im Grid kommen noch die Erweiterungen des Web Services Resource Framework hinzu. Der Web Services Container des Globus Toolkit benötigt neben der Service Klasse noch eine Klasse für die Ressource. Diese wird JNDI Dokument angegeben.

Listing 6.13: deploy-jndi-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">
<service name="ECS">
  <resource name="home" type="org.globus.wsrf.impl.ServiceResourceHome">
  <resourceParams>
    <parameter>
      <name>factory</name>
      <value>org.globus.wsrf.jndi.BeanFactory</value>
    </parameter>
  </resourceParams>
  </resource>
</service>
</jndiConfig>
```

Der Prototyp des EC Service implementiert nur den Service, da er keinen Zustand benötigt. Für diesen Fall kann man als Ressource *org.globus.wsrf.impl.ServiceResourceHome* angeben.

6.2.3 Compilieren

Das Compilieren des EC Service geschieht mit Hilfe von Apache Ant [ANT07] einem Build Tool, das ähnlich wie der UNIX Befehl *make* den gesamten Vorgang steuert. Die Schritte werden im XML basierten *Buildfile* beschrieben.

Das Buildfile muss nicht selbst geschrieben werden. Das Script *globus-build-service* wurde als Teil der Globus Service Build Tools (GSBT) entwickelt und erlaubt die automatische Erstellung eines Buildfile und Compilieren von Globus Diensten ([SOT06] S. 72).

Durch folgenden Befehl werden die Java Stubs aus der WSDL generiert und der gesamte Service compiliert.

> *globus-build-service.sh -d de/jenshellwig/services/ecs/ -s schema/ECS/ECS.wsdl*

Alle Dateien, die für das Deployment und den Betrieb des EC Service notwendig sind, werden in die Archivdatei *de_jenshellwig_services_ecs.gar* gepackt.

6.2.4 Deployment

Das Globus Toolkit enthält einen Befehl zum Deployment des oben genannten Archivs. Durch den folgenden Befehl wird das Archiv entpackt, alle Dateien an den richtigen Ort kopiert und der Globus Web Services Container so konfiguriert, dass er den Dienst beim nächsten Neustart lädt.

> *globus-deploy-gar de_jenshellwig_services_ecs.gar*

7 Leistungsbewertung

Der vorliegende Entwurf beschreibt ein komplettes System zur verlässlichen Speicherung VO-kritischer Daten im Grid. Es ist aber noch nicht geklärt, wie gut er die Anforderungen erfüllt, die in Kapitel 1.3 gestellt wurden.

7.1 OGSA

Der Entwurf passt zur Definition des Grid Computing von Ian Foster (siehe Kapitel 1.1.1). Das System verwendet offene und standardisierte Protokolle des Globus Toolkit. Funktionalität, die von den Diensten des Globus Toolkit angeboten wird, wird nicht neu implementiert. Überall, wo es möglich ist, werden Dienste als Web Service betrieben. Das GridFTP Protokoll ist die einzige Ausnahme. Die Entwickler des Globus Toolkit haben es aus Geschwindigkeitsgründen nicht als Web Service konzipiert. Auch ich habe diese Abweichung von der OGSA in Kauf genommen, da ich der Meinung bin, dass GridFTP vor allem bei datenintensiven Anwendungen signifikant schneller ist als vergleichbare Web Services.

7.2 Sicherheit

Das Management der Zugriffsrechte über den CAS Service funktioniert. Durch den Einsatz von richtig konfigurierten EC Proxies kann die VO garantieren, dass die Zugriffsrechte auch eingehalten werden.

Es wurde auch die Sicherheit gegenüber kompromittierten Datenspeichern gefordert. Wenn ein Angreifer einen Datenspeicher unter Kontrolle bringt, so kann er Blockgruppen auslesen oder löschen. Aus einzelnen Blockgruppen kann er aber keine Rückschlüsse auf den Inhalt von Dateien ziehen. Gelöschte oder veränderte Blockgruppen kann das System wiederherstellen.

Hat der Angreifer Zugang zu mehreren Datenspeichern, können Probleme entstehen. Unter Umständen kann der Angreifer Dateien zerstören oder auslesen. Durch das gleichzeitige Löschen vieler Blockgruppen können Dateien unbrauchbar werden. Durch die Kenntnis genügend vieler Blockgruppen kann eine Datei rekonstruiert werden. Wenn ein Angreifer n Blockgruppen einer Datei kennt, fehlen ihm zur Dekodierung nur die Indizes der Blockgruppen. Falls er kleine Ausschnitte der Datei kennt, kann er diese Ausschnitte selbst kodieren und mit den Blockgruppen vergleichen. Dadurch erhält er die Indizes und kann die ganze Datei dekodieren. Bei bekannten Dateiformaten können zum Beispiel Informationen aus dem Header, die für alle Dateien gleich sind, nützlich sein. Textdateien enthalten mit Sicherheit oft gebrauchte Wörter, die Hinweise auf die Indizes geben können.

Natürlich können auch EC Proxy und EC Service angegriffen werden. Der Entwurf sieht vor, dass beide Dienste über ein Zertifikat einen VO-weiten Zugang zu allen Datenspeichern erhalten. Die EC Proxies und der EC Service müssen gut gesichert werden.

Zusammenfassend kann man sagen, dass die Anforderungen an die Sicherheit zum Großteil erfüllt werden. Im Allgemeinen ist die Anzahl der Datenspeicher wesentlich höher als die Anzahl der EC Proxies. Somit verringert sich die Zahl der Systeme, die gesichert werden müssen.

7.3 Zuverlässigkeit

Das zweite Anliegen dieser Arbeit war die Sicherung der Daten gegen Ausfall. Ich möchte dazu für ein Beispielszenario die Mean Time Between Failure (MTBF) für das gesamte System berechnen.

Die Mean Time between Failure für den einzelnen Datenspeicher sei 100 Tage. Normalerweise haben Datenspeicher eine sehr viel längere Verfügbarkeit. Ziel dieser Arbeit ist aber eine verlässliche Speicherung von Daten auf unzuverlässigen Speichern. Deshalb wird hier ein pessimistischer Wert gewählt.

Die Mean Time Betweeen Repair (MTBR) sei 1 Tag. In unserem Fall sorgt das System automatisch für die Wiederherstellung der ausgefallenen Ressource,. Wenn die VO große Datenmengen besitzt, kann es vorkommen, dass es einen Tag dauert bis die Daten eines ausgefallenen Datenspeichers wiederhergestellt sind.

Die Redundanz sei 150%. Für jede Datei werden 24 Blockgruppen erstellt (benötigt werden 16).

Das System fällt aus, wenn mehr als 8 Datenspeicher gleichzeitig ausfallen. Das muss innerhalb eines Tages passieren, bevor die Daten wiederhergestellt worden sind. Es reicht also einen einzigen Tag zu betrachten.

Die Wahrscheinlichkeit, dass ein bestimmter Datenspeicher an einem bestimmten Tag ausfällt beträgt:

$$p = 0.01$$

Die Binomialverteilung gibt an, wie groß die Wahrscheinlichkeit ist, dass von n Ereignissen mit der Wahrscheinlichkeit p genau k eintreten.

Wahrscheinlichkeit, dass genau k von n Datenspeichern ausfallen:

$$P_k = \binom{n}{k} * p^k * (1-p)^{n-k}$$

Wahrscheinlichkeit, dass 8 oder mehr Datenspeicher von 24 ausfallen:

$$P_{gesamt} = \sum_{k=9}^{2} 4 = 24 P_k$$

Durch Einsetzen aller Werte erhalten wir:

$$P_{gesamt} = 1.14 * 10^{-12}$$

Das ergibt eine Mean Time Between Failure des gesamten Systems von:

$$MTBF_{gesamt} = 877 * 10^9 Tage$$

Mit einem Datenverlust ist also alle 2 Milliarden Jahre zu rechnen. Um die gleiche Ausfallwahrscheinlichkeit durch vollständige Replikation zu erreichen, müsste man von jeder Datei 6 Replikate auf unterschiedlichen Datenspeichern halten, d.h. man bräuchte eine Redundanz von ca. 600%. Das entspricht dem vierfachen Speicherverbrauch im Vergleich zum vorgestellten System. Das System kann eine bessere Aussfallsicherheit garantieren als die einzelnen Datenspeicher. Es wird der Forderung von Ian Foster nach nichttrivialer Dienstgüte gerecht.

7.4 Verwaltungsaufwand

Die Einrichtung einer Grid Infrastruktur auf Basis des Globus Toolkit ist nicht trivial und durchaus aufwändig. Der vorliegende Entwurf sieht den Einsatz von einigen verschiedenen Grid-Diensten vor, die konfiguriert, eingesetzt und gewartet werden müssen. Falls die VO außer zur Speicherung von Daten keine anderen Grid-Anwendungen benötigt, muss man den Verwaltungsaufwand als hoch ansehen.

Falls die VO bereits das Globus Toolkit und sowohl MDS als auch CAS einsetzt, ist der Einrichtungsaufwand vertretbar. Das anschließende Betreiben ist mit wenig Aufwand verbunden. Die Konfiguration neuer Datenspeicher ist Dank der geringen Anforderungen einfach. Die automatische Verteilung der Daten erhöht die Flexibilität stark. Zur Erhöhung des Gesamtspeicherplatzes müssen nur neue Datenspeicher eingebunden werden, verwendet werden sie automatisch.

7.5 Performanz

In Kapitel 1.3 wurden keine Anforderungen an die Geschwindigkeit gestellt. Uns interessiert natürlich trotzdem, ob das System auch schnelle Datenübertragungen ermöglicht.

Zur Messung der maximalen Geschwindigkeit, in der Blöcke enkodiert und dekodiert werden können, wurde ein Programm geschrieben. *test_linearencode.c* misst die Zeit, die benötigt wird um 1600000 Blöcke zu enkodieren und wieder zu dekodieren. Die Enkodierung findet nicht vollständig statt. Von jeweils 32 möglichen Blöcken werden nur 16 erstellt. Dass heißt es werden nur so viele Daten erzeugt, wie für die Dekodierung nötig sind. Um den Einfluss des Prozessorcache gering zu halten, werden zwei Speicherbereiche von jeweils ca. 400MByte angelegt. Im ersten Schritt wird der erste Speicherbereich enkodiert und im zweiten Speicherbereich abgelegt. Im zweiten Schritt werden die Daten vom zweiten in den ersten dekodiert. Folgende Ergebnisse wurden auf einem AMD Athlon64 X2 mit einer Taktrate von 2.0GHz ermittelt. Compiliert wurde das Programm mit gcc (Version 4.1.2) und größtmöglicher Optimierung (-O3).

- **Enkodieren** 12 Sekunden. Durchsatz: 264 MBit/s
- **Dekodieren** 19 Sekunden. Durchsatz: 168 MBit/s

Der Durchsatz entspricht in etwa den Datentransferraten, die man in einfachen Netzwerken erreicht. Für viele Anwendungen reicht das aus. Es kann aber VO-kritische Daten geben, die schneller übertragen werden müssen.

7.6 Skalierbarkeit

Eine Grundanforderung an alle Grid-Systeme ist die Skalierbarkeit. Hier interessiert uns vor allem die Erhöhung der Anzahl der Dateien in der Verzeichnisstruktur der VO. Die Anzahl der Dateien und Verzeichnisse wirkt sich auf die Geschwindigkeit des EC Service aus. Je größer die Datenbank, desto länger dauert es, sie zu durchsuchen. Da der Prototyp keine Datenbankanbindung besitzt, fehlen praktische Erfahrungen zur maximal möglichen Anzahl von Dateien und Verzeichnissen. Es ist allerdings abzusehen, dass es virtuelle Organisationen gibt, deren Datenmenge nicht von einem einzigen EC Service verwaltet werden kann. Ich sehe die schlechte Skalierbarkeit als das größte Problem dieses Entwurfs an.

8 Ausblick

Obwohl der Entwurf vollständig ist, handelt es sich um einen Vorschlag. Andere Systeme können auf dieser Arbeit aufsetzen. Viele Konzepte bleiben auch dann noch gültig, wenn der Einsatzbereich verändert oder die Architektur angepasst wird.

Die Leistungsbewertung hat gezeigt, dass es durchaus Probleme und Spielraum für Verbesserungen gibt.

8.1 Verbesserungen

Wie wir gesehen haben, bietet das System ausgezeichnete Zuverlässigkeit bei Ausfall von Hardware. In den Bereichen der Sicherheit gegenüber Angreifern, der Performanz und der Skalierbarkeit sehe ich noch Verbesserungsbedarf.

8.1.1 Sicherheit

Die Verwendung eines einzigen Zertifikats für die gesamte VO impliziert die Speicherung des privaten Schlüssels auf allen teilnehmenden EC Proxies und dem EC Service. Im Allgemeinen rät man von solch einem Vorgehen eher ab. Es gibt eine Lösung, die aber mit etwas höherem Konfigurationsaufwand verbunden ist.

Es lässt sich nicht vermeiden, dem EC Service vollen Zugriff auf alle Datenspeicher zu gewähren. Nur so kann er autonom auf Veränderungen der VO reagieren. Die EC Proxies aber müssen nicht zu jeder Zeit auf alle Datenspeicher zugreifen können. Mit dem Mechanismus der Delegation kann der EC Service temporäre Rechte an EC Proxies abgeben. Wenn ein EC Proxy auf einige Datenspeicher zugreifen will, lässt er sich zunächst vom EC Service für jeden Datenspeicher ein eigenes Proxy-Zertifikat vom EC Service signieren. Der EC Proxy erhält somit nur Zugriff auf Datenspeicher, die er gerade braucht.

Ein Angreifer, der einen EC Proxy unter Kontrolle bringt, kann zwar immer noch auf alle Dateien zugreifen, weil er beliebig viele neue Proxy-Zertifikate anfordern kann. Sobald jedoch bekannt wird, dass der EC Proxy kompromittiert wurde, kann er gesperrt werden.

8.1.2 Performanz

Die Anforderungen an die Geschwindigkeit der Datenübertragung sind stark von der Anwendung abhängig. Das Grid Computing wird gerne als besonders schnell und leistungsfähig dargestellt. Datenübertragungen um die 200MBit passen nicht in dieses Bild.

Folgende Liste zeigt Möglichkeiten, das System leistungsfähiger zu machen:

- **Zusätzliche EC Proxies** steigern den Gesamtdurchsatz.
- **Multithreading innerhalb des EC Proxy.** Mehrprozessorsysteme für die EC Proxies können den Durchsatz steigern.
- **Parallele Nutzung mehrerer EC Proxies.** Die GridFTP Erweiterungen lassen es zu, mehrere EC Proxies gleichzeitig für dieselbe Datei zu verwenden. Der vorliegende Prototyp unterstützt das noch nicht.
- **Optimierung der Linear Block Coding Library.** Die Geschwindigkeit lässt sich durch gezielte Optimierung des Quellcodes der Linear Block Coding Library noch steigern.

- **Verkleinerung der Kodiermatrix.** Die Anzahl mathematischer Operationen ist in etwa proportional zur Anzahl der Spalten in der Kodiermatrix. Durch Verkleinerung der Matrix verringert sich die Ausfallsicherheit, aber der Durchsatz steigt.
- **Andere Erasure Codes**: Der LT Code [LUB02] ist ein Erasure Code, der nicht garantieren kann, dass n Blockgruppen zur Dekodierung ausreichen. Dafür ist er signifikant schneller.

Es gibt also einige Ansätze zur Leistungssteigerung, die auch in Kombination angewandt werden können. Durch die Hinzunahme weiterer Hardware können große Leistungssteigerungen erzielt werden.

8.1.3 Skalierbarkeit

Die Datenbank des EC Service ist nicht skalierbar. Bevor der Entwurf implementiert wird, sollte noch eine Optimierung des Datenbankschemas und der Abfragen erfolgen. Durch die Indizierung von Attributen, nach denen oft in den Tabellen gesucht wird, kann eine Leistungssteigerung erzielt werden. Ausserdem können das Schema und die Abfragen verändert werden. Vielleicht stellt sich heraus, dass die Baumstruktur der Verzeichnisse anders und effizienter gelöst werden kann.

Echte Skalierbarkeit erreicht man aber erst, wenn vermieden werden kann, dass alle Anfragen über dieselbe Datenbank laufen. Es sind also mehrere EC Services nötig, die über eine verteilte Datenbank auf einen gemeinsamen Datenbestand zugreifen.

8.2 Anpassungen

Sowohl Griblocks DISK (siehe Kapitel 2.3) als auch der SRB (siehe Kapitel 2.4) besitzen gewisse Parallelen in der Architektur. Die Trennung zwischen eigentlicher Datenhaltung und der Datenverwaltung ist ihnen gemeinsam. Die Hauptunterschiede ergeben sich durch die unterschiedlichen Protokolle und Datenbanksysteme. Auch der vorliegende Entwurf passt gut zu der Architektur von Gridblocks DISK oder SRB.

Gridblocks DISK löst das Problem der Zuverlässigkeit und Datenverteilung. Durch Erweiterungen um die Grid Security Infrastructure und das VO-Management kann Gridblocks DISK an die Anforderungen des Grid Computing angepasst werden.

Der **SRB** ist bereits für den Einsatz in Grid vorbereitet. Er orientiert sich zwar nicht an der OGSA, ist aber kompatibel zur Grid Security Infrastructure. Erweiterungen um das VO-Management und der Einsatz von Erasure Codes könnten sich lohnen.

Der Entwurf bleibt auch gültig, wenn man den Einsatzbereich anpasst. Das GridFTP Protokoll wird überwiegend zum Datenaustausch zwischen Grid Anwendungen verwendet. Dadurch ist es gut für typische Einsatzszenarien im Grid geeignet. Andere Einsatzgebiete erfordern andere Protokolle. Besonders sei hier die Verwendung als Dateisystem erwähnt, wie es der SRB verwirklicht. Es gibt einen Treiber für Linux der GridFTP als Dateisystem einbinden kann. Durch Anpassungen des Entwurfs kann aber auch ein eigenständiges verteiltes Grid File System [GFS07] entstehen.

A WSDL des EC Service

Listing A.1: ECS.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ECS"
    targetNamespace="http://jenshellwig.de/namespaces/ECS"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://jenshellwig.de/namespaces/ECS"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/"
         "wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


<!--===============================================================

                      T Y P E S

  ===============================================================-->
<types>
<xsd:schema targetNamespace="http://jenshellwig.de/namespaces/ECS"
  xmlns:tns="http://jenshellwig.de/namespaces/ECS"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">


  <!-- COMMON TYPES -->

  <xsd:element name="blockgroup">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="url" type="xsd:string"/>
        <xsd:element name="size" type="xsd:int"/>
        <xsd:element name="index" type="xsd:int"/>
        <xsd:element name="active" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="fileInfo">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="filename" type="xsd:string"/>
        <xsd:element name="size" type="xsd:int"/>
        <xsd:element name="atime" type="xsd:int"/>
        <xsd:element name="mtime" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="parameters">
    <xsd:complexType>
      <xsd:sequence>
```

```
      <xsd:element ref="tns:fileInfo"/>
      <xsd:element name="redundancy" type="xsd:int"/>
      <xsd:element ref="tns:blockgroup" minOccurs="0" maxOccurs="32"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="directory">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:fileInfo" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- errors that can occur at the Storage Element -->
<xsd:element name="resultSE">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <!-- messages from ec proxy -->
      <xsd:enumeration value="SUCCESS"/>
      <xsd:enumeration value="ReadAccessDenied"/>
      <xsd:enumeration value="WriteAccessDenied"/>
      <xsd:enumeration value="AccessDenied"/>
      <xsd:enumeration value="FileNotFound"/>
      <xsd:enumeration value="ConnectionReset"/>
      <xsd:enumeration value="ConnectionRefused"/>
      <xsd:enumeration value="LowPerformance"/>
      <xsd:enumeration value="HostNotFound"/>
      <xsd:enumeration value="NoSpace"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<!-- errors that can occur at the EC Service -->
<xsd:element name="resultECS">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="SUCCESS"/>
      <xsd:enumeration value="ReadLock"/>
      <xsd:enumeration value="WriteLock"/>
      <xsd:enumeration value="DatabaseError"/>
      <xsd:enumeration value="FileNotFound"/>
      <xsd:enumeration value="FileExists"/>
      <xsd:enumeration value="RedundancyError"/>
      <xsd:enumeration value="InvalidHandle"/>
      <xsd:enumeration value="MoveToSelf"/>
      <xsd:enumeration value="Timeout"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<!-- REQUESTS AND RESPONSES -->

<xsd:element name="getParameters">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="filename" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="getParametersResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:parameters"/>
      <xsd:element ref="tns:resultECS"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ls">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="dirname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="lsResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:directory"/>
      <xsd:element ref="tns:resultECS"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="sendInfo">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="url" type="xsd:string"/>
      <xsd:element ref="tns:resultSE"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="sendInfoResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:resultECS"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="open">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="filename" type="xsd:string"/>
      <xsd:element name="mode" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="openResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="handle" type="xsd:int"/>
      <xsd:element ref="tns:parameters"/>
      <xsd:element ref="tns:resultECS"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="keepalive">
  <xsd:complexType>
```

```
      <xsd:sequence>
        <xsd:element name="handle" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="keepaliveResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:resultECS"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="close">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="handle" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="closeResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:resultECS"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="delete">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="filename" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="deleteResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:resultECS"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="mkdir">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="dirname" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="mkdirResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:resultECS"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="rmdir">
    <xsd:complexType>
      <xsd:sequence>
```

```
        <xsd:element name="dirname" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="rmdirResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:resultECS"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="mv">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="from" type="xsd:string"/>
        <xsd:element name="to" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="mvResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:resultECS"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>


  <!-- RESOURCE PROPERTIES -->

  <xsd:element name="desiredRedundancy" type="xsd:int"/>
  <xsd:element name="averageRedundancy" type="xsd:int"/>
  <xsd:element name="minimumRedundancy" type="xsd:int"/>
  <xsd:element name="maximumRedundancy" type="xsd:int"/>
  <xsd:element name="numberOfStorageElements" type="xsd:int"/>
  <xsd:element name="overallCapacity" type="xsd:int"/>
  <xsd:element name="freeCapacity" type="xsd:int"/>

  <xsd:element name="ECSResourceProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:desiredRedundancy"/>
      <xsd:element ref="tns:averageRedundancy"/>
      <xsd:element ref="tns:minimumRedundancy"/>
      <xsd:element ref="tns:maximumRedundancy"/>
      <xsd:element ref="tns:numberOfStorageElements"/>
      <xsd:element ref="tns:overallCapacity"/>
      <xsd:element ref="tns:freeCapacity"/>
    </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  <xsd:element name="getECSResourceProperties">
    <xsd:complexType/>
  </xsd:element>

</xsd:schema>
</types>


<!--============================================================
```

```
                    M E S S A G E S

  ==============================================================-->
<message name="getParametersInputMessage">
  <part name="parameters" element="tns:getParameters"/>
</message>
<message name="getParametersOutputMessage">
  <part name="parameters" element="tns:getParametersResponse"/>
</message>

<message name="lsInputMessage">
  <part name="parameters" element="tns:ls"/>
</message>
<message name="lsOutputMessage">
  <part name="parameters" element="tns:lsResponse"/>
</message>

<message name="sendInfoInputMessage">
  <part name="parameters" element="tns:sendInfo"/>
</message>
<message name="sendInfoOutputMessage">
  <part name="parameters" element="tns:sendInfoResponse"/>
</message>

<message name="openInputMessage">
  <part name="parameters" element="tns:open"/>
</message>
<message name="openOutputMessage">
  <part name="parameters" element="tns:openResponse"/>
</message>

<message name="keepaliveInputMessage">
  <part name="parameters" element="tns:keepalive"/>
</message>
<message name="keepaliveOutputMessage">
  <part name="parameters" element="tns:keepalive"/>
</message>

<message name="closeInputMessage">
  <part name="parameters" element="tns:close"/>
</message>
<message name="closeOutputMessage">
  <part name="parameters" element="tns:closeResponse"/>
</message>

<message name="deleteInputMessage">
  <part name="parameters" element="tns:close"/>
</message>
<message name="deleteOutputMessage">
  <part name="parameters" element="tns:closeResponse"/>
</message>

<message name="mkdirInputMessage">
  <part name="parameters" element="tns:close"/>
</message>
<message name="mkdirOutputMessage">
  <part name="parameters" element="tns:closeResponse"/>
</message>

<message name="rmdirInputMessage">
```

```
  <part name="parameters" element="tns:close"/>
</message>
<message name="rmdirOutputMessage">
  <part name="parameters" element="tns:closeResponse"/>
</message>

<message name="mvInputMessage">
  <part name="parameters" element="tns:close"/>
</message>
<message name="mvOutputMessage">
  <part name="parameters" element="tns:closeResponse"/>
</message>



<message name="getECSResourcePropertiesInputMessage">
  <part name="parameters" element="tns:getECSResourceProperties"/>
</message>
<message name="getECSResourcePropertiesOutputMessage">
  <part name="parameters" element="tns:ECSResourceProperties"/>
</message>



<!--============================================================

                       P O R T T Y P E

  ============================================================-->
<portType name="ECSPortType"
    wsrp:ResourceProperties="tns:ECSResourceProperties">

  <operation name="getParameters">
    <input message="tns:getParametersInputMessage"/>
    <output message="tns:getParametersOutputMessage"/>
  </operation>

  <operation name="ls">
    <input message="tns:lsInputMessage"/>
    <output message="tns:lsOutputMessage"/>
  </operation>

  <operation name="sendInfo">
    <input message="tns:sendInfoInputMessage"/>
    <output message="tns:sendInfoOutputMessage"/>
  </operation>

  <operation name="open">
    <input message="tns:openInputMessage"/>
    <output message="tns:openOutputMessage"/>
  </operation>

  <operation name="keepalive">
    <input message="tns:keepaliveInputMessage"/>
    <output message="tns:keepaliveOutputMessage"/>
  </operation>

  <operation name="close">
    <input message="tns:closeInputMessage"/>
    <output message="tns:closeOutputMessage"/>
  </operation>
```

```
  <operation name="delete">
    <input message="tns:deleteInputMessage"/>
    <output message="tns:deleteOutputMessage"/>
  </operation>

  <operation name="mkdir">
    <input message="tns:mkdirInputMessage"/>
    <output message="tns:mkdirOutputMessage"/>
  </operation>

  <operation name="rmdir">
    <input message="tns:rmdirInputMessage"/>
    <output message="tns:rmdirOutputMessage"/>
  </operation>

  <operation name="mv">
    <input message="tns:mvInputMessage"/>
    <output message="tns:mvOutputMessage"/>
  </operation>

  <operation name="getECSResourceProperties">
    <input message="tns:getECSResourcePropertiesInputMessage"/>
    <output message="tns:getECSResourcePropertiesOutputMessage"/>
  </operation>

</portType>

</definitions>
```

B WSDL des SE Properties Service

Listing B.1: SEProperties.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ECS"
       targetNamespace="http://jenshellwig.de/namespaces/ECS"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://jenshellwig.de/namespaces/ECS"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/"
            "wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


<!--============================================================

                      T Y P E S

  ============================================================-->
<types>
<xsd:schema targetNamespace="http://jenshellwig.de/namespaces/ECS"
  xmlns:tns="http://jenshellwig.de/namespaces/ECS"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- RESOURCE PROPERTIES -->
  <xsd:element name="storageElement">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="totalSpace" type="xsd:int"/>
        <xsd:element name="freeSpace" type="xsd:int"/>
        <xsd:element name="toRemove" type="xsd:int"/>
        <xsd:element name="url" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="SEResourceProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:storageElement" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  <xsd:element name="getSEResourceProperties">
    <xsd:complexType/>
  </xsd:element>

</xsd:schema>
</types>


<!--============================================================
```

```
                       M E S S A G E S

  ===============================================================-->
<message name="getSEResourcePropertiesInputMessage">
  <part name="parameters" element="tns:getECSResourceProperties"/>
</message>
<message name="getSEResourcePropertiesOutputMessage">
  <part name="parameters" element="tns:ECSResourceProperties"/>
</message>



<!--===============================================================

                       P O R T T Y P E

  ===============================================================-->
<portType name="SEPortType"
    wsrp:ResourceProperties="tns:SEResourceProperties">
  <operation name="getSEResourceProperties">
    <input message="tns:getSEResourcePropertiesInputMessage"/>
    <output message="tns:getSEResourcePropertiesOutputMessage"/>
  </operation>

</portType>

</definitions>
```

Literaturverzeichnis

[AGG07] *GT 4.0: Information Services: Aggregator Framework*, 7 2007, http://www.globus.org/toolkit/docs/4.0/info/aggregator/index.pdf .

[ALL07] ALLCOCK, WILLIAM E.: *Programming with the Globus Toolkit GridFTP Client Library*, 2007, ftp://info.mcs.anl.gov/pub/tech-reports/reports/P1285.pdf .

[ANT07] *Apache Ant 1.7.0 Manual*, 7 2007, http://ant.apache.org/manual/index.html .

[ASY07] *Globus Toolkit 3.2 Developer's Guide: Asynchronous Event Handling*, 5 2007, http://www-unix.globus.org/toolkit/docs/3.2/developer/globus-async.html .

[CAS07] *GT 4.0: Security: Community Authorization Service*, 7 2007, http://www.globus.org/toolkit/docs/4.0/security/cas/index.pdf .

[FOS01] FOSTER, IAN, C. KESSELMAN und S. TUECKE: *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. In: *International J. Supercomputer Applications*, 2001.

[FOS04] FOSTER, IAN und CARL KESSELMAN: *The Grid. Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, ISBN-13: 978-1558609334, 2004.

[GFD03] ALLCOCK, W.: *GridFTP: Protocol Extensions to FTP for the Grid*, http://www.ogf.org/documents/GWD-R/GFD-R.020.pdf .

[GFS07] *Grid File System Working Group (GFS-WG)*, http://phase.hpcc.jp/ggf/gfs-rg/ .

[GPFS06] *An Introduction to GPFS*, 2006, http://www-03.ibm.com/systems/clusters/software/whitepapers/gpfs-intro.pdf .

[GT4] *The Globus Alliance*, http://www.globus.org/toolkit/about.html .

[IF02] FOSTER, IAN: *What is the Grid? A Three Point Checklist*. Argonne National Laboratory, 2002, http://www-fp.mcs.anl.gov/ foster/Articles/WhatIsTheGrid.pdf .

[IND07] *GT 4.0: Information Services: Index*, 7 2007, http://www.globus.org/toolkit/docs/4.0/info/index/index.pdf .

[JAK04] JAKOBY, ANDREAS: *Kodierung und Sicherheit*. 2004, http://www.tcs.uni-luebeck.de/pages/jakoby/veranstaltungen/vorlesungen/CodeSichWS0405.pdf .

[LA97] KOECHER, MAX: *Lineare Algebra und analytische Geometrie*. Springer, Berlin, ISBN-13: 978-3540629030, 1997.

[LUB02] LUBY, M.: *Lt Codes*. In: *Symposium on Foundations of Computer Science*, Seite 271. IEEE Computer Society, 2002.

[OGSA06] D., BERRY, A. DJAOUI, A. GRIMSHAW, B. HORN, F. MACIEL, F. SIEBENLIST, R. SUBRAMANIAM, J. TREADWELL und J. VON REICH: *The Open Grid Services Architecture, Version 1.5*. 2006, http://www.ggf.org/documents/GFD.80.pdf .

[PIT06] PITKANEN, M., J. KARPPINEN und M. SWANY: *GridBlocks DISK - Distributed Inexpensive Storage with K-availability*. In: *HPDC Workshop on Next-Generation Distributed Data Management*, 2006, http://gridblocks.hip.fi/opencms/export/sites/gridblocks/downloads/hpdc06_paper.pdf .

[SIL05] SILVA, VLADIMIR: *Grid Computing For Developers*. Charles River Media, ISBN-13: 978-1584504245, 2005.

[SOT06] SOTOMAYOR, BORJA und L. CHILDERS: *Globus Toolkit 4 Programming Java Services*. Morgan Kaufmann Publishers, ISBN-13: 978-0-12-369404-1, 2006.

[SRB07] *SRB User Manual*, 5 2007, http://www.sdsc.edu/srb/index.php/SRB_User_Manual .

[TAN02] TANENBAUM, ANDREW S.: *Computer Networks*. Prentice Hall PTR, ISBN 0130661023, 2002.

[TRI07] *GT 4.0 WS MDS Trigger Service*, 7 2007, http://www.globus.org/toolkit/docs/4.0/info/trigger/ index.pdf .

[UML04] JECKLE, MARIO, C. RUPP, J.HAHN, B. ZENGLER und S.QUEINS: *UML2 Glasklar*. Carl Hanser Verlag Muenchen Wien, ISBN 3-446-22575-7, 2004.

[WS04] CZAJKOWSKI, KARL, D. FERGUSON, I. FOSTER, J. FREY, S. GRAHAM, I. SEDUKHIN, D. SNELLING, S. TUECKE und W. VAMBENEPE: *The WS-Resource Framework*, 2004, http://www.globus.org/wsrf/specs/ws-wsrf.pdf .