INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwicklung eines Grid Fault Tolerant Execution Services

Vitaly Hachuzky

Draft vom 31. März 2011

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwicklung eines Grid Fault Tolerant Execution Services

Vitaly Hachuzky

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Michael Schiffers
Abgabetermin:	31. März 2011

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

München, den 31. März 2011

.....

(Unterschrift des Kandidaten)

Abstract

Ian Foster und Carl Kesselman stellten im Jahre 1999 ein neues Konzept mit dem Namen *Grid Computing* vor, das neben Computer auch andere Arten von (IT-)Ressourcen (wie Software, Datenbanken, Rechenleistungen, Speicherplatz oder spezielle Hardware) beinhaltet und sie miteinander vernetzen kann. Aufgrund des beliebigen und weltweiten Zugriffs auf Ressourcen über das Internet sollte die Vereinigung aller Grid Systemen zu einer Art von Generalisierung des World Wide Web werden (7).

Verlässlichkeit von Grid-Ressourcen und -Diensten in *Virtuellen Organisationen* (VO) sind für jedes VO-Mitglied sowie für potentielle VO-Mitglieder von besonderer Bedeutung. Leider, fehlt zur Zeit eine einheitliche Implementierung der Behandlung von Grid Faults bei unterschiedlich existierenden *Grid Middleware Technologien* (wie, z.B., Globus Toolkit, gLite, Unicore, etc.). Diese Vereinheitlichung wäre aber notwendig für weitere Entwicklungen und Verbreitungen von Grid Konzepten.

Diese Diplomarbeit untersucht die Möglichkeiten der dynamischen Migration mit anschließendem Re-Scheduling eines Grid Jobs im Fall eines Ressource Crash-Fehlers. Dafür wurde eine neue Abstraktionsschicht eingeführt, die in Form eines *Grid Fault Tolerant Execution Services* (GFTES) jedem VO-Mitglied zur Verfügung gestellt werden kann. Es wurde dabei ein Failover Re-Scheduling Konzept herausgearbeitet und prototypisch implementiert, das die Ausführung des Grid Jobs von verschiedenen Grid Middleware Technologien entkoppelt und in einer darüber liegenden Schicht integriert.

Inhaltsverzeichnis

1. Einleitung

Grid Computing (GC) ist eine wichtige Schlüsseltechnologie. Sie bietet vielen Wissenschaftlern und Ingenieuren in verschiedenen Forschungs- und Industriegebieten eine Möglichkeit, große Zahl von anspruchsvollen Problemen in einer komplexen heterogenen Umgebung zu lösen, sowie auf einer neuen Ebene miteinander zu kooperieren.

Durch eine Reihe von auf etablierten Protokollen aufgebauten Service-Schnittstellen kombiniert GC High Performance Capability und High Throughput Computing zusammen mit Data Intensive Computing und On-Demand Computing in ein für Zusammenarbeit fähiges Netz.

Grid Computing ist nicht mehr nur auf den Einsatz in der wissenschaftlichen Gemeinde beschränkt. So gibt es viele Beispiele erfolgreicher Grid Projekte in der Industrie und immer mehr große IT-Firmen beteiligen sich an der Entwicklung von Grid Standards und Technologien. GC und Virtualisierung von IT-Infrastrukturen sind eng verwandt und bieten Möglichkeiten zur Kostenreduktion durch eine verbesserte Ausnutzung vorhandener Ressourcen.

Die *Grid Middleware* (GM) spielt bei dem Grid Computing eine entscheidende Rolle, da sie einem Grid Benutzer den Zugriff auf unterschiedliche heterogene Ressourcen ermöglicht. Eine sehr wichtige Bedeutung hat dabei auch das Konzept von Virtuellen Organisationen(VO).

Eine VO ist eine Form von Organisation, bei der sich rechtlich unabhängige Unternehmungen und Einzelpersonen über das Internet für einen gewissen Zeitraum zu einem gemeinsamen Geschäftsverbund zusammenschließen. Gegenüber Dritten tritt VO wie eine einheitliche Organisation auf. Durch die Virtualität ist der physische Standort der einzelnen Teilnehmer nicht von Bedeutung.

In einer heterogenen Umgebung ist die Wahrscheinlichkeit für Fault relativ hoch. Deswegen ist die Fehlertoleranz die notwendige Eigenschaft aller Grid Middleware Technologien, welche aus verschiedenen Bereichen der Wissenschaft stammen und zum Teil mit unterschiedlichen Anforderungen entwickelt worden waren. Standardisierung der Grid Architektur und der Grid Protokolle ist ein wichtiger Schritt auf dem Weg zum globalen Grid, da nur dadurch die Interoperabilität zwischen den verschiedenen Grid Middleware Arten erreicht werden kann.

1.1. Motivation

Als Basisidee für Grid Computing dient das Konzept der *Service Orientierte Architektur*. In einer Service Orientierten Architektur werden die Komponenten eines verteilten Systems durch ihre Schnittstellen und ihr Verhalten definiert. Die Implementation spielt dabei

nur eine untergeordnete Rolle (22). Die Interaktion zwischen den einzelnen Diensten regeln vordefinierte Protokolle. Web Services (WS) sind *zustandslos*, d.h. es werden keine Zustandsinformationen über einzelne Clients gespeichert. Jede Anfrage, die von einem Client gestellt wird, muss alle Informationen enthalten, um diese Anfrage zu bearbeiten und es dürfen keine weiteren Informationen auf dem Server, z.B. in Form von Session-Daten, für die Interpretation der Anfrage notwendig sein. Es weiß also nur der Client, an welcher Stelle in der Anwendung er sich befindet und was die nächsten sinnvollen Schritte sind (23). Dies kann in manchen Einsatzgebieten eine große Einschränkung darstellen. Aus diesem Grund können (bestehende) Web-Dienste die Funktionen eines Web Service Ressource Frameworks (WSRF) implementieren und so zustandsbehaftete Funktionalität bekommen.

Die *Open Grid Service Architecture* (OGSA) ist eine erweiterbare Systemarchitektur für das Erbringen, das Zusammenspiel und den Konsum von Diensten im Grid. Das von OGSA definierte Komponentenmodell macht es für Anwendungen möglich, auf eine einfache Weise die im Grid angebotenen Dienste zu nutzen und miteinander zu koppeln. Es geht besonders auf die grundlegenden Probleme in Grid-Umgebungen ein, die bisher zum Teil für jedes Projekt individuell gelöst werden mussten. So standardisiert OGSA die Identifizierung, Authentifizierung und Autorisierung von Teilnehmern und Diensten, das Auffinden und den Aufruf von Diensten, das Aushandeln von Verfahren, die Überwachung und Kontrolle verteilter Anwendungen und die Einrichtung und Unterstützung dynamischer Arbeitsgruppen.

Die *Execution Management Services* (OGSA-EMS) befassen sich mit Lebenszyklus von Aufgaben (Units of Work), die im Grid ausgeführt werden sollen (25). Unter „Aufgaben“ werden OGSA-Anwendungen, Jobs ebenso wie Legacy-Anwendung, verstanden. Das Management dieser Units of Work umfasst die Identifizierung geeigneter Ausführungskandidaten, deren Lokalisierung, Vorbereitung für die Ausführung, Instanziierung und Ausführung sowie die Beendung.

Mit einer steigenden Anzahl von Rechenknoten erhöht sich gleichzeitig die Wahrscheinlichkeit von Fehlern (Fault). Da es viele verschiedene Arten von Fehler gibt, ist Fault Detection sowie Fault Tolerance eine der wichtigsten Aufgaben von jeder OGSA implementierenden Grid Middleware. Leider, gibt es zur Zeit keine einheitlichen Standards dafür, wie Fehlertoleranz funktionieren soll. Vielfältigkeit und Unterschiedlichkeit der Arten von Grid Middleware Technologien stellten einen Engpass für ihre Benutzung, da sogar die Job Beschreibung bei jedem der Grid Middleware einen eigenen Standard hat.

1.2. Aufgabenstellung

Ziel dieser Arbeit ist die Entwicklung eines *Grid Fault Tolerant Execution Services* (GFTES) für heterogene Grids. Im Rahmen dieser Entwicklung wird für Crash-Fehler untersucht, wie degeneratives Failover durch dynamische Migration fehlerhafter Jobs und anschließendem Re-Scheduling realisiert werden kann. Fokussiert wird auf den Anwendungsfall „Migration auf Ressourcen anderer Administrationsdomänen derselben Virtuellen Organisation (VO)“. Dazu sind zuerst folgende Fragen zu beantworten:

1. Was ist ein Fault?

2. Was ist eine Fault Tolerance?
3. Wie kann Fehlertoleranz Middleware-unabhängig realisiert werden?
4. Was passiert im Fall eines Fehlers bei der Grid Ressource und welche Reaktion darauf folgen soll?

Zur Implementierung von GFTES für heterogene Grid Middleware wird aufbauend auf den bestehenden Grid Middleware Technologien ein neuer Web-Dienst im höheren Abstraktionsschicht gebaut, der die gesamte Kommunikation zwischen dem User und Grid Middleware übernimmt. Abbildung 1.1 zeigt die schematische Darstellung dieses neuen Services, welcher im folgenden GFTES genannt wird.

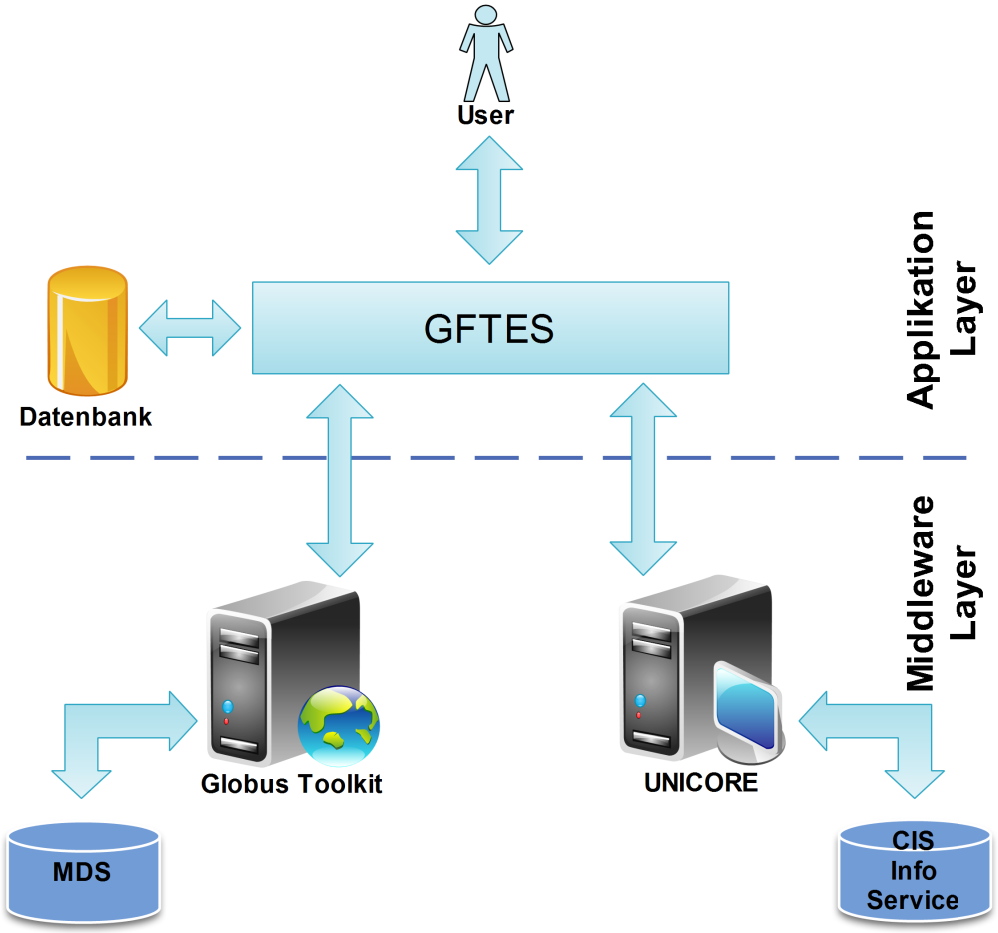

Abbildung 1.1.: Grid Fault Tolerant Execution Service (GFTES)

Bei Entwicklung von GFTES werden insbesondere folgende Komponenten in dieser Diplomarbeit behandelt, die in der Abbildung 1.1 zusammen mit Beispielen von darunter liegenden Grid Middleware Technologien abgebildet sind:

- Grid Fault Tolerant Execution Service

- Datenstruktur zum Speichern aller Informationen, die für Job Execution und Job Recovery relevant sind
- Schnittstellen zu einzelnen GM Technologien
- API zum Entwickeln von Client-Software
- Deployment von GFTES in einer Testumgebung

Das entwickelte System soll unabhängig von der verwendeten Grid Middleware sein und nur auf bestehende Grid-Befehle zugreifen. Die GM darf dabei nicht verändert werden. Die Anbindung an die Grid Middleware Technologien erfolgt dabei ausschließlich über die von Grid Middleware bereitgestellten Funktionen.

Eine Unterstützung für Grid-Portalsoftware, welche über einen Webbrowser oder über eine eigene Anwendung mit einer GUI gesteuert wird, ist in dieser Diplomarbeit nicht vorgesehen.

1.3. Vorgehensweise

Diese Diplomarbeit besteht aus folgenden drei Teilen: Analyse der Anforderungen, Entwurf des Systems und prototypische Implementierung des Grid Fault Tolerant Execution Services. In Abbildung 1.2 ist die konsequente Vorgehensweise nochmals schematisch dargestellt.

Analyse der Anforderungen: in der ersten Phase der Analyse werden zunächst die Grundlagen des Grid Computings im Allgemein dargestellt. Dazu werden die wichtigsten Begriffe wie Grid Computing, Fehler und Fehlertoleranz, Job und Re-Scheduling eingeführt. Als nächstes, wird ein Überblick über verschiedene Grid Middleware Implementierungen, wie z.B. Globus Toolkit, UNICORE und gLite, verschafft. Besondere Aufmerksamkeit bei dieser Analyse wird den Fehlertoleranz- und Migrationsmechanismen dieser GM gewidmet.

Zusätzlich werden die Szenarien für die Fehlerentdeckung mit anschließenden Re-Scheduling und Job Migration anhand eines Beispiels gezeigt. Anhand dieser Szenarien wird ersichtlich, dass bestehende Lösungen nicht adäquat sind und daher erweitert werden müssen. Mit Hilfe dieser Szenarien werden Use Cases entwickelt, welche die genauen funktionalen Anforderungen an Grid Fault Tolerant Execution Service definieren.

Entwurf des Systems: anhand der Anforderungen wird der GFTES entworfen. Der Service mit seinen Schnittstellen wird als Objektmodell definiert und die Schnittstellen werden spezifiziert. Des Weiteren wird eine Datenstruktur zur Speicherung der relevanten Informationen in der GFTES-Datenbank entwickelt.

Implementierung eines Prototyps: für die Realisierung wird ein Testbed erzeugt, welches alle Komponente zum Testen von GFTES enthält. Das entwickelte Konzept wird in diesem Testbed prototypisch implementiert und anschließend wird der GFTES System anhand der aufgelisteten Anforderungen bewertet.

1.4. Übersicht

Diese Diplomarbeit gliedert sich in sieben Kapiteln. In der Abbildung 1.2 ist die Struktur dieser Diplomarbeit schematisch präsentiert.

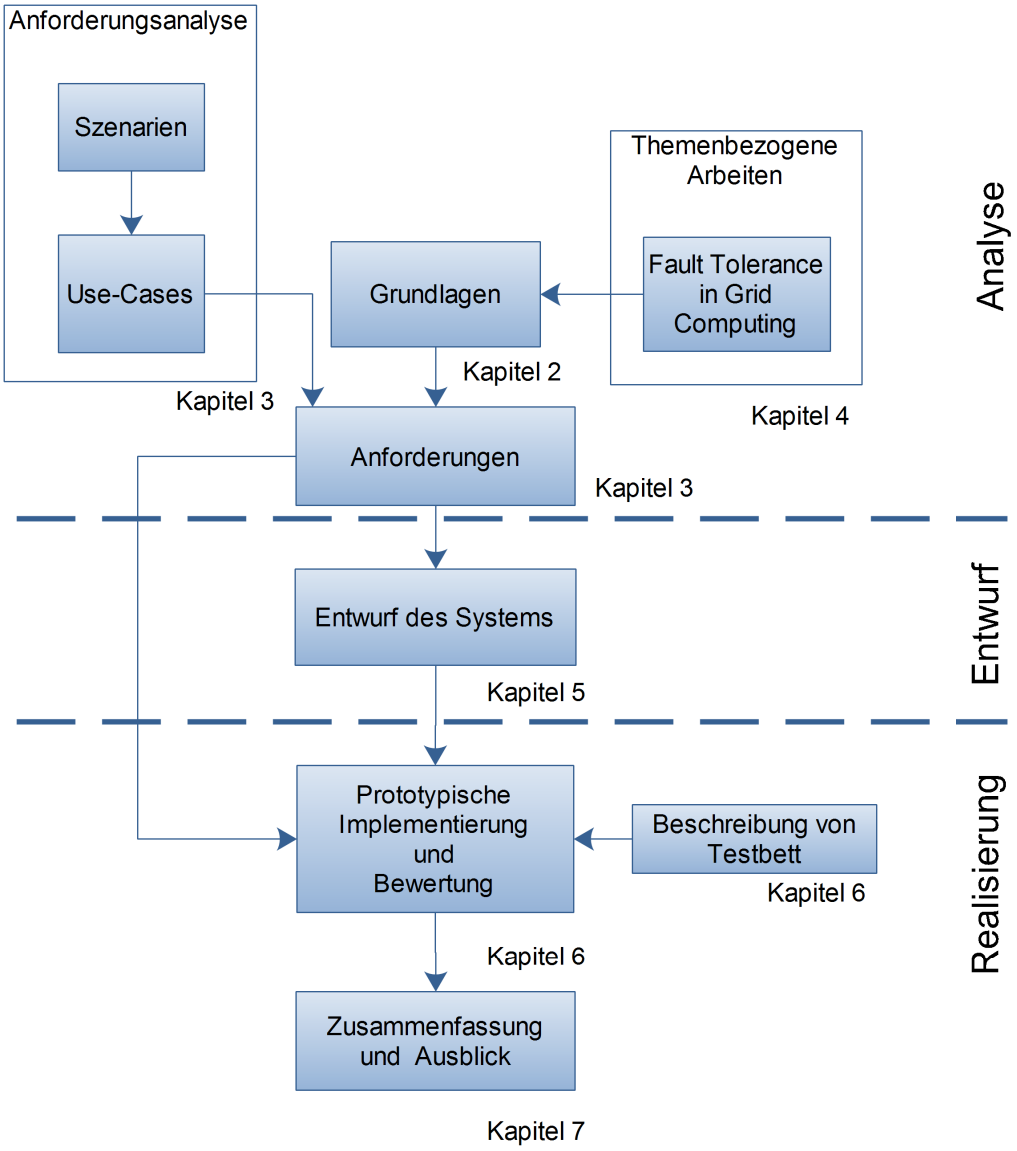

Abbildung 1.2.: Gliederung und Vorgehensweise

Das **zweite Kapitel** beschäftigt sich mit Grundkonzepten, die in dieser Arbeit verwendet werden. Hier werden die Begriffe Grid Computing, Web Service, Job Management, Faults Taxonomy, Fault Tolerance, Job Migration eingeführt.

Das **dritte Kapitel** beschreibt Ergebnisse der Anforderungsanalyse. Am Anfang werden die Szenarien beschrieben, die für den GFTES erzeugt wurden. Als nächstes werden die aus den Szenarien abgeleiteten Use Cases beschrieben. Anschließend werden die funktionalen und

nicht-funktionalen Anforderungen aufgelistet.

Das **vierte Kapitel** stellt die aktuellen Grid Middleware Technologien vor. Es wird analysiert, ob sie die im Kapitel drei definierten Anforderungen erfüllen.

Das **fünfte Kapitel** beschreibt den konzeptuellen Entwurf von GFTES sowie seiner Schnittstellen. In diesem Teil wird der Objektentwurf des GFTES vorgestellt.

Das **sechste Kapitel** erläutert anfänglich das Testbed, wo das aus dem Kapitel fünf erstellte System prototypisch implementiert wird. Danach werden die wichtigsten Teile des Quellcodes vorgestellt und erklärt. Am Ende des Kapitels werden Details zu Deployment des GFTES auf einem Zielsystem gegeben.

Das **siebte Kapitel** fasst die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick und Anregungen für die nachfolgenden Arbeiten.

Teil I.

Analyse

2. Grundlagen

In diesem Kapitel werden wesentliche Definitionen und Standards vorgestellt, die zu einem besseren Verständnis für Problematik und Inhalt weiterer Kapiteln dieser Arbeit beitragen sollen.

Zuerst wird das Konzept von *Grid* und *Grid Services* dargestellt. Dieses Thema ist zweifellos wesentlich größer und umfangreicher als hier beschrieben werden kann. Deswegen werden in Rahmen dieses Kapitels nur die Grundideen solcher Dienste präsentiert.

Ein weiterer wichtiger Begriff ist *Execution Service*, der im danach folgendem Abschnitt beschrieben wird.

Im Weiteren wird das Thema der *Fehlertoleranz* angefasst. Dieses Thema beschäftigt Wissenschaftlern schon seit langer Zeit (ein Beispiel solcher Arbeit ist (20)), da unzählige Arten von möglicher Fehler existieren. Im Rahmen dieser Arbeit beschäftigen wir uns nur mit einigen Fehlerklassen, die am Ende dieses Kapitels genannt werden.

2.1. Grids und Grid Computing

Der Begriff *Grid Computing* ist leider nicht eindeutig definiert und wird auch mit zum Teil unterschiedlichen Bedeutungen verwendet. Traditionell stammt der Begriff *Grids* aus dem Stromnetz, in das mehrere Erzeuger Strom einleiten und aus dem mehreren Verbraucher den Strom herausnehmen.

Ein Computing Grid ist ein Netz aus Rechenressourcen, die zur Verfügung gestellt werden und von Benutzern bei Bedarf belegt werden. Zu häufigen Anwendungsszenarien zählt man heute die Berechnung von großen Problemen aus dem technisch-wissenschaftlichen Bereich. Die eigentliche Vision geht aber weiter, da nicht nur Rechenkapazität sondern auch Speicher, Programme und Informationen in einem globalen System geteilt werden sollen.

Als Grid Computing wird die Aufgabenlösung in einem Netz von verbundenen Computern mit gemeinsamer Nutzung von Rechenleistung und Speicherplatz bezeichnet. Die Vernetzung erfolgt, wie beim World Wide Web, über das Internet. Der Unterschied liegt in der Nutzung: das WWW verwendet Internet zum Informationsaustausch und Grid Computing verwendet es zum Teilen von Rechenleistung. Dadurch entsteht eine Art von virtuellen Supercomputer, der zur Lösung besonders rechenintensiver Probleme verwendet werden kann.

Die Einsatzbereiche für Grids sind vielfältig und reichen von wissenschaftlichen Bereichen wie Biologie, Pharmaforschung oder Teilchenphysik über kommerzielle Bereiche wie e-Commerce bis zum Erstellen von aufwendigen Animationen in der Filmbranche (3).

Die primäre Aufgabe eines Grids ist die gemeinsame Nutzung von Ressourcen. Dazu müssen Richtlinien für alle beteiligten Individuen oder Institutionen erstellt werden, die die Rechte und Bedingungen für die Nutzung der Ressourcen eindeutig festlegen. Ein solcher Zusammenschluss von Beteiligten und vordefinierten Richtlinien wird *Virtuelle Organisation* genannt.

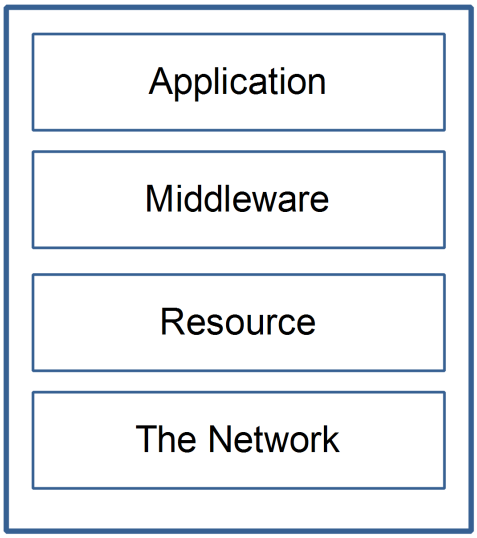

Abbildung 2.1.: Grid Architektur (8)

Virtuelle Organisationen sind ein zentraler Bestandteil des Grid Computings. Verschiedene Institutionen oder Individuen können sich zu einer Virtuellen Organisation zusammenschließen und treten somit nach außen als ein einheitliches Unternehmen auf. Dadurch können die verschiedenen Eigenschaften der Beteiligten flexibel kombiniert werden, um ein konkretes Problem zu lösen.

Die *Architektur eines Grids* kann mit Schichtenmodell wie in der Abbildung 2.1 beschrieben werden. Jede Ebene hat eine eigene Funktionalität. Die Netz Ebene - unterste Schicht - ist für Verbindung von Ressourcen zuständig. Die Resource Ebene stellt die Ressourcen wie Computern oder Speichersysteme dar. In der Middleware Ebene befinden sich die Grid Middleware Dienste die den Zugriff auf die darunter liegende Ressource verwalten und koordinieren. Schließlich auf der Anwendungsebene befinden sich die Anwendungen der Benutzer, die auf die Grid Ressource zugreifen möchten.

Web Services (WS) bilden die Grundlage von Grids, die auf dem Konzept von Service-orientierter Architektur basieren. Ein Web Service benötigt für sein Wesen eine Umgebung, in der er ausgeführt werden kann. Diese Umgebung wird ein *Web Application Server* oder auch Container, oder Web Service Container genannt. Web Services sind von Natur aus zustandslos (*stateless*) und speichern keine Informationen über vergangene Interaktionen mit dem Client. Jede Anfrage, die vom Client gestellt wird, muss alle Informationen enthalten, um diese Anfrage verstehen zu können. Nur dem Client ist bekannt, an welcher Stelle in der Anwendung er sich gerade befindet und was die nächsten sinnvollen Schritte sind. Eine

schematische Darstellung eines zustandslosen Web Services findet man in Abbildung 2.2.

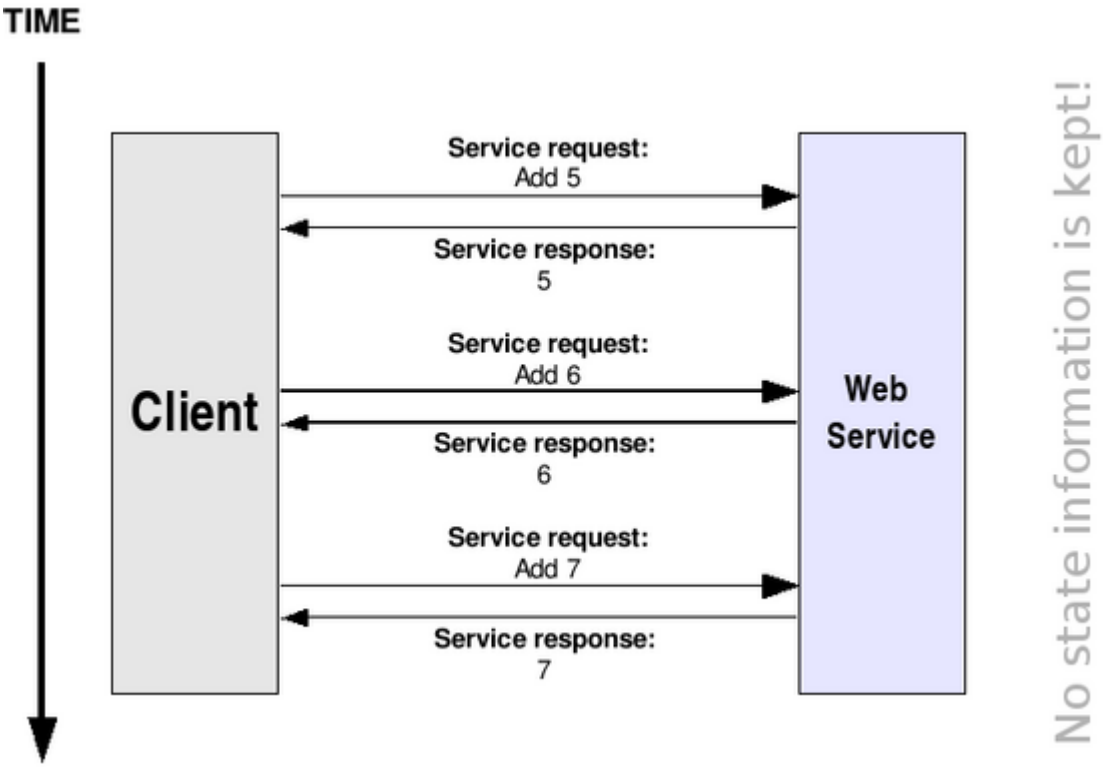

Abbildung 2.2.: Zustandloser Web Service (27)

Die Zustandslosigkeit von Web Services kann in manchen Einsatzgebieten eine große Einschränkung darstellen. Deswegen können (bestehende) Web-Services die Funktionen von *Web-Service Ressource Framework* (WSRF) implementieren und in dieser Weise die zustandsbehaftete (*stateful*) Funktionalität bekommen wie in der Abbildung 2.3 gezeigt ist.

Dabei werden Daten persistent (persistent = beständig, transient = flüchtig) in Ressourcen (Container für Zustandsinformationen) so gespeichert, dass einzelne Clients gezielt auf die persistent gespeicherten Daten zugreifen und diese verändern können. Dabei sind die Daten nicht an einen einzelnen Client gebunden, sondern können auch von „neuen“ Clients weiter verwendet werden.

Die Adressierung der Ressource erfolgt über das sogenannte *Implied Resource Pattern*. Jede Ressource bekommt einen eigenen und eindeutigen *Ressource-Identifikator*, der in den Reference Properties gespeichert wird. Welche Reference Property verwendet wird, wird in der Endpoint Reference des Web Services definiert und in der Anfrage verwendet. Diese Referenzierung wird über einen entsprechenden Eintrag im SOAP-Nachrichtenheader definiert. Ein Beispiel einer SOAP-Nachricht kann man im Listing 2.1 sehen.

```
1 <?xml version="1.0" encoding="utf-8"?>
```

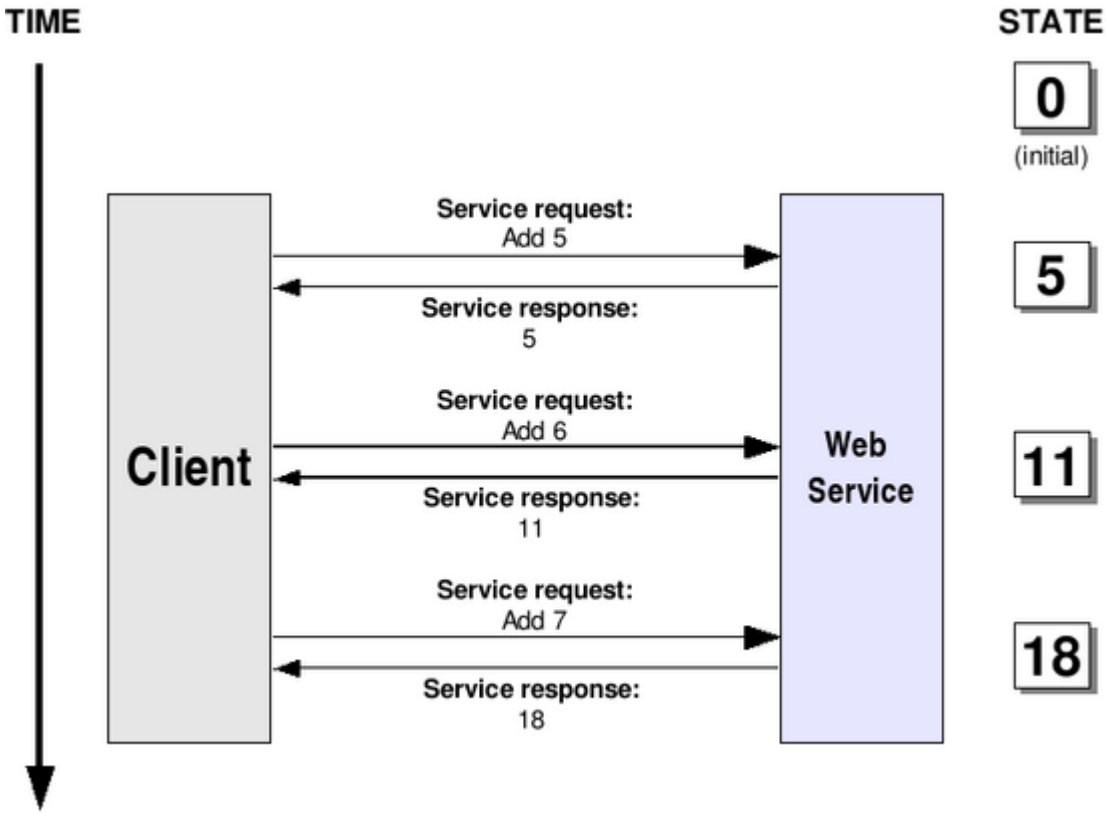

Abbildung 2.3.: Zustandsbehafteter Web Service (27)

```
2 <soap:Envelope nxmlns:soap="http://www.w3.org/2001/12/soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
3   <!-- Header der SOAP-Nachricht -->
4   <soap:Header>
5     <m:Trans nxmlns:m="http://w3schools.com/transaction/"
          soap:mustUnderstand="1" soap:actor="http://www.example.com/handle
          ">1234</m:Trans>
6     <wsa:EndpointReference>
7       <wsa:Address>http://bsp.de/wsrfWebServiceAdresse</wsa:Address>
8       <wsa:ReferenceProperties>
9         <tns:resourceID>A</tns:resourceID>
10      </wsa:ReferenceProperties>
11    </wsa:EndpointReference>
12  </soap:Header>
13  <!-- Inhalt der SOAP-Nachricht -->
14  <soap:body>
15  </soap:body>
16 </soap:Envelope>
```

Listing 2.1: Ein Beispiel von SOAP Message

Im Listing 2.1 kann man sehen, dass eine EndpointReference aus einer Adresse und ReferenceProperties besteht. Die Adressinformationen werden damit direkt in die WS-Nachrichten

gepackt: der SOAP-Header enthält Routing-Informationen zum Service (Address) und Aktionsinformationen für Services (ReferenceProperties). Man muss auch beachten, dass Endpoint References die gleiche URI haben können. Werden aber gleichzeitig unterschiedliche Reference Properties verwendet, handelt es sich auch um unterschiedliche WebServices. Enthält der Endpoint eine resourceID, wird er als WS-Resource Qualified Endpoint Reference genannt.

2.2. Grid Execution Services

Eine mögliche Softwarearchitektur für Grids ist die von Ian Foster 2002 mitentwickelte *Open Grid Services Architecture* (OGSA). Diese wird in Ansätzen bereits in Ihrem Vorläufer, der Open Grid Services Infrastructure (OGSI), beschrieben. Deren Grundidee ist die Darstellung von beteiligten Komponenten (Rechner, Speicherplatz, Mikroskope, etc.) als Grid-Services in einer offenen Komponentenarchitektur. Die Zusammenhänge zwischen OGSA, WSRF und zustandsbehaftete Web Services ist in Abbildung 2.4 gezeigt.

Ein *Grid-Job* ist eine Instanz vom Programm, das Benutzer im Grid ausführen lassen möchte. Typischerweise wird ein Job durch eine Job-Beschreibung in der festgelegten Sprache, wie *Resource Specification Language* (RSL) oder *Job Submission Description Language* (JSDL), definiert. Die Job-Beschreibung bestimmt neben der eigentlichen Applikation weitere Parameter (wie beispielsweise Stege-In und Stage-Out Daten oder Ressourcenanforderungen hinsichtlich der für die Ausführung des Jobs verwendeten Hardware).

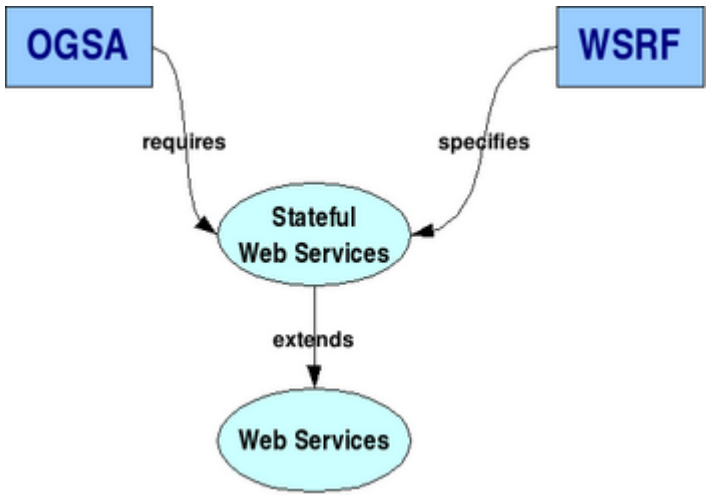

Abbildung 2.4.: Zusammenhänge zwischen OGSA, WSRF und zustandsbehaftete Web Services (28)

OGSA-BES Arbeitsgruppe bei Open Grid Forum hat in ihrer *Basic Execution Service Spezifikation*(9) eine Reihe von Web-Service-Schnittstellen beschrieben. Diese Spezifikation bestimmt das Interface für Erzeugung, Überwachung und Verwaltung von Computing Jobs, die in JSDL beschrieben werden. Ein Basic Execution Service führt diesen Job auf einer bestimmten Computing-Ressource aus. Als solche Ressource können beispielsweise einfache

PC-Arbeitsplatz, ein Rechner Cluster oder sogar ein anderer Basic Execution Service sein. Beim Entwerfen dieser Spezifikation versuchte die OGSA-BES-Arbeitsgruppe Balance zu erreichen, damit verschiedene BES-Implementationen von sehr unterschiedlichen Clients auf gleicher Art und Weise erreicht werden können und gleichzeitig zusätzliche Erweiterungen von OGSA-BES möglich sind. Diese Ergänzungen können in drei unterschiedlichen Bereichen gemacht werden: im Zustandsmodell (State Model), im Informationsmodell (Information Model) und im Ressourcenmodell (Resource Model).

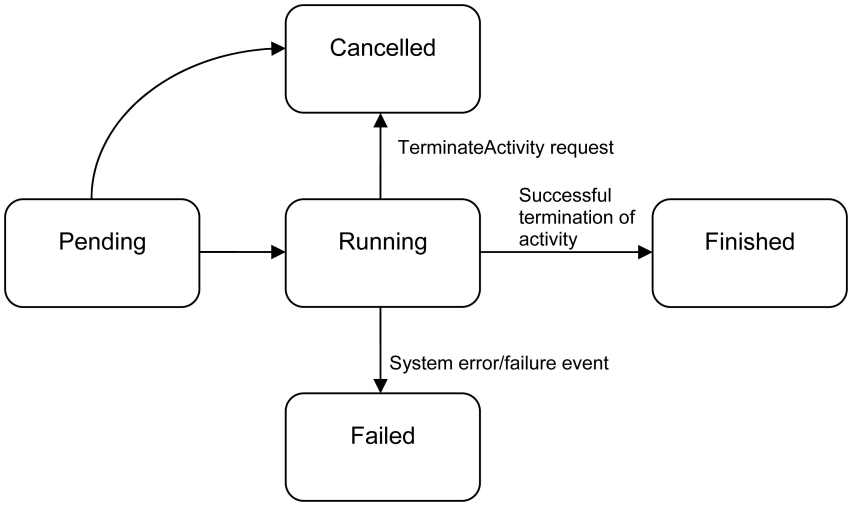

Abbildung 2.5.: Das Basis-Zustandsmodell des Basic Execution Service (9)

- **Das Zustansmodell**: Während der Ausführung beschreibt man den aktuellen Stand der Bearbeitung des Jobs durch seine Zustände. Verschiedene BES-Implementationen können unterschiedliche Reihe von Zustände haben. Um diese Unterschiede zu verallgemeinern, schreibt die BES-Spezifikation die Hauptzustände vor, sowie die Mechanismen wie diese Zustände erweitern werden können. Die Übergänge zwischen Haupzuständen sind in der Abbildung 2.5 dargestellt.

- **Das Informationsmodell**: Die Benutzer bekommen Zugriff auf verschiedene Informationen über den Execution Service sowie den aktuellen Stand ihrer Jobs. Unterschiedliche BES-Implementationen stellen verschiedene Informationen zur Verfügung. Die Spezifikation beschreibt die Reihe von Informationsattributen, die jeder Implementierung von BES Port-Type erkennen kann. Viele solche Attribute sind optional und können erweitert werden.

- **Das Ressourcenmodell**: Verschiede Modelle von Ressourcen werden durch unterschiedliche Profile ausgeprägt. Die BES Spezifikation schreibt nur allgemeine Funktion vor, um alle Attribute in einem einzigen Dokument abfragen zu können. Gleichzeitig schließt man nicht aus, dass unterschiedliche BES-Implementationen die anderen Mechanismen unterstützen können. Die anderen Port-Typen können mit der Port-Typen aus der BES-Spezifikation kombiniert werden.

Die in OGSA-BES Spezifikation beschriebenen Interfaces müssen auch von Grid Fault Tolerant Execution Service implementiert werden, damit der Zugriff auf seiner Funktionalität geregelt und benutzerfreundlich gestaltet wird. Dabei ist die Implementierung der Erweiterungen im Laufe des Entwerfens des Grid Execution Service nicht ausgeschlossen.

2.3. Fehlertoleranz

Lieferung von korrekten Computing und Kommunikation-Services beschäftigt wissenschaftliche Gemeinde schon seit langer Zeit. Ein korrekter Service wird genau dann geliefert, wenn dieser seine Funktionen erfüllt. Wenn man einen Service mit der Reihe interner und externer Zustände beschreibt, dann kann man die *Fault* (Fehlerursache) als ein unerwünschter Zustand des Systems definieren, der zu einer Fehler führen kann. In diesem Fall ist *Error* (Fehler) - ein Systemzustand, der nicht mehr seiner Spezifikationen entspricht. Ein *Failure* (Funktionsausfall) ist in dieser Umgebung dann ein Systemzustand, wenn weitere Diensterbringung nicht mehr möglich sind.

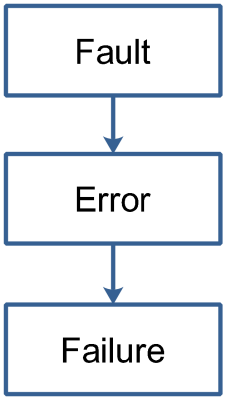

Abbildung 2.6.: Zusammenhang von Fault, Error und Failure (16)

Wenn funktionale Spezifikationen des Systems die Reihe der Funktionen beinhaltet, kann der Ausfall einer von denen oder mehrerer Funktionen gleichzeitig das System im *degradierten* Zustand hinterlassen, d.h. dass seine Funktionalität stark beschränkt wird. Diese Beschränkung der Funktionalität nennt man *Partial Failure*.

Die Definition von *Zuverlässigkeit* (Dependability) kann qualitativ und quantitativ beschreiben werden. Die qualitative Beschreibung von Dependability ist die Fähigkeit des Services solche Dienste zu liefern, denen man zurecht vertrauen kann. Die alternative, quantitative, Definition von Zuverlässigkeit: ist die Fähigkeit des Systems die Service-Failure zu vermeiden, wenn diese Funktionsausfälle viele zu oft auftreten.

In der letzten fünfzig Jahre wurden mehrere Mitteln für das Erreichen des Zuverlässigkeitszustands entwickelt. Diese Mitteln kann man in vier Hauptgruppen aufteilen:

- **Fault Prevention**: Mittel zur Vorbeugung von Systemausfälle

- **Fault Tolerance**: Mittel zum Verhindern von Service Failure beim Entstehen einer Fault
- **Fault Removal**: Mittel zur Reduktion der Anzahl und der Frequenz von Fehlern
- **Fault Forcasting**: Mittel zur vorherigen Schätzung zukünftiger Anzahl, Häufigkeit und der wahrscheinlichen Folgen eines Fault

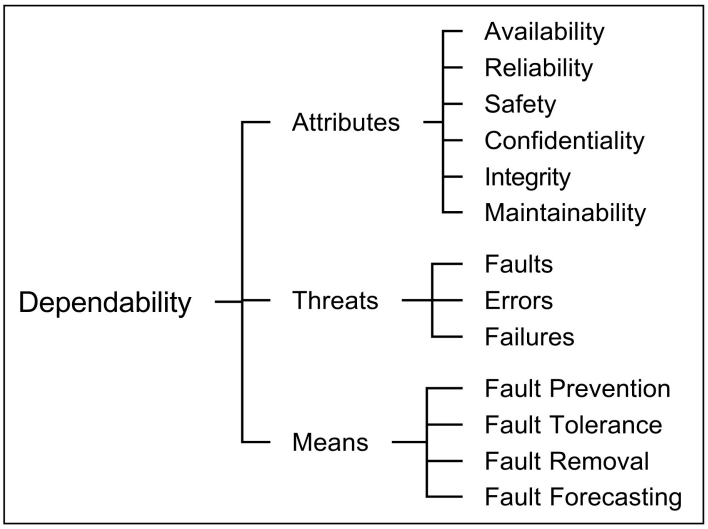

Abbildung 2.7.: Zuverlässigkeitstaxonomy (16)

Das komplette Schema von der Zuverlässigkeitstaxonomy kann der Abbildung 2.7 entnommen werden.

Alle Fehler, die das System während seines gesamten Lebenszyklus beeinflussen können, kann man in acht Klassen aufteilen, so wie es in der Abbildung 2.8 gezeigt ist. Man unterscheidet folgende Klassifikationskriterien:

1. *Phasen* des Lebens von System:
   - **Development Faults** entstehen während der Entwicklungszeit
   - **Operational Faults** entstehen während der Lieferung des Services
2. *Entstehungsort* von Fault:
   - **Internal Faults** entstehen innerhalb des Systems
   - **External Faults** entstehen außerhalb des Systems beispielsweise durch Kommunikation mit einem anderen System
3. *Phenomenologicale* Ursache von Fault:
   - **Natural Faults** sind von einem Natur-Phänomen ohne Teilnahme eines Menschen verursacht
   - **Human-Made Faults** sind Ergebnis menschlicher Aktivität

4. *Dimension* einer Fehler:
   - **Hardware (physische) Faults** sind in einer Hardware des Rechners entstanden
   - **Software (Information) Faults** sind von Software, Programmen, Daten, etc. verursacht
5. Art einer Fault:
   - **Malicious Faults** sind von einer Person mit bösartigen Absichten verursacht
   - **Non-Malicious Faults** sind von einer Person ohne irgendwelche böse Absichten verursacht
6. Vorhaben einer Person, die diese Fault verursacht hat:
   - **Deliberate Faults** sind von einer Person bewusst verursacht
   - **Non-Deliberate Faults** sind von einer Person unbewusst verursacht
7. *Kompetenz* der Person, die dieser Fault verursacht hat:
   - **Accidential Faults** sind zufällig verursachten
   - **Incompetence Faults** sind wegen der nicht ausreichenden Kompetenz des Personals verursacht
8. *Nachhaltigkeit* einer Fault
   - **Permante Faults** werden im System dauernd wiederholt
   - **Transient Faults** sind nur vorübergehend im System

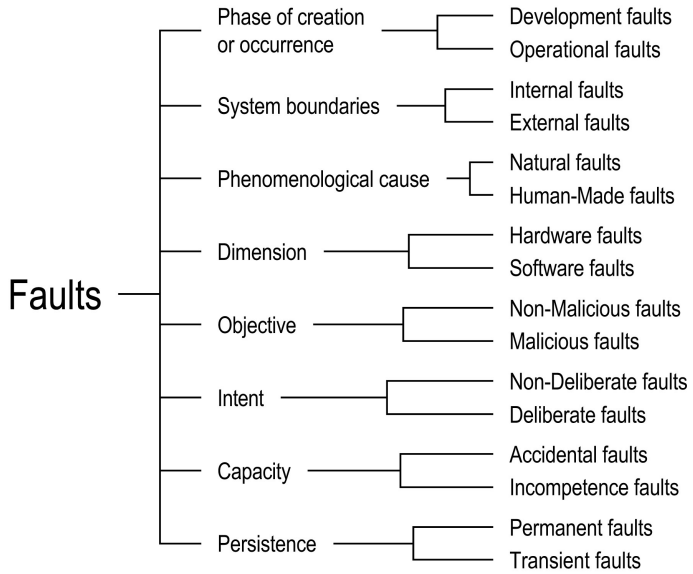

Abbildung 2.8.: Taxonomy der Fehlerklassen (16)

Zweifellos ist die Zahl möglicher Kombinationen dieser Fehlerklassen sehr groß. Man kann aber nicht wegen alle Fehler tolerant sein. Zum Beispiel wenn bei der Entwicklung des System

eine Development Fehler vorgekommen ist, wird das System einfach nicht korrekt funktionieren und kann deswegen alle seine Funktionen nicht erfüllen. Das gleiche trifft auch auf die internen Fehler zu. Sie können zum Beispiel entstehen, wenn beispielsweise ein Teil der benötigten Bibliotheken nicht gefunden werden. Genauso unberechenbar sind die Folgen eines Natural Faults. Das System kann leider nicht gegenüber allen möglichen Fault sowie ihren verschiedenen Kombinationen tolerant sein. Daher werden wir uns im Rahmen dieser Arbeit nur mit einer bestimmten Gruppe von Faults beschäftigen. Grid Fault Tolerant Execution Service, der im Rahmen dieser Arbeit entwickelt werden soll, wird ausschließlich gegen operationellen externen von Menschen verursachten Hardware und Software Faults tolerant. Einige Beispiele solcher Faults sind:

- Ressource Crash
- Netzverzögerung
- Netzausfall
- Gewünschte Ressource nicht gefunden
- Ressource weicht von Beschreibung ab

Es gibt viele Strategien für Ausprägung der Konzepte von Fault Tolerance. Man kann beispielsweise zwei oder mehr Instanzen eines gleichen Prozesses auf unterschiedlichen Rechner parallel laufen lassen. Beim Ausfall eines der Rechner wird die Instanzen auf dem anderen Rechner weiter fortgesetzt, bis einer der Prozesse fertig ist. Dann werden die anderen Prozesse abgebrochen. Diese Strategie ist offensichtlich verschwenderisch im Bezug auf die Ressourcen, die für die Execution eines einzigen Jobs benutzt werden. Eine andere Strategie ist es, bei Erkennung der Ressource Fault, den Prozess einfach noch mal zu starten. Der Nachteil dieser Strategie ist die gebrauchte Zeit, die für Ausführung eines einzigen Prozesses gebraucht wird. Da diese Zeit aus offensichtlichen Gründen schlecht berechenbar ist, kann man die gewöhnliche Prozess-Scheduling-Strategie nur sehr ineffizient verwenden. Besonders in dem Fall, wenn die Wahrscheinlichkeit des Resssourcenausfalls sehr hoch ist. Eine andere beliebte Strategie ist die Taktik von Prozess-Checkpointing mit anschließender Prozessmigration. Diese Strategie ist bedeutsam für diese Arbeit und wird deswegen genauer beschrieben.

2.4. Prozessmigration

Prozessmigration ist ein sehr wichtiges Thema für alle Grid Systeme. Normalerweise beinhaltet ein Grid viele verschieden geographisch verteilte Ressourcen, die leider nicht ausfallsicher sind. Um dieses Problem zu lösen benötigt man ein Verfahren für Übertragung des Computing Prozesses von einer Grid Ressource zur andere. Dieses Verfahren nennt man Prozess Checkpointing mit anschließender Migration. Man unterscheidet im Kontext von Grids zwei Arten von Migration: Datenmigration und Prozessmigration. Die **Datenmigration** ist der Import/Export von Dateien von einer Grid Ressource zu einer anderen. Bei der **Prozess-Migration** geht es um die Migration des Anwendungsprozesses von einer Grid Ressource zu einer anderen. In dieser Arbeit geht es ausschließlich um die Prozessmigration. Auf die Datenmigration wird im Weiteren nicht eingehen. Diese Methode ist ein bedeutungsvolles Mittel

für die Bereitstellung der verlässlichen ausfalltoleranten Web Services. Migration ist Akt der Übertragung des Prozesses zwischen zwei Rechenressourcen während der Ausführung. Das abstrakte Schema der Migration ist in der Abbildung 2.9 dargestellt.

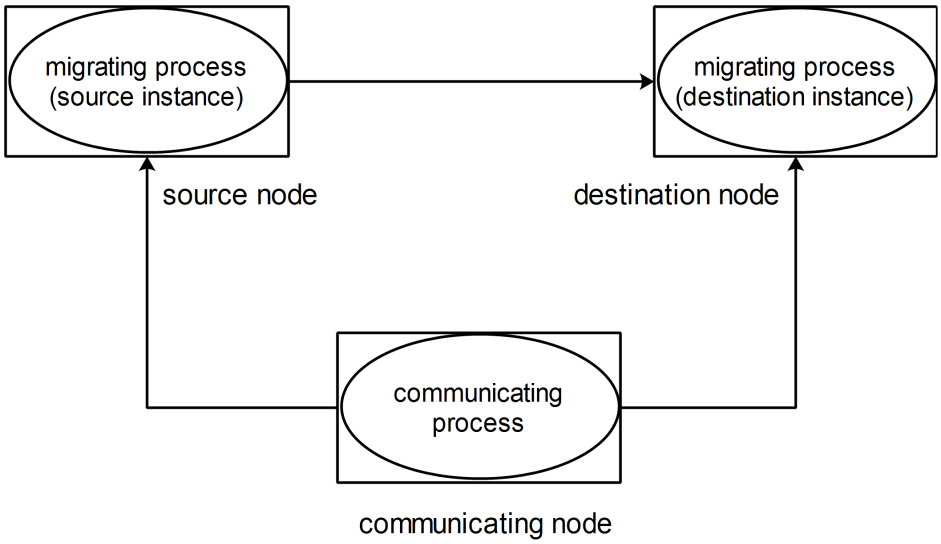

Abbildung 2.9.: High-Level Ansicht der Prozessmigration (18)

Prozessmigration besteht aus folgenden Schritte: Extraktion der *Prozesszustände* aus dem Quellsystem, Übertragung dieser Zustände zum Zielsystem, Erzeugung einer neuen Instanz dieses Prozesses mit Übernahme der Prozesszustände aus dem alten Prozess, Aktualisierung der Kommunikationskanäle des Prozesses und schließlich die Fortsetzung des Prozesses auf dem neuen Rechner. Die übertragenen Zustände können beispielsweise aus folgenden Informationen bestehen: Prozess-Adressraum, Execution Point (Register Content), Kommunikationszustände (geöffnete Dateien und Kommunikationskanäle), andere betriebssystemabhängige Zustände oder einfache Werte von Programmvariablen (falls das Programm seinen vorherigen Zustand aus solchen Werten wiederherstellen kann). Das Speichern solcher Prozesszustände in einer Datei nennt man *Prozess Checkpointing*. Und die Wiederherstellung des Prozesses aus dem Checkpoint nennt man *Prozess Recovery*.

Man unterscheidet vier Methoden für Implementierung von Checkpointing:

- der Programmier implementiert die Checkpointing-Mechanismen manuell
- die Checkpointing-Mechanismen werden während der Compilierung von Compiler eingeführt
- das Betriebsystem besitzt eigene Checkpoiting-Mechanismen
- die Checkpointing-Mechanismen werden von Run-Time Library unterstützt

Jeder dieser Mechanismen hat eigene Vor- und Nachteile. Der Auswahl der konkreten Checkpointing-Methode ist von der Aufgabe abhängig.

2.5. Zusammenfassung

In diesem Kapitel wurde die Begrifflichkeit für besseres Verständnis dieser Arbeit eingeführt und einige Einschränkungen herausgearbeitet, die für dieser Arbeit gelten. Diese Einschränkungen werden hier noch mal genannt:

- man beschäftigt sich im Laufe dieser Arbeit mit Fault Tolerance ausschließlich gegenüber operationellen externen von Menschen verursachten Hardware und Software Faults
- diese Fehler werden ausschließlich im Rahmen einer Grid Umgebung untersucht
- man beschäftigt sich im Rahmen dieser Arbeit mit dynamische Migration nur der Klasse von Grid-Anwendungen, wo nicht Daten migriert werden, sondern Prozesse

Alle anderen verwandten Themen werden in dieser Arbeit nicht betrachtet.

3. Anforderungsanalyse

In diesem Kapitel werden die Anforderungen für den Grid Fault Tolerant Exekution Service herausgearbeitet. Zuerst werden die Szenarien anhand der Aufgabenstellung aus dem Kapitel 1.2 identifiziert. Danach wird man die aus dieser Szenarien folgenden Use Cases beschreiben. Neben den Use Cases für Job Execution und Job Migration, werden auch einige andere Use Cases beschrieben, die für das Management von GFTES relevant sind. Abschließend werden die aus den Use Cases folgenden Anforderungen herauskristallisiert, die von GFTES erfüllt werden sollen. Direkt nach der Beschreibung funktionaler Anforderungen, werden auch nicht-funktionale Anforderungen benannt.

3.1. Szenarien

Aus der Aufgabenstellung lassen sich im wesentlichen zwei Szenarien identifizieren. Das erste Szenario ist ein normaler Ablauf der Grid Job Execution. Das zweite Szenario beschäftigt sich mit Crash-Fehler Entdeckung und mit anschließenden Re-Scheduling und Migration des Jobs, wobei dieses zweite Szenario auf dem ersten aufbaut. Für beide Szenarien gelten folgende Anfangsbedingungen:

- Der Benutzer ist im Besitz eines für die VO gültigen Proxy-Zertifikates. Die Erstellung des Proxy-Zertifikates wird im Rahmen dieser Arbeit nicht behandelt.
- Benutzer verfügt über ein Client-Programm und kann mit diesem Programm verschiedene Funktionen des Grid Execution Services aufzurufen.
- Jede vom Benutzer vorgelegte Job Beschreibung ist JSDL-Valid.
- Der Job Prozess speichert periodisch seine Zustände in der Checkpoint-Datei ab. Der Name dieser Checkpoint-Dateien ist eindeutig in der Job Beschreibung definiert.Der Job Prozess kann seinen vorherigen Zustand aus der Checkpoint-Datei automatisch wiederherstellen.

Im Folgenden werden zwei visionäre Szenarios für die Ausführung des Grid Jobs im Normalfall und bei der Erkennung des Ressource-Crash vorgestellt. Diese Szenarios sind zwar abstrakt, aber nicht künstlich. Ein Beispiel von erstem Szenario ist die Ausführung von dem Grid Job, wo eine große Video-Datei in das andere Format konvertiert werden soll. Dieser Job braucht das spezielle Konverter-Programm, das nur auf einem einzigen Rechner des Resource Providers installiert ist. Aus dem ersten Szenario wird das zweite Szenario folgen, falls dieser Rechner während der Ausführung von Grid Job ausfällt. Im zweiten Szenario muss der Grid Job zu einer anderen Grid Ressource migriert werden.

3.1.1. Normalverlauf der Job Ausführung

Das folgende visionäre Szenario beschreibt beispielhaft den „normalen“ Ablauf der Ausführung des Grid Jobs. Der Benutzer ist ein Mitglied von fiktiver VO „*AbstractGrid*“.Der Benutzer möchte seinen Grid Job auf einem Hochleistungsrechner ausführen lassen.

Szenario	Normalverlauf der Grid Job Ausführung
Akteure	Benutzer, Grid Execution Service, Globus Toolkit
Ereignisfluss	1. Benutzer ist ein Mitglied von „AbstractGrid“ VO und möchte seinen Job auf einem der verfügbaren Hochleistungsrechner ausführen lassen. 2. Benutzer legt korrekte Job Beschreibung vor. 3. Benutzer gestattet in der Job Beschreibung die Nutzung von ähnlichen Ressourcen falls die gewünschten Ressourcen ihm gar nicht oder nur teilweise zur Verfügung gestellt werden können. 4. Benutzer verfügt über einen gültigen Proxy-Zertifikat. 5. Grid Execution Service versucht in „AbstractGrid“ VO passende Ressourcen zu finde. 6. Aus allen gefundenen Ressourcen wird der zur Job Beschreibung best passende ausgesucht. Die ausgewählte Ressource ist nur um 20% weniger Leistungsstark als die gewünschte. Diese Ressource wird von einem Resource Provider anhand von Globus Toolkit Middleware zur Verfügung gestellt. 7. Grid Execution Service bereitet eine Job Beschreibung vor, die nach Globus Toolkit Standard angepasst ist. Die Job Beschreibung wird bei der Resource Provider vorgelegt und der Job wird ausgeführt. 8. Nach der erfolgreichen Ausführung des Jobs, werden dem an Benutzer eine entsprechende Mitteilung sowie die Ergebnisse übergeben.

Tabelle 3.1.: Normalverlauf der Grid Job Ausführung

3.1.2. Verlauf der Job Ausführung im Fall eines Ressource-Crashes

Dieses visionäre Szenario baut sich direkt auf dem obigen Szenario auf. Währende der Ausführung des Jobs bei Resource Provider fällt die benutzte Ressource aus.

Szenario	Job Ausführung im Fall eines Ressource-Crashes
Akteure	User, Grid Execution Service, Globus Toolkit, UNICORE
Ereignisfluss	1. Die Ausführung des Jobs hat begonnen. 2. Während der Ausführung des Jobs, werden periodisch Checkpoint-Dateien erzeugt. Grid Execution Service beobachtet Erschaffung der neuen Checkpoint-Dateien und speichert sie ab. 3. Gleichzeitig bewacht der Grid Execution Service den Ressourcenstatus, um zu erkennen, wenn die Ressource ausfällt. 4. Die Ressource fällt aus. 5. Grid Execution Service merkt den Ressource-Ausfall und entscheidet sich für den Job Re-Scheduling mit anschließender Migration . 6. Re-Scheduling Prozedur wird durchgeführt. Der Job wird zurück in die Queue gesetzt. Man unternimmt die Suche nach neuen Ressourcen noch einmal. 7. Passende Ressourcen werden gefunden. Der Resource Provider, der die Ressourcen in VO zur Verfügung stellt, nutzt UNICORE Grid Middleware. 8. Neue Job Beschreibung wird erstellt, die nun nach UNICORE Standard gestaltet ist. Dabei wurde die zuletzt gespeicherte Checkpoint-Datei in die Job Beschreibung als Stage-In Datei eingeführt. Die Job Beschreibung wird bei Resource Provider vorgelegt. 9. Der Job Prozess stellt seinen Verlauf aus der letzten Checkpoint-Datei wieder her. 10. Die Überwachung sowie periodische Abspeichern von Checkpoint-Dateien wird fortgesetzt. 11. Der Benutzer wird nach Beenden seines Jobs benachrichtigt.

Tabelle 3.2.: Verlauf der Job Ausführung im Fall eines Ressource-Crashes

3.2. Use Cases

In diesem Abschnitt werden Use Cases aus den zuvor erstellten Szenarien identifiziert. Die Abbildung 3.1 stellt die wichtigsten Anforderungen in Form eines Use-Case Diagramms dar. Neben zwei zentralen Use Cases, werden noch einige weitere Use Cases beschrieben. Die Management Use-Cases werden in Abschnitt 3.2.9 eingeführt.

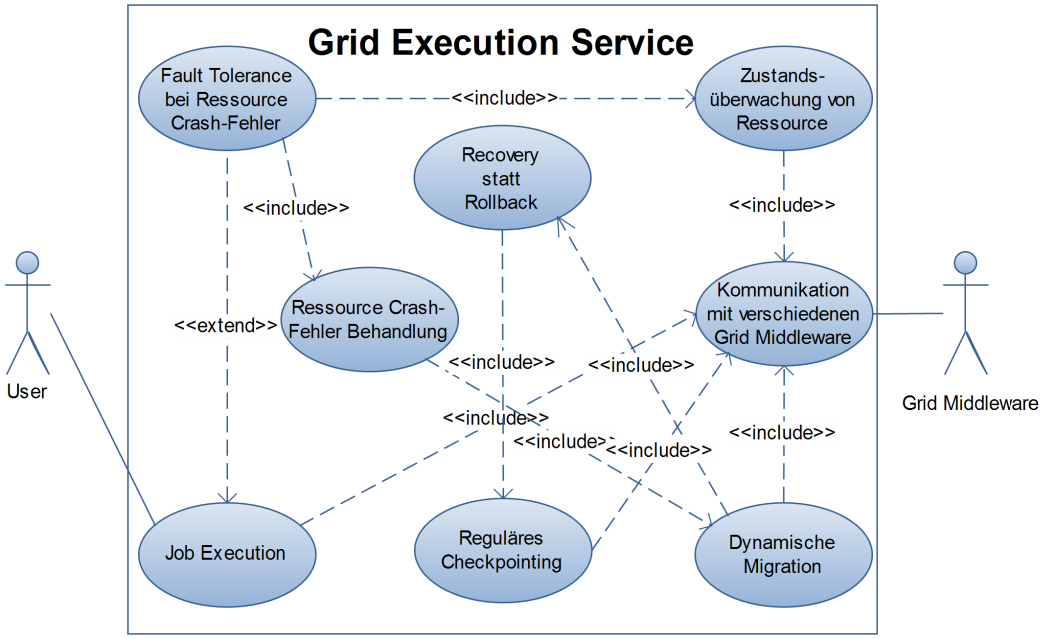

Abbildung 3.1.: Dieses Use Case Diagramm stellt die Anwendungsfälle *Normalverlauf der Job Ausführung* und *Verlauf der Job Ausführung im Fall eines Ressource-Crashes* dar. Dabei werden die Akteure „User“ und „Grid Middleware“ über Grid Execution Service miteinander verbunden.

3.2.1. Job Execution

Primary Use Case	Job Execution
Beschreibung	Dieser Use Case beschreibt einen normalen Verlauf der Job Ausführung
Akteure	User, Grid Middleware

Ereignisfluss	1. User sendet ein Job-Submission Request mit der Job Beschreibung und eigenem Proxy-Zertifikat. 2. Grid Execution Service findet passende Ressource, erzeugt angemessene Job Beschreibung für den Resource Provider und legt Job-Beschreibungsdokument vor. 3. Grid Middleware leitet diesen Job an die entsprechende Rechen-Ressource weiter. 4. Grid Execution Service überwacht die Ressource um zu erkennen, wenn die Ressource-Crashed. Periodisch werden die Checkpotints-Dateien von Grid Ressource abgeholt und abgespeichert. 5. Wenn Job erfolgreich durchgelaufen ist, teilt dies der Grid Execution Service dem Benutzer mit.
Anfangsbedingungen	• Der Benutzer verfügt über ein Client-Programm und kann mit diesem Programm verschiedene Funktionen des Grid Execution Services aufrufen • User besitzt ein gültiges Proxy-Zertifikat für die VO • Job Prozess speichert periodische seine Zustände in der Checkpoint-Datei ab. Der Name dieser Checkpoint-Dateien ist eindeutig in der Job Beschreibung definiert. Job Prozess kann seinen vorherigen Zustand aus der Checkpoint-Datei automatisch wiederherstellen
Abschlussbedingungen	• User erhält eine Nachricht über erfolgreichen Durchlauf seines Jobs ODER • User erhält eine Nachricht, mit Erklärung, warum die Ausführung seines Jobs fehlgeschlagen ist.

Tabelle 3.3.: Use Case: Job Execution

Bei diesem Use Case gibt es einige Ferhlerfälle, die von einander unterschieden werden müssen. Jeder dieser Fehlerfälle zieht eine eigene Meldung nach sich, damit der User über den Grund des Scheiterns der Job Ausführung informiert wird. Mögliche Grunde sind:

- User darf nicht auf die Service von Grid Execution Service zugreifen.
- Proxy-Zertifikat ist nicht mehr gültig oder abgelaufen.
- Job beinhaltet möglicherweise einige Fehler.
- Die Grid-Provider sind nicht verfügbar.

Dabei sind einige Secondary Use Cases durchaus wahrscheinlich:

Secondary Use Case	Ressourcen wurden nicht gefunden
Beschreibung	Dieser Use Case beschreibt den Fall, wenn ein oder mehrere Ressourcen aus der Job Beschreibung nicht gefunden wurden
Akteure	User, Grid Middleware
Ereignisfluss	1. Eine oder mehreren Ressource die zur Job Beschreibung exakt gepasst hätten, können nicht gefunden werden. 2. Grid Execution Service überprüft in der Job Beschreibung, ob die Verwendung von ähnlichen statt identischen Ressourcen vom Benutzer zugelassen wurde. 3. Wenn die Nutzung von ähnlichen Ressourcen auch gestattet ist, versucht Grid Execution Service die best passenden Ressourcen aus den verfügbaren zu nehmen.
Anfangsbedingungen	• Die gleiche Anfangsbedingungen wie im Use Case „Job Execution“ • Eine oder mehrere Ressourcen, die exakt zu Job Beschreibung passen, werden nicht gefunden.
Abschlussbedingungen	• Job Durchlauf wird gestartet ODER • Benutzer bekommt eine Meldung darüber, dass eine oder mehreren Ressourcen nicht gefunden wurden

Tabelle 3.4.: Use Case: Ressourcen wurden nicht gefunden

Ein anderer Use Case beschreibt die Situation, wenn die angeforderten Ressourcen sich in einer anderen als der Benutzer VO befinden und können für den Job deswegen nicht benutzt

werden.

Secondary Use Case	Ressource befindet sich in einer anderen VO als der Benutzer
Beschreibung	Dieser Use Case beschreibt den Fall, wenn ein oder mehreren angeforderten Ressourcen sich in einer anderen VO befinden als User.
Akteure	User, Grid Middleware
Ereignisfluss	1. Grid Execution Service findet Ressource die exakt zu Job Beschreibung passt. Diese Ressource befindet sich aber in einer anderer VO als der Benutzer und kann von ihm nicht benutzt werden. 2. Grid Execution Service überprüft, ob die Nutzung von ähnlichen statt identischen Ressource für Job gestattet ist. 3. Falls ähnliche Ressource zugelassen sind, wird der weiteren Verlauf wie der User Case 3.4 fortgesetzt.
Anfangsbedingungen	• Die Ressource die gebraucht wird, befindet sich in einer anderen VO als der Benutzer.
Abschlussbedingungen	• Die Suche wird mit gelockerten Ressource-Anforderungen fortgesetzt ODER • der Benutzer bekommt die Fehlermeldung darüber, dass Ressourcen nicht gefunden werden können.

Tabelle 3.5.: Use Case: Ressource befindet sich in eine anderer VO als der Benutzer

3.2.2. Fault Tolerance bei Ressource Crash-Fehler

Primary Use Case	Fault Tolerance bei Ressource Crash-Fehler
Beschreibung	Ressource fällt während der Job Ausführung aus
Akteure	Grid Middleware

Ereignisfluss	1. Der Job wird wie im Use Case „Job Execution“ gestartet und während der Ausführung fallen eine oder mehreren Ressourcen aus 2. Grid Execution Service überwacht den Zustand der Ressourcen und merkt eine Crash-Fehler (weiter darüber im Use Case „Zustandsüberwachung von Ressourcen“) 3. Es wird die Routine für Crash-Fehler Behandlung gestartet (weiter dazu im Use Case „Ressource Crash-Fehler Behandlung“)
Anfangsbedingungen	• Job Execution wird gestartet • Grid Middleware stellt einen Service für Ressource-Überwachung bereit • Eine oder mehrere benutzte Ressource fallen aus
Abschlussbedingungen	• Re-Scheduling Routine mit anschließender Migration wird gestartet.

Tabelle 3.6.: Use Case: Fault Tolerance bei Ressource Crash-Fehler

3.2.3. Ressource Crash-Fehler Behandlung

Primary Use Case	Ressource Fault Behandlung
Beschreibung	Crash-Fault wurde erkannt und muss behandelt werden
Akteure	Grid Middleware
Ereignisfluss	1. Während der Job Ausführung wird automatisch Überwachung des Ressource Status gestartet. 2. Nach dem Ressource-Crash wird automatisch die Re-Scheduling Prozedur gestartet (weiter im Use Case „Job Migration“)

Anfangsbedingungen	• Die Ausführung des Jobs wurde gestartet. • Grid Middleware stellt ein Service für Monitoring der Ressourcen zur Verfügung • Eine oder mehreren Ressourcen fallen gleichzeitig aus
Abschlussbedingungen	• Job Re-Scheduling Routine wird gestartet

Tabelle 3.7.: Use Case: Ressource Fault Behandlung

3.2.4. Dynamische Migration

Primary Use Case	Job Migration
Beschreibung	Grid Execution Service entscheidet sich für Migration eines Jobs
Akteure	User, Grid Middleware
Ereignisfluss	1. Grid Execution Service startet die Job Re-Scheduling Routine. 2. Grid Execution Service setzt Job zurück in die Queue und sucht nach passenden Ressourcen erneut. 3. Sobald/falls die Ressourcen gefunden wurden, erstellt Grid Execution Service eine neue Job Beschreibung (abhängig von der Grid Middleware, die von Resource Provider verwendet wird) und legt sie bei Resource Provider vor. Zusammen mit allen Stage-In Dateien wird auch die letzte Checkpoint-Datei in die Job Beschreibung eingetragen. 4. Auf der neuen Ressource versucht Job seinen vorherigen zustand aus der Checkpoint-Datei wiederherzustellen (weitere Informationen im Use Case „Job Recovery“)
Anfangsbedingungen	• Job beinhaltet die Checkpoint-Routine und speichert periodisch seine Zustände in Checkpoint-Datei ab.

Abschlussbedingungen	• Neue Ressourcen werden gefunden. Job wird aus der Checkpoint-Datei wiederhergestellt ODER • Neue Ressourcen werden gefunden und Job wird neu gestartet ODER • User bekommt eine Fehlermeldung.

Tabelle 3.8.: Use Case: Dynamische Migration

Bei diesem Use Case gibt es einige Fehlerfälle die sich von einander unterscheiden. Zu jedem dieser Fälle gehört eine eigene Meldung, die an den Benutzer verschickt wird. Diese Fehlermeldungen sind:

- Proxy-Zertifikat ist ungültig oder abgelaufen
- Grid-Services sind nicht erreichbar

In der Use Case „Dynamische Migration“ sind bei der Suche nach neuen Ressourcen folgende Secondary Use Cases möglich „Ressource wurde nicht gefunden“, „Eine oder mehreren Ressourcen befinden sich in einer anderen VO als der Benutzer“. Die Beschreibung dieser Use Cases ist gleich der Beschreibung 3.4 und 3.5 die bereits zuvor dargestellt wurden
.

3.2.5. Zustandsüberwachung von Ressourcen

Primary Use Case	Ressource Monitoring
Beschreibung	Grid Execution Service überwacht die Job ausführenden Ressourcen um zu erkennen, wenn eine Ressource ausfällt
Akteure	User, Grid Middleware
Ereignisfluss	1. Job Ausführung ist gestartet 2. Grid Execution Service befragt periodisch Grid Middleware um zu merken, wenn die benutzten Rechen-Ressourcen ausfallen
Anfangsbedingungen	• Grid Middleware stellt Ressource Monitoring Service zur Verfügung.

Abschlussbedingungen	• Falls die entsprechenden Ressourcen keine Antworten auf die Statusabfragen mehr liefern, wird die Re-Scheduling Prozedur gestartet.

Tabelle 3.9.: Use Case: Ressource Monitoring

3.2.6. Job Recovery

Primary Use Case	Job Recovery
Beschreibung	Job versucht seinen Prozess-Verlauf aus der Checkpoint-Datei wiederherzustellen
Akteure	Grid Middleware
Ereignisfluss	1. Job wird zusammen mit Checkpoint-Dateien, falls diese vorhanden sind, an die Grid Ressource übergeben. 2. Der Job Prozess versucht seine Zustände aus der Checkpoint-Dateien wiederherzustellen.
Anfangsbedingungen	• Job Prozess beinhaltet eine Routine um während des Durchlaufs seine Zustände in Checkpoint-Dateien abzuspeichern sowie eine Recovery-Routine, die bei der Wiederherstellung aus der Checkpoint-Dabei gestartet wird. • Checkpoint-Datei wird zusammen mit allen Stage-In Dateien an die entsprechende Grid Ressource kopiert.
Abschlussbedingungen	• Job wird fortgesetzt ODER • Job wird neu gestartet, falls keine Recovery möglich ist.

Tabelle 3.10.: Use Case: Job Recovery

Aus dem Primary Use Case „Job Recovery“ kann auch der folgende Secondary Use Case folgen „Checkpoint-Datei ist nicht vorhanden oder beschädigt“.

Secondary Use Case	Checkpoint-Datei ist nicht vorhanden oder beschädigt
Beschreibung	Dieser Use-Case beschreibt einen Fall wenn die gespeicherten Checkpoint-Datei für den Job nicht vorhanden ist oder beschädigt ist
Akteure	Grid Middleware
Ereignisfluss	1. Der Job Prozess findet heraus, dass die Checkpoint-Datei fehlt oder Job aus der Checkpoint-Datei nicht wiederhergestellt werden kann (z.B. die Datei ist beschädigt) 2. Der Job Prozess wird neu gestartet
Anfangsbedingungen	• Checkpoint-Datei ist nicht vorhanden oder ist beschädigt
Abschlussbedingungen	• Job wird neu gestartet.

Tabelle 3.11.: Use Case: Checkpoint-Datei ist nicht vorhanden oder ist beschädigt

3.2.7. Requläres Checkpointing

Primary Use Case	Reguläres Checkpointing
Beschreibung	Grid Execution Service speichert periodisch die Checkpoint-Datei ab. Diese Datei wird von Job Prozess periodisch erzeugt.
Akteure	Grid Middleware
Ereignisfluss	1. Der Job Prozess speichert seine Zustände in Checkpoint-Dateien periodisch ab. 2. Der Grid Execution Service holt diese Checkpoint-Dateien und speichert sie in der Datenbank.

Anfangsbedingungen	• Die Job-Anwendung verfügt über eine Checkpointing-Routine. • Namen der Checkpoint-Datei ist in der Job Beschreibung vorgegeben.
Abschlussbedingungen	• Checkpoint-Datei wird in der DB abgespeichert ODER • Es wird keine Checkpoint-Datei gefunden

Tabelle 3.12.: Use Case: Reguläres Checkpointing

Zu der Primary Use Case Requläre Checkpointing sind auch zwei anderen Secondary Use Cases möglich: keine Checkpoint-Datei wird erstellt und Checkpoint-Datei kann nicht kopiert werden. In beiden Fällen wird nichts unternommen und im Fall einer Migration muss der Job neu gestartet werden.

3.2.8. Kommunikation mit verschiedenen Grid Middleware

Primary Use Case	Kommunikation mit mehreren Grid Middleware Technologien
Beschreibung	Grid Execution Service kann die Ressource von mehreren Resource Provider benutzen, unabhängig davon, welche der Grid Middleware Arten bei diesem Resource Provider installiert ist
Akteure	Grid Middleware

Ereignisfluss	1. Grid Execution Service muss eine gewisse Funktion an einem der Grid Middleware Services aufrufen oder Informationen von diesem Service bekommen. 2. Die Entsprechen SOAP-Nachricht wird von Grid Execution Service zusammengestellt und an Endpoint des Grid Middleware Services verschickt. 3. Der Befehl wird vom Service ausgeführt und das Ergebnis wird an Grid Execution Service als Response-Message zurückgeliefert. 4. Response-Message wird empfangen und von Grid Execution Service verstanden.
Anfangsbedingungen	• WSDL-Dokument von der Grid Web Service ist bekannt • Grid Execution Service kennt Endpoint-Adresse des gebrauchten Web-Service • Grid Execution Service kann SOAP-Nachrichten zusammenstellen, verschicken, empfangen und verstehen
Abschlussbedingungen	• Response-Message wird empfangen ODER • Es kommt keine Response-Message an ODER • Grid Middleware Service liefert eine Fehlermeldung zurück

Tabelle 3.13.: Use Case: Kommunikation mit verschiedenen Grid Middleware

Bei diesem Use Case gibt es eine Reihe von möglichen Fehlern. Jeder von diesen Fehlern verursacht eine Fehlermeldung, die an den Benutzer geschickt wird. Diese Fehlermeldungen sind:

- WSDL-Dokument von gewünschtem Grid Service ist unbekannt
- Endpoint des Web Services ist unbekannt
- Grid Service antworten nicht auf die Request-Message oder sind nicht verfügbar

3.2.9. Management Use Cases

In folgendem Abschnitt werden die Management Use Cases beschrieben, die für Verwaltung des Grid Execution Services bedeutsam sind.

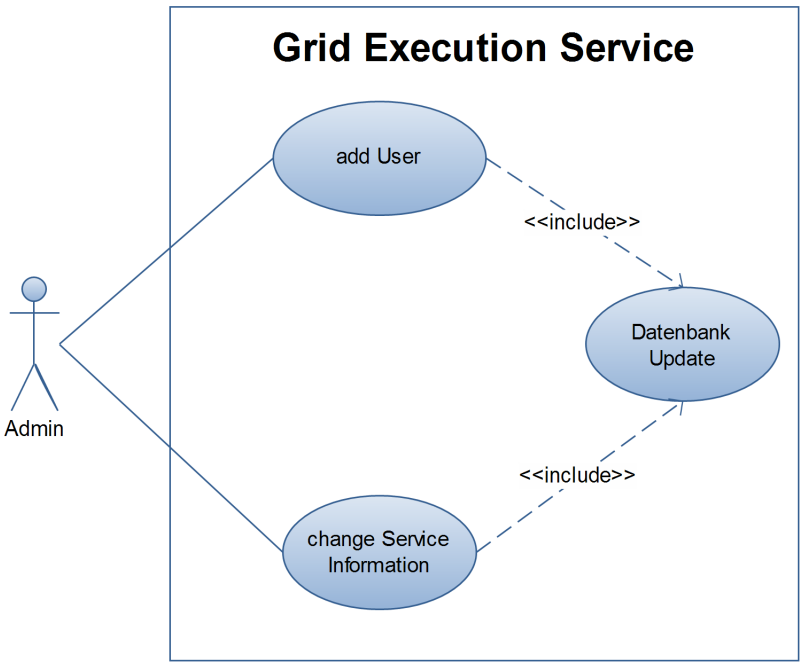

Abbildung 3.2.: Ausgewählte Use Cases für die Verwaltung von Grid Execution Service

Die Datenstruktur des Grid Execution Services soll möglichst dynamisch und flexible sein. Die User Cases für Einfügen, Löschen und Verwalten der Informationen über Grid Services und System Benutzer sollen Grid Execution Service dabei helfen, die gewünschte Verwaltungselastizität zu erreichen.

Die Management Use Cases kann man in zwei Gruppen aufteilen:

- Einfügen, Löschen und Verwalten von Informationen über Grid Services. Hier werden z.B. die Informationen über neue Endpoints eingefügt sowie geändert
- Einfügen, Löschen und Verwalten von Benutzerinformationen.

In der Abbildung 3.2 sind beide oben genannten Gruppen anhand der ausgewählten Management Use Cases dargelegt. Die restlichen Use Cases aus der obigen Auflistung sind den zwei in der Abbildung dargestellten sehr ähnlich und sind deshalb im UML Use Case Diagramm ausgespart worden. Zusätzlich wird immer erwähnt, welche anderen Use Cases analog dazu sind. Jeder Use Case ruft einen „Datenbank Update“ Use Case auf, um die Änderungen bzw. neue Informationen in die Datenbank aufzunehmen.

Erstellen eines neuen Benutzers

Use Case	add User
Akteure	Admin
Ereignisfluss	1. Admin möchte einen neuen Benutzer erstellen und schickt den entsprechenden Request-Befehl an Endpoint von Grid Execution Service. 2. Grid Execution Service überprüft die Berechtigung des Admin-User, erstellt neuen Benutzer und speichert Daten in die Datenbank.
Anfangsbedingungen	• Der Benutzer ist als Administrator angemeldet.
Abschlussbedingungen	• Der Benutzer erhält Response-Message mit Bestätigung ODER • Der Benutzer bekommt eine Fehlermeldung ODER • Es wird keine Response-Message empfangen

Tabelle 3.14.: Use Case: *add User*

Vergleichsweise laufen auch zwei anderen Use Cases - Löschen und Verwalten von Benutzerinformationen.

Änderung der Informationen über Grid Services

Use Case	change Service Information
Akteure	Admin

Ereignisfluss	1. Administrator möchte die Informationen über Grid Service ändern und sendet ein entprechendes Request-Message. 2. Grid Execution Service überprüft die Berechtigung des Benutzers, Ändert die Service Informationen und antwortet mit Response-Message.
Anfangsbedingungen	• Der Benutzer ist als Administrator angemeldet und ist berechtigt, die gewünschte Änderungen in der Datenbank durchzuführen
Abschlussbedingungen	• Der Benutzer erhält Response-Message mit Bestätigung ODER • Der Benutzer erhält Response-Message mit Fehlermeldung ODER • Es kommt keine Response-Message an

Tabelle 3.15.: Use Case: *change Resource Role*

Es gibt gleichartige Use Cases für das Einfügen und Löschen der Informationen über Grid Middleware Dienste.

3.2.10. Nicht-funktionale Anforderungen

Dieser Abschnitt beschreibt die nicht-funktionalen Anforderungen nach dem ISO/IEC 9126 Standard (11). Diese nicht-funktionalen Anforderungen, sollten bei einem produktiven Einsatz von Grid Execution Service ebenfalls erfüllt werden:

- **Portabilität**: Grid Execution Service soll so gestaltet werden, dass sie auf heterogenen Hardware und auf verschiedenen Betriebssystemen funktionieren kann
- **Benutzerfreundlichkeit**: Einfache Bedienung sowie eine klare und gut dokumentierte Schnittstelle hilft einem potentiellen User bei der Einarbeitung und Weiternutzung dieses Services
- **Skalierbarkeit**: Das System soll auch mit vielen Benutzer und Jobs ohne große Verzögerungen und Ausfälle umgehen können

- **Zuverlässigkeit**: Datenstruktur des Grid Execution Service soll Fault Tolerant gestaltet werden
- **Performance**: Die Antwortzeit von Grid Execution Service soll so kurz wie möglich sein. Der Durchsatz an Anfragen soll berechenbar sein, damit auch viele Clients diesen Service gleichzeitig verwenden können
- **Sicherheit**: Grid Execution Service darf nur von autorisierten Benutzern verwaltet und benutzt werden

3.3. Anforderungen

Aus den oben beschriebenen Use Cases kann man nun die Anforderungsliste für den Grid Execution Service zusammenstellen.

- **Funktionale Anforderungen**:
  1. Fault Tolerance zu Ressource Crash-Fehler
  2. Recovery durch dynamische Migration statt Rollback
  3. fein-granulare Zustandsüberwachung durch reguläres Checkpointing
  4. Nutzung von ähnlichen und identischen Ressourcen
  5. unterschiedliche Resource Provider
- **Nicht-funktionale Anforderungen**:
  7. Unabhängigkeit von der Art der benutzten Grid Middleware
  8. Portabilität
  9. Benutzerfreundlichkeit
  10. Skaliebarkeit
  11. Zuverlässigkeit
  12. Performance
  13. Sicherheit

Im nächstem Kapitel werden verschiedene Grid Lösungen untersucht, die sich mit dem Thema Fault Tolerance bereits beschäftigt hatten. Besonders wird darauf aufmerksam gemacht, ob diese Grid Software Lösungen die Anforderungen aus der obigen Liste erfüllen.

4. Themenbezogene Arbeiten

In diesem Kapitel werden einige Grid Middleware Technologien präsentiert. Dabei wird besondere Aufmerksamkeit der Frage geschenkt, ob und wie die Fault Tolerance Konzepte in Grid Middleware realisiert wurden. Zugleich wird überprüft, ob die Funktionsweise dieser Software den Anforderungen aus dem Kapitel 3.3 genügt. Für die Grundlagen der Fault Tolerance, sei auf die Kapitel 2.3 hingewiesen.

4.1. Fault Tolerance in Grid Middleware

Im diesem Teil werden gängige Grid Middleware Technologien beschrieben und anschließend den Anforderungen aus dem Kapitel 3.3 gegenübergestellt.

4.1.1. Globus Toolkit

Globus Toolkit (GT) (6) ist ein Framework einer Grid Middleware, d.h., es besteht aus einer Reihe von Werkzeugen, auf deren Basis Grids und Grid-Applikationen erstellt werden können. Globus Toolkit implementiert eine Reihe von Standards, vor allem aber die Open Grid Service Architecture (OGSA)(15).

GT ist modular aufgebaut, so dass einzelne Funktionalitäten möglichst unabhängig voneinander benutzt werden können. Seit der Version 4 des Globus Toolkit sind viele (aber noch nicht alle) Funktionalitäten als Web Services konform zum Web Service Resource Framework (WSRF) als Standard implementiert(15).

Im Kontext von Fault Tolerance unterscheidet man im Globus Toolkit zwei Ebene: die Data Ebene und die Execution Ebene (31).

Im Bezug auf Datenkonsistenz sollten *Reliable File Transfer* Service (RFT) und *Data Replication Service* (DRS) erwähnt werden. Reliable File Transfer Service garantiert die erfolgreiche Übertragung von Dateien, selbst wenn ein Fehler während der Übertragung auftritt. Dabei werden in der Datenbank der Status der Übertragung sowie alle relevanten Information gespeichert und im Fall eines Fehlers wird die Übertragung wiederholt.
Data Replication Service bietet eine automatische Replikation von Dateien auf verschiedenen Ressourcen. Für den Fall, das einige Dateien fehlen, wird ihr Replikat anhand von Replica Location Service identifiziert und automatisch auf die Ressource kopiert.

Auf Execution Ebene bietet Globus Toolkit den *Grid Resource Allocation and Management* Service (GRAM) an. GRAM reicht dem Benutzer eine entsprechende Schnittstelle für Start und Überwachung seiner Jobs.

Die Fault Tolerance Konzepte in Globus Toolkit müssen von Ressource Scheduler unterstützt werden. Ein der bekanntesten Scheduler ist *Condor* (26). Condor bietet Mechanismen für Checkpointing und Job Migration an.

Im Bezug auf die Anforderungen an Grid Execution Service aus Kapitel 3.3, sieht man einige fehlenden Use Cases im System. Der Globus Toolkit kann zwar mit Hilfe von Condor die Job Checkpoins abspeichern und Jobs migrieren, doch die Checkpoint-Datei kann leider nur auf der Ressource mit dem Condor Scheduler wiederhergestellt werden.

Außerdem kann Globus Toolkit mit anderen Grid Middleware Systemen nur sehr beschränkt zusammenarbeiten. In einem hypothetischen Fall, wenn eine für den Job benötige Ressource zwar in gleicher VO doch unter anderer Grid Middleware Technologie zur Verfügung stehen würde, wird Globus Toolkit auf die Ressource verzichten müssen.

Darüber hinaus kann man sehen, das einige Anforderungen an Grid Execution Service die im Rahmen dieser Arbeit gestellt wurden, von Globus Toolkit gar nicht erfüllt werden. Hier werden noch mal die Anforderungen aufgelistet, die von Globus Toolkit nicht erfüllt werden:

- Globus Toolkit kann nicht die Prozesse dynamisch migrieren
- Globus Toolkit unterstützt keinen Zustandsüberwachung durch reguläres Checkpointing
- Globus Toolkit unterstützt nicht die Nutzung von ähnlichen Grid Ressourcen
- Die Grid Services von anderen Grid Middleware werden von Globus Toolkit nicht unterstützt
- Die Installation von Globus Toolkit ist nicht benutzerfreundlich und kann viel Zeit kosten

4.1.2. UNICORE

UNICORE (Uniform Interface to Computing Resource) (30) ist eine ready-to-run Grid Middleware, die aus zwei Teilen besteht. Diese Teile sind UNICORE Client und UNICORE Server. UNICORE stellt verteilte Rechen- und Data-Ressourcen in einer umfassenden und sicheren Weise im Intranet oder im Internet zur Verfügung.

Die Fault Tolerance ist im UNICORE sehr beschränkt implementiert. Die Job Migration wird gar nicht unterstützt (12).
Die Vorbeugung und Recovery Methoden für den Fall wenn Grid Dienste ausfallen würden oder im Fall eines Staging-Error stehen dem Benutzer noch nicht zur Verfügung, und befinden sich noch in der Entwicklungsphase (19).

Der Verlust von Speicher, nicht abgefangene Ausnahmen, Deadlocks etc. können anhand von Target System Interface (TSI) entdeckt werden. Vermisste Bibliotheken und Job Crashes können mit Hilfe von NJS entdeckt werden.

Recovery im Fall eines Job Crashes wird erst in der zukünftigen Versionen implementiert. Auf der Workflow- und User Ebenen werden Datenfehler und Ausnahmen auch mittels Network Job Supervisor Scheduler erkannt.

Für die Fault-Vorbeugung auf Workflow- und Task-Ebenene steht in UNICORE der Site Monitor (SIMON) zu Verfügung. SIMON ist für Task Management sowie für die Abschätzung der Ressourcezuverlässigkeit zuständig. Im Fall eines Job Fehler versucht NJS der Job auf der gleichen Ressource noch einmal zu starten.

Im Bezug auf die Anforderungen an Grid Execution Service aus dem Kapitel 3.3 kann man einige fehlende Use Cases erblicken. Man kann zwar Fehler auf unterschiedliche Ebenen erkennen, doch die Migration wird gar nicht unterstützt. Die Checkpointing ist nicht vorgesehen und im Fall eines Fehlers versucht die UNICORE Middleware den Job auf der gleichen Ressource neu zu starten.

Genauso wie Globus Toolkit kann UNICORE mit keinen Grid Middleware kooperieren und ist somit nur auf die Ressourcen mit installierten UNICORE Clients angewiesen. Dei Anforderungen, die von UNICORE nicht erfüllt werden, sind hier noch mal aufgelistet:

- UNICORE kann nich die Grid-Anwendungen dynamisch migrieren
- UNICORE unterstützt genauso wie Globus kein Checkpointing
- UNICORE unterstützt nicht die Nutzung von Grid Ressourcen, die ähnlich der gewünschten aus der Job Beschreibung sind
- UNICORE unterstützt nicht die Nutzung der Web Services von anderen Grid Middleware

4.1.3. gLite

Die gLite - Lightweight Middleware for Grid Computing (21) - bietet einen Framework für Scheduling und Ausführung von Grid Jobs, Zugang und Bearbeitung von Daten und Zugriff auf Informationen über Grid Infrastruktur und Grid Applikationen.

Die Middleware läuft auf Scientific Linux Plattform [1] und ist auf C++ geschrieben. Auf einem Rechner mit einem anderen Betriebssystem kann gLite nicht installiert werden.

Die gLite Grid Services kann man in fünf Gruppen aufteilen: Access Services, Security Services, Information und Monitoring Services, Data Services und Job Management Services (13).

Der Prozess von Job Submission umfasst in gLite viele Operationen, beginnend von der Erstellung und Delegierung der Credentials, gefolgt von der Übertragung der Dateien, Einrichtung aller Software und Hardware Ressourcen bis zur Job Ausführung und Empfang der Job Ergebnisse (4).

[1] `http://www.scientificlinux.org/`

Im Bezug auf Fault Tolerance behandelt gLite leider nur einen kleinen Teil aller möglichen Fehler. Bei Entduckung eines Fehlers wird der Job immer neu gestartet(Resubmission/Retransmission).

Die Erfüllung der Konzepte von Fault Tolerance besteht aus zwei Schritten: Fault Erkennung und Fault Recovery. Die Überwachung von Job Submission findet in gLite anhand des Logging und Bookkeeping Servers (L&B) statt.

Wenn man die Funktionalität von gLite mit den Anforderungen aus dem Kapitel 3.3 vergleicht, kommt man zu dem Schluss, dass sowohl einige funktionale als auch nicht-funktionale Anforderungen in gLite nicht erfüllt sind. gLite ist abhängig von Betriebssystem und kann nur auf Scientific Linux Plattform installiert werden. Alle Fehler werden nur mit Neustart behandelt indem der Job noch einmal auf die gleiche Ressource geschickt wird. Zum Schluss muss man sagen, dass die Möglichkeit der Kommunikation und Zusammenarbeit mit anderer Grid Middleware in gLite gar nicht vorgesehen ist.

Als kleine Zusammenfassung werden hier noch mal die Anforderungen aufgelistet, die von gLite nicht erfüllt wurden:

- gLite kann genauso wie Globus Toolkit und UNICORE die Job-Anwendung nicht dynamisch migrieren
- gLite unterstützt kein Checkpointing und dehalb auch kein Recovery aus der Checkpoint-Datei
- gLite darf bei der Suche nach passenden Ressourcen von den Ressource-Parameter aus der Job Beschreibung nicht abweichen (heißt, keine Nutzung von ähnlichen Ressourcen).
- gLite benutzt nur eigene Web Services, aber nicht die von anderen Grid Middleware
- gLite arbeitet nur auf Scientific Linux (heißt keine Portabilität)
- Die Installation von gLite ist nicht benutzerfreundlich und braucht viel Zeit

4.1.4. Grid(Lab) Resource Management System (GRMS)

Grid(Lab) Resource Management System (GRMS) (29) ist ein Meta-Scheduling System für verteilte Recheninfrastrukturen. GRMS wurde für einen sicheren Umgang mit Ressource Management Aufgaben programmiert. Als Beispiel solcher Aufgaben sind Load-Balancing in einem Cluster-System, Einrichtung der Laufzeitumgebung vor und nach der Jobausführung, eine Job Submission und Job Kontrolle auf einer entlegenen Ressource, Datenmanagement und vieles mehr (14).

Grid(Lab) Resource Management System basiert sich auf Low-Level Globus Service aus dem Globus Toolkit Middleware. GRMS bindet sich an Globus Core Service anhand einer Reihe von C und Java API's. Insbesondere werden Grid Resource Allocation and Management (GRAM) Service, GridFTP und Grid Resource Information Servie (GRIS)/Grid IndexInformation Service (GIIS) aus dem Globus genutzt.

Im Zusammenhang mit Fault Tolerance in Grid(Lab) Resource Management System muss man den Rescheduling Plug-In genauer unter die Lupe nehmen, der bei GRMS für Migration und Re-Scheduling zuständig ist. Vor der Migration wird für jeden Job zuerst eine Checkpoint-Datei eingelagert. Wegen der höheren Heterogenität der Grid Ressource und Grid Jobs wurde bei der Entwicklung von Grid(Lab) Ressource Management System davon ausgegangen, dass jede Job Applikation eine einfache Web-Schnittstelle anbieten muss, um auf den Erzeugungsbefehl von GRMS entsprechend zu reagieren.

Das Re-Scheduling Plug-in verwendet Migration von laufenden Jobs, um die Ressource für „schwebende“ Jobs freizumachen. Im Unterschied zu den Anforderungen aus dem Kapitel 3.3 wird Checkpointing der migrierten Jobs nicht periodisch sondern direkt vor der Migration durchgeführt. Falls auf der Ressource eine Fehler auftreten sollte, wird der Job einfach neu gestartet. Zusätzlich muss man erwähnen, dass bei der Auswahl von Ressourcen für Migration nicht nur die angeforderten Ressourcen, sondern auch die Ressourcen mit „lockeren“ Anforderungen einbezogen werden.

Grid(Lab) Resource Management System (GRMS) erfüllt zwar viele aber leider nicht alle Anforderungen aus dem Kapitel 3.3. Da GRMS kein reguläres Checkpointing unterstützt, werden von ihm auch keine Checkpoint-Dateien regulär gespeichert. Das heißt, dass im Fall eines Ressource-Crashes wird keine Rocovery durch dynamische Migration unterstützt.

4.2. Weitere verwandte Arbeiten

Zweifellos gibt es eine Reihe von anderen Arbeiten, die sich mit dem Thema von Fehlertolerance beschäftigen. Es gibt zum Beispiel viele Arbeiten zum Thema von Fault Tolerance bei Web Services (17), die in dieser Arbeit aber nicht weiter charakterisiert werden, da die meisten dieser Web Services zustandslos sind und folglich für die Implementierung eines Grid Fault Tolerant Execution Service weniger von Bedeutung sind. Nach meinem besten Wissen gibt es zur Zeit keine verwandte Arbeiten, die sich mit der Thema der dynamischen Migration von Grid-Anwendungen auf der Basis von regulärem Checkpointing als einem Mittel für Fault Tolerance zu Ressource Crash-Fehler in einer Grid-Umgebung beschäfigen. Deswegen werden diese anderen Arbeiten hier nicht beschrieben.

4.3. Zusammenfassung

Auf der Basis von Informationen aus den themenrelevanten Arbeiten kann man die Vergleichstabelle 4.1 erstellen, um Übereinstimmung zwischen den Anforderungen aus dem Kapitel 3.3 und verschiedenen Arten von Grid Middleware Technologien zu illustrieren.

Anforderungen	**Globus**	**UNICORE**	**gLite**	**GRMS**
Fault Tolerance zu Ressource Crash-Fehler	✓	✓	✓	✓
Recovery durch dynamische Migration statt Rollback	—	—	—	—
fein-granulierte Zustandsüberwachung durch reguläres Checkpointing	—	—	—	—
Nutzung von ähnlichen und identischen Ressourcen	—	—	—	✓
unterschiedliche Resource Provider	✓	✓	✓	✓
Unabhängigkeit von der Art der benutzten Grid Middleware	—	—	—	✓
Portabilität	✓	✓	—	✓
Benutzerfreundlichkeit	—	✓	—	✓
Skaliebarkeit	✓	✓	✓	✓
Zuverlässigkeit	✓	✓	✓	✓
Performance	✓	✓	✓	✓
Sicherheit	✓	✓	✓	✓

Tabelle 4.1.: Vergleichstabelle der Entsprechung gängigste Arten von Grid Middleware Technologien der Anforderungen aus dem Kapitel 3.3

Aus der Vergleichstabelle 4.1 ist erkennbar, dass es nur wenige der Anforderungen an Fault Tolerance erfüllt werden und somit ein Bedarf an die Implementierung von Grid Fault Tolerant Execution Service besteht. Im nachfolgenden Kapitel wird ein solcher Service zuerst mit Hilfe eines UML Diagramms entworfen und danach werden die bedeutendste Teile dieses Entwurfs prototypisch implementiert und getestet.

Teil II.

Entwurf

5. Systementwurf

In diesem Kapitel wird der Grid Fault Tolerant Execution Service modelliert und an die im Kapitel 3.3 geschriebenen Anforderungen angepasst. Für die Arbeit mit mehreren Grid Middleware-Technologien ist die flexible Struktur von GFTES GM Adapter nicht zu unterschätzen. Durch diese Flexibilität bleiben die anderen Bestandteile von GFTES unverändert, selbst wenn die benutzten Grid Middleware geändert werden. Diese GFTES Struktur wird im ersten Abschnitt des Kapitels eingeführt.

Im zweiten Abschnitt werden die grundlegenden Entscheidungen bezüglich der Datenspeicherung sowie des globalen Kontrollflusses präsentiert.

Im nachfolgendem Teil wird die allgemeine Softwarearchitektur entwickelt sowie die Subsysteme von GFTES vorgestellt.

Abschließend werden alle zuvor getroffenen Entscheidungen in den Objektentwurf aufgenommen und dort zu einem Onbjektmodell verfeinert. Zusätzlich werden die Schnittstellen zwischen den Objekten und den verschiedenen Subsystemen festgelegt.

Um das System zu entwickeln, das die Anforderungen aus dem Kapitel 3.3 erfüllen wird, muss man auf die folgenden Fragen beantworten:

1. Welche Struktur soll GFTES haben, um ausreichende Flexibilität und Portierbarkeit zu bekommen?
2. Welche Komponenten soll GFTES haben und welche Anforderungen durch diese Komponenten erfüllt werden?
3. Wie funktionieren die Komponenten von GFTES um die Anforderungen zu erfüllen?

Die Antworten auf diese Fragen werden im Laufe dieses Kapitel herausgearbeitet.

5.1. Struktur von GFTES

In diesem Abschnitt wird die Struktur von GFTES eingeführt. In der Abbildung 5.1 sind drei Bestandteile von Grid Fault Tolerant Execution Service vorgestellt:

- **Core**: beinhaltet die *Business Logic* von GFTES. Wenn GFTES auf die andere Grid Middleware oder das andere Datenbank-System umgestellt wird, bleibt die Business Logic immer die selbe.
- **Datenbankmanagementsystem** (DBMS): ist für die Verwaltung aller Daten in der Datenbank zuständig. Falls GFTES auf eine neue Datenbank-Software oder einen neuen Datenbank-Dienst umgestellt wird, muss man nur diesen Teil entsprechend anpassen müssen.

- **Adapter**: ist für alle Informationflüsse sowie Verbindungen zwischen GFTES und dem Ressource Provider zuständig. Für den Anschluss einer neuen Grid Middleware an GFTES muss nur dieser Teil angepasst werden.

Eine solche Struktur sowie die streng definierten Schnittstellen für die Zusammenarbeit von allen Subsystemen bringen in GFTES ausreichende Dynamik und Anpassbarkeit, damit beim Wechseln aller externen Akteuren (wie z.B. der Grid Middleware Technologie), keine Änderungen vom Kern-Teil benötigt werden.

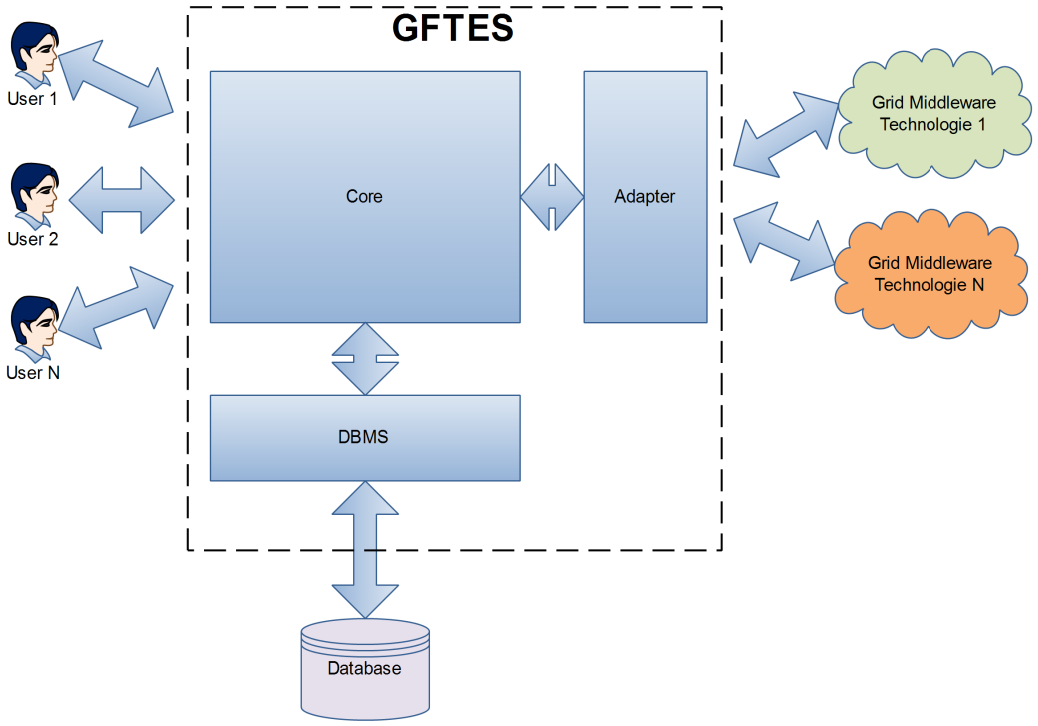

Abbildung 5.1.: Allgemeine Struktur von GFTES

5.2. Entwurfsziele und Entwurfsentscheidungen

Das gesamte System wird in Abschnitt 5.4 in mehrere unabhängige Subsysteme aufgeteilt. In dieser Sektion werden vier systemübergreifende Punkte: *Datenverwaltung*, *Checkpointing*, *Job Beschreibung* und *globaler Kontrollfluss* erklärt.

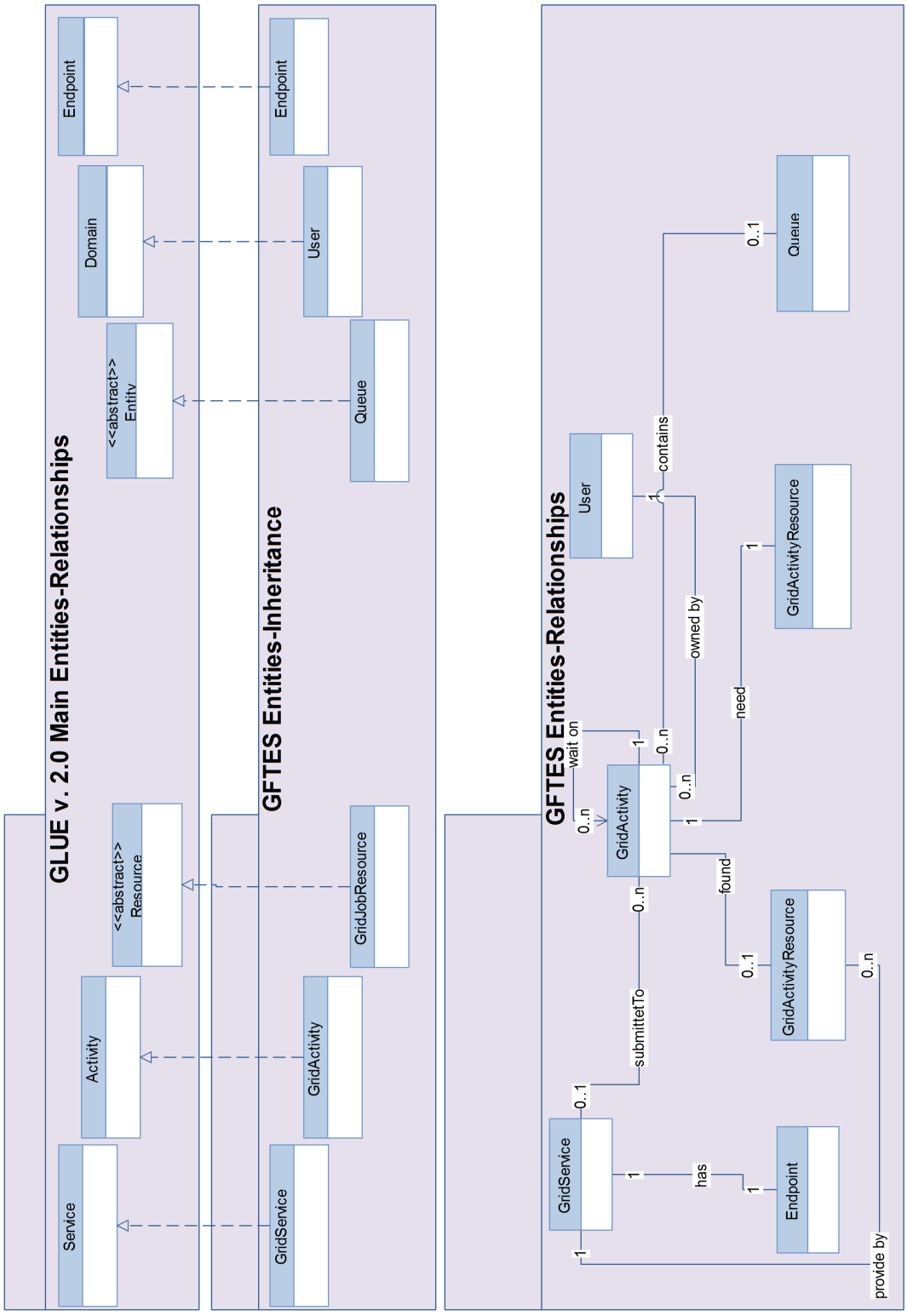

Abbildung 5.2.: Objekte und Beziehungen für GFTES Datenmodell

5.2.1. Datenverwaltung

Das Datenmodell von GFTES basiert auf dem GLUE v. 2.0 (1) Informationsmodell, das für Bedürfnisse von GFTES erweitert wurde. In der Abbildung 5.2 sind die Beziehungen zwischen allen abgeleiteten Objekten vorgestellt.
Auf dem Basis von GLUE 2.0 Objekte wurden für das Datenmodell folgende Informationsobjekte erstellt:

- **GridService**: beinhaltet Informationen über die benutzten Grid Services
- **Endpoint**: beinhaltet Daten über den Endpunkt eines Grid Services
- **User**: beinhaltet Informationen über die Benutzer von GFTES
- **GridActivity**: beinhaltet Informationen über die Job Beschreibung sowie die Ausführungsumgebung und Checkpoint Dateien
- **GridActivityResource**: beinhaltet Informationen über die Hardware-Ressourcen, die für den Job gesucht und gefunden wurden
- **Queue**: beinhaltet die Warteschlange für neue Jobs

Dieses Datenmodell wird in einer SQL Datenbank komplett GLUE v. 2.0 kompatibel umgesetzt. Jedes Objekte bekommt eigene Datenbanktabelle. Wenn ein Objekt die mehrwertigen Attribute besitzt, wird für jedes dieser Attribute eine eigene SQL Tabelle angelegt. In der Abbildung 5.3 sind alle Datenobjekt mit dazu gehörenden Attribute dargestellt.

5.2.2. Checkpointing

Checkpointing ist eine sehr wichtige Methode bei der Entwicklung eines Fault Tolerance Execution Services jeder Art. Ein Grid Job kann prinzipiell eine beliebige Gestalt haben. Einige Beispiele von einem Grid Job sind: ein Java Programm oder ein C++ Programm, die Aufgabe ein Diagramm aus der Input-Daten zu erstellen oder die Aufgabe ein Video aus einem Format in ein anderes zu konvertieren. Diese Vielfalt von möglichen Grid Job Arten macht erschwert den Auswahl einer einzigen Checkpointing Strategie bei dem Entwurf von Grid Fault Tolerant Execution Service.

Eine Grid Middleware Technologie kann auch eigene Checkpointing-Mechanismen haben, die aber nur für diese Middleware gelten. Die Möglichkeiten für Checkpointing wurden bereits im Kapitel 2 beschrieben. Recovery des Prozesses aus der Checkpoint-Datei, die in einem anderen Betriebssystem erstellt wurde, kann im neuen Betriebssystem einfach unmöglich sein.

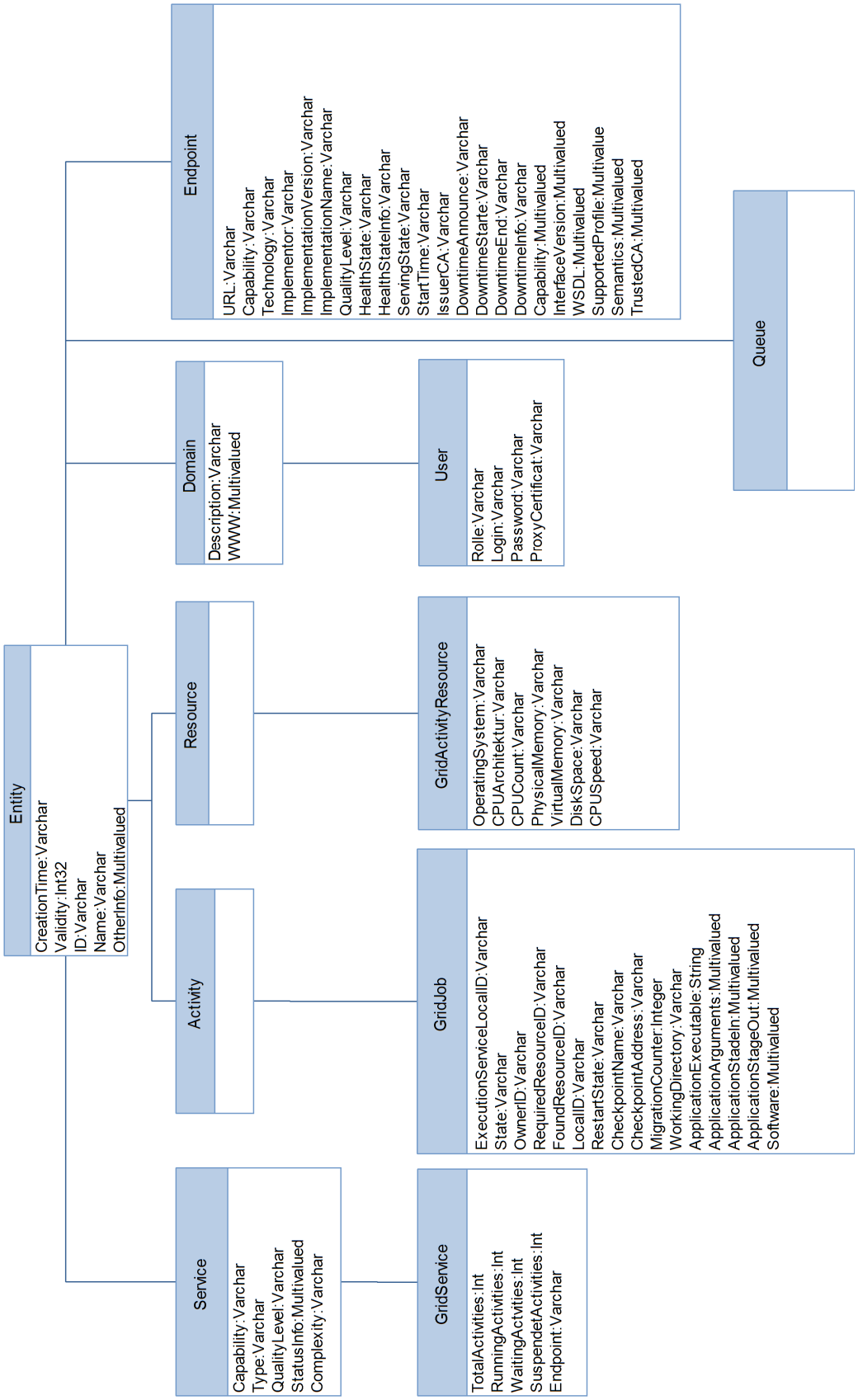

Abbildung 5.3.: Attribute und Ableitungen für GFTES Datenmodell

Ich sehe generell nur drei Wege für die Lösung dieses Kompatibilitätsproblems:

- man erstellt einen Konverter für Checkpoint-Dateien
- man arbeitet nur mit Grid Programmen, die für das Checkpointing geeignet sind
- man arbeitet mit allen Grid Jobs und akzeptiert die Tatsache, dass nicht alle Jobs die Checkpoint-Dateien erstellen können. Außerdem, kann eine Checkpoint-Datei (zum Beispiel wegen einem Übertragungsfehler) beschädigt sein. In diesem Fall ist die Recovery genauso unmögliche wie in einem Fall, wenn die Checkpoint-Datei nicht erstellt wurde. In diesem Fall ist man gezwungen, den Job einfach neu starten.

Im Rahmen dieser Arbeit wird angenommen, dass jeder Job periodisch eine Checkpoint-Datei erstellen kann. Die zweite Annahme betrifft Recovery des Jobs nach der Migration. Man geht im Rahmen dieser Arbeit davon aus, dass der Job spezielle interne Mechanismen für die automatische Wiederherstellung aus der Checkpoint-Datei bereitstellt. Dafür muss die Checkpoint-Datei zusammen mit den anderen Input-Dateien auf die Grid-Ressource übertragen werden. Damit Grid Fault Tolerant Execution Service die Checkpoint-Dateien in seiner Datenbank periodisch abspeichert, muss der Benutzer den Name dieser Checkpoint-Datei in der Job Beschreibung angeben. GFTES überwacht die Erstellung der neuen Checkpoint-Datei während der gesamten Laufzeit des Jobs und speichert sie periodisch in der Datenbank. Bei der Job Migration wird der Pfad zu der Checkpoint-Datei in das Job Description Document eingefügt. Nachdem diese Datei auf die Grid Ressource übertragen wird, wird die Job-Anwendung die Recovery automatisch starten.

5.2.3. Job Beschreibung

Für alle Jobs wurde in Rahmen dieser Arbeit die spezielle Sprache für Erstellung der Job Beschreibung entwickelt. Als Basis für diese Sprache dient *Job Description Submission Language* (JSDL) (2).

Eine typische GFTES Job Beschreibung, wie man im Listing 5.1 sehen kann, besteht es aus folgenden Teilen, die auch aus der JSDL Spezifikation (v. 1.0) bekannt sind :

- **JobIdentification**: die allgemeine Beshreibung des Jobs in normale menschliche Sprache sowie sein Name
- **Application**: die Beschreibung der Ausführungsumgebung
- **Resources**: die Beschreibung der gewünschten Ressourcen
- **DataAttributes**: die Beschreibung von Input und Output Dateien

```
1 <JobDefinition>
2   <JobDescription>
3     <JobIdentification ...="" />?
4     <Application ...="" />?
5     <Resources ...="" />?
6     <DataAttributes ...="" />*
7   </JobDescription>
```

```
</JobDefinition>
```

Listing 5.1: Job Description Document

Ein Beispiel von der Job Beschreibung kann man im Listing 5.2 sehen.

```
<?xml version="1.0" encoding="utf-8"?>
<jsdl:JobDefinition
        xmlns="http://www.example.org/"
        xmlns:jsdl="http://localhost/jsdl/2011/01/jsdl"
  xsi:schemaLocation="http://localhost/jsdl/">
  <jsdl:JobIdentification>
    <jsdl:JobName>
      Calculation of the Fibonacci Number
    </jsdl:JobName>
    <jsdl:Description>
      Simple application invocation:
      User wants to run the java programm 'fibo.class'
      to get a Fibonacci number.

      This Fibonacci number will be stored to 'output.xml' file.
      All staged-in data will be copied to working directory.

      In case of error, message schould be produced to the same
      'output.xml' file.
    </jsdl:Description>
  </jsdl:JobIdentification>
  <jsdl:Resource>
    <jsdl:CPUCount>2</jsdl:CPUCount>
    <jsdl:PhysicalMemory>20971520</jsdl:PhysicalMemory>
  </jsdl:Resource>
  <jsdl:SoftwareRequirements>
    <SoftwareDescription>Java</SoftwareDescription>
    <SoftwareVersion>1.6</SoftwareVersion>
  </jsdl:SoftwareRequirements>
  <jsdl:Application>
    <jsdl:ExecutableName>
      java
    </jsdl:ExecutableName>
    <jsdl:Argument>fibo</jsdl:Argument>
    <jsdl:WorkingDirectory>
      /sandbox/test/files/temporary
    </jsdl:WorkingDirectory>
    <jsdl:InCaseOfResourceCrash>recovery</jsdl:InCaseOfResourceCrash>
    <jsdl:CheckpointName>checkpoint.xml</jsdl:CheckpointName>
  </jsdl:Application>
  <jsdl:DataAttributes>
    <jsdl:File>
      <jsdl:FileName>fibo.class</jsdl:FileName>
      <jsdl:Source>
        /sandbox/test/files/fibo.class
      </jsdl:Source>
      <jsdl:DeleteOnTermination/>
```

```
48      </jsdl:File>
49      <jsdl:File>
50        <jsdl:FileName>input.xml</jsdl:FileName>
51        <jsdl:Source>
52          /sandbox/test/files/input.xml
53        </jsdl:Source>
54        <jsdl:DeleteOnTermination/>
55      </jsdl:File>
56      <jsdl:File>
57        <jsdl:FileName>output.xml</jsdl:FileName>
58        <jsdl:Source>
59          /sandbox/test/files/output.xml
60        </jsdl:Source>
61        <jsdl:DeleteOnTermination/>
62      </jsdl:File>
63      <jsdl:File>
64        <jsdl:FileName>checkpoint.xml</jsdl:FileName>
65        <jsdl:Source>
66          /sandbox/test/files/checkpoint.xml
67        </jsdl:Source>
68        <jsdl:DeleteOnTermination/>
69      </jsdl:File>
70    </jsdl:DataAttributes>
71  </jsdl:JobDefinition>
```

Listing 5.2: Job Description Document nach JSDL Standart

5.2.4. Globaler Kontrollfluss

Grid Fault Tolerant Execution Service ist dafür ausgelegt, dass dieses System von mehreren Benutzern sowie Administratoren gleichzeitig benutzt werden kann. Dafür werden alle Informationen in der GFTES-Datenbank immer in einem konsistentem Zustand gehalten und die Änderungen von Job Status werden sofort in der Datenbank abgebildet.

Den globalen Kontrollfluss kann man im Aktivitätsdiagramm 5.4 sehen. Man kann grob die folgenden Ablaufschritte unterscheiden:

1. die Zugriffsberechtigung vom User prüfen
2. den Job annehmen
3. die passenden Ressourcen bei Grid Middleware finden
4. das Job Description Document für gefundene Ressource zusammenstellen und bei dem Ressource Provider vorlegen
5. Job-Ausführung beobachten und wenn nötig, automatisch in die neue Ressource-Umgebung umziehen/migrieren lassen.

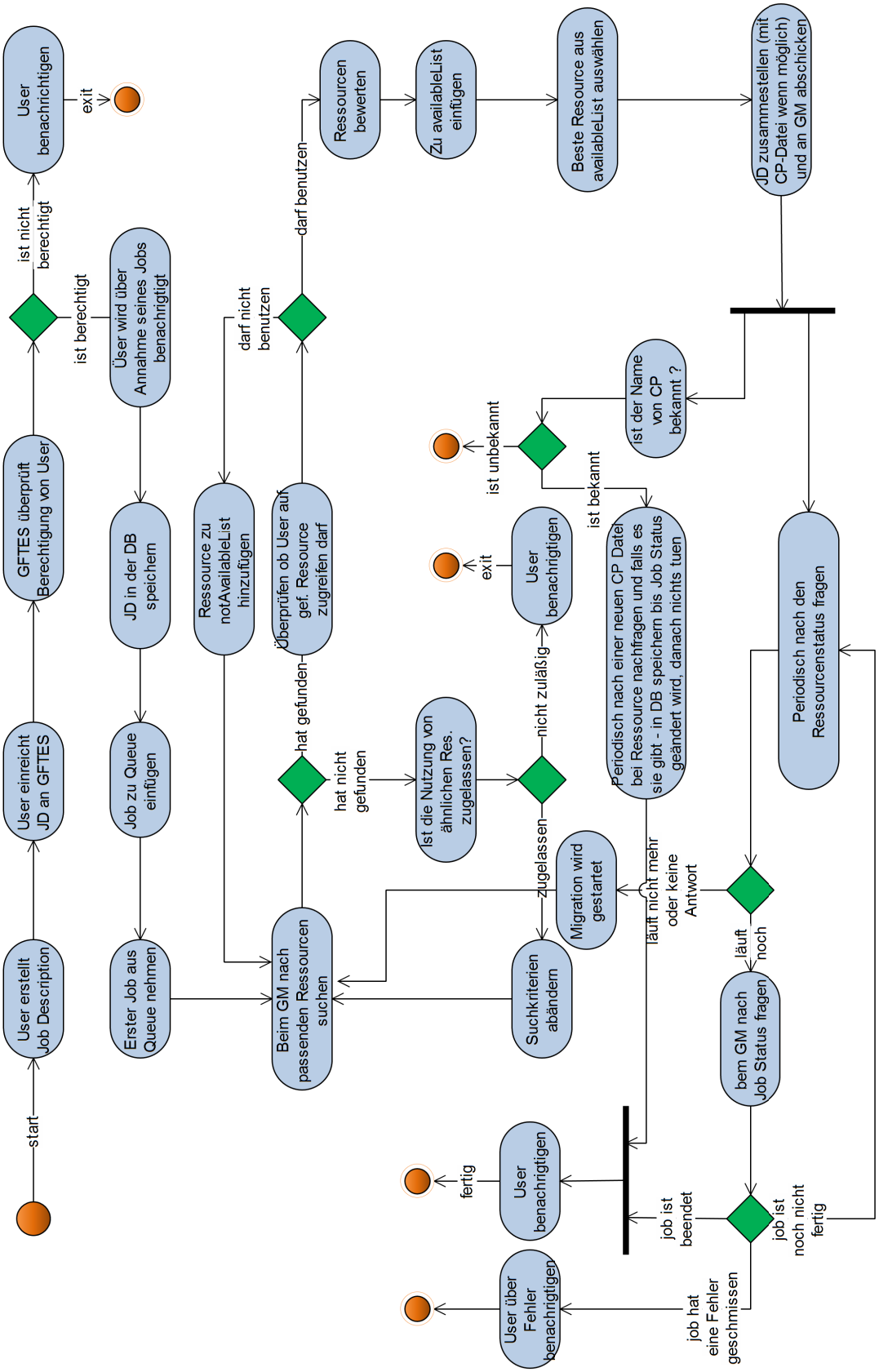

Abbildung 5.4.: GFTES Aktivitätsdiagram

5.3. Softwarearchitektur

Die Softwarearchitektur von Grid Fault Tolerant Execution Services basiert auf dem Entwurfmuster „Model View Controller (MVC)“ (Abbildung 5.5) (24). Die Trennung von Model, View und Controller entspricht dem generellen Prinzip der Aufteilung von Zuständigkeiten, das der Modularisierung von Programmen dient (10, Seite 50).

MVC unterteilt jede interaktive Applikation in drei Komponenten: das Modell beinhaltet die Kernfunkionalität und die Daten, der View zeigt dem Benutzer die Informationen und der Controller verarbeitet die Benutzereingaben. View und Controller gemeinsam definieren die Benutzerschnittstelle. Ein Mechanismus zur Weiterleitung von Änderungen sichert die Konsistenz zwischen der Benutzerschnittstelle und dem Modell (5, Seite 105).

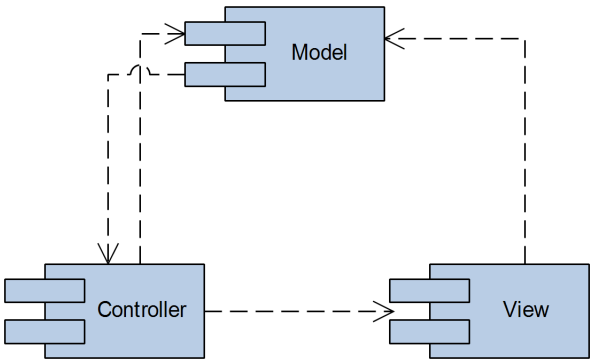

Abbildung 5.5.: Das Entwurfmuster „Model View Controller“ (24)

Der große Vorteil dieses Konzepts besteht darin, dass sich alle drei Teile unterschiedlich entwickeln und einsetzen lassen. Die Darstellungs-Komponenten können weiterentwickelt werden, und das Model ändert sich nicht. In GFTES ist dies besonders aufgrund des wechselnden Grid Middleware Technologien interessant, die sich zur Laufzeit dynamisch ändern können. Die Business Logic im Model muss man dabei nicht neu anpassen. Aufgrund seiner Flexibilität wurde dieses Entwurfmuster für die prototypische Implementierung von GFTES ausgewählt.

5.4. Subsysteme

In der Abbildung 5.6 sind drei Subsysteme von GFTES gezeigt. Der Service besteht aus drei Packages. Jeder Package entspricht einem Subsystem. Im Package *DBMS* sind die Klassen für Verwaltung von Daten in der Datenbank enthalten. Im Package *Adapter* findet man die Klassen für Interaktionen mit verschiedenen Grid Middleware Technologien. Und im Package *Core* befinden sich die Klassen, die Business Logic von GFTES bilden.

Die Verbindung zwischen den drei Subsystemen geschieht über die vordefinierten Schnittstellen, die im nächsten Teil genauer dargestellt werden. Großer Wert wurde auf möglichst

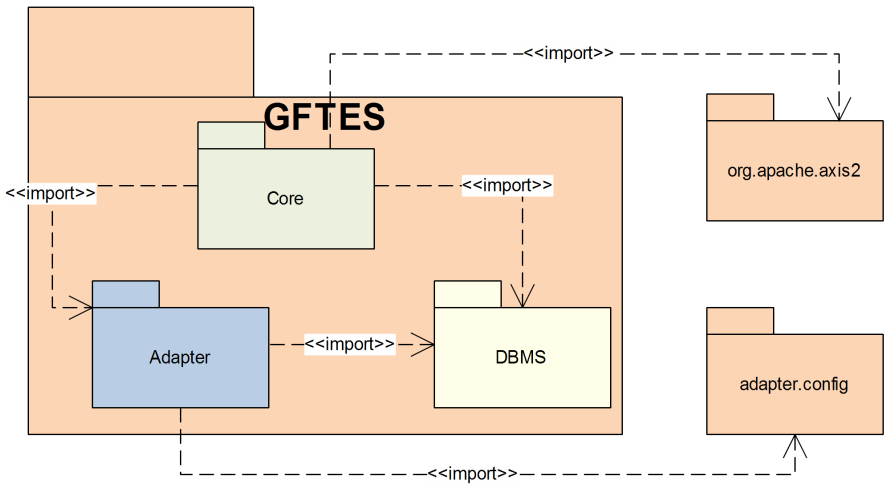

Abbildung 5.6.: Das GFTES Package Diagramm

geringe Anzahl von Verbindungen zwischen den verschiedenen Teilen gelegt.

In der Abbildung 5.6 sind alle drei Packages mit ihren Verbindungen innerhalb des Grid Fault Tolerant Execution Service sowie außerhalb des GFTES gezeigt.

5.5. Objektentwurf

In diesem Abschnitt wird das Objektmodell von Grid Fault Tolerant Execution Service beschrieben, das im Rahmen dieser Diplomarbeit entwickelt wurde. In der Abbildung 5.7 sind alle Klassen der drei Packages **Core**, **DBMS** und **Adapter** sowie die Beziehungen zwischen diesen Klassen abgebildet. Die Methoden und Attribute werden diesen Klassen erst später im Lauf der Darstellung von Sequenzdiagrammen hinzugefügt. Das zusätzliche Package **Config** beinhaltet nur die Client-Stub Klassen und wurde für das Testen von GFTES künstlich erzeugt. Sein Inhalt ist für die Beschreibung der Struktur und der internen Business Logic von Grid Fault Tolerant Execution Service unerheblich und wird deswegen im Rahmen dieser Arbeit nicht beschrieben. Der Package *gftes.adapter.config* kann man beispielsweise durch die Bibliotheken aus dem JavaGAT-Projekt [1] ersetzen. Eine andere Alternative ist die automatische Generierung der Stub-Klassen auf der Basis vom WSDL-Dokument des Web-Services. Für diese Generierung kann man das Programm **wsdl2java** [2] benutzen.

[1] Mehr zum **JavaGAT**-Projekt findet man unter `http://gforge.cs.vu.nl/gf/project/javagat/frs/`
[2] Mehr über **wsdl2java** findet man unter `http://ws.apache.org/axis/java/user-guide.html`

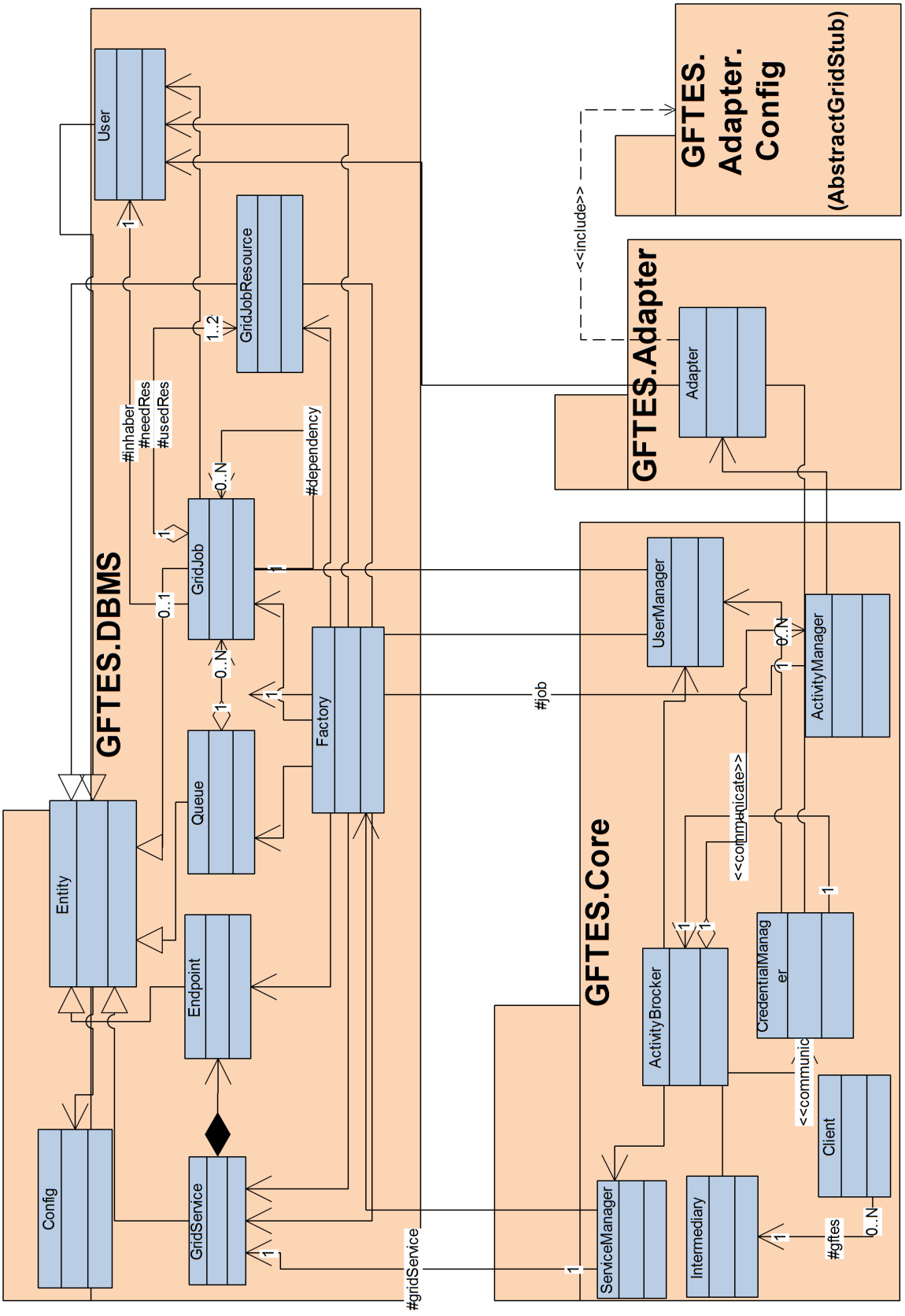

Abbildung 5.7.: Das Klassen Diagramm von GFTES

Eine der nicht-funktionalen Anforderungen aus dem Kapitel 3.3 ist die Anfoderung an **Portabilität**. Im Bezug auf Grid Fault Tolerant Execution Service diese Anforderung bedeutet, dass GFTES auf jedem Betriebssystem installiert werden kann, ohne damit man das Programm umständlich ändern muss. Das betrifft auch die externen benutzen Komponenten wie die Datenbank. Man kann beispielsweise eine SQL Datenbenk durch eine XML-Datenbank ersetzen. Deswegen wurde GFTES im Form einer modularen Struktur entworfen, wo jeder Teil durch eine neue Implementation ersetzt werden kann. Und die Kommunikation zwischen den Klassen verschiedener Teile ausschließlich mit Hilfe der streng definierten Schnittstellen stattfindet. Diese Schnittstellen bilden die Basis für Flexibilität von GFTES sowie die Erfüllung von GFTES der Anforderung an Portabilität. In dieser Weise wird es möglich, die Ressourcen von vielen Resource Provider zu benutzen, sogar wenn diese Provider unterschiedliche Grid Middleware Technologien verwenden. In diesem Fall wird einfach der *Adapter* Paket erweitert, ohne damit die anderen beiden Packages auch geändert werden müssen. Diese Interface werden im Laufe dieser Arbeit herausgearbeitet.
In Weiterem werden die Sequenzdiagrammen vorgestellt, die für besseres Verständnis innerer Mechanismen sowie der Funktionen der Klassen von GFTES sorgen sollen. Es wird dabei auch erklärt, welche Anforderungen aus dem Kapitel 3.3 mit Hilfe dieser Klassen erfüllt wurden.

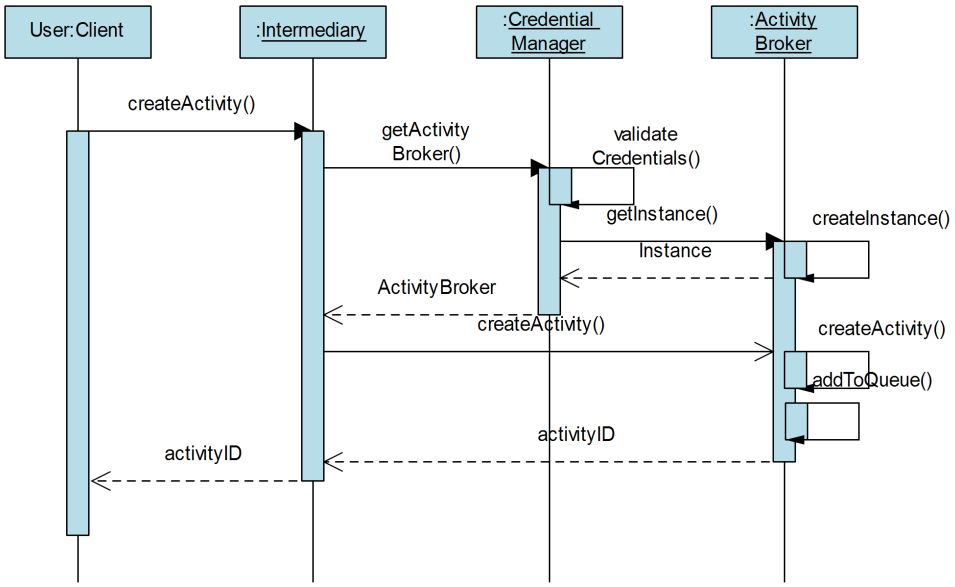

Abbildung 5.8.: Sequence Diagramm: Job Submission

In der Abbildung 5.8 kann man das Einreichen eines neuen Jobs an GFTES sehen. Die Klasse *Intermediary* definiert das Frontend-Interface von GFTES. Die Anforderung **Benutzerfreundlichkeit** wird in Rahmen dieser Arbeit so verstanden, dass man die Frontend-Schnittstelle von GFTES nach einem allgemein anerkannten Standard entwickeln soll. Für die Grid Execution Services wurde dieser Standard in der OGSA BES Spezifikation beschrieben (9). Diese Spezifikation definiert Standardfunktionen und Standardattribute, die von jedem Grid Execution Service realisiert werden sollen. Die Implementierung des Interfaces nach dem OGSA BES Standard wird für das bessere Verständnis zwischen verschiedener Grid Services beitragen. Diese Standardfunktionen wurden im Interface *IntermediaryInterface* beschrieben und in der Klasse *Intermediary* implementiert.

Die Authentifizierung und Autorisierung findet in der Klasse *CredentialManager* statt. Die Klasse *CredentialManager* dient zur Erfüllung der nicht-funkionalen Anforderung an **Sicherheit**, da nur die berechtigten Benutzer auf die Funktionalität von GFTES zugreifen dürfen. Nach der Autorisierung wird der *ActivityBroker* den Job zur Queue eingefügt. An den Client wird der lokale *ActivityID* zurückgeschickt. Mit diesem ActivityID kann der Client auf die Zustandsinformationen seines Jobs zuzugreifen, sowie die Ausführung des Jobs abbrechen.

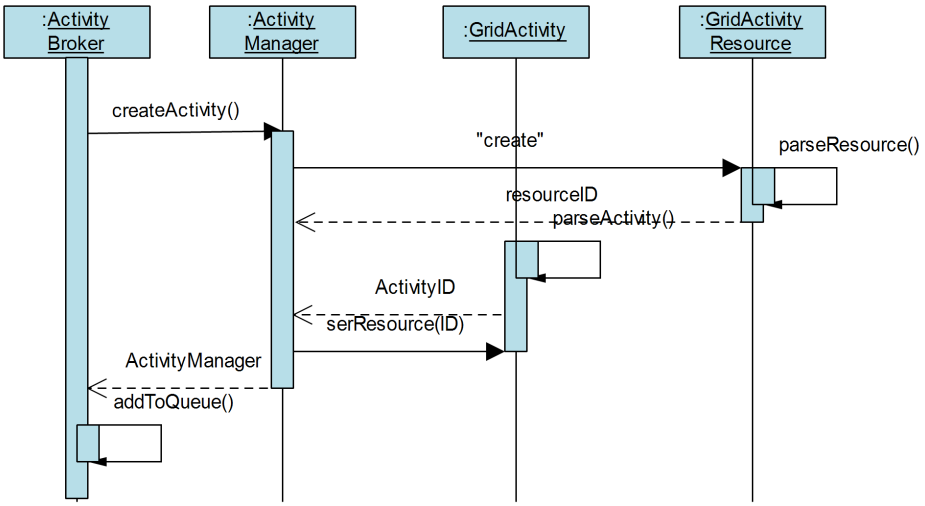

Abbildung 5.9.: Sequence Diagramm: Job wird in der Datenbank gespeichert und anschließend in die Queue eingefügt

Die Anforderungen an **Skalierbarkeit** und **Performance** bedeuten, dass viele Benutzer gleichzeitig auf den Service zugreifen dürfen, sowie dass die Jobs nicht sequenziell sondern parallel oder quasi-parallel ausgeführt werden. Um diese Anforderungen zu erfüllen, wurden die Klassen *ActivityBroker* und *ActivityManager* in eigenen Threads gestartet. So können mehrere Jobs parallel auf unterschiedlichen Ressourcen gestartet und verwaltet werden. In der Abbildung 5.9 sind die Ablaufschritte für die weitere Bearbeitung der Job Beschreibung dargestellt (noch bevor der ID dieses Jobs der Queue zugefügt wird). Das Objekt der Klasse *ActivityBroker* erstellt den neuen *AcitivityManager*, um die Bearbeitung von neuer Job Beschreibung zu starten. Dabei werden die benötigen Informationen aus dem XML-Struktur erkannt und in der Datenbank gespeichert. Dafür dienen zwei Objekte *GridActivity* und *GridActivityResourcen*, die Mechanismen zum Parsen und Speichern von Daten besitzen. Jeder Job bekommt einen eindeutigen ID, der für die Unterscheidung und die Wiederherstellung von Informationen aus der Datenbank benutzt werden kann. Dieser ID wird von dem *ActivityBroker* in der Queue gespeichert.

Der *ActivityBroker* wird in einem eigenen Thread gestartet, um die wartenden Jobs aus der Queue zu bearbeiten. Der *ActivityManager* wird ebenso in dem eigenen Thread gestartet, nachdem alle Informationen zu dem Job aus der Datenbank wiederhergestellt wurden. Dieser Prozess ist in der Abbildung 5.10 gezeigt.

Im nächsten Schritt muss der *ActivityManager* passende Ressourcen finden, das Job Description Document erstellen lassen und anschließend, nachdem dieses Dokument bei der gefundenen Grid Ressource vorgelegt wurde, das Monitoring des Jobzustandes und die Überwachung der Erzeugung von neuen Checkpoint-Dateien zu starten. Die funktionale Anforderung an **fein-granulare Zustandsüberwachung durch reguläres Checkpointing** erfüllt man, wenn das System regulär nach neuen Checkpoint-Dateien abfragt. In dieser Weise bleibt das System mit der Grid Ressource ständig im Kontakt und kann schnell verstehen, wenn die Ressource ausfällt. Diese periodischen Abfragen sind die Aufgabe des *ActivityManagers*, der die Checkpoint-Dateien in der Datenbank speichert. Für den Fall, wenn die Job-Anwendung keine Checkpointing-Mechanismen besitzt, wird parallel das Monitoring des Jobzustandes durchgeführt.

Um die Anforderung an *Fault Tolerance bei Crash-Fehler* zu erfüllen muss der GFTES nicht nur erkennen, wann die Grid Ressource ausgefallen ist, sondern auch wie man mit dem Crash-Fehler umgehen soll. Im Rahmen dieser Arbeit wurde das Konzept „Recovery statt Rollback“ implementiert. Sobald der *ActivityManager* die Tatsache versteht, dass die benutzte Grid Ressource ausgefallen ist, wird er Re-Scheduling Prozedur starten. *ActivityManager* sucht nach neuen Grid Ressourcen und, sobald diese Ressource gefunden wird, den Job zusammen mit der letzten Checkpoint-Datei an die neue Grid Ressource schickt. Dort wird der Job wiederhergestellt und fortgesetzt. Dieses Verfahren hilft die Anforderung an **Recovery durch dynamische Migration statt Rollback** zu erfüllen.

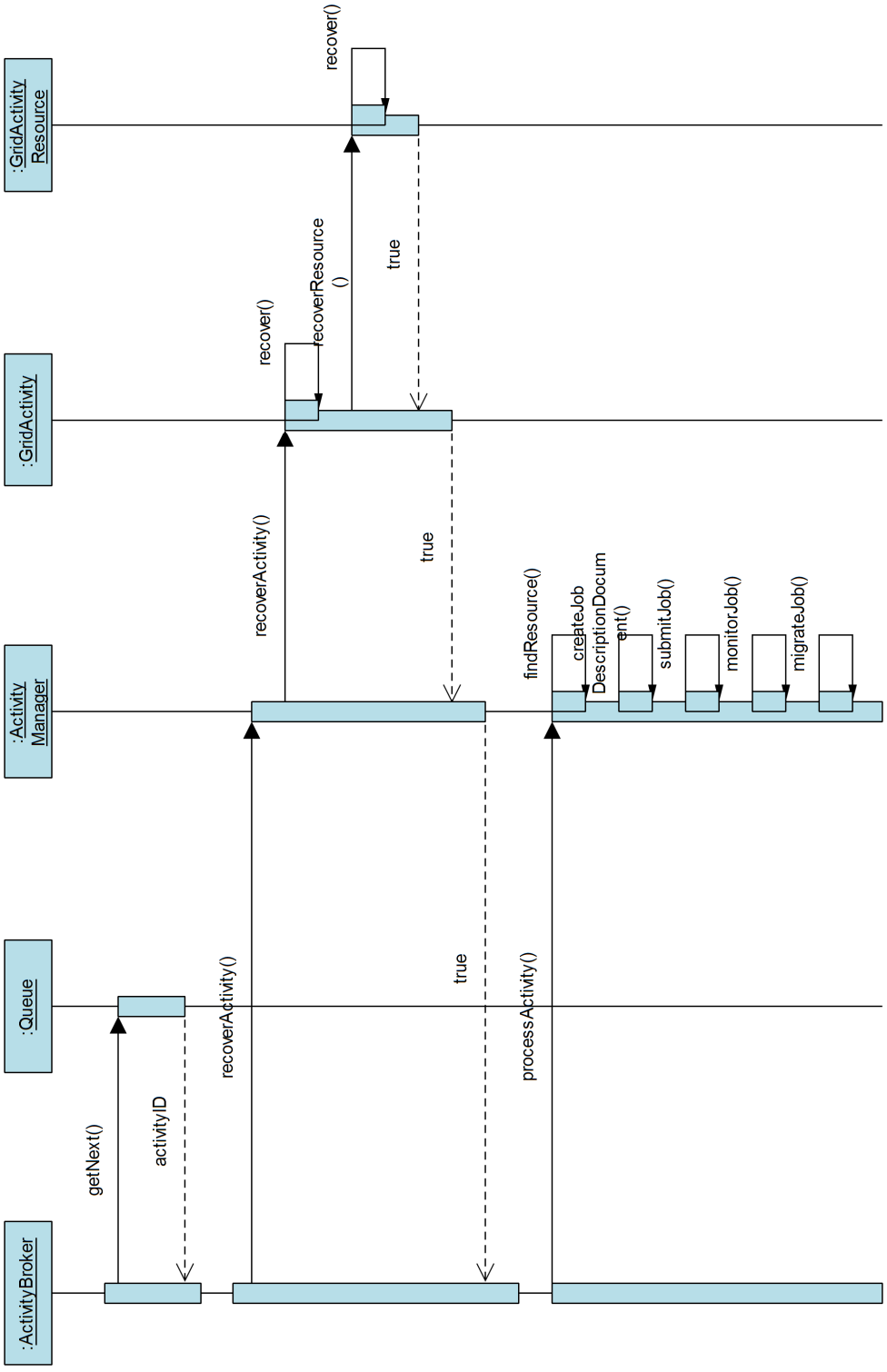

Abbildung 5.10.: Sequence Diagramm: der nächste Job aus der Queue wird bearbeitet

Die Anforderung **nicht nur identische sondern auch ähnliche Ressourcen zu nutzen** wird dadurch erfüllt, dass der *ActivityManager* den automatischen Auswahl der best passenden aus allen verfügbaren Grid Ressourcen in dem Fall startet, wenn die gewünschte Ressource nicht verfügbar ist. Dafür besitzt der *ActivityManager* die spezielle Funktion für Ressourcenbewertung. Dieser Vorgang hilft die Zeitverzögerung bei der Ausführung des Jobs zu vermeiden, was für die längeren Jobs besonders viel Zeit sparen kann.

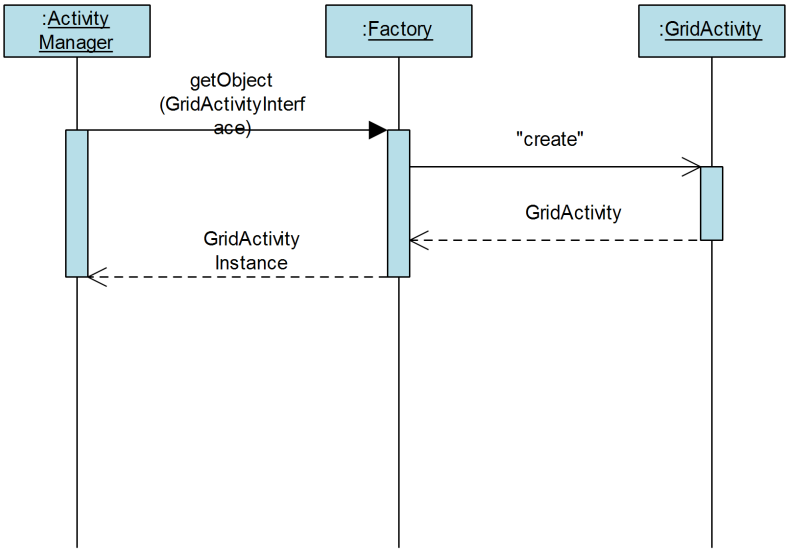

Abbildung 5.11.: Sequence Diagramm: Erzeugung eines neuen Objekts von der Klasse aus dem Package *DBMS*

In der Abbildung 5.11 ist gezeigt, wie das Erstellen eines Objekts aus dem Package *DBMS* stattfindet. Die Klasse *ActivityManager* kennt nur den *GridActivityInterface* aber nicht die Klasse *GridActivity*. Wenn der *ActivityManager* eine Implementation dieses Interfaces braucht, fragt er das *Factory* an. Falls man nun die Klasse *GridActivity* beispielsweise durch die Klasse *GridActivity2* ersetzen, wird der *ActivityManager* diese Ersetzung gar nicht merken, da die Kommunikation zwischen den zwei Instanzen dieser Klassen ausschließlich mit Hilfe von definiertem Interface *GridActivityInterface* stattfindet. Wenn der *AcitvityManager* die Informationen aus der Datenbank braucht, wird er die statische Methode aus der Klasse *Factory* mit der Angabe von gewünschtem Interface als Parameter aufrufen und der *Factory* erzeugt ein richtiges Objekt.

Für den *ActivityManager* spielen die Unterschiede zwischen den verschiedenen Resource Provider sowie zwischen den verschiedenen Grid Middleware Technologien keine Rolle. Diese Unterschiede werden von internen Mechanismen des *Adapters* verbergt. Durch diese Eigenschaft des *Adapters* werden die Anforderungen an **die unterschiedlichen Resource Provider** und **der Unabhängigkeit von der Art der benutzten Grid Middleware** erfüllt. In der Abbildung 5.12 sind die Standardfunktionen für jede Implementation von *Adapter* dargestellt. Der *Adapter* kann für jeder Grid Middleware das passende Job Des-

cription Document erstellt und übernimmt die gesamte Kommunikation zwischen der Grid Middleware und dem GFTES *Core*.

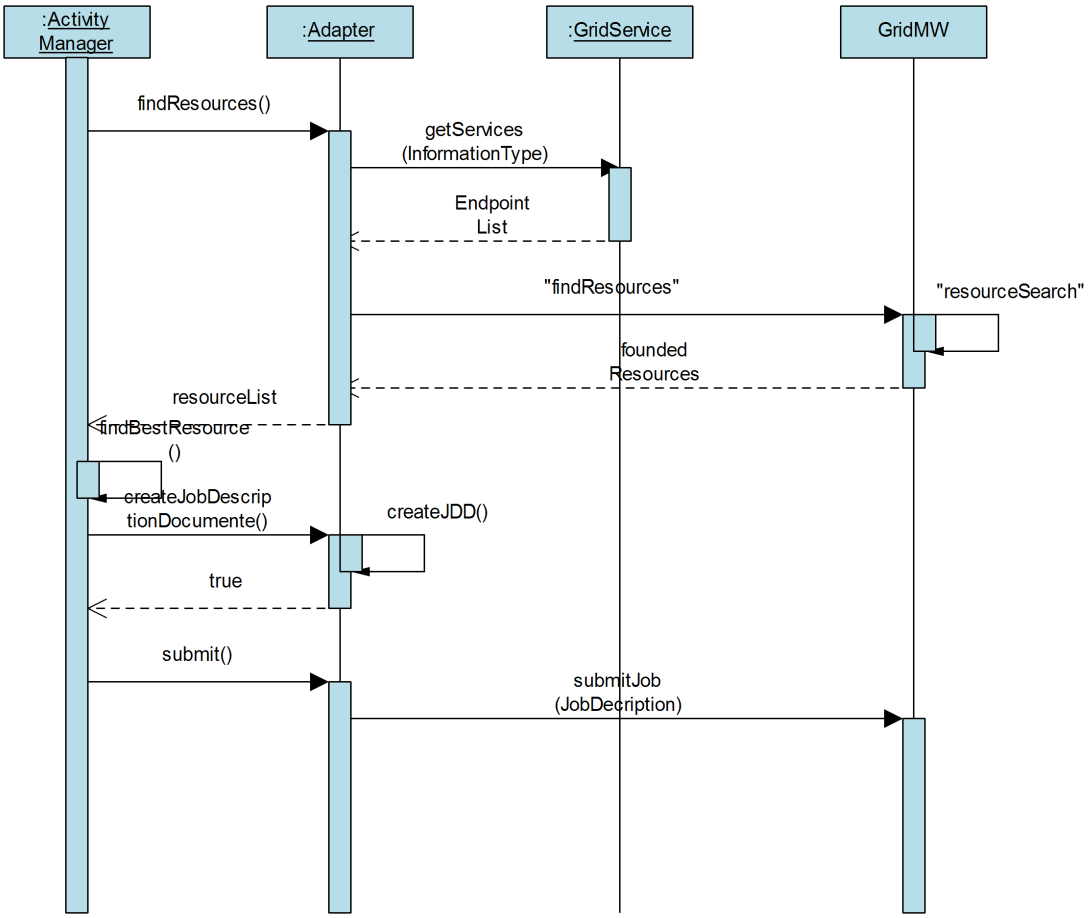

Abbildung 5.12.: Sequence Diagramm: Job wird an Grid Middleware übergeben

In der Abbildung 5.13 ist der Prozess von Job-Zustandsüberwachung dargestellt, der vom *ActivityManager* gemacht wird. Falls die Grid Ressource ausfällt, wird der *ActivityManager* das schnell merken. Danach wird für laufenden Job die Re-Scheduling Routine mit anschließender Migration-Routine gestartet.

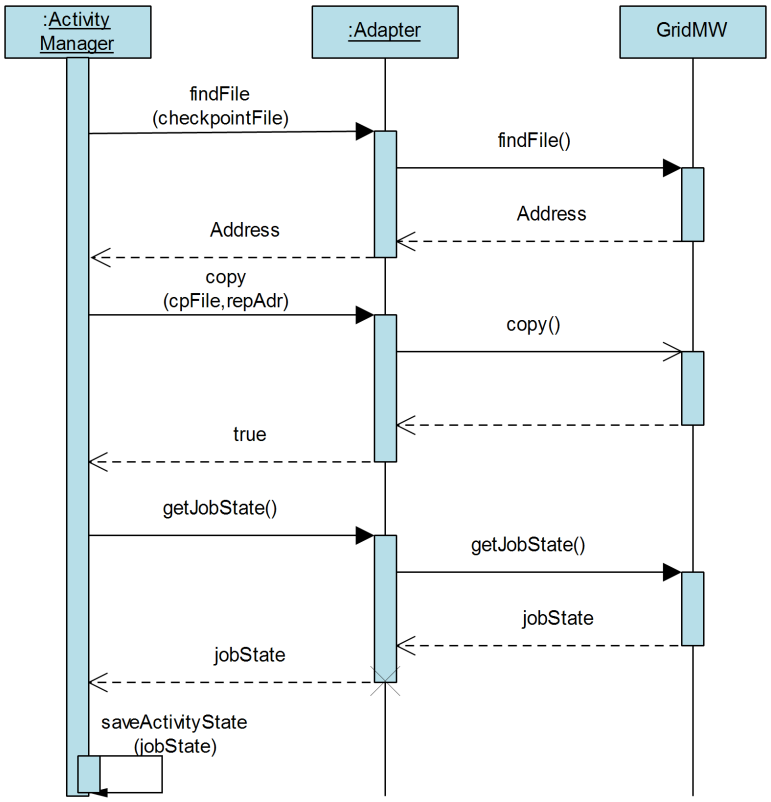

Abbildung 5.13.: Sequence Diagramm: Job wird von GFTES überwacht

In den Abbildungen 5.14, 5.15 und 5.16 sind alle drei Packages zusammen mit den Interfaces und den Klassen sowie die Klassenbeziehungen innerhalb jedes Pakets dargestellt. Alle Klassen und Interfaces sind mit den Methoden gezeigt, die teilweise aus den obigen Sequenzdiagrammen bekannt sind.

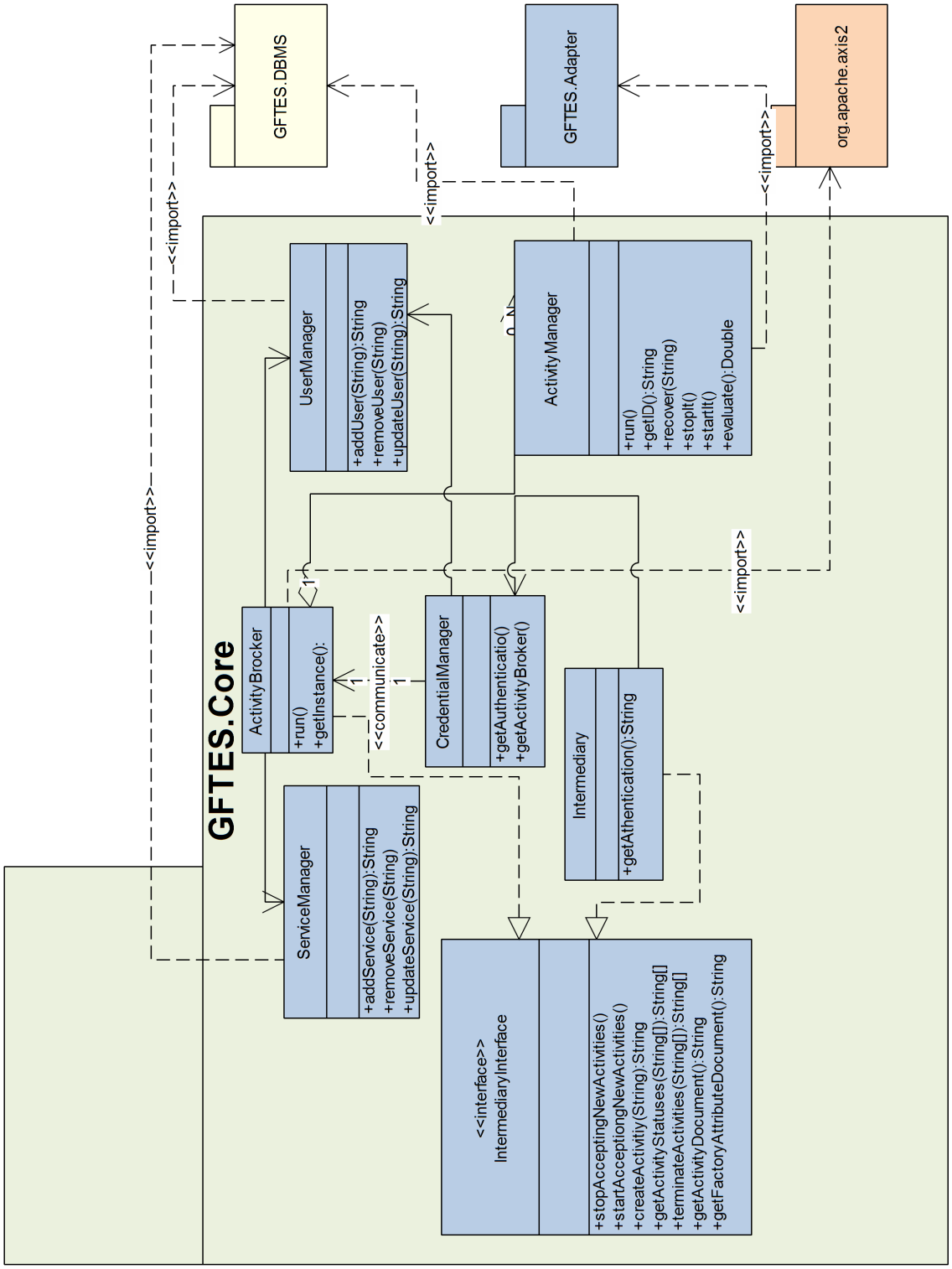

Abbildung 5.14.: Klassendiagramm: Klasse und Interface von Core Package

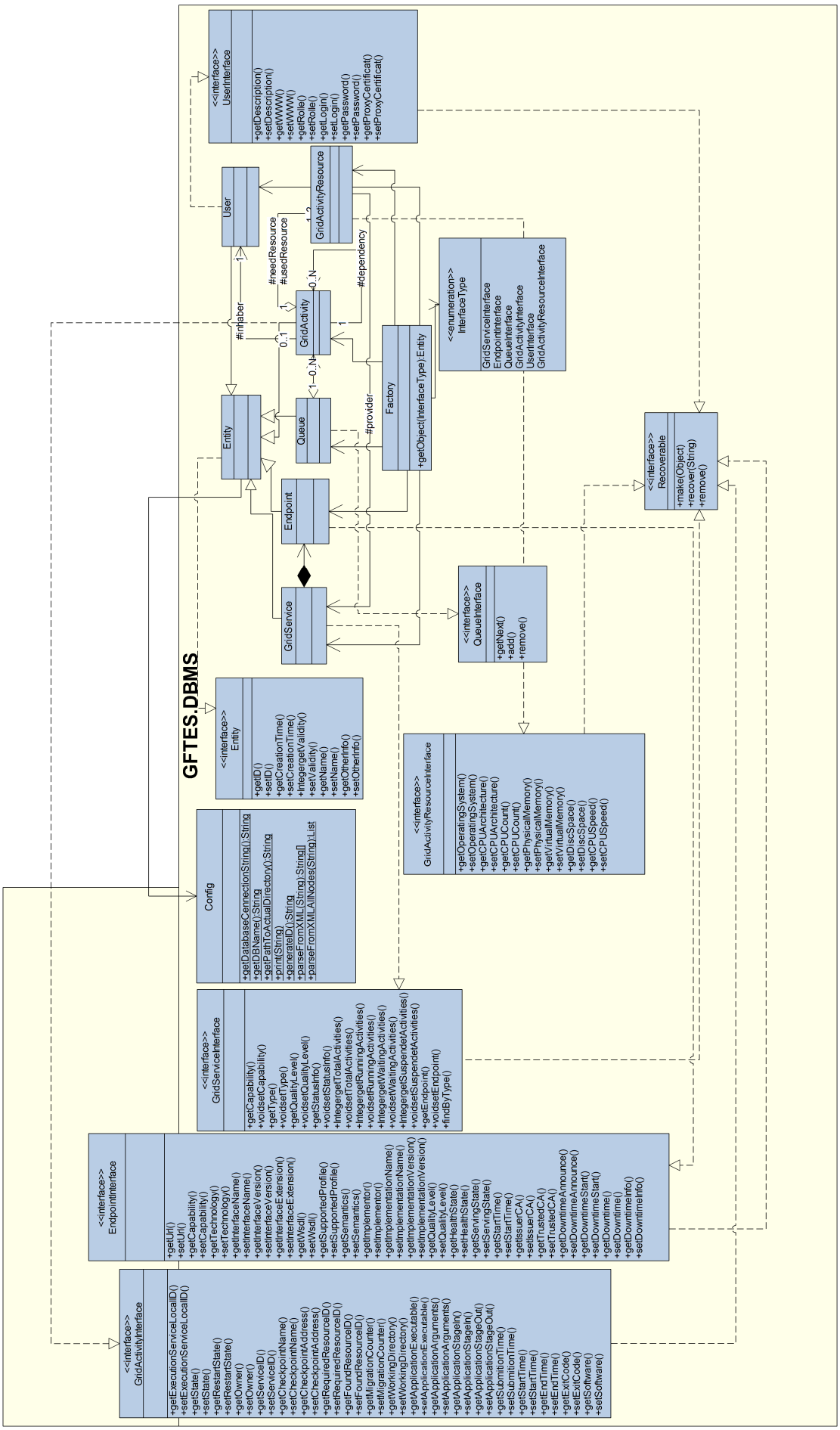

Abbildung 5.15.: Klassendiagramm: Klassen und Interface von DBMS Package

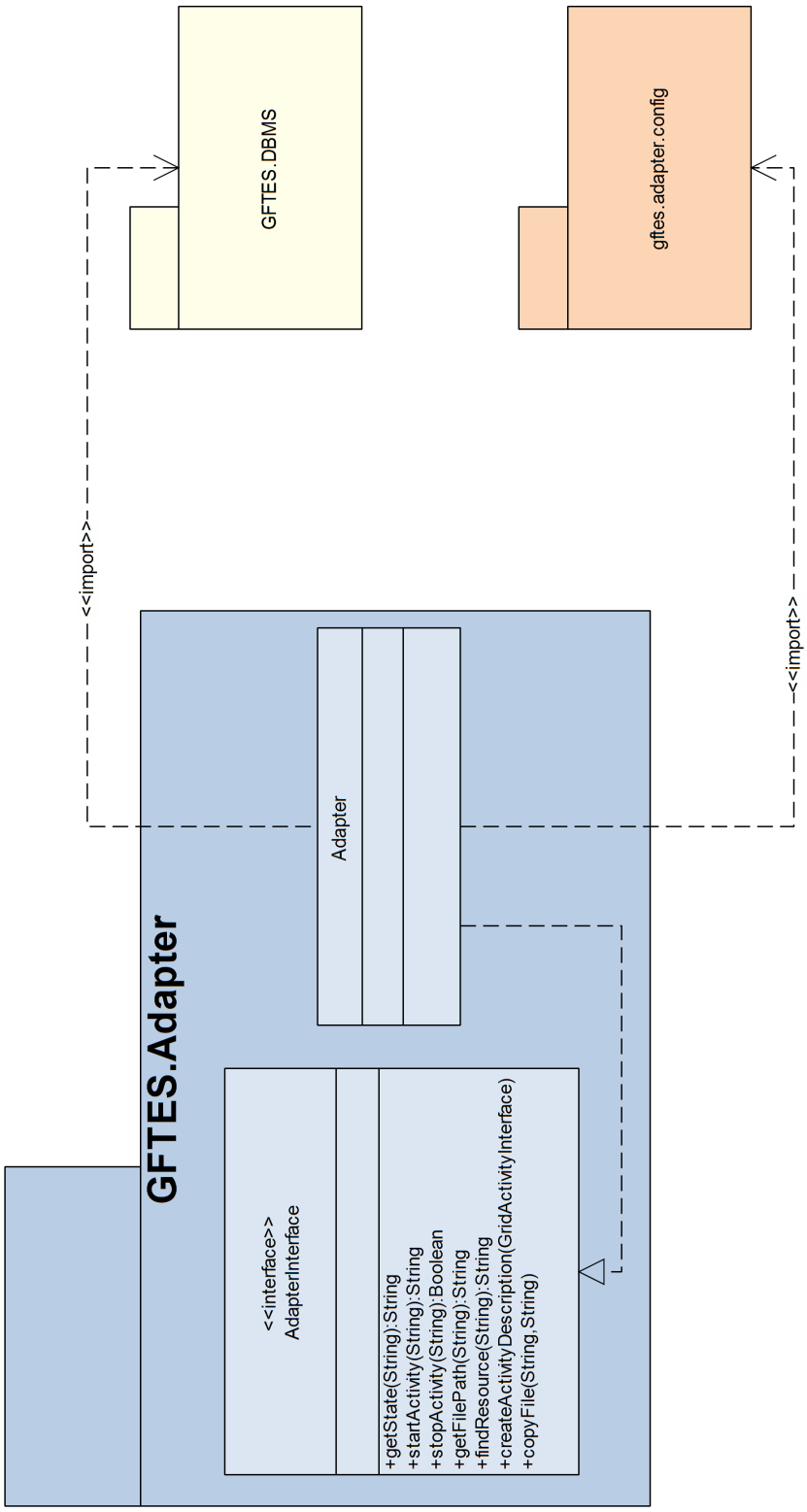

Abbildung 5.16.: Klassendiagramm: Klasse und Interface von Adapter Package

5.6. Zusammenfassung

In diesem Kapitel wurde das System GFTES entworfen. Die Struktur von GFTES wurde in der Form von UML-Diagrammen präsentiert. Man hat die Mechanismen für die Erfüllung der Anforderungen aus dem Kapitel 3.3 dargestellt. Diese Mechanismen sind hier noch mal kurz zusammengefasst.

Die Anforderung an die **Fault Tolerance zu Ressource Crash-Fehler** wird dadurch erfüllt, dass GFTES mit Hilfe von regulären Zustandsabfragen einen Ressource Fault schnell verstehen und den Job zu andere Grid Ressource migrieren kann.

Die Anforderung an das **Recovery durch dynamische Migration statt Rollback** wird dadurch erfüllt, dass der GFTES die Checkpoint-Dateien regulär abspeichert und Crash-Fall den Job-Prozess auf der neuen Ressource wiederherstellen kann.

Die Anforderung an die **fein-granulare Zustandsüberwachung durch reguläres Checkpointing** wird ebenso mit Hilfe von regulärem Speichern von Checkpoint-Dateien erfüllt, da dadurch der Ressource-Crash schnell erkannt wird.

Die Anforderung an die **Nutzung von ähnlichen und identischen Ressourcen** wird dadurch erfüllt, dass der GFTES bei Scheduling und Re-Scheduling auch die Ressourcen benutzen kann, die nicht identisch der Ressourcen aus der Job Beschreibung sind.

Die Anforderung an **unterschiedliche Resource Provider** wird mit Hilfe von der Business Logic des GFTES erfüllt, die es zulässt, bei der Auswahl von Grid Ressource, die Informationen von vielen Ressource Provider in den Auswahl einzubeziehen.
Die Anforderung an die **Unabhängigkeit von der Art der benutzten Grid Middleware** wird von GFTES dadurch erfüllt, dass GFTES eine Reihe Adapter-Klassen haben kann, die Unterschiede zwischen verschiedenen Grid Middleware Technologien verdecken.

Die Anforderung an die **Benutzerfreundlichkeit** wird durch die Implementierung der Funktionen aus der OGSA-BES Spezifikation erfüllt.

Die Anforderung an die **Performance** wird dadurch erfüllt, dass mehreren Grid Jobs parallel ausgeführt werden können.

Die Anforderung an die **Skaliebarkeit** wird dadurch erfüllt, dass GFTES die gleichzeitige Arbeit von mehreren User zulässt.

Die Anforderung an die **Sicherheit** wird dadurch erfüllt, dass nur die Arbeit von autorisierten Benutzern zugelassen ist.
Die Anforderung **Zuverlässigkeit** wird dadurch erfüllt, dass alle internen Fehler im System abgefangen werden, ohne damit die Arbeit von anderen System-Teile oder verwalteten Grid Jobs beeinflusst wird.
In dieser Weise hilft die entworfene Objektstruktur alle Anforderungen aus dem Kapitel 3.3 zu erfüllen.

Anhand dieser Informationen kann man nun die Tabelle 4.1 über Erfüllung der Anforderungen aus dem Kapitel 3.3 erweitern, wie es in der Tabelle 5.1 gezeigt ist.

Anforderungen	**Globus**	**UNICORE**	**gLite**	**GRMS**	**GFTES**
Fault Tolerance zu Ressource Crash-Fehler	✓	✓	✓	✓	✓
Recovery durch dynamische Migration statt Rollback	—	—	—	—	✓
fein-granulare Zustandsüberwachung durch reguläres Checkpointing	—	—	—	—	✓
Nutzung von ähnlichen und identischen Ressourcen	—	—	—	✓	✓
unterschiedliche Resource Provider	✓	✓	✓	✓	✓
Unabhängigkeit von der Art der benutzten Grid Middleware	—	—	—	✓	✓
Portabilität	✓	✓	—	✓	✓
Benutzerfreundlichkeit	—	✓	—	✓	✓
Skaliebarkeit	✓	✓	✓	✓	✓
Zuverlässigkeit	✓	✓	✓	✓	✓
Performance	✓	✓	✓	✓	✓
Sicherheit	✓	✓	✓	✓	✓

Tabelle 5.1.: Vergleichstabelle der Entsprechung gängigsten Arten von Grid Middleware Technologien den Anforderungen aus dem Kapitel 3.3

Im nachfolgenden Kapitel wird die prototypische Implementierung von Grid Fault Tolerant Execution Service zusammen mit dem Testbed und der Bewertung einzelner Teile beschrieben.

Teil III.

Realisierung

6. Implementierung

Dieses Kapitel beschreibt ein prototypische Implementierung von Grid Fault Tolerant Execution Service auf dem Fundament des im Kapitel 5 konstruierten Entwurfes. Man beschäftigt sich im ersten Teil des Kapitels mit der Aufbau der Testumgebung, die für die Entwicklung des GFTES benötigt wird. Anschließend werden die wichtigsten Bestandteile aus dem Quellcode genauer beschrieben. Der letzte Abschnitt ist dem Deployment des GFTES in einer Grid-Umgebung gewidmet.

6.1. Aufbau des Testbeds

Das im Kapitel 5 projektierte System wird in einer Testumgebung prototypisch realisiert. Dieses Testbed besteht aus einer künstlichen Blackbox, die das abstrakte Grid Middleware imitieren wird. Diese Blackbox wird auf einem 32-Bit System mit Ubuntu 9.04 [1] eingerichtet. Dazu wird für Ubuntu eine virtuelle Umgebung mit Hilfe von VirtualBox [2] erstellt. In der Testumgebung werden folgende Software Pakete für die Implementierung von GFTES installiert:

Software	**Version**	**Homepage**
Java Development Kit	1.6	`http://www.oracle.com/technetwork/java`
GBU Bash	3.2.48	`http://www.gnu.org/software/bash/`
Apache Ant	1.8.1	`http://ant.apache.org/`
Apache Axis2	1.5.4	`http://axis.apache.org/axis2/java/core/`
MySQL Server	5.1	`http://www.mysql.de/`
MySQL Connector Java	5.1.15	`http://dev.mysql.com/downloads/connector/j/`

Tabelle 6.1.: Benötigte Softwarepakete

Für das Testen von GFTES wird ein Web Service mit dem Name *AbstractGrid* implementiert, der das Verhalten einer abstrakten Grid Middleware simulieren kann. Im Listing 6.1 findet man das WSDL-Dokument dieses Web Services.

[1] `http://old-releases.ubuntu.com/releases/jaunty/ubuntu-9.04-desktop-i386.iso`
[2] `http://www.virtualbox.org/wiki/Downloads`

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:tns="http://testws.gftes.apps.grid.java.org/" xmlns:xsd="http:
       //www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"
        targetNamespace="http://testws.gftes.apps.grid.java.org/" name="
       StringService">
3    <types>
4      <xsd:schema>
5        <xsd:import namespace="http://testws.gftes.apps.grid.java.org/"
             schemaLocation="http://localhost:7890/AbstractGrid?xsd=1"/>
6      </xsd:schema>
7      <xsd:schema>
8        <xsd:import namespace="http://jaxb.dev.java.net/array"
             schemaLocation="http://localhost:7890/AbstractGrid?xsd=2"/>
9      </xsd:schema>
10   </types>
11   <message name="findResources">
12     <part name="parameters" element="tns:findResources"/>
13   </message>
14   <message name="findResourcesResponse">
15     <part name="parameters" element="tns:findResourcesResponse"/>
16   </message>
17   <message name="isAlive">
18     <part name="parameters" element="tns:isAlive"/>
19   </message>
20   <message name="isAliveResponse">
21     <part name="parameters" element="tns:isAliveResponse"/>
22   </message>
23   <message name="getAttributes">
24     <part name="parameters" element="tns:getAttributes"/>
25   </message>
26   <message name="getAttributesResponse">
27     <part name="parameters" element="tns:getAttributesResponse"/>
28   </message>
29   <message name="getActivityState">
30     <part name="parameters" element="tns:getActivityState"/>
31   </message>
32   <message name="getActivityStateResponse">
33     <part name="parameters" element="tns:getActivityStateResponse"/>
34   </message>
35   <message name="startActivity">
36     <part name="parameters" element="tns:startActivity"/>
37   </message>
38   <message name="startActivityResponse">
39     <part name="parameters" element="tns:startActivityResponse"/>
40   </message>
41   <message name="stopActivity">
42     <part name="parameters" element="tns:stopActivity"/>
43   </message>
44   <message name="stopActivityResponse">
45     <part name="parameters" element="tns:stopActivityResponse"/>
46   </message>
```

```
47  <message name="getCheckpointFile">
48    <part name="parameters" element="tns:getCheckpointFile"/>
49  </message>
50  <message name="getCheckpointFileResponse">
51    <part name="parameters" element="tns:getCheckpointFileResponse"/>
52  </message>
53  <portType name="StringService">
54    <operation name="findResources">
55      <input message="tns:findResources"/>
56      <output message="tns:findResourcesResponse"/>
57    </operation>
58    <operation name="isAlive">
59      <input message="tns:isAlive"/>
60      <output message="tns:isAliveResponse"/>
61    </operation>
62    <operation name="getAttributes">
63      <input message="tns:getAttributes"/>
64      <output message="tns:getAttributesResponse"/>
65    </operation>
66    <operation name="getActivityState">
67      <input message="tns:getActivityState"/>
68      <output message="tns:getActivityStateResponse"/>
69    </operation>
70    <operation name="startActivity">
71      <input message="tns:startActivity"/>
72      <output message="tns:startActivityResponse"/>
73    </operation>
74    <operation name="stopActivity">
75      <input message="tns:stopActivity"/>
76      <output message="tns:stopActivityResponse"/>
77    </operation>
78    <operation name="getCheckpointFile">
79      <input message="tns:getCheckpointFile"/>
80      <output message="tns:getCheckpointFileResponse"/>
81    </operation>
82  </portType>
83  <binding name="StringServiceSoapHttpPortBinding" type="
        tns:StringService">
84    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style
        ="document"/>
85    <operation name="findResources">
86      <soap:operation soapAction=""/>
87      <input>
88        <soap:body use="literal"/>
89      </input>
90      <output>
91        <soap:body use="literal"/>
92      </output>
93    </operation>
94    <operation name="isAlive">
95      <soap:operation soapAction=""/>
96      <input>
97        <soap:body use="literal"/>
```

```
98        </input>
99        <output>
100         <soap:body use="literal"/>
101       </output>
102     </operation>
103     <operation name="getAttributes">
104       <soap:operation soapAction=""/>
105       <input>
106         <soap:body use="literal"/>
107       </input>
108       <output>
109         <soap:body use="literal"/>
110       </output>
111     </operation>
112     <operation name="getActivityState">
113       <soap:operation soapAction=""/>
114       <input>
115         <soap:body use="literal"/>
116       </input>
117       <output>
118         <soap:body use="literal"/>
119       </output>
120     </operation>
121     <operation name="startActivity">
122       <soap:operation soapAction=""/>
123       <input>
124         <soap:body use="literal"/>
125       </input>
126       <output>
127         <soap:body use="literal"/>
128       </output>
129     </operation>
130     <operation name="stopActivity">
131       <soap:operation soapAction=""/>
132       <input>
133         <soap:body use="literal"/>
134       </input>
135       <output>
136         <soap:body use="literal"/>
137       </output>
138     </operation>
139     <operation name="getCheckpointFile">
140       <soap:operation soapAction=""/>
141       <input>
142         <soap:body use="literal"/>
143       </input>
144       <output>
145         <soap:body use="literal"/>
146       </output>
147     </operation>
148   </binding>
149   <service name="StringService">
150     <port name="StringServiceSoapHttpPort" binding="
```

```
        tns:StringServiceSoapHttpPortBinding">
151       <soap:address location="http://localhost:7890/AbstractGrid"/>
152     </port>
153   </service>
154 </definitions>
```

Listing 6.1: Web Service für Grid Middleware Simulation

Die wichtigsten Funktionen von *AbstractGrid* sind: `isAlive` – Prüfung, ob die Ressource noch funktioniert, `getActivityState` – Abfrage des Status von einem Job, `startActivity` – für Erstellung neuer Jobs, `stopActivity` – Abbruch von einem Job, `getCheckpointFile` – damit bekommt man die Adresse der Checkpoint-Datei, `getAttributes` – liefert alle Attribute des AbstractGrids und `findResources` – Suche nach Ressource die einer Beschreibung genügen.

Als Test-Aufgabe wird das Java Programm für die Berechnung des *N*-ten Mitglieds der Fibonacci-Folge. Das Programm speichert periodisch die Werte von *N-1*-tem und *N-2*-tem Mitglied in einer Checkpoint-Datei mit dem Namen `checkpoint.xml` und kann dementsprechend seine Zustände aus dieser Datei wiederherzustellen. In der Datei `input.xml` wird die Nummer des Mitglieds von Fibonacci-Folge angegeben, der berechnet werden soll. Das Ergebnis wird in die Datei `output.xml` gespeichert. Die Job Beschreibung für die Berechnung einer Fibonacci-Zahl findet man im Listing 6.2.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <jsdl:JobDefinition
3          xmlns="http://www.example.org/"
4          xmlns:jsdl="http://localhost/jsdl/2011/01/jsdl"
5    xsi:schemaLocation="http://localhost/jsdl/">
6    <jsdl:JobIdentification>
7      <jsdl:JobName>
8        Calculation of the Fibonacci Number
9      </jsdl:JobName>
10     <jsdl:Description>
11       Simple application invocation:
12       User wants to run the java programm 'fibo.class'
13       to get a Fibonacci number.
14
15       This Fibonacci number will be stored to 'output.xml' file.
16       All staged-in data will be copied to working directory.
17
18       In case of error, message schould be produced to the same
19       'output.xml' file.
20     </jsdl:Description>
21   </jsdl:JobIdentification>
22   <jsdl:Resource>
23     <jsdl:CPUCount>2</jsdl:CPUCount>
24     <jsdl:PhysicalMemory>20971520</jsdl:PhysicalMemory>
25   </jsdl:Resource>
26   <jsdl:SoftwareRequirements>
27     <SoftwareDescription>Java</SoftwareDescription>
28     <SoftwareVersion>1.6</SoftwareVersion>
```

```
29    </jsdl:SoftwareRequirements>
30    <jsdl:Application>
31      <jsdl:ExecutableName>
32        java
33      </jsdl:ExecutableName>
34      <jsdl:Argument>fibo</jsdl:Argument>
35      <jsdl:WorkingDirectory>
36        /sandbox/test/files/temporary
37      </jsdl:WorkingDirectory>
38      <jsdl:InCaseOfResourceCrash>recovery</jsdl:InCaseOfResourceCrash>
39      <jsdl:CheckpointName>checkpoint.xml</jsdl:CheckpointName>
40    </jsdl:Application>
41    <jsdl:DataAttributes>
42      <jsdl:File>
43        <jsdl:FileName>fibo.class</jsdl:FileName>
44        <jsdl:Source>
45          /sandbox/test/files/fibo.class
46        </jsdl:Source>
47        <jsdl:DeleteOnTermination/>
48      </jsdl:File>
49      <jsdl:File>
50        <jsdl:FileName>input.xml</jsdl:FileName>
51        <jsdl:Source>
52          /sandbox/test/files/input.xml
53        </jsdl:Source>
54        <jsdl:DeleteOnTermination/>
55      </jsdl:File>
56      <jsdl:File>
57        <jsdl:FileName>output.xml</jsdl:FileName>
58        <jsdl:Source>
59          /sandbox/test/files/output.xml
60        </jsdl:Source>
61        <jsdl:DeleteOnTermination/>
62      </jsdl:File>
63      <jsdl:File>
64        <jsdl:FileName>checkpoint.xml</jsdl:FileName>
65        <jsdl:Source>
66          /sandbox/test/files/checkpoint.xml
67        </jsdl:Source>
68        <jsdl:DeleteOnTermination/>
69      </jsdl:File>
70    </jsdl:DataAttributes>
71  </jsdl:JobDefinition>
```

Listing 6.2: Job Beschreibung für die Berechnung von Fibonacci Zahl

Im Deploymentdiagramm 6.1 sind die wichtigsten Teile des Aufbaues der Testumgebung dargestellt. Im folgenden werden alle Schritte des Testablaufs aufgelistet:

1. GFTES Client reicht Job Beschreibung (Listing 6.2) mit dem Job ein, der das Java Programm `fibo.class` mit der Input-Datei `input.xml` auf der Grid Ressource auszuführen soll.

2. Für das Testen der Funktionalität von GFTES wird der Web Service `AbstractGrid` benutzt (WSDL-Dokument kann man in Listing 6.1 finden). Die zwei Instanzen dieses Service werden lokal auf zwei unterschiedlichen Ports gestartet. Für die Konfiguration von `AbstractGrid` ist die Datei `config.xml` zuständig, die jewels bei dem Servicestart als Argument übergeben wird.

3. GFTES nimmt die Job Beschreibung entgegen und sucht bei `AbstractGrid` nach einer Ressource, die zur Beschreibung am besten passt. Wenn die Ressource gefunden wurde, erstellt GFTES eine Job Beschreibung und schickt diese an eine der `AbstractGrid`-Einheiten. Dort wird das Java Programm `fibo.class` ausgeführt.

4. Während der Ausführung speichert das Java Programm regulär seine Zustände in der Datei `checkpoint.xml`. In `AbstractGrid` ist ein Mechanismus eingebaut, der den Ausfall der Grid Ressource simulieren soll. Dafür wird mit Hilfe eines Zufallsgenerators eine Zeit ausgewählt, in der der Web Service für eine gewisse Zeit keine Antwort auf die ankommenden Anfragen gibt. Alternativ wird der Web Service manuell abgeschaltet.

5. Die Checkpoint-Datei wird von GFTES periodisch in der Checkpoint-Datenbank gespeichert.

6. Wenn GFTES merkt, dass eine der Ressourcen ausgefallen ist, versucht er den gleichen Job auf der anderen `AbstractGrid` Ressource zu starten. Das Programm `fibo.class` versucht jeweils seinen letzt Zustand aus der letzten Checkpoint-Datei wiederherzustellen.

Dieser Test soll übeprüfen wie GFTES den funktionalen Anforderungen aus dem Kapitel 3.3 entspricht und auf die Crash-Fehler von Grid Ressource reagiert.

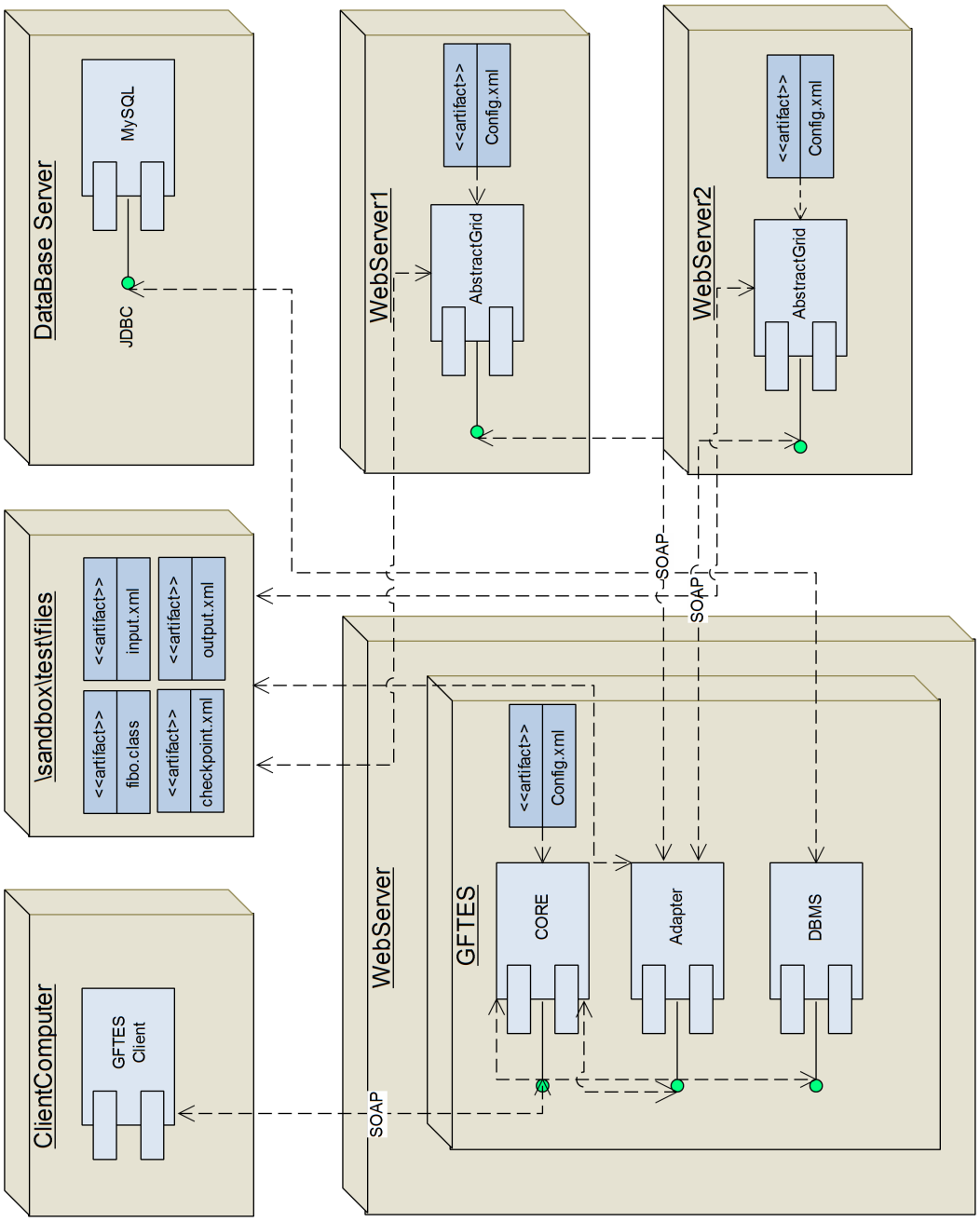

Abbildung 6.1.: Deployment Diagram: Aufbau der Testumgebung für GFTES

6.2. Grid Fault Tolerant Execution Service

In diesem Abschnitt werden die wichtigsten Teile der prototypischen Implementierung von GFTES beschrieben. Zu diesen Mechanismen zählt man die Erkennung der Informationen aus der Job Beschreibung, Speicherung der Informationen in der Datenbank, Suche nach einer passenden Ressource etc. Gleichzeitig wird beschrieben, welche Anforderungen aus dem Kapitel 3.3 mit diesem Teil erfüllt werden.

6.2.1. Datenbankmanagementsystem (DBMS)

Für seine Arbeit braucht der Grid Fault Tolerant Execution Service sämtliche gespeicherte Informationen über die Grid Ressources, Users, Grid Aktivitäten etc. Deswegen wurde man im Rahmen der prototypischen Implementierung des Systems den **MySQL-Server 5.1.15** für die Verwaltung alle diesen Informationen ausgewählt. Eine der nicht-funktionalen Anforderungen – **Portabilität** bedeutet, dass der GFTES sowohl unter anderen Betriebssystemen installiert werden kann, als auch für die Arbeit mit anderen Datenbanken geeignet ist. Um die Arbeit mit unterschiedlichen Datenbanksystemen zu ermöglichen, wurde im Rahmen dieser Arbeit das System von Interfaces entwickelt, die von Klassen aus dem Package `DBMS` implementiert werden müssen. Diese Interface ermöglichen den Klassen aus den anderen GFTES Packages, die Informationen aus dem Datenbanksystem unabhängig von der Art der Datenbank zu bekommen, zu speichern und zu verwalten. Man kann MySQL durch den Datenbank-Web Service ersetzen, ohne damit die anderen Teile von GFTES außerhalb `gftes.dbms` Package geändert werden sollen.

Da MySQL die gleichzeitige Arbeit von mehreren Benutzer unterstützt, trägt der Auswahl gerade dieser Datenbank bei, um die Anforderungen an **Skalierbarkeit** und **Performance** zu erfüllen. Mit diesem Datenbanksystem können mehreren Instanzen von `ActivityManager`, die Informationen aus der Datenbank gleichzeitig abzufragen und zu verwalten.

Alle von GFTES benötigten Informationen in der Datenbank werden nach der Datenbankschema 5.3 gespeichert. Für den Zugang zu dieser Informationen sowie für die Aktualisierung alter oder das Einfügen neuer Informationen wurden im Package `gftes.dbms` die folgenden Klassen entwickelt: `Endpoint`, `GridService`, `GridActivity`, `Queue`, `GridActivityResource`, `User`. Jede dieser Klassen implementiert gleichzeitig zwei Interfaces: den eigenen Interface, wie z.B. `UserInterface`, in dem die allgemeinen Operationen für Verwaltung der Attribute aus der entsprechenden Datenbank-Tabelle definiert sind, und den `Recoverable`-Interface (siehe Listing 6.3), in dem die Operationen für Speichern und für das Zurückbekommen ebenso wie für das Updaten von Informationen in der Datenbank definiert sind.

```
1 package gftes.dbms;
2
3 public interface Recoverable {
4         public void make(Object object);
5
6         public void recover(String id);
7
8         public void remove();
9 }
```

Listing 6.3: `Recoverable.java`

Zusätzlich wurde für Erstellung einer Klasse aus dem Package `gftes.dbms` ein Factory-Pattern angewendet, wie im Listing 6.4 gezeigt ist. Für Erzeugung eines Objeks von der Klasse aus dem Package `gftes.dbms` wird die Methode `getEntity()` mit der Eingabe des Typs von Inteface als Argument aufgerufen. Alle diese Interface sind im Enum Type mit dem Namen `InterfaceType` (Listing 6.5) festgeschrieben.

```
1 package gftes.dbms;
2
3 public class Factory {
4         public static Object getEntity(InterfaceType interfaceType){
5                 switch(interfaceType){
6                         case GridServiceInterface:
7                                 return new GridService();
8                         case EndpointInterface:
9                                 return new Endpoint();
10                        case    QueueInterface:
11                                return new Queue();
12                        case    GridActivityInterface:
13                                return new GridActivity();
14                        case    UserInterface:
15                                return new User();
16                        case    GridActivityResourceInterface:
17                                return new GridActivityResource();
18                        default: return null;
19                }
20        }
21 }
```

Listing 6.4: `Factory.java`

Das im Rahmen der prototypischen Implementierung von GFTES entwickelte Datenbank-Schema (SQL Skript dafür ist im Appendix A beigelegt) ist die Realisierung des Datenbankmodells, das im Diagramm 5.2 präsentiert ist. Dieses Datenbankmodell wurde auf der Basis von GLUE v. 2.0 Modell entwickelt. Jeder der mehrwertigen Attributen (mehr dazu findet man in GLUE Spezifikation (1)) bakommt in der GFTES Datenbank-Scheme eine eigene Tabelle mit Verlinkung auf die Haupttabelle per `id`.

```
1  package gftes.dbms;
2
3  /**
4   * This Enum Type describe all possible interface in
5   * GFTES DBMS Package
6   * @author Vitaly Hachuzky
7   *
8   */
9  public enum InterfaceType {
10          GridServiceInterface,
11          EndpointInterface,
12          QueueInterface,
13          GridActivityInterface,
14          UserInterface,
15          GridActivityResourceInterface,
16          EntitiyInterface
17 }
```

Listing 6.5: `InterfaceType.java`

Der Zugang zu den Informationen aus jeder Tabelle wird durch die entsprechenden get und set Methoden geleistet. So, wenn man z.B. durch einen `id` auf das Attribut `password` in der Tabelle `User` zugreifen möchte, muss man zuerst die `recover()` Methode aus der Klasse `User` aufrufen und erst danach mit Hilfe der Methoden `getPassword()` und `setPassword()` auf den Wert zugreifen. Wenn man den wert zurück in der Datenbank speichern möchte, muss man die Methode `make()` mit Parameter `null` aufrufen.

Die Methode `make()` dient auch dafür, um alle benötigen Informationen aus der Job Beschreibung zu erkennen und in der Datenbank zu speichern. Beim Aufruf dieser Methode aus der Klasse `GridActivity` mit Angebe von Job Description Dokument als Parameter werden die entsprechenden Informationen aufgefasst und automatisch in der Datenbank gespeichert.

6.2.2. Repository für Checkpoint-Dateien

Bei der prototypischen Realisierung einer der funktionalen Anforderungen, der Anforderung an die fein-granularer Zustandsüberwachung durch reguläres Checkpointing, stößt man auf ein Problem beim Speichern von Checkpoint Dateien. Dieses Problem wurde dadurch gelöst, dass man in der `config.xml` Datei (ein Beispiel dieser Datei findet man im Listing 6.6) einen Pfad zu dem Repository-Verzeichnis auf der Festplatte angibt. Dort werden die Checkpoint-Dateien gespeichert. In der gleichen Datei `config.xml` werden auch alle benötigen Informationen für den Zugang zur MySQL-Datenbank (wie z.B. Login, Password, URI, etc.) angegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
        <Database>
                <Host>localhost</Host>
                <Port>3306</Port>
                <DBName>gftesdbo</DBName>
                <Login>gftes</Login>
                <Password>123a456b</Password>
        </Database>
        <ActivityBroker>
                <TimeUnit>10</TimeUnit>
        </ActivityBroker>
        <Checkpoint>
                <RepositoryPath>
                /sandbox/cpRep
                </RepositoryPath>
        </Checkpoint>
</Configuration>
```

Listing 6.6: `config.xml`

6.2.3. Adapter

Die Aufgabe der Klassen aus dem Package `gftes.adapter` ist es die Routine für die Kommunikation von GFTES mit anderen Grid Middleware Technologien zu verbergen. Hier geht es an erste Stelle um die Klassen aus dem Package `gftes.core`, die für Business Logic in Grid Fault Tolerant Execution Service zuständig sind.

```
package gftes.adapter;

import gftes.dbms.GridActivityInterface;
import java.util.*;

public interface AdapterInterface {

        //get actual state of grid job
        public String getState(String url, String id)
        throws Exception;

        //start new grid job
        public String startActivity(String url, String
           activityDescription)
        throws Exception;

        //stop grid job
        public Boolean stopActivity(String url, String id)
        throws Exception;

        //find resource for grid job
        public Hashtable[] findResource(String url, String parameters)
        throws Exception;
```

```
23
24          //create Job Description Document
25          public String createActivityDescription(GridActivityInterface
                activity)
26          throws Exception;
27
28          //find file on the foreign resource
29          public String getFilePath(String ulr, String id, String fileName
                )
30          throws Exception;
31
32          //make a copy of file from foreign resource to new location
33          public void copyFile (String src, String destination)
34          throws Exception;
35  }
```

Listing 6.7: `AdapterInterface.java`

Jede Adapter-Klasse implementiert die Methoden aus dem Inteface `AdapterInterface`, welches im Listing 6.7 gezeigt ist. Damit man möglich abstrahiert von konkreten Grid Middleware die Konzepte von Grid Fault Tolerant Execution Service im Rahmen prototypische Implementierung testen kann, wurde es, wie im Kapitel 6.1 schon mal erwähnt wurde, ein Web Service mit dem Namen `AbstractGrid` konstruiert, der ein Verhalten von Grid Middleware simulieren sollte. Deswegen wurden die aktuelle Klasse `Adapter` gezielt für die Arbeit mit `AbstractGrid` programmiert. Man kann aber leicht eine eigene `Adapter` Klasse für die Verständigung mit einer anderen Grid Middleware, wie z.B. Globus Toolkit, gLite, UNICORE etc., implementierten z.B. einen **GAT** [3] Framework (Grid Application Toolkit) verwendet. Grid Application Toolkit ist eine Bibliothek von Plug-ins für die Kommunikation mit verschiedenen Web-Services diverser Grid Middleware Technologien.

Durch die Implementierung der Methoden aus `AdapterInterface` wird nun vom Package `gftes.adapter` eine der Funktionale Anforderungen, und zwar **verschiedener Grid Middleware** erfüllt, da GFTES nun mit unterschiedlicher Grid Middleware Arten kommunizieren kann.

Dabei wird auch die Anforderung an die **Kommunikation mit unterschiedlichen Resource Providern** erfüllt, falls in der Datenbank mehr als nur ein Service Provider eingetragen ist. So wurde es in unserem Testbed gemacht, wo zwei Instanzen von Web Service `AbstractGrid` zwar lokal aber auf unterschiedlicher Ports gestartet werden, um unterschiedliche Provider zu simulieren. Jede dieser Instanzen liefert eine eigene Beschreibung von verfügbaren Ressourcen. Nun können die eingebaute Mechanismen von GFTES für Bewertung von verschiedenen verfügbaren Ressourcen getestet werden. Somit wird auch die Anforderung nach die **Nutzung von ähnlichen und identischen Ressourcen** realisiert.

Die Methoden `getFilePath()` und `copyFile()` aus der Klasse `Adapter` bilden die Grundlage für die Realisierung zwei anderen Anforderungen. Mit Hilfe von diesen Methoden kann man die Checkpoint-Dateien nun finden und in Repository speichern. In dieser Weise wer-

[3] Mehr über GAT-Projekt erfährt man unter dieser Web-Adresse `http://www.gridlab.org/WorkPackages/wp-1/`

den die Anforderung an die **fein-granulare Zustandsüberwachung durch reguläres Checkpointing** erfüllt.

6.2.4. Core

Die Klassen aus dem Package `gftes.core` bilden die Business Logic von GFTES. Im folgenden werden die Funktionen jeder einzelner Klasse genauer erklärt.

Die Klasse `Intermediary` ist das Frontend des GFTES und bildet nach OGSA-BES Spezifikation (9) ein **benutzerfreundliches** (eine der nicht-funktionale Anforderungen aus dem Kapitel 3.3) Interface von Grid Fault Tolerant Execution Service. Das WSDL-Dokument von GFTES findet man im Appendix B. Das Interface bietet die notwendigen Operationen, um einen Grid Job zu starten oder zu stoppen, den Status des Grid Jobs abzufragen etc.

Eine der nich-funkionalen Anforderungen ist die **Sicherheit** von GFTES. Das bedeutet, dass nur die User, die für die Nutzung von GFTES berechtigt sind, auf seine Funktionen zugreifen dürfen. Deswegen wurde in der Datenbank eine spezielle Tabelle `User` angelegt, in der jedem Benutzer ein Password zugeteilt wird. Die Autorisierung läuft dann wie folgt ab: wenn der User auf eine Methode aus der Klasse `Intermediary` zugreifen möchte, muss er zuerst seine Session durch Aufruf der Methode `getAuthentication()` mit Angabe von Login und Passwort als Parameter initialisieren. Die Angaben werden vom `CredentialManager` überprüft und, falls diese richtig sind, wird ein Zugang zu der Klasse `ActivityBroker` gewährt.

`ActivityBroker` ist eine Singleton-Klasse. Sie wird im eigenen Thread gestartet und trägt die zwei wichtigsten Funktionen. Die erste Funktion ist es, dass wenn ein neuer Grid Job ankommt, wird vom `ActivityBroker` ein neuer `AcitivityManager` erstellt, um die Grid Job Informationen in der Datenbank zu speichern und in die Queue einzufügen. Die zweite Funktion vom `ActivityBroker` ist die Aktivierung von wartenden Jobs aus der Queue. Falls ein wartender Job gefunden wird, startet `ActivityBroker` für ihn einen neuen `ActivityManager` in einem neuen Thread. Dadurch wird eine höhere Stufe von **Skalierbarkeit** und **Performance** erreicht und in dieser Weise zwei der wichtigsten nicht-funktionalen Anforderungen erfüllt.

Wie im vorherigen Absatz bereits erwähnt wurde, bekommt jeder Grid Job einen eigenen `ActivityManager`, der in eigenem Thread gestartet wird. Gerade diese Klasse ist für die Erfüllung der restlichen funktionalen Anforderungen zuständig. Mithilfe vom `Adapter` wird die komplette Kommunikation mit den anderen Grid Diensten durchgeführt.
Eine der funktionalen Anforderungen, für die der `ActivityManager` zuständig ist, ist die **Nutzung nicht nur identischer Ressourcen**, die in der Job Beschreibung gewünscht sind, **sondern auch ähnlicher Ressourcen**. Die Daten über die verfügbaren Ressourcen, die z.B. ein gleiche Betriebsystem besitzen, werden von der Grid Middleware geliefert. Die Aufgaben von dem `ActivityManager` ist es diese Angaben zu bewerten, um die best mögliche Ressource zu finden. Dafür wurde in der Klasse `ActivityManager` die Funktion `evaluate()` implementiert. Als theoretische Grundlage für die Bewertung der verfügbaren Ressource wurde das Formel 6.1 genommen, die von Kurowki in et al. (14) vorgeschlagen wurde.

$$f(\overline{C}) = \frac{1}{\sum_{i=1}^{n} w_i} w_i^* C_i \tag{6.1}$$

mit n - die Anzahl von bewerteten Kriterien, C - ist der Kriterienvektor und w - Vektor der Gewichte von Kriterien, um die Wichtigkeit verschiedener Kriterien zu unterscheiden.

Im Rahmen der prototypischen Implementierung wurden als Kriterien in dieser Arbeit die folgenden Ressource-Parameter genommen: **physischer Speicher**, **CPU-Frequenz** und **Anzahl von CPU**. Die Gewichtung dieser Parameter wurde aus der experimentellen Arbeit von Grid(Lab)-Team (14) übernommen: der physische Speicher hat das Gewicht 3, die Anzahl der CPU ist mit dem 1 gewichtet und die CPU-Frequenz ist auch mit 1 gewichtet. Die Bewertung von anderen Parameter ist auch möglich, die wird aber im Rahmen dieser Arbeit nicht realisiert.

Nach der Bewertung von allen Informationen über die verfügbaren Grid Ressourcen, wird die best passende Ressource ausgewählt. Der Job kann nun auf dieser Ressource gestartet werden. Der Objekt `Adapter` besitzt die spezielle Funktion, um auf der Basis der ausgewählten Grid Ressource und der Informationen aus der Job Beschreibung das passende Job Description Document zusammenzustellen und dem `AbstractGrid` zu übergeben.

Als ein Beweis der erfolgreichen Übergabe des Jobs bekommt der GFTES von `AbstractGrid` den Lokalen Job ID. Dieser Job ID wird in der Datenbank gespeichert und wird im weiteren angewendet, um die Informationen zu der Jobausführung zu bekommen. Das betrifft an erster Stelle die Information über den aktuellen Jobzustand. Nachdem der Job auf der Grid Ressource gestartet wurde, startet der `ActivityManager` die Monitoring-Routine, die periodisch (in diesem Prototyp alle 5 Minuten) eine Abfrage des Jobzustands und neuer Checkpoint-Datei sendet (nur falls der Name dieser Dateien in der Job Beschreibung angegeben ist). Wenn eine neue Checkpoint-Datei gefunden wird, wird sie im Repository gespeichert.

Durch die regulären Abfragen wird die funktionale Anforderung an die **fein-granulare Zustandsüberwachung durch reguläres Checkpointing** erfüllt, sowie die Grundlage für Erfüllung zwei anderen Anforderungen aufgebaut. Hier geht es um die Anforderung an die **Fault Tolerance im Fall eines Crash-Fehler** und an das **Recovery durch dynamische Migration statt Rollback**. Falls der Adapter keine Rückmeldung erhält, bedeutet das ein Ressource-Fault. In diesem Fall startet der `ActivityManager` die Suche nach die passende Ressource noch mal, um den Job auf der anderen Stelle fortzusetzen. Da aber in der Datenbank nun auch die Checkpoint-Datei registriert ist, wird der Adapter in der Job Beschreibung diese Checkpoint-Datei als eine der Input-Dateien angeben. Diese Datei wird dann zusammen mit anderen Input-Dateien in das Arbeitsverzeichnis vom `AbstractGrid` kopiert und die Job-Anwendung kann nun aus dieser Checkpoint-Datei seinen letzten Zustand wiederherstellen und den Durchlauf fortsetzen. Selbstverständlich wird ohne der Checkpoint-Datei keine Wiederherstellung sondern Neustart des Jobs bei dem `AbstractGrid` durchgeführt. Nach der Wiederherstellung setzt der `ActivityManager` seine Überwachung weiter fort.

Durch die flexible Struktur von GFTES werden die restlichen der funktionalen Anforderungen erfüllt. Das gleiche betrifft auch die letzten nicht-funktionalen Anforderung aus dem ISO/IEC 9126-Standart (11). Die höhere Stufe von `Portabilität` wird dadurch erreicht, dass GFTES im Rahmen dieser Arbeit als ein Web Service prototypisch realisiert wurde und kann auf jedem Betriebsystem mit dem AXIS2-Server gestartet werden. Außerdem kann er die verschiedenen Datenbanken und die verschiedenen Adapter-Systemen benutzen. Die `Zuverlässigkeit` wird dadurch erreicht, das die meisten möglichen Fehler und Ausnahmen im Code abgefangen werden, ohne damit die Arbeit von GFTES damit gestört wird und der Service ausfällt. Die speziellen Stress-Tests von GFTES werden aber im Rahmen der prototypischen Realisierung nicht durchgeführt.

6.3. Ergebnisse aus dem Testlauf

Als Test von GFTES wurde der Job für die Berechnung von dem 300-ten Mitglied der Fibonacci-Folge genommen. Die Input-Datei `input.xml` findet man im Listing 6.8.

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <root>
3   <sequenceNumber>
4     300
5   </sequenceNumber>
6 </root>
```

Listing 6.8: `input.xml`

Die Job Beschreibung für die Berechnung von 300-te Fibonacci-Zahl findet man im Listing 6.2. Im Laufe der Ausführung wurde von Job Programm periodisch die Checkpoint-Datei erzeugt, die von GFTES in Repository gespeichert wurden. Ein Beispiel solcher Checkpoint-Datei kann man im Listing 6.9 sehen.

```
1 <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
2 <checkpoint time="28.02.2011␣12:24:09">
3     <counter>3.54224848179262E20</counter>
4     <currentFib>3.54224848179262E20</currentFib>
5     <nextFib>5.731478440138172E20</nextFib>
6     <zaehler>100.0</zaehler>
7     <i>100</i>
8 </checkpoint>
```

Listing 6.9: `checkpoint.xml`

Während der Ausführung vom Job wurden die `AbstractGrid` Web Services periodisch gestoppt, um einen Crash von Grid Ressourcen zu simulieren. Jedes Mal wurde der Job zu einer anderen Grid Ressource migriert bis der Job erfolgreich beendet wurde. Das Ergebnis der Ausführung von Job ist die Datei `output.xml`, die im Listing 6.10 dargestellt ist.

```
1 <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
2 <fibonaccy>
3     <number>300.0</number>
4     <value>2.2223224462942035E62</value>
5 </fibonaccy>
```

Listing 6.10: `output.xml`

6.4. Deployment von GFTES

In dieser Sektion wird die Installation von dem Grid Fault Tolerant Execution Service schrittweise beschrieben.

1. Als Erstes installiert man alle benötigen Software Komponenten. Dazu zählt man Java, Ant, Apache Axis2 Server, MySQL Server und MySQL Connector Java Bibliothek. Die Installationsanleitungen findet man auf den entsprechenden Projekt Web-Seiten die in der Tabelle 6.1 dargestellt sind.

2. Der nächste Schritt ist es die entsprechenden Umgebungsvariablen JAVA_HOME, ANT_HOME, AXIS2_HOME zu setzen, die auf die Verzeichnisse zeigen, wo die entsprechende Software installiert wurde. Diese Prozedur ist von Betriebsystem zu Betriebsystem unterschiedlich. Unter Ubuntu kann man diese Variablen in der Datei `.bashrc` im Home-Verzeichnis des Benutzers initialisieren.

3. Nun legt man alle für die Arbeit von GFTES benötigen SQL-Tabellen an. Der Skript dafür findet man im Appendix A.

4. Der weitere Schritt ist es im Home-Verzeichnis des Benutzers ein neues Verzeichnis mit dem Namen `gftes/` anzulegen und die Datei `config.xml` (siehe Listing6.6) mit den Einstellungen für den Zugang zur Datenbank in dieses Verzeichnis hinein zu kopieren. Diese Einstellungen müssen selbstverständlich an die Zugangsdaten der Datenbank angepasst werden.

5. Als Nächstes setzt man im CLASSPATH die lokalen Pfade aller Bibliotheken, die für die Arbeit von GFTES benötigt werden. Das sind: der ODBC Konnektor für Java `mysql-connector-java-5.1.15-bin.jar`, und die Dateien `activation-1.1.jar`, `axiom-api-1.2.8.jar`, `axiom-dom-1.2.8.jar`, `axiom-impl-1.2.8.jar`, `axis2-adb-1.5.1.jar`, `axis2-kernel-1.5.1.jar`, `axis2-transport-http-1.5.1.jar`, `axis2-transport-local-1.5.1.jar`, `commons-codec-1.3.jar`, `wsdl4j-1.6.2.jar`, `commons-fileupload-1.2.jar`, `commons-httpclient-3.1.jar`, `wstx-asl-3.2.4.jar`, `commons-logging-1.1.1.jar`, `geronimo-stax-api_1.0_spec-1.0.1.jar`, `httpcore-4.0.jar`, `mail-1.4.jar`, `neethi-2.0.4.jar`, `woden-api-1.0M8.jar`, `woden-impl-dom-1.0M8.jar`, `XmlSchema-1.4.3.jar` die im Verzeichnis `$AXIS2_HOME/lib/` gefunden werden können. Diese Bibliotheken werden beim Kompilieren benötigt. Sie können auf der beigelegten CD im Verzeichnis `lib/` gefunden werden.

6. Als Nächste muss das Projekt kompiliert und in eine `gftes.aar` Datei verpackt werden. Die Datei `gftes.aar` ist im Grunde die gleiche Datei wie jede `.jar`-Datei, sie hat aber eine Namenserweiterung. Während dieser Arbeit wurde zwar die Entwicklungsumgebung **Eclipse** [4] verwendet, in der die Funktionen zum Kompilieren von Java-Dateien sowie für Erzeugung von `.jar`-Dateien vorhanden sind, man kann aber einen `build.xml`-Skript aus dem Verzeichnis `ant/` verwenden, die zusammen mit dem `ant`-Programm zum gleichen Zweck verwendet werden kann. Wichtig ist aber, dass in `gftes.aar` das Verzeichnis `META-INF` mit der Datei `services.xml` vorhanden ist, die vom Axis-Server benötigt wird. Der Inhalt dieser Datei findet man im Listing 6.11.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <service name="gftes" scope="application" targetNamespace="http://gftes/
       ">
3      <messageReceivers>
4          <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
5                           class="org.apache.axis2.rpc.receivers.
                                RPCInOnlyMessageReceiver"/>
6          <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
7                           class="org.apache.axis2.rpc.receivers.
                                RPCMessageReceiver"/>
8      </messageReceivers>
9      <schema schemaNamespace="http://gftes/xsd"/>
10     <parameter name="ServiceClass">gftes.core.Intermediary</parameter>
11 </service>
```

Listing 6.11: `META-INF/services.xml`

7. Sobald die Datei `gftes.aar` vorhanden ist, wird diese Datei in das Verzeichnis `$AXIS2_HOME/repository/services` kopiert und der AXIS-Server kann nun gestartet werden.

[4] `http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliossr2`

7. Zusammenfassung und Ausblick

7.1. Zusammenfassung

Fehlertoleranz ist eine der wichtigsten Eigenschaften beim Entwurf von Applikationen, die Vorteile von Grid Computing nutzen möchten, wo die Zuverlässigkeit einzelner Grid Ressourcen nicht garantiert werden kann. Die Größe und Komplexität solcher Applikationen im Vergleich zu anderen Programmen steigt dadurch enorm. Die Wahrscheinlichkeit eines Fehlers oder eines Ressource Crashes ist besonders hoch, wenn man in Betracht zieht, dass die Berechnungen einiger Grid Jobs über mehrere Tage hindurch laufen können. Heterogene Natur von Grid Nodes bedeutet, dass viele Grid Applikationen in der Grid Umgebung funktionieren, wo Kommunikationsfehler zwischen zwei Nodes potenziell sehr wahrscheinlich sind. Zugleich bedeutet die dynamische Natur von Grid (die Ressourcen können jeder Zeit kommen und gehen), dass Grid Applikationen einen Bedarf an Fähigkeit haben, diese transiente Verfügbarkeit von Grid Ressourcen zu tolerieren. Die verschiedenen Arten von modernen Grid Middleware (wie Globus Toolkit, gLite, UNICORE) weisen zur Zeit keine einheitlichen Standarten von Fehlertoleranz auf und gehen mit Ressource Fault sehr unterschiedlich um. Diese Diplomarbeit beschäftigt sich mit dem Entwurf eines Grid Fault Tolerant Execution Services, der für Grid Jobs eine unifizierte Fault Tolerante Executionstelle bilden soll, unabhängig von der Grid Middleware Technologien, die von jedem einzelnen Resource Provider benutzt wird.

Zunächst wurde im Kapitel 2 die für diese Arbeit relevanten Grundbegriffe wie *Grid Computing*, *Fault*, *Fault Tolerance*, *Grid Execution Service* und *Prozessmigration* vorgestellt. Im Kapitel 3 hat man anhand der Aufgabenstellung die verschiedenen Szenarien aufgebaut. Auf dem Grund dieser wahrscheinlichen Szenarien wurden Use Cases identifiziert, die im nächsten Schritt geholfen haben, die Anforderungen an den Grid Fault Tolerant Execution Service herauszuarbeiten. Zusätzlich zu den funktionalen Anforderungen wurde noch eine Reihe von nicht-funkionalen Anforderungen nach den internationalen ISO 9126-1 Standard (11) ermittelt. Im nachfolgenden Kapitel 4 wurden die bereits existierenden Grid Middleware Projekte auf ihre taugliche Beschaffenheit, im Bezug auf die soeben ermittelten Anforderungen, hin untersucht. Dabei wurde gezeigt, dass alle diese Projekte mindestens eine Anforderung nicht erfüllen und somit es ein Bedarf an die Entwicklung des neuen Grid Fault Tolerant Execution Services besteht.

Dieser Service wurde aus der definierten Anforderungen im Kapitel 5 entwickelt. Eine besondere Aufmerksamkeit bei dem Entwurf des GFTES war der Anpassbarkeit des System an neue Grid Middleware Arten geschenkt. Auf dem Basis GLUE v. 2.0 Spezifikation (1) wurde ein Datenbankschema herausgearbeitet, das für Speichern von relevanten Informationen über Grid Jobs, User und Grid Diensten fähig ist. Anschließend wurde im Kapitel 6 ein Testbed präsentiert, in dem der Grid Fault Tolerant Execution Service prototypisch realisiert und getestet wurde. Im Kapitel 6.2 wurden die wichtigsten Teile des Quelle-Codes

dargestellt. Die Funktionen und Fähigkeiten verschiedener Teile von GFTES wurden anhand der Erfüllung von Anforderungen aus dem Kapitel 3.3 erklärt. Der komplette Code von Grid Fault Tolerant Execution Service kann man auf der beigelegten CD finden.

7.2. Ausblick

Die Zielsetzung dieser Arbeit war die Entwicklung eines Grid Fault Tolerant Execution Service unter der Voraussetzung, dass keine der bereits existierenden Grid Middleware Technologien allen der gestellten Anforderungen entspricht. Der in dieser Diplomarbeit entwickelte Grid Fault Tolerant Execution Service bietet seinem Benutzer eine Möglichkeit, einen Grid Job auf den Grid Ressourcen verschiedener Resource Provider zu starten, sogar wenn diese Resource Provider heterogene Grid Middleware benutzen. Spezielle viel Aufmerksamkeit wurde der fein-granulierten Überwachung der Grid Jobs durch reguläres Checkpointing sowie der Nutzung von Ressourcen geschenkt, die von der Beschreibung vom Grid Job abweichen können.

Im Laufe dieser Diplomarbeit wurden neben den Kernfunktionen auch einige zusätzliche Erweiterungen erkannt. Die Entwicklung solcher Erweiterungen würde einen Wertzuwachs für Grid Fault Tolerant Execution Service bilden und könnte die Aussichten von GFTES auf die Anwendung im professionellen Grid Umfeld verbessern. Folgende Ergänzungen wären dabei sinnvoll:

- Entwicklung einer benutzerfreundlichen GUI für die Arbeit mit Grid Fault Tolerant Execution Service.
- Entwicklung eines graphischen Interfaces für Administration von Grid Fault Tolerant Execution Service.
- Implementierung und Integrierung von GFTES Adaptern für neue Arten von Grid Middleware.
- Integration in GFTES eines Pre-Compilers für automatische Einbau einer Checkpointing-Routine.
- Einbau der Mechanismen für automatische Erzeugung und Verwaltung von Benutzerzertifikaten.

Appendix A: SQL Skript für's Erstellen der Tabellen in MySQL Datenbank.

```
1  /*****************BEGIN: Grid Service ******************/
2  create table gftesdbo.service
3          (
4                  creationTime varchar(14),
5                  validity int default 0,
6                  id varchar (255) not null,
7                  name varchar (255),
8                  capability varchar (255),
9                  type varchar (255),
10                 qualityLevel varchar (255),
11                 complexity varchar (255),
12                 primary key (id)
13         )
14 create table gftesdbo.service_otherinfo
15         (
16                 id varchar (255) not null,
17                 otherinfo varchar  (255) not null,
18                 primary key (id, otherinfo),
19                 foreign key (id) references gftesdbo.service(id)
20                 on delete cascade
21         )
22 create table gftesdbo.service_statusInfo
23         (
24                 id varchar(255) not null,
25                 statusInfo varchar (255) not null,
26                 primary key (id, statusInfo),
27                 foreign key (id) references gftesdbo.service(id)
28                 on delete cascade
29         )
30 create table gftesdbo.gridService
31         (
32                 id varchar (255) not null,
33                 totalActivities integer default 0,
34                 runningActivities integer default 0,
35                 waitingActivities integer default 0,
36                 suspendetActivities integer default 0,
37                 endpoint varchar(255) not null,
38                 primary key (id),
39                 foreign key (endpoint) references gftesdbo.endpoint(id),
40                 foreign key (id) references gftesdbo.service(id)
41                 on delete cascade
42         )
43 /************************END Services ***********************/
```

```
44 /****************BEGIN: GridActivity ******************/
45 create table gftesdbo.activity
46         (
47                 creationTime varchar(14),
48                 validity int default 0,
49                 id varchar (255) not null,
50                 name varchar (255),
51                 description varchar(255),
52                 primary key (id)
53         )
54
55 create table gftesdbo.activity_otherinfo
56         (
57                 id varchar(255) not null,
58                 otherinfo varchar(255) not null,
59                 primary key (id, otherinfo),
60                 foreign key (id) references gftesdbo.activity(id)
61                 on delete cascade
62         )
63
64 create table gftesdbo.gridActivity
65         (
66                 id varchar(255) not null,
67                 executionServiceLocalID varchar(255),
68                 state varchar(255) default 'unknown',
69                 restartState varchar(255) default 'false',
70                 serviceID varchar(255),
71                 ownerID varchar(255) not null,
72                 checkpointName varchar(255),
73                 checkpointAddress varchar(255),
74                 requiredResourceID varchar(255) not null,
75                 foundResourceID varchar(255),
76                 workingDirectory varchar(255),
77                 applicationExecutable varchar(255) not null,
78                 migrationCounter integer default 0,
79                 primary key (id),
80                 foreign key (id) references gftesdbo.activity(id)
81                 on delete cascade,
82                 foreign key (requiredResourceID) references gftesdbo.
                       resource(id),
83                 foreign key (gettedResourceID) references gftesdbo.
                       resource(id),
84                 foreign key (ownerID) references gftesdbo.user(id)
85         )
86 create table gftesdbo.gridActivity_applicationArguments
87         (
88                 id varchar(255) not null,
89                 placeNumber integer not null,
90                 applicationArgument varchar(255) not null,
91                 primary key (id, placeNumber),
92                 foreign key (id) references gftesdbo.gridActivity(id)
93                 on delete cascade
94         )
```

```
95
96  create table gftesdbo.gridActivity_applicationStageIn
97          (
98                  id varchar(255) not null,
99                  applicationStageIn varchar(255) not null,
100                 primary key (id, applicationStageIn),
101                 foreign key (id) references gftesdbo.gridActivity(id)
102                 on delete cascade
103         )
104
105 create table gftesdbo.gridActivity_applicationStageOut
106         (
107                 id varchar(255) not null,
108                 applicationStageOut varchar(255) not null,
109                 primary key (id, applicationStageOut),
110                 foreign key (id) references gftesdbo.gridActivity(id)
111                 on delete cascade
112         )
113
114 create table gftesdbo.gridActivity_software
115         (
116                 id varchar(255) not null,
117                 softwareDescription varchar(255) not null,
118                 softwareVersion varchar(255),
119                 primary key (id, softwareDescription, softwareVersion),
120                 foreign key (id) references gftesdbo.gridActivity(id)
121                 on delete cascade
122         )
123
124 /****************END: GridActivity ******************/
125
126 /****************BEGIN: GridActivityResource ******************/
127 create table gftesdbo.resource
128         (
129                 creationTime varchar(14),
130                 validity int default 0,
131                 id varchar (255) not null,
132                 name varchar (255),
133                 description varchar(255),
134                 primary key (id)
135         )
136
137 create table gftesdbo.resource_otherinfo
138         (
139                 id varchar(255) not null,
140                 otherinfo varchar(255) not null,
141                 primary key (id, otherinfo),
142                 foreign key (id) references gftesdbo.resource(id)
143                 on delete cascade
144         )
145
146 create table gftesdbo.gridActivityResource
147         (
```

```
148                 id varchar(255) not null,
149                 operatingSystem varchar(255) default 'unknown',
150                 cpuArchitecture varchar(255) default 'unknown',
151                 totalCPUCount integer default 0,
152                 totalPhysicalMemory integer default 0,
153                 totalVirtualMemory integer default 0,
154                 totalDiscSpace integer default 0,
155                 totalCPUSpeed varchar(255) default "0.0",
156                 primary key (id),
157                 foreign key (id) references gftesdbo.resource(id)
158                 on delete cascade
159         )
160 /*****************END: GridActivityResource ******************/
161
162 /*****************BEGIN: User ************************/
163 create table gftesdbo.domain
164         (
165                 creationTime varchar(14),
166                 validity int default 0,
167                 id varchar (255) not null,
168                 name varchar (255),
169                 description varchar(255),
170                 primary key (id)
171         )
172
173 create table gftesdbo.domain_otherinfo
174         (
175                 id varchar(255) not null,
176                 otherinfo varchar(255) not null,
177                 primary key (id, otherinfo),
178                 foreign key (id) references gftesdbo.domain(id)
179                 on delete cascade
180         )
181
182 create table gftesdbo.domain_www
183         (
184                 id varchar(255) not null,
185                 www varchar(255) not null,
186                 primary key (id, www),
187                 foreign key (id) references gftesdbo.domain(id)
188                 on delete cascade
189         )
190 create table gftesdbo.user
191         (
192                 id varchar(255) not null,
193                 rolle varchar(255),
194                 login varchar(255),
195                 password varchar(255),
196                 proxyCertificat varchar(255),
197                 primary key (id),
198                 foreign key (id) references gftesdbo.domain(id)
199                 on delete cascade
200         )
```

```
201 /****************END: User ************************/
202
203 /****************BEGIN: Queue *********************/
204 create table gftesdbo.queue
205         (
206                 creationTime varchar(14),
207                 validity int default 0,
208                 id varchar (255) not null,
209                 name varchar (255),
210                 primary key (id)
211         )
212
213 create table gftesdbo.queue_otherinfo
214         (
215                 id varchar(255) not null,
216                 otherinfo varchar(255) not null,
217                 primary key (id, otherinfo),
218                 foreign key (id) references gftesdbo.queue(id)
219                 on delete cascade
220         )
221 /****************END: Queue ********************** /
222
223 /****************BEGIN: Endpoint ******************/
224 create table gftesdbo.endpoint
225         (
226                 creationTime char(14),
227                 validity int default 0,
228                 id varchar (255) not null,
229                 name varchar (255),
230                 url varchar(1000) not null default 'unknown' ,
231                 technology varchar(1000),
232                 interfaceName varchar(255) not null default 'unknown',
233                 implementor varchar(255),
234                 implementationName varchar(255),
235                 implementationVersion varchar(255),
236                 qualityLevel varchar(255),
237                 healthState varchar(255),
238                 healthStateInfo varchar(255),
239                 servingState varchar(255),
240                 startTime varchar(14),
241                 issuerCA varchar(255),
242                 downtimeAnnounce varchar(14),
243                 downtimeStart varchar(14),
244                 downtimeEnd varchar(14),
245                 downtimeInfo varchar(255),
246                 primary key (id)
247         )
248 create table gftesdbo.endpoint_otherinfo
249         (
250                 id varchar(255) not null,
251                 otherinfo varchar(255) not null,
252                 primary key (id, otherinfo),
253                 foreign key (id) references gftesdbo.endpoint(id)
```

```
254                 on delete cascade
255         )
256
257 create table gftesdbo.endpoint_capability
258         (
259                 id varchar(255) not null,
260                 capability varchar(255) not null,
261                 primary key (id, capability),
262                 foreign key (id) references gftesdbo.endpoint(id)
263                 on delete cascade
264         )
265
266 create table gftesdbo.endpoint_interfaceVersion
267         (
268                 id varchar(255) not null,
269                 interfaceVersion varchar(255) not null,
270                 primary key (id, interfaceVersion),
271                 foreign key (id) references gftesdbo.endpoint(id)
272                 on delete cascade
273         )
274 create table gftesdbo.endpoint_interfaceExtension
275         (
276                 id varchar(255) not null,
277                 interfaceExtension varchar(255) not null,
278                 primary key (id, interfaceExtension),
279                 foreign key (id) references gftesdbo.endpoint(id)
280                 on delete cascade
281         )
282
283 create table gftesdbo.endpoint_wsdl
284         (
285                 id varchar(255) not null,
286                 wsdl varchar(255) not null,
287                 primary key (id, wsdl),
288                 foreign key (id) references gftesdbo.endpoint(id)
289                 on delete cascade
290         )
291
292 create table gftesdbo.endpoint_supportedProfile
293         (
294                 id varchar(255) not null,
295                 supportedProfile varchar(255) not null,
296                 primary key (id, supportedProfile),
297                 foreign key (id) references gftesdbo.endpoint(id)
298                 on delete cascade
299         )
300 create table gftesdbo.endpoint_semantics
301         (
302                 id varchar(255) not null,
303                 semantics varchar(255) not null,
304                 primary key (id, semantics),
305                 foreign key (id) references gftesdbo.endpoint(id)
306                 on delete cascade
```

```
307         )
308
309 create table gftesdbo.endpoint_trustedCA
310         (
311                 id varchar(255) not null,
312                 trustedCA varchar(255) not null,
313                 primary key (id, trustedCA),
314                 foreign key (id) references gftesdbo.endpoint(id)
315                 on delete cascade
316         )
317
318
319 /*****************END: Endpoint ********************** /
```

Appendix B: WSDL Dokument von Grid Fault Tolerant Execution Service.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
       xmlns:axis2="http://gftes/" xmlns:ns1="http://org.apache.axis2/xsd"
       xmlns:ns="http://gftes/xsd" xmlns:wsaw="http://www.w3.org/2006/05/
       addressing/wsdl" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
       xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:mime="http://
       schemas.xmlsoap.org/wsdl/mime/" xmlns:soap="http://schemas.xmlsoap.
       org/wsdl/soap/" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
       " targetNamespace="http://gftes/">
3      <wsdl:documentation>gftes</wsdl:documentation>
4      <wsdl:types>
5          <xs:schema attributeFormDefault="qualified" elementFormDefault="
               qualified" targetNamespace="http://gftes/xsd">
6              <xs:complexType name="Exception">
7                  <xs:sequence>
8                      <xs:element minOccurs="0" name="Exception" nillable=
                           "true" type="xs:anyType"/>
9                  </xs:sequence>
10             </xs:complexType>
11             <xs:element name="terminateActivities">
12                 <xs:complexType>
13                     <xs:sequence>
14                         <xs:element maxOccurs="unbounded" minOccurs="0"
                               name="activityIdentifier" nillable="true"
                               type="xs:string"/>
15                     </xs:sequence>
16                 </xs:complexType>
17             </xs:element>
18             <xs:element name="terminateActivitiesResponse">
19                 <xs:complexType>
20                     <xs:sequence>
21                         <xs:element maxOccurs="unbounded" minOccurs="0"
                               name="return" nillable="true" type="xs:string
                               "/>
22                     </xs:sequence>
23                 </xs:complexType>
24             </xs:element>
25             <xs:element name="getFactoryAttributesDocumentResponse">
26                 <xs:complexType>
27                     <xs:sequence>
28                         <xs:element minOccurs="0" name="return" nillable
                               ="true" type="xs:string"/>
29                     </xs:sequence>
```

```
30                </xs:complexType>
31            </xs:element>
32            <xs:element name="getAuthentication">
33                <xs:complexType>
34                    <xs:sequence>
35                        <xs:element minOccurs="0" name="login" nillable=
                              "true" type="xs:string"/>
36                        <xs:element minOccurs="0" name="password"
                              nillable="true" type="xs:string"/>
37                    </xs:sequence>
38                </xs:complexType>
39            </xs:element>
40            <xs:element name="getAuthenticationResponse">
41                <xs:complexType>
42                    <xs:sequence>
43                        <xs:element minOccurs="0" name="return" type="
                              xs:boolean"/>
44                    </xs:sequence>
45                </xs:complexType>
46            </xs:element>
47            <xs:element name="getActvityStatuses">
48                <xs:complexType>
49                    <xs:sequence>
50                        <xs:element maxOccurs="unbounded" minOccurs="0"
                              name="activityIdentifier" nillable="true"
                              type="xs:string"/>
51                    </xs:sequence>
52                </xs:complexType>
53            </xs:element>
54            <xs:element name="getActvityStatusesResponse">
55                <xs:complexType>
56                    <xs:sequence>
57                        <xs:element maxOccurs="unbounded" minOccurs="0"
                              name="return" nillable="true" type="xs:string
                              "/>
58                    </xs:sequence>
59                </xs:complexType>
60            </xs:element>
61            <xs:element name="getActivityDocuments">
62                <xs:complexType>
63                    <xs:sequence>
64                        <xs:element maxOccurs="unbounded" minOccurs="0"
                              name="activityIdentifier" nillable="true"
                              type="xs:string"/>
65                    </xs:sequence>
66                </xs:complexType>
67            </xs:element>
68            <xs:element name="getActivityDocumentsResponse">
69                <xs:complexType>
70                    <xs:sequence>
71                        <xs:element minOccurs="0" name="return" nillable
                              ="true" type="xs:string"/>
72                    </xs:sequence>
```

```
73                  </xs:complexType>
74              </xs:element>
75              <xs:element name="createActivity">
76                  <xs:complexType>
77                      <xs:sequence>
78                          <xs:element minOccurs="0" name="xml" nillable="
                                true" type="xs:string"/>
79                      </xs:sequence>
80                  </xs:complexType>
81              </xs:element>
82              <xs:element name="createActivityResponse">
83                  <xs:complexType>
84                      <xs:sequence>
85                          <xs:element minOccurs="0" name="return" nillable
                                ="true" type="xs:string"/>
86                      </xs:sequence>
87                  </xs:complexType>
88              </xs:element>
89          </xs:schema>
90      </wsdl:types>
91      <wsdl:message name="getActivityDocumentsRequest">
92          <wsdl:part name="parameters" element="ns:getActivityDocuments"/>
93      </wsdl:message>
94      <wsdl:message name="getActivityDocumentsResponse">
95          <wsdl:part name="parameters" element="
                ns:getActivityDocumentsResponse"/>
96      </wsdl:message>
97      <wsdl:message name="stopAcceptingNewActivitiesRequest"/>
98      <wsdl:message name="stopAcceptingNewActivitiesResponse"/>
99      <wsdl:message name="getFactoryAttributesDocumentRequest"/>
100     <wsdl:message name="getFactoryAttributesDocumentResponse">
101         <wsdl:part name="parameters" element="
                ns:getFactoryAttributesDocumentResponse"/>
102     </wsdl:message>
103     <wsdl:message name="getActvityStatusesRequest">
104         <wsdl:part name="parameters" element="ns:getActvityStatuses"/>
105     </wsdl:message>
106     <wsdl:message name="getActvityStatusesResponse">
107         <wsdl:part name="parameters" element="
                ns:getActvityStatusesResponse"/>
108     </wsdl:message>
109     <wsdl:message name="createActivityRequest">
110         <wsdl:part name="parameters" element="ns:createActivity"/>
111     </wsdl:message>
112     <wsdl:message name="createActivityResponse">
113         <wsdl:part name="parameters" element="ns:createActivityResponse"
                />
114     </wsdl:message>
115     <wsdl:message name="startAcceptingNewActivitiesRequest"/>
116     <wsdl:message name="startAcceptingNewActivitiesResponse"/>
117     <wsdl:message name="terminateActivitiesRequest">
118         <wsdl:part name="parameters" element="ns:terminateActivities"/>
119     </wsdl:message>
```

```
120     <wsdl:message name="terminateActivitiesResponse">
121         <wsdl:part name="parameters" element="
                ns:terminateActivitiesResponse"/>
122     </wsdl:message>
123     <wsdl:message name="getAuthenticationRequest">
124         <wsdl:part name="parameters" element="ns:getAuthentication"/>
125     </wsdl:message>
126     <wsdl:message name="getAuthenticationResponse">
127         <wsdl:part name="parameters" element="
                ns:getAuthenticationResponse"/>
128     </wsdl:message>
129     <wsdl:portType name="gftesPortType">
130         <wsdl:operation name="getActivityDocuments">
131             <wsdl:input message="axis2:getActivityDocumentsRequest"
                    wsaw:Action="urn:getActivityDocuments"/>
132             <wsdl:output message="axis2:getActivityDocumentsResponse"
                    wsaw:Action="urn:getActivityDocumentsResponse"/>
133         </wsdl:operation>
134         <wsdl:operation name="stopAcceptingNewActivities">
135             <wsdl:input message="axis2:stopAcceptingNewActivitiesRequest
                    " wsaw:Action="urn:stopAcceptingNewActivities"/>
136             <wsdl:output message="
                    axis2:stopAcceptingNewActivitiesResponse" wsaw:Action="
                    urn:stopAcceptingNewActivitiesResponse"/>
137         </wsdl:operation>
138         <wsdl:operation name="getFactoryAttributesDocument">
139             <wsdl:input message="
                    axis2:getFactoryAttributesDocumentRequest" wsaw:Action="
                    urn:getFactoryAttributesDocument"/>
140             <wsdl:output message="
                    axis2:getFactoryAttributesDocumentResponse" wsaw:Action="
                    urn:getFactoryAttributesDocumentResponse"/>
141         </wsdl:operation>
142         <wsdl:operation name="getActvityStatuses">
143             <wsdl:input message="axis2:getActvityStatusesRequest"
                    wsaw:Action="urn:getActvityStatuses"/>
144             <wsdl:output message="axis2:getActvityStatusesResponse"
                    wsaw:Action="urn:getActvityStatusesResponse"/>
145         </wsdl:operation>
146         <wsdl:operation name="createActivity">
147             <wsdl:input message="axis2:createActivityRequest"
                    wsaw:Action="urn:createActivity"/>
148             <wsdl:output message="axis2:createActivityResponse"
                    wsaw:Action="urn:createActivityResponse"/>
149         </wsdl:operation>
150         <wsdl:operation name="startAcceptingNewActivities">
151             <wsdl:input message="
                    axis2:startAcceptingNewActivitiesRequest" wsaw:Action="
                    urn:startAcceptingNewActivities"/>
152             <wsdl:output message="
                    axis2:startAcceptingNewActivitiesResponse" wsaw:Action="
                    urn:startAcceptingNewActivitiesResponse"/>
153         </wsdl:operation>
```

```
154         <wsdl:operation name="terminateActivities">
155             <wsdl:input message="axis2:terminateActivitiesRequest"
                    wsaw:Action="urn:terminateActivities"/>
156             <wsdl:output message="axis2:terminateActivitiesResponse"
                    wsaw:Action="urn:terminateActivitiesResponse"/>
157         </wsdl:operation>
158         <wsdl:operation name="getAuthentication">
159             <wsdl:input message="axis2:getAuthenticationRequest"
                    wsaw:Action="urn:getAuthentication"/>
160             <wsdl:output message="axis2:getAuthenticationResponse"
                    wsaw:Action="urn:getAuthenticationResponse"/>
161         </wsdl:operation>
162     </wsdl:portType>
163     <wsdl:binding name="gftesSoap11Binding" type="axis2:gftesPortType">
164         <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                style="document"/>
165         <wsdl:operation name="getActivityDocuments">
166             <soap:operation soapAction="urn:getActivityDocuments" style=
                    "document"/>
167             <wsdl:input>
168                 <soap:body use="literal"/>
169             </wsdl:input>
170             <wsdl:output>
171                 <soap:body use="literal"/>
172             </wsdl:output>
173         </wsdl:operation>
174         <wsdl:operation name="stopAcceptingNewActivities">
175             <soap:operation soapAction="urn:stopAcceptingNewActivities"
                    style="document"/>
176             <wsdl:input>
177                 <soap:body use="literal"/>
178             </wsdl:input>
179             <wsdl:output>
180                 <soap:body use="literal"/>
181             </wsdl:output>
182         </wsdl:operation>
183         <wsdl:operation name="getFactoryAttributesDocument">
184             <soap:operation soapAction="urn:getFactoryAttributesDocument
                    " style="document"/>
185             <wsdl:input>
186                 <soap:body use="literal"/>
187             </wsdl:input>
188             <wsdl:output>
189                 <soap:body use="literal"/>
190             </wsdl:output>
191         </wsdl:operation>
192         <wsdl:operation name="getActvityStatuses">
193             <soap:operation soapAction="urn:getActvityStatuses" style="
                    document"/>
194             <wsdl:input>
195                 <soap:body use="literal"/>
196             </wsdl:input>
197             <wsdl:output>
```

```
198                 <soap:body use="literal"/>
199             </wsdl:output>
200         </wsdl:operation>
201         <wsdl:operation name="createActivity">
202             <soap:operation soapAction="urn:createActivity" style="
                    document"/>
203             <wsdl:input>
204                 <soap:body use="literal"/>
205             </wsdl:input>
206             <wsdl:output>
207                 <soap:body use="literal"/>
208             </wsdl:output>
209         </wsdl:operation>
210         <wsdl:operation name="startAcceptingNewActivities">
211             <soap:operation soapAction="urn:startAcceptingNewActivities"
                     style="document"/>
212             <wsdl:input>
213                 <soap:body use="literal"/>
214             </wsdl:input>
215             <wsdl:output>
216                 <soap:body use="literal"/>
217             </wsdl:output>
218         </wsdl:operation>
219         <wsdl:operation name="terminateActivities">
220             <soap:operation soapAction="urn:terminateActivities" style="
                    document"/>
221             <wsdl:input>
222                 <soap:body use="literal"/>
223             </wsdl:input>
224             <wsdl:output>
225                 <soap:body use="literal"/>
226             </wsdl:output>
227         </wsdl:operation>
228         <wsdl:operation name="getAuthentication">
229             <soap:operation soapAction="urn:getAuthentication" style="
                    document"/>
230             <wsdl:input>
231                 <soap:body use="literal"/>
232             </wsdl:input>
233             <wsdl:output>
234                 <soap:body use="literal"/>
235             </wsdl:output>
236         </wsdl:operation>
237     </wsdl:binding>
238     <wsdl:binding name="gftesSoap12Binding" type="axis2:gftesPortType">
239         <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
                 style="document"/>
240         <wsdl:operation name="getActivityDocuments">
241             <soap12:operation soapAction="urn:getActivityDocuments"
                    style="document"/>
242             <wsdl:input>
243                 <soap12:body use="literal"/>
244             </wsdl:input>
```

```
245             <wsdl:output>
246                 <soap12:body use="literal"/>
247             </wsdl:output>
248         </wsdl:operation>
249         <wsdl:operation name="stopAcceptingNewActivities">
250             <soap12:operation soapAction="urn:stopAcceptingNewActivities
                    " style="document"/>
251             <wsdl:input>
252                 <soap12:body use="literal"/>
253             </wsdl:input>
254             <wsdl:output>
255                 <soap12:body use="literal"/>
256             </wsdl:output>
257         </wsdl:operation>
258         <wsdl:operation name="getFactoryAttributesDocument">
259             <soap12:operation soapAction="
                    urn:getFactoryAttributesDocument" style="document"/>
260             <wsdl:input>
261                 <soap12:body use="literal"/>
262             </wsdl:input>
263             <wsdl:output>
264                 <soap12:body use="literal"/>
265             </wsdl:output>
266         </wsdl:operation>
267         <wsdl:operation name="getActvityStatuses">
268             <soap12:operation soapAction="urn:getActvityStatuses" style=
                    "document"/>
269             <wsdl:input>
270                 <soap12:body use="literal"/>
271             </wsdl:input>
272             <wsdl:output>
273                 <soap12:body use="literal"/>
274             </wsdl:output>
275         </wsdl:operation>
276         <wsdl:operation name="createActivity">
277             <soap12:operation soapAction="urn:createActivity" style="
                    document"/>
278             <wsdl:input>
279                 <soap12:body use="literal"/>
280             </wsdl:input>
281             <wsdl:output>
282                 <soap12:body use="literal"/>
283             </wsdl:output>
284         </wsdl:operation>
285         <wsdl:operation name="startAcceptingNewActivities">
286             <soap12:operation soapAction="
                    urn:startAcceptingNewActivities" style="document"/>
287             <wsdl:input>
288                 <soap12:body use="literal"/>
289             </wsdl:input>
290             <wsdl:output>
291                 <soap12:body use="literal"/>
292             </wsdl:output>
```

```
293         </wsdl:operation>
294         <wsdl:operation name="terminateActivities">
295             <soap12:operation soapAction="urn:terminateActivities" style
                    ="document"/>
296             <wsdl:input>
297                 <soap12:body use="literal"/>
298             </wsdl:input>
299             <wsdl:output>
300                 <soap12:body use="literal"/>
301             </wsdl:output>
302         </wsdl:operation>
303         <wsdl:operation name="getAuthentication">
304             <soap12:operation soapAction="urn:getAuthentication" style="
                    document"/>
305             <wsdl:input>
306                 <soap12:body use="literal"/>
307             </wsdl:input>
308             <wsdl:output>
309                 <soap12:body use="literal"/>
310             </wsdl:output>
311         </wsdl:operation>
312     </wsdl:binding>
313     <wsdl:binding name="gftesHttpBinding" type="axis2:gftesPortType">
314         <http:binding verb="POST"/>
315         <wsdl:operation name="getActivityDocuments">
316             <http:operation location="gftes/getActivityDocuments"/>
317             <wsdl:input>
318                 <mime:content type="text/xml" part="getActivityDocuments
                        "/>
319             </wsdl:input>
320             <wsdl:output>
321                 <mime:content type="text/xml" part="getActivityDocuments
                        "/>
322             </wsdl:output>
323         </wsdl:operation>
324         <wsdl:operation name="stopAcceptingNewActivities">
325             <http:operation location="gftes/stopAcceptingNewActivities"/
                    >
326             <wsdl:input>
327                 <mime:content type="text/xml" part="
                        stopAcceptingNewActivities"/>
328             </wsdl:input>
329             <wsdl:output>
330                 <mime:content type="text/xml" part="
                        stopAcceptingNewActivities"/>
331             </wsdl:output>
332         </wsdl:operation>
333         <wsdl:operation name="getFactoryAttributesDocument">
334             <http:operation location="gftes/getFactoryAttributesDocument
                    "/>
335             <wsdl:input>
336                 <mime:content type="text/xml" part="
                        getFactoryAttributesDocument"/>
```

```
337             </wsdl:input>
338             <wsdl:output>
339                 <mime:content type="text/xml" part="
                        getFactoryAttributesDocument"/>
340             </wsdl:output>
341         </wsdl:operation>
342         <wsdl:operation name="getActvityStatuses">
343             <http:operation location="gftes/getActvityStatuses"/>
344             <wsdl:input>
345                 <mime:content type="text/xml" part="getActvityStatuses"/
                        >
346             </wsdl:input>
347             <wsdl:output>
348                 <mime:content type="text/xml" part="getActvityStatuses"/
                        >
349             </wsdl:output>
350         </wsdl:operation>
351         <wsdl:operation name="createActivity">
352             <http:operation location="gftes/createActivity"/>
353             <wsdl:input>
354                 <mime:content type="text/xml" part="createActivity"/>
355             </wsdl:input>
356             <wsdl:output>
357                 <mime:content type="text/xml" part="createActivity"/>
358             </wsdl:output>
359         </wsdl:operation>
360         <wsdl:operation name="startAcceptingNewActivities">
361             <http:operation location="gftes/startAcceptingNewActivities"
                    />
362             <wsdl:input>
363                 <mime:content type="text/xml" part="
                        startAcceptingNewActivities"/>
364             </wsdl:input>
365             <wsdl:output>
366                 <mime:content type="text/xml" part="
                        startAcceptingNewActivities"/>
367             </wsdl:output>
368         </wsdl:operation>
369         <wsdl:operation name="terminateActivities">
370             <http:operation location="gftes/terminateActivities"/>
371             <wsdl:input>
372                 <mime:content type="text/xml" part="terminateActivities"
                        />
373             </wsdl:input>
374             <wsdl:output>
375                 <mime:content type="text/xml" part="terminateActivities"
                        />
376             </wsdl:output>
377         </wsdl:operation>
378         <wsdl:operation name="getAuthentication">
379             <http:operation location="gftes/getAuthentication"/>
380             <wsdl:input>
381                 <mime:content type="text/xml" part="getAuthentication"/>
```

```
382             </wsdl:input>
383             <wsdl:output>
384                 <mime:content type="text/xml" part="getAuthentication"/>
385             </wsdl:output>
386         </wsdl:operation>
387     </wsdl:binding>
388     <wsdl:service name="gftes">
389         <wsdl:port name="gftesHttpSoap11Endpoint" binding="
                axis2:gftesSoap11Binding">
390             <soap:address location="http://localhost:8080/axis2/services
                    /gftes.gftesHttpSoap11Endpoint/"/>
391         </wsdl:port>
392         <wsdl:port name="gftesHttpSoap12Endpoint" binding="
                axis2:gftesSoap12Binding">
393             <soap12:address location="http://localhost:8080/axis2/
                    services/gftes.gftesHttpSoap12Endpoint/"/>
394         </wsdl:port>
395         <wsdl:port name="gftesHttpEndpoint" binding="
                axis2:gftesHttpBinding">
396             <http:address location="http://localhost:8080/axis2/services
                    /gftes.gftesHttpEndpoint/"/>
397         </wsdl:port>
398     </wsdl:service>
399 </wsdl:definitions>
```

Appendix C: Normative Schema für Job Beschreibung in GFTES.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://localhost/jsdl/2011/01/jsdl"
    xmlns:jsdl="http://localhost/jsdl/2011/01/jsdl"
    targetNamespace="http://localhost/jsdl/2011/01/jsdl"
    elementFormDefault="qualified">
  <!--=============================================================-->
  <!-- SIMPLE TYPES -->
  <xsd:simpleType name="ProcessorArchitectureEnumeration">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="sparc"/>
      <xsd:enumeration value="powerpc"/>
      <xsd:enumeration value="x86"/>
      <xsd:enumeration value="x86_32"/>
      <xsd:enumeration value="x86_64"/>
      <xsd:enumeration value="parisc"/>
      <xsd:enumeration value="mips"/>
      <xsd:enumeration value="ia64"/>
      <xsd:enumeration value="arm"/>
      <xsd:enumeration value="other"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!--=============================================================-->
  <xsd:simpleType name="OperatingSystemTypeEnumeration">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Unknown"/>
      <xsd:enumeration value="MACOS"/>
      <xsd:enumeration value="ATTUNIX"/>
      <xsd:enumeration value="DGUX"/>
      <xsd:enumeration value="DECNT"/>
      <xsd:enumeration value="Tru64_UNIX"/>
      <xsd:enumeration value="OpenVMS"/>
      <xsd:enumeration value="HPUX"/>
      <xsd:enumeration value="AIX"/>
      <xsd:enumeration value="MVS"/>
      <xsd:enumeration value="OS400"/>
      <xsd:enumeration value="OS_2"/>
      <xsd:enumeration value="JavaVM"/>
      <xsd:enumeration value="MSDOS"/>
      <xsd:enumeration value="WIN3x"/>
      <xsd:enumeration value="WIN95"/>
      <xsd:enumeration value="WIN98"/>
      <xsd:enumeration value="WINNT"/>
```

```
44        <xsd:enumeration value="WINCE"/>
45        <xsd:enumeration value="NCR3000"/>
46        <xsd:enumeration value="NetWare"/>
47        <xsd:enumeration value="OSF"/>
48        <xsd:enumeration value="DC_OS"/>
49        <xsd:enumeration value="Reliant_UNIX"/>
50        <xsd:enumeration value="SCO_UnixWare"/>
51        <xsd:enumeration value="SCO_OpenServer"/>
52        <xsd:enumeration value="Sequent"/>
53        <xsd:enumeration value="IRIX"/>
54        <xsd:enumeration value="Solaris"/>
55        <xsd:enumeration value="SunOS"/>
56        <xsd:enumeration value="U6000"/>
57        <xsd:enumeration value="ASERIES"/>
58        <xsd:enumeration value="TandemNSK"/>
59        <xsd:enumeration value="TandemNT"/>
60        <xsd:enumeration value="BS2000"/>
61        <xsd:enumeration value="LINUX"/>
62        <xsd:enumeration value="Lynx"/>
63        <xsd:enumeration value="XENIX"/>
64        <xsd:enumeration value="VM"/>
65        <xsd:enumeration value="Interactive_UNIX"/>
66        <xsd:enumeration value="BSDUNIX"/>
67        <xsd:enumeration value="FreeBSD"/>
68        <xsd:enumeration value="NetBSD"/>
69        <xsd:enumeration value="GNU_Hurd"/>
70        <xsd:enumeration value="OS9"/>
71        <xsd:enumeration value="MACH_Kernel"/>
72        <xsd:enumeration value="Inferno"/>
73        <xsd:enumeration value="QNX"/>
74        <xsd:enumeration value="EPOC"/>
75        <xsd:enumeration value="IxWorks"/>
76        <xsd:enumeration value="VxWorks"/>
77        <xsd:enumeration value="MiNT"/>
78        <xsd:enumeration value="BeOS"/>
79        <xsd:enumeration value="HP_MPE"/>
80        <xsd:enumeration value="NextStep"/>
81        <xsd:enumeration value="PalmPilot"/>
82        <xsd:enumeration value="Rhapsody"/>
83        <xsd:enumeration value="Windows_2000"/>
84        <xsd:enumeration value="Dedicated"/>
85        <xsd:enumeration value="OS_390"/>
86        <xsd:enumeration value="VSE"/>
87        <xsd:enumeration value="TPF"/>
88        <xsd:enumeration value="Windows_R_Me"/>
89        <xsd:enumeration value="Caldera_Open_UNIX"/>
90        <xsd:enumeration value="OpenBSD"/>
91        <xsd:enumeration value="Not_Applicable"/>
92        <xsd:enumeration value="Windows_XP"/>
93        <xsd:enumeration value="z_OS"/>
94        <xsd:enumeration value="other"/>
95      </xsd:restriction>
96    </xsd:simpleType>
```

```
97  <xsd:simpleType name="InCaseOfResourceCrashTypeEnumarion">
98    <xsd:restriction base="xsd:string">
99      <xsd:enumeration value="restart"/>
100     <xsd:enumeration value="recover"/>
101     <xsd:enumeration value="ignore"/>
102   </xsd:restriction>
103 </xsd:simpleType>
104 <!--=================================================================-->
105 <xsd:simpleType name="Description_Type">
106   <xsd:restriction base="xsd:string"/>
107 </xsd:simpleType>
108 <!--=================================================================-->
109 <!-- COMPLEX TYPES: Definitions for the RangeValueType -->
110 <xsd:complexType name="JobDefinition_Type">
111   <xsd:sequence>
112     <xsd:element ref="jsdl:JobDescription"/>
113   </xsd:sequence>
114   <xsd:attribute name="id" type="xsd:ID" use="optional"/>
115 </xsd:complexType>
116 <!--=================================================================-->
117 <xsd:complexType name="JobDescription_Type">
118   <xsd:sequence>
119     <xsd:element ref="jsdl:JobIdentification" minOccurs="0"/>
120     <xsd:element ref="jsdl:Application" minOccurs="0"/>
121     <xsd:element ref="jsdl:SoftwareRequirements" minOccurs="0"/>
122     <xsd:element ref="jsdl:Resources" minOccurs="0"/>
123     <xsd:element ref="jsdl:DataAttributes" minOccurs="0" maxOccurs="
          unbounded"/>
124   </xsd:sequence>
125 </xsd:complexType>
126 <!--=================================================================-->
127 <xsd:complexType name="JobIdentification_Type">
128   <xsd:sequence>
129     <xsd:element ref="jsdl:JobName" minOccurs="0"/>
130     <xsd:element ref="jsdl:Description" minOccurs="0"/>
131   </xsd:sequence>
132 </xsd:complexType>
133 <!--=================================================================-->
134 <xsd:complexType name="Application_Type">
135   <xsd:sequence>
136     <xsd:element ref="jsdl:ExecutableName" minOccurs="1" maxOccurs="1"
          />
137     <xsd:element ref="jsdl:Argument" minOccurs="1" maxOccurs="1"/>
138     <xsd:element ref="jsdl:WorkingDirectory" minOccurs="1" maxOccurs="
          1"/>
139     <xsd:element ref="jsdl:InCaseOfResourceCrash" minOccurs="0"
          maxOccurs="1"/>
140     <xsd:element ref="jsdl:CheckpointFileName" minOccurs="0" maxOccurs
          ="1"/>
141   </xsd:sequence>
142 </xsd:complexType>
143 <!--=================================================================-->
144 <xsd:complexType name="SoftwareRequirements_Type">
```

```
145     <xsd:sequence>
146       <xsd:element ref="SoftwareDescription" minOccurs="0" maxOccurs="1"
              />
147       <xsd:element ref="SoftwareVersion" minOccurs="0" maxOccurs="1"/>
148     </xsd:sequence>
149   </xsd:complexType>
150   <!-- ================================================================ -->
151   <!-- COMPLEX TYPES : Resource related types -->
152   <xsd:complexType name="Resources_Type">
153     <xsd:sequence>
154       <xsd:element ref="jsdl:OperatingSystem" minOccurs="0"/>
155       <xsd:element ref="jsdl:CPUArchitecture" minOccurs="0"/>
156       <xsd:element ref="jsdl:CPUCount" minOccurs="0"/>
157       <xsd:element ref="jsdl:CPUSpeed" minOccurs="0"/>
158       <xsd:element ref="jsdl:PhysicalMemory" minOccurs="0"/>
159       <xsd:element ref="jsdl:VirtualMemory" minOccurs="0"/>
160       <xsd:element ref="jsdl:DiskSpace" minOccurs="0"/>
161     </xsd:sequence>
162   </xsd:complexType>
163   <!-- ================================================================ -->
164   <!-- Complex types: Data staging -->
165   <xsd:complexType name="DataAttributes_Type">
166     <xsd:sequence>
167       <xsd:element ref="jsdl:File" minOccurs="0" maxOccurs="unbounded"/>
168     </xsd:sequence>
169   </xsd:complexType>
170   <xsd:complexType name="File_Type">
171     <xsd:sequence>
172       <xsd:element ref="jsdl:FileName" minOccurs="1" maxOccurs="1"/>
173       <xsd:element ref="jsdl:Source" minOccurs="0" maxOccurs="unbounded"
              />
174       <xsd:element ref="jsdl:Target" minOccurs="0" maxOccurs="unbounded"
              />
175     </xsd:sequence>
176   </xsd:complexType>
177   <!-- ================================================================ -->
178   <xsd:element name="JobDefinition" type="jsdl:JobDefinition_Type"/>
179   <xsd:element name="JobDescription" type="jsdl:JobDescription_Type"/>
180   <xsd:element name="JobIdentification" type="
          jsdl:JobIdentification_Type"/>
181   <xsd:element name="JobName" type="xsd:string"/>
182   <xsd:element name="Description" type="xsd:string"/>
183   <xsd:element name="Application" type="jsdl:Application_Type"/>
184   <xsd:element name="ExecutableName" type="xsd:string"/>
185   <xsd:element name="Argument" type="xsd:string"/>
186   <xsd:element name="WorkingDirectory" type="xsd:string"/>
187   <xsd:element name="SoftwareRequirements" type="
          jsdl:SoftwareRequirements_Type"/>
188   <xsd:element name="SoftwareDescription" type="xsd:string"/>
189   <xsd:element name="SoftwareVersion" type="xsd:string"/>
190   <xsd:element name="InCaseOfResourceCrash" type="
          InCaseOfResourceCrashTypeEnumarion"/>
191   <xsd:element name="CheckpointFileName" type="xsd:string"/>
```

```
192   <xsd:element name="Resources" type="jsdl:Resources_Type"/>
193   <xsd:element name="OperatingSystem" type="
          jsdl:OperatingSystemTypeEnumeration" />
194   <xsd:element name="CPUArchitecture" type="
          jsdl:ProcessorArchitectureEnumeration" />
195   <xsd:element name="CPUCount" type="xsd:integer" />
196   <xsd:element name="CPUSpeed" type="xsd:float"/>
197   <xsd:element name="PhysicalMemory" type="xsd:integer" />
198   <xsd:element name="VirtualMemory" type="xsd:integer" />
199   <xsd:element name="DiskSpace" type="xsd:integer" />
200   <xsd:element name="DataAttributes" type="jsdl:DataAttributes_Type" />
201   <xsd:element name="File" type="jsdl:File_Type" />
202   <xsd:element name="FileName" type="xsd:string"/>
203   <xsd:element name="Source" type="xsd:anyURI"/>
204   <xsd:element name="Target" type="xsd:anyURI"/>
205 </xsd:schema>
```

Abbildungsverzeichnis

Literaturverzeichnis

[1] Andreozzi, S. ; Burke, S. ; Ehm, F. ; Field, L. ; Galang, G. ; Konya, B. ; Litmaath, M. ; Millar, P. ; Navarro, J.: *GLUE 2.0 Specification*

[2] Anjomshoaa, A. ; Brisard, F. ; Drescher, M. ; Fellows, D. ; Ly, A. ; McGough, S. ; Pulsipher, D. ; Savva, A.: Job submission description language (jsdl) specification, version 1.0. In: *Open Grid Forum, GFD* Bd. 56, 2005

[3] Chong, Anthony ; Sourin, Alexei ; Levinski, Konstantin: Grid-based computer animation rendering. In: *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*. New York, NY, USA : ACM, 2006 (GRAPHITE '06). – ISBN 1–59593–564–9, 39–47

[4] Cuttone, F. ; Di Stefano, A. ; Morana, G.: Designing a Dependable Job Submission System in gLite. In: *2010 19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises* IEEE, 2010. – ISSN 1524–4547, S. 258–262

[5] Dustdar, S. ; Gall, H. ; Hauswirth, M.: *Software-Architekturen für verteilte Systeme*. Springer, 2003 `http://books.google.de/books?id=Vwd08ThG90cC`. – ISBN 9783540430889

[6] Globus-Team: *Globus Toolkit Homepage*. `http://www.globus.org/toolkit/`, 2011. – [Online; accessed 07-02-2011]

[7] Grandinetti, L.: *Grid computing: the new frontier of high performance computing*. Elsevier, 2005 (Advances in parallel computing). – 54 S. `http://books.google.com/books?id=O7d-D4DtNX4C`. – ISBN 9780444519993

[8] GridTalkProject Team, CERN: *Grid: The Architecture*. `http://www.gridcafe.org/grid-architecture.html`, 2011. – [Online; accessed 10-01-2011]

[9] Grimshaw, A. ; Newhouse, S. ; Pulsipher, D. ; Morgan, M.: OGSA Basic Execution Service Version 1.0. In: *Open Grid Forum*, 2006

[10] Haase, O.: *Kommunikation in verteilten Anwendungen: Einführung in Sockets, Java RMI, CORBA und Jini*. Oldenbourg Wissensch.Vlg, 2008 `http://books.google.de/books?id=uKgzz-833vIC`. – ISBN 9783486584813

[11] IEC, ISO: 9126-1: Software Engineering–Qualität von Softwareprodukten-Teil 1: Qualitätsmodell. In: *International Standardisation Organisation, Berlin, Beuth Verlag* (2001)

[12] Klemm, M. ; Veldema, R. ; Philippsen, M.: An Automatic Cost-based Framework for Seamless Application Migration in Grid Environments. In: *Proceedings of the 20th IASTED International Conference* Bd. 631, S. 072–219

[13] Kretsis, A. ; Kokkinos, P. ; Varvarigos, E.: Developing scheduling policies in gLite middleware. In: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid* IEEE Computer Society, 2009, S. 20–27

[14] Kurowski, K. ; Ludwiczak, B. ; Nabrzyski, J. ; Oleksiak, A. ; Pukacki, J.: Dynamic grid scheduling with job migration and rescheduling in the GridLab resource management system. In: *Scientific Programming* 12 (2004), Nr. 4, S. 263–273. – ISSN 1058–9244

[15] Köln, Uni: *SuGI-Lernportal: Globus Toolkit.* `http://sugi.d-grid.de/de/grid-computing/grid-middlewares-uebungssysteme/globus-toolkit.html`, 2011. – [Online; accessed 07-02-2011]

[16] Laprie, J.C. ; Randell, B.: Fundamental concepts of dependability. In: *Report N01145, LAAS-CNRS* (2001)

[17] Liang, D. ; Fang, C.L. ; Chen, C. ; Lin, F.: Fault tolerant web service. In: *Software Engineering Conference, 2003. Tenth Asia-Pacific* IEEE, 2005. – ISBN 0769520111, S. 310–319

[18] Milojicic, D.S. ; Douglis, F. ; Paindaveine, Y. ; Wheeler, R. ; Zhou, S.: Process migration. In: *ACM Computing Surveys* 32 (2000), Nr. 3, S. 241–299. – ISSN 0360–0300

[19] Plankensteiner, K. ; Prodan, R. ; Fahringer, T. ; Kertesz, A. ; Kacsuk, P.K.: Fault-tolerant behavior in state-of-the-art grid workflow management systems. In: *Institute for Computer Science University of Innsbruck* (2007)

[20] Pradhan, DK: Fault-tolerant computing. In: *Computer* (1980), S. 6–7. – ISSN 0018–9162

[21] Project, EGEE: *gLite Project.* `http://glite.cern.ch/`, 2010. – [Online; accessed 21-02-2011]

[22] Reinefeld, A. ; Schintke, F.: „Dienste und Standards für das Grid-Computing “. In: *J. von Knop, W. Haferkamp (Hrsg.)* 18, S. 293–304

[23] Rodriguez, Alex: *RESTful Web services: The basics.* `https://www.ibm.com/developerworks/webservices/library/ws-restful/`, 2008. – [Online; accessed 25-01-2011]

[24] Silva, L.M. ; Silva, J.G.: System-level versus user-defined checkpointing. In: *Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on* IEEE, 2002. – ISBN 0818692189, S. 68–74

[25] Stahl, H. ; Karsch, S. ; Reiser, H. ; Atug, M. ; Skerka, R. ; Leventi-Peetz, A. ; Plaga, R.: Vorstudie Grid Sicherheits-Infrastruktur (GSI). Ergebnisse des Arbeitspakets 1: Relevante Grid-Szenarien und ihr Schutzbedarf. (2006)

[26] Team, Condor: *Condor Project.* `http://www.cs.wisc.edu/condor/`, 2010. – [Online; accessed 20-02-2011]

[27] Team, Globus: *Globus Documentation Project*. `http://gdp.globus.org/gt4-tutorial/multiplehtml/ch01s03.html`, 2011. – [Online; accessed 19-02-2011]

[28] Team, Globus: *Globus Documentation Project*. `http://gdp.globus.org/gt4-tutorial/multiplehtml/ch01s01.html`, 2011. – [Online; accessed 10-01-2011]

[29] Team, GridLab: *Grid(Lab) Resource Management System (GRMS)*. `http://www.gridlab.org/WorkPackages/wp-9/`, 2010. – [Online; accessed 25-02-2011]

[30] Team, UNICORE: *UNICORE Project*. `http://www.unicore.eu/unicore/`, 2010. – [Online; accessed 20-02-2011]

[31] Tudor, D. ; Cretu, V. ; Ciocarlie, H.: A view on fault tolerant techniques applied for mediogrid. In: *Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT2007, Cluj-Napoca, Romania*, 2007