Fachhochschule
Bonn-Rhein-Sieg

Fachbereich Informatik
Fachhochschule Bonn-Rhein-Sieg
Grantham-Allee 20
D-53757 Sankt Augustin

Master Thesis

im Studiengang
Master of Science in Computer Science

Portierung einer Anwendung in Grid-Architekturen anhand des Beispiels „DataFinder“

von
Christoph Schmitz

Erstbetreuer: Prof. Dr. Manfred Kaul
Zweitbetreuer: Prof. Dr. Rudolf Berrendorf

08. Oktober 2007

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Sankt Augustin, 9. Oktober 2007

Christoph Schmitz

Abbildungsverzeichnis

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
BMBF	Bundesministerium für Bildung und Forschung
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DLR	Deutsches Zentrum für Luft- und Raumfahrt
HTTP	Hypertext Transfer Protocol
JSF	Java Server Faces
JSR	Java Specification Request
MVC	Model-View-Controller
NFS	Network File System
OGSA	Open Grid Service Architecture
OGSI	Open Grid Service Infrastructure
P2P	Peer-to-Peer
QoS	Quality of Service
SISTEC	Abteilung Simulations- und Softwaretechnik
SOA	Service Oriented Architecture
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier

VO	Virtuelle Organisation
WebDAV	Web-based Distributed Authoring and Versioning
WSDL	Web Service Description Language
WSFL	Web Service Flow Language
WSIF	Web Service Invocation Framework
WSRF	Web Service Resource Framework
WSRP	Web Services for Remote Portlets
XML	Extensible Markup Language

Inhaltsverzeichnis

Kurzfassung

In einem Grid steht Benutzern mit entsprechendem Zugang eine Vielzahl verteilter Ressourcen zur Verfügung. Die daraus entstehenden wirtschaftlichen und technischen Vorteile rechtfertigen die Portierung von bestehenden Desktop-Anwendungen.

Die vorliegende Arbeit befasst sich mit der Fragestellung, welche Einflussfaktoren bei der Portierung von Desktop-Anwendungen in ein Grid eine Rolle spielen können und wie diese in Hinblick auf die Machbarkeit zu bewerten sind.

Basierend auf den zugrunde liegenden Softwarearchitekturen werden Architekturmerkmale von Desktop-Anwendungen identifiziert und Hypothesen darüber entwickelt, welche Aspekte den Portierungsprozess beeinflussen.

Am Beispiel der Portierung der Anwendung „DataFinder“ der Abteilung Simulations- und Softwaretechnik des DLR werden die entwickelten Hypothesen überprüft. Die Erkenntnisse aus der Beispielportierung werden ausführlich dargestellt und anschließend kritisch diskutiert.

Abstract

A Grid provides users with a service for sharing computational power and data storage capacity over the Internet. The resulting economic and technical benefits justify the porting of existing desktop applications into the grid architecture.

This master thesis addresses the question which aspects might influence the porting process and to which degree. Based on the underlying software architecture the characteristics of desktop applications are identified. Subsequently several hypothesis are generated.

The developed hypotheses are validated based on the experiences of porting the application “DataFinder” from the DLR. The results of the porting are being described and critically reviewed.

1 Einleitung

Seit das “Grid-Computing” Anfang der 90er Jahre seinen Kinderschuhen entwachsen ist und sich über das wissenschaftliche Umfeld hinaus verbreitet hat, begegnet man diesem Schlagwort fast so häufig wie dem Begriff “Web” zu den Anfangszeiten des Internets. Vergleichbar sind auch die Erwartungshaltungen, die sich mit dem Grid-Computing verbinden. So wie das Internet den Umgang mit Informationen innerhalb kurzer Zeit revolutionierte, soll das Grid-Computing den Umgang mit Ressourcen, d.h. die Nutzung von Rechnern und Rechnernetzen, revolutionieren. Zur Grid-Community zählen in erster Linie wissenschaftliche Institutionen, die auf in diesem Gebiet forschen. In zunehmenden Maße entdecken aber auch Unternehmen die Eigenschaften des Grid sich. So wird Grid-Computing in der Automobilbranche zur Simulation des Fahrverhaltens von neuen Automobilmodellen eingesetzt. Mit der Verbreitung der Idee des Grids werden auch Ideen für dessen Verwendung entwickelt. Unternehmen setzen Grid-Infrastrukturen beispielsweise innerhalb ihres Intranets ein, um Ressourcen unternehmensweit verfügbar zu machen. Mit der Einführung dieser neuen Technologie tritt häufig die Herausforderung der Integration vorhandener Ressourcen in diese Infrastruktur auf. Ein wichtiger Aspekt ist dabei die Frage, wie eine bestehende Anwendung in ein Grid integriert werden kann.

Die vorliegende Arbeit greift diese Fragestellung auf und spezifiziert sie durch die Eingrenzung auf Desktop-Anwendungen. Konkret soll der Frage nachgegangen werden, ob und wie die Integration bzw. Portierung einer Desktop-Anwendung in eine Grid-Architektur gelingen kann. Dabei wird untersucht, welche Eigenschaften einer Desktop-Anwendung Auswirkungen auf den Prozess der Integration haben und wie diese im Detail aussehen. Dazu wird ein

Ansatz für die Ermittlung dieser Eigenschaften entwickelt. Anhand der Portierung der Anwendung *DataFinder* der Abteilung Simulations- und Softwaretechnik des DLR wird ein konkretes Praxisbeispiel untersucht.

Die Arbeit beginnt in Kapitel Zwei mit einer Annäherung an das Thema Grid-Computing. Grundlegende Begriffe werden geklärt und das Modell des Grids erläutert, auf das sich die Arbeit bezieht. Eine Beschreibung der Architektur und der Technologien des Grids schließen das zweite Kapitel ab.

Im dritten Kapitel wird zunächst der Fragestellung nach gegangen, aus welchen Gründen eine Desktop-Anwendung in ein Grid integriert werden soll. Es folgen eine Definition und eine Beschreibung allgemeiner Eigenschaften einer Desktop-Anwendung. Anschließend wird versucht, generelle Merkmale einer Desktop-Anwendung zu bestimmen, die für den Portierungsprozess ins Grid eine Rolle spielen können. Anhand dieser Ergebnisse werden die Merkmale abschließend auf ihre Auswirkungen in Bezug auf die Integration untersucht.

Das vierte Kapitel beschreibt die Portierung der Anwendung *DataFinder*. Dabei wird die Anwendung beschrieben und die Vorgehensweise bei der Umsetzung vorgestellt. Eine Bewertung des Portierungsprozesses anhand der zuvor ermittelten Merkmale beendet das Kapitel.

Den Abschluß der Arbeit bildet das fünfte Kapitel mit einer Bewertung des entwickelten Ansatzes und einem Ausblick auf zukünftig zu untersuchende Fragestellung.

2 Grid: Architektur und Technologien

2.1 Historie & Definition

Grid-Computing verdankt seinen Namen der Assoziation mit dem Stromnetz, im Englischen *electric power grid*. Das Stromnetz stellt eine Infrastruktur bereit, in der Strom allgegenwärtig ist, aus vielen verschiedenen Quellen kommt und über das Netzwerk an den Endverbraucher geliefert wird. Der Endverbraucher kann den Strom nutzen, ohne wissen zu müssen, welche Quelle ihn erzeugt hat. Er benötigt lediglich eine Schnittstelle mit dem Stromnetz, die in Form der Steckdose gegeben ist. Ähnlich diesem Bild sieht die Vision vom Grid-Computing eine vergleichbare Infrastruktur für Rechen- und Dienstleistung vor, die von einzelnen Ressourcen (Rechner, Datenbanken, Software) über ein Netzwerk (Internet) an den Endanwender in Form von Datenzugriffen und Ressourcennutzung erbracht wird. Hierbei steht das Bild des einfachen Zugriffs durch die Steckdose im Vordergrund. Rechen- und Dienstleistungen sollen ebenso einfach durch gemeinsame Schnittstellen abgerufen werden können.

Die Vorläufer für das Grid-Computing reichen bis in die 60er Jahre zurück, in denen erstmals Ansätze für die Verteilung rechenintensiver Aufgaben beschrieben wurden. Seitdem wurden viele Ideen und Konzepte im Bereich der verteilten Systeme entwickelt, die mit dem Begriff Grid-Computing in Verbindung gebracht werden. Beispiele hierfür sind das Cluster-Computing, Ansätze zum verteilten Rechnen (Distributed Computing) und das Peer-to-Peer (P2P) Konzept. Alle Ansätze weisen aufgrund ihrer Eigenschaften eine Verwandschaft zum Begriff Grid auf, obwohl sie verschiedene Ziele verfolgen.

Eigenschaft	Stromnetz	Grid
Infrastruktur	Verbindung verschiedener Kraftwerke und Verbraucher über Stromleitungen	Verbindung verschiedener Ressourcen über das Internet
Verfügbarkeit	Lückenlose Anbindung aller Haushalte ans Stromnetz	Infrastruktur durch vielfältige Plattformen mit Unterstützung für das Internet
Transparenz	Quelle des Stroms für den Verbraucher unbedeutend	Herkunft der Ressource (z.B. Rechenleistung) unbedeutend
Abbrechnung	Abbrechnung basiert auf der verbrauchten Menge an Strom	Abbrechnung basiert auf der Nutzungszeit für Ressourcen (z.B. Rechenzeit)

Tabelle 2.1: Analogie Stromnetz und Grid, nach [CER07]

- **Cluster Computing** - Unter Cluster Computing versteht man die lokale Vernetzung von meist homogenen Rechnersystemen unter der Kontrolle einer einzigen administrativen Domäne mit dem Ziel, Hochverfügbarkeit und hohe Rechenleistung zu erreichen [BM05, S. 51]. Die Systeme der jährlich erscheinenden Liste der 500 schnellsten Supercomputer [TOP07] sind z.B. Cluster-Systeme.

- **Distributed Computing** - Beim Distributed Computing liegt der Schwerpunkt auf der Lastverteilung einer rechenintensiven Aufgabe auf viele u.U. heterogene Teilnehmer, die freie Kapazitäten für die Bearbeitung einer Teilaufgabe erübrigen können. Die Kontrolle über die Verteilung dieser Teilaufgaben liegt bei einer zentralen Instanz. Eines der ersten Beispiele für Distributed Computing war das Projekt *SETI@home* [UoC07] zur Suche nach ausserirdischen Lebensformen initiiert von der Universität Berkeley in Kalifornien.

- **P2P Computing** - *SETI@home* kann auch als Beispiel für P2P dienen, dann allerdings als Sonderform eines zentralisierten P2P-Netzwerkes. Die ursprüngliche Idee von P2P war der Datenaustausch zwischen zwei gleichberechtigten Partnern (Peer-to-Peer), die aber bald aufgrund der technischen Entwicklung auf eine Vielzahl von Teilnehmern ausgedehnt werden

konnte. Charakteristisch für P2P-Netzwerke ist die Dezentralisierung des Netzwerkes in Bezug auf die Kontrolle und Verteilung der Teilnehmer sowie die Gleichberechtigung aller Teilnehmer, die sowohl als Client als auch als Server fungieren können [HR06, S. 446].

Die Anfang der 90er Jahre aufgekommenen Begriffe *Hypercomputing* und *Metacomputing* lassen sich ebenfalls der Grid-Thematik zuordnen. Sie bezeichneten Ansätze zur Vernetzung von Hochleistungsrechnern über schnelle Datenleitungen mit dem Ziel der Lösung von komplexen Rechenaufgaben durch koordinierte Ressourcennutzung. An dieser Stelle sei speziell auf das I-WAY Projekt [D+96] verwiesen, das maßgeblich zur Entwicklung des späteren Globus Toolkit [TGA07] beigetragen hat. Diese Bemühungen können als Vorläufer für den späteren Begriff Grid-Computing angesehen werden, für den es bis dahin noch keine eigentliche Definition gab. Nach anfänglichen Erfolgen im Bereich der Hochleistungsrechner wurden die Konzepte auf kleinere Maßstäbe übertragen und erreichten schließlich mit den Desktop-Systemen eine potenziell große Nutzergemeinde [RS04]. Zu diesem Zeitpunkt, im Jahr 1999, legten Ian Foster und Carl Kesselmann in ihrem Buch *The Grid: Blueprint for a New Computing Infrastructure* [FK99] erstmals eine Definition der Begriffe Grid und Grid-Computing vor. Mit dieser Definition und der drei Jahre später erschienenen Checkliste [Fos02] für die Eigenschaften eines Grids steht eine Erläuterung der Begriffe Grid und Grid-Computing zur Verfügung, die in der Literatur weitgehend akzeptiert ist. [Tay05, S. 67]

Demnach stellt ein Grid eine skalierbare Infrastruktur zur Integration verteilter, heterogener Ressourcen dar, mit der Zielsetzung eine möglichst verlässliche, konsistente und transparente Nutzung dieser Ressourcen zu ermöglichen. Die Art der Ressourcen umfasst dabei Rechner verschiedenster Bauart und Leistung, Datenspeichersysteme, spezielle Geräte wie Sensoren und Softwareanwendungen. Die lokale Autonomie der Ressourcen bleibt gewahrt, d.h. die Infrastruktur des Grids hat keinen Einfluß auf lokale Entscheidungen und die Ressourcen werden weiterhin von den jeweiligen Eigentümern verwaltet. Die Checkliste von Foster greift die seiner Meinung nach wichtigsten Eigenschaften dieser Definition auf, um zu beschreiben, was ein Grid ausmacht [Fos02].

- Koordinierte Ressourcen, die keiner zentralen Kontrolle unterliegen - Ein Grid integriert und koordiniert Ressourcen und Teilnehmer aus unterschiedlichen administrativen Domänen und adressiert dabei u.a. Aufgaben in Bezug auf Sicherheit, Verfahrensrichtlinien und Mitgliedschaft.
- Verwendung von offenen, standardisierten und allgemeinen Protokollen - Diese Protokolle und Schnittstellen dienen zur Handhabung der einzelnen Aufgaben, wie Authentifizierung oder das Auffinden von auf Ressourcen. Dabei ist es unerlässlich, dass die verwendeten Protokolle offen und standardisiert sind, um keine anwendungsspezifische Infrastruktur zu schaffen.
- Bereitstellung nichttrivialer Dienstgüte (Quality of Service,QoS) - Ein Grid ermöglicht die koordinierte Nutzung seiner Ressourcen um QoS bzgl. Antwortverhalten, Durchsatz oder z.B. Verfügbarkeit zu gewährleisten.

Betrachtet man die oben angesprochen Formen von verteilten Systemen, wird deutlich, dass sie nach dieser Definition nicht als Grid bezeichnet werden können. Ein Cluster erfüllt sicherlich die Forderung nach nichttrivialer QoS, ist aber in den meisten Fällen zentral administriert und verfügt jederzeit über die volle Übersicht über das Systems. Damit würde sich ein Cluster als Ressource innerhalb eines Grids eignen, nicht jedoch als eigenständiges Grid in Sinne der Definition. Projekte aus dem Bereich des Distributed Computing entsprechen in ihrer eigentlichen Bedeutung auch nicht der ersten Eigenschaft, da die Verteilung der Aufgaben zentral gesteuert wird, und gehören so auch nicht zu den Grids. Dieses Problem ist bei traditionellen P2P-Netzwerken durch ihre dezentrale Kontrolle nicht gegeben. Allerdings sind die verwendeten Protokolle oft proprietär und zu spezifisch auf ihren Anwendungsbereich zugeschnitten. [Fos02] Damit erfüllen sie nicht die notwendige Eigenschaft offener und allgemeiner Protokolle.

Die vorliegende Arbeit stützt sich auf die vorangegangene Definition des Grids.

2.1.1 Klassifizierung von Grids

Ähnlich zu den verschiedenen Definitionen zum Begriff Grid, lassen sich in der Literatur verschiedene Klassifizierungen für Grids finden. In [JEF04] findet sich ein Beispiel für eine infrastrukturelle, unternehmensorganisatorische Einteilung unterschiedlicher Grids. Es wird zwischen *Infra-Grid*, *Intra-Grid*, *Extra-Grid* und *Inter-Grid* unterschieden. Das *Infra-Grid* stellt eine auf eine Abteilung eines Unternehmens begrenzte Infrastruktur in einer kontrollierten Umgebung mit definierten Verfahrensregeln und Sicherheitsmaßnahmen dar. Bei diesem sehr zentralen Ansatz liegt der Fokus des Grids auf der Gewährleistung von Performanz spezifischer QoS, wie Durchsatz und Rechenleistung. Die Begrenzung auf eine einzige administrative Domäne steht bei dieser Kategorisierung im Widerspruch zur Definition von [Fos02]. Das *Intra-Grid* beschreibt die Integration von Ressourcen mehrerer verschiedener Abteilungen desselben Unternehmens. Auch hier liegt das Hauptaugenmerk auf der Optimierung der Leistung des Grids, da der administrative Aufwand durch die Zusammengehörigkeit zum selben Unternehmen in Grenzen gehalten wird. Mit *Extra-Grid* ist ein Grid gemeint, dass seine Infrastruktur über mehrere, vertrauenswürdige Partnerunternehmen erstreckt und darum auch als *Partner Grid* bezeichnet wird. Im Gegensatz zu den vorherigen beiden Arten liegt hier der Schwerpunkt auf der Entwicklung von sogenannten Service Level Agreements, um die Verwaltung der Ressourcen regeln zu können. Die Steigerung dessen bildet das *Inter-Grid*. Dieses Grid stellt eine Infrastruktur zur gemeinsamen Nutzung von Ressourcen und Daten in einem öffentlichen Netzwerk mit fremden Unternehmen bereit. Die Auslegung dieses Grids basiert hauptsächlich auf kurzfristigen Nutzungszeiträumen verschiedener Ressourcen und erfordert eine flexible und anpassungsfähige Form der Administration. Diese Art des Grids kommt der Definition von Foster am nächsten.

Eine weitere Unterscheidung von Grids kann anhand ihrer Anwedungsschwerpunkte vorgenommen werden, wie in [JF03] beschrieben. Dabei wird zwischen *Data Grids*, *Computational Grids* und *Provisioning Grids* unterschieden. *Data Grids* dienen vorwiegend der Speicherung und Verwaltung von Daten bzw. Informationen auf einer Vielzahl lose gekoppelter Ressourcen, wie Datenbanken

und Storage-Servern. Aufgaben des Grids sind vor allem der sichere Transport und die effiziente Bereitstellung der Daten durch Mechanismen zur Integritätsprüfung und Replikation. Im Gegensatz dazu dient ein *Computational Grid* in erster Linie der koordinierten Nutzung von verteilten Ressourcen und damit verbundenen Rechen- und Speicherkapazitäten zur Bearbeitung von rechenintensiven Aufgaben. Aufgabe dieses Grids ist vor allem das Scheduling von Arbeitsabläufen. *Provisioning Grids* stellen eine Kombinationen der beiden vorangegangen Arten dar. Diese Grids bieten sowohl den Zugriff auf verteilte Daten als auch auf verteilte Ressourcen und entsprechen damit der Definition von Foster. In [RS02] werden Grids in *Information Grid*, *Resource Grid* und *Service Grid* unterteilt. Als *Information Grid* werden dabei das WWW und P2P-Netzwerke angesehen, da sie jedem angeschlossenen Rechner weltweit Zugriff auf Informationen bzw. Daten bieten auf Basis einer verteilten, dynamischen und dezentralisierten Infrastruktur. Diese Klassifizierung steht im Widerspruch zu der Definition von Foster, wird aber der Vollständigkeit halber aufgeführt. Ein *Resource Grid* dient, ähnlich dem *Computational Grid*, dem Zugriff auf verteilte physikalische Ressourcen, wie Server oder Sensoren. Im Gegensatz zum *Information Grid*, bei dem ein anonymer Benutzer Zugriff auf die bereitgestellten Daten hat, ist hier der Zugriff auf autorisierte Benutzer beschränkt. Aufgabe des Grids ist es, diesen Nutzern einen effizienten und transparenten Zugriff zur Verfügung zu stellen, unabhängig davon, wo sich die Ressourcen befinden. Das *Service Grid* realisiert diese Transparenz auf anwendungsspezifischer Ebene. Es bietet Services und Anwendungen unabhängig von ihrer Implementierung oder zugrundeliegenden Plattform an. Die Kombination von *Service Grid* und *Resource Grid* erfüllt die Definition von Foster.

Die vorliegende Arbeit bezieht sich auf den Aspekt des *Service Grid*.

2.1.2 Definition: Virtuelle Organisation

Zusammen mit den Begriffen Grid und Grid-Computing führt [FKT01] die sogenannte *virtuellen Organisation (VO)* ein. Eine VO bezeichnet eine Menge von Personen bzw. Institutionen, die die koordinierte, gemeinsame Nutzung

ihrer Ressourcen auf Grundlage veränderbarer Richtlinien (Policies), wie Zugangsrechten, vereinbaren. Der Charakter einer virtuellen Organisation ist dabei jederzeit dynamisch in Bezug auf Größe und Dauer, d.h. eine VO kann nur vorübergehend bestehen und es können stetig neue Teilnehmer hinzukommen bzw. wegfallen. Der Anwendungsbereich einer VO ist nicht festgelegt, so dass sich die Struktur und die Art der Teilnehmer je nach Anwendungsfall stark unterscheiden können.

Ein Beispiel für eine eine virtuelle Organisation im kommerziellen Bereich kann ein Autohersteller und seine Zulieferer sein. Sie schließen sich zu einer VO zusammen, um Daten, Software und Rechenleistung auszutauschen, die zur Entwicklung der Modelle eines Prototyps notwendig sind. Auf wissenschaftlicher Ebene können VOen z.B. in der Bioinformatik genutzt werden, um rechenintensive Aufgaben wie die Strukturvorhersage von Molekülen zu beschleunigen. Auch für organisatorische Aufgaben lassen sich virtuelle Organisationen nutzen. So lasse sich z.B. die Teilnehmer einer Lehrveranstaltung als VO definieren, in der die Teilnehmer unterschiedliche Rollen (z.B. Dozent, Hörer) und Rechte (z.B. Zugang zu Lehrmaterial oder Rechnern) haben. [BNS04]

Trotz der großen Unterschiede in Bezug auf den Anwendungsbereich einer VO, finden sich Eigenschaften, die alle Ausprägungen gemeinsam haben. Die Mitglieder einer virtuellen Organisation sind sich mehr oder weniger unbekannt. Die Vertrauensbildung beruht auf eventuell vorhandenen früheren Beziehungen. Alle Mitglieder wollen ihre Ressourcen austauschen, um eine bestimmte Aufgabe zu bearbeiten. Dieser Austausch von Ressourcen geht dabei über einfachen Dokumentenaustausch hinaus und schließt den Zugriff auf verschiedenste Art von Hardware ein. Der Austausch für die Ressourcen unterliegt festgelegten Richtlinien, die sich je nach Teilnehmer innerhalb der VO unterscheiden können. [FKT01]

Die denzentrale Natur einer virtuellen Organisation macht es notwendig, dass die lokale Autonomie der Ressourcen gewahrt bleibt. Die Eigentümer der Ressourcen behalten die Kontrolle über ihre Ressourcen und entscheiden, zu welchem Zeitpunkt und unter welchen Bedingungen (z.B. zu welchen Kosten) sie die Ressourcen innerhalb der VO zur Verfügung stellen. So ist sicherge-

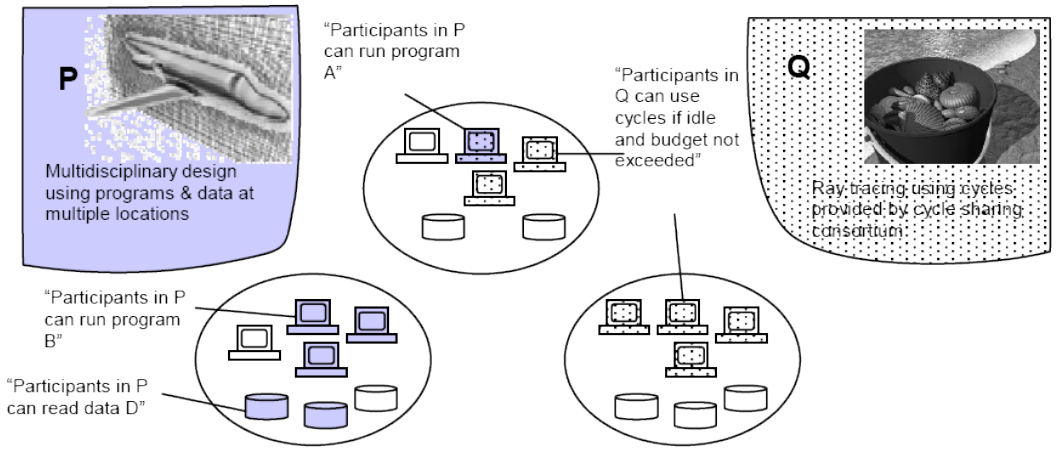

Abbildung 2.1: Szenario einer virtuellen Organisation, aus [FKT01]

stellt, dass Richtlinien mit anderen Teilnehmern der VO ausgehandelt werden können, die den eigenen Bedürfnissen entsprechen. Außerdem erleichtert dies den Beitritt zu einer virtuellen Organisation im Vergleich zu einer Abgabe der Kontrolle über die eigenen Ressourcen. [BNS04]

2.2 Architektur

Der folgende Abschnitt geht auf die Architektur eines Grids und den daraus resultierenden Eigenschaften für Grid-Umgebungen ein. Das Schichtenmodell und die standardisierte Grundlage für Grid-Anwendungen werden in den wichtigsten Punkten dargestellt.

2.2.1 Schichtenmodell

Die Bildung von heterogenen, virtuellen Organisationen erfordert eine Architektur, die in erster Linie die Zusammenarbeit unterschiedlichster Systeme (Ressourcen) innerhalb dieser Organisationen ermöglicht. Daher ist Interoperabilität der zentrale Punkt dieser Architektur. Ohne einen standardisierten Weg über den alle Teilnehmer einer VO miteinander kommunizieren können, ist die Bildung von virtuellen Organisationen, wie im vorangegangenen Abschnitt beschrieben, nicht möglich. Die einzelnen Teilnehmer wären gezwungen Mittel und Wege zu finden, um untereinander kommunzieren zu können, ohne dass sichergestellt wäre, dass diese Maßnahmen auch für andere Teilnehmer funktionieren würden. Die dynamische Natur eines Grids könnte so nicht entstehen und auf diesem Wege entstandene VOen wären nur von begrenztem Nutzen [FKT01]. Dies zeigt die Bedeutung von Interoperabilität bei der Bildung eines Grids. Als vergleichendes Beispiel kann an dieser Stelle das Internet dienen. Das Internet ermöglicht globalen Informationsaustausch zwischen beliebigen Nutzern. In ähnlicher Weise sollen in einem Grid Ressourcen zwischen beliebigen Teilnehmern ausgetauscht werden können (siehe Abschnitt 1).

Um Interoperabilität in einer verteilten und vernetzen Umgebung zu erreichen, werden standardisierte Protokolle eingesetzt. Protokolle sind in einer solchen Umgebung notwendig, da sie festlegen, wie sich die einzelnen Komponenten miteinander verbinden und Daten austauschen, um ein bestimmtes Verhalten zu erreichen, ohne dass dabei vorausgesetzt ist, wie dieses Verhalten realisiert wird. [BNS04] Durch den Fokus auf eine externe Sicht, d.h. auf die Interaktion zwischen den Komponenten, anstatt auf eine interne Sicht, also z.B. spezifische Implementierungen, garantieren Protokolle die notwendige

Flexibilität in Bezug auf die Bildung einer virtuellen Organisation. Der dynamische Charakter von virtuellen Organisationen macht es erforderlich, dass neue Ressourcen schnell und einfach in die Umgebung integriert und wieder aus ihr entfernt werden können. Deswegen darf der Mechanismus, der diese Ressourcen miteinander verbindet, keine tiefgreifenden Änderungen an der Infrastruktur (z.B. die lokale Kontrolle oder bestimmte Verhaltensregeln) der Teilnehmer verlangen. Aufgrund ihrer integrierenden Eigenschaft stellen Protokolle eine Erfüllung dieser Anforderung dar.

In der Grid-Community hat sich ein Schichten-Modell für die Grid-Architektur durchgesetzt, das sich stark an den Internetprotokollen orientiert. Es wurde zum ersten Mal 1998 von Ian Foster in dem Artikel *The Anatomy of the Grid: Enabling Scalable Virtual Organizations* [FKT01] vorgestellt und beschreibt die Basis-Komponenten der Architektur und wie diese Komponenten miteinander interagieren. Komponenten mit ähnlichen Eigenschaften bzw. mit ähnlicher Semantik werden in einer Schicht zusammengefasst. Sie bauen auf den Eigenschaften der Komponenten der darunter liegenden Schicht auf und bieten ihre Eigenschaften an die nächst höhere Schicht an. Im Gegensatz zur Protokoll-Architektur des Internets ist hierbei der Zugriff über benachbarte Schichten hinweg möglich, d.h. höher liegende Schichten können direkt auf tiefere Schichten zugreifen. Einzige Ausnahme dabei bildet die unterste Schicht, das sogenannte Fabric-Layer, auf die nicht direkt zugegriffen werden kann.

Abbildung 2.2 zeigt einen Vergleich zwischen den Schichten der Grid-Architektur und der Protokoll-Architektur des Internets. Die Pfeile symbolisieren den direkten Zugriff auf tiefere Schichten. Wie man sehen kann, verengen sich die mittleren Schichten der Grid-Architektur im Gegensatz zu den oberen und unteren Schichten. Diese “Sanduhr-Eigenschaft” findet ihre Entsprechung in der Protokoll-Architektur des Internets, bei der die beiden mittleren Schichten, Transport- und Internetschicht, den Kern der Architektur bilden und eine kleine Menge von Protokollen definieren, die die Verbindungen zwischen den höheren und tieferen Schichten regeln [FKT01]. Bei der Grid-Architektur stellen die beiden Schichten Connectivity-Layer und Resource-Layer den Kern dar. Sie definieren eine begrenzte Menge an Basis-Protokollen, die einerseits auf verschie-

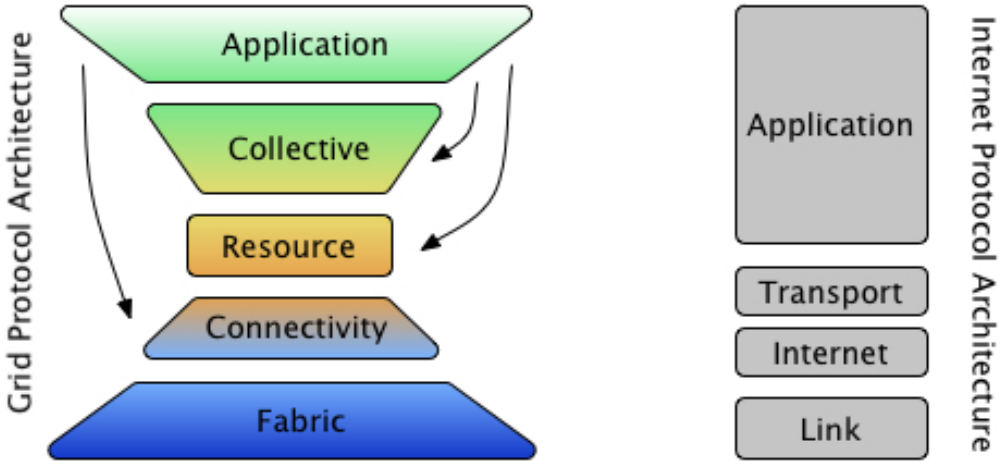

Abbildung 2.2: Vergleich Grid-Architektur und Internet-Architektur, nach [FKT01, S. 206]

denen Ressourcen-Typen der darunter liegenden Schicht (Fabric-Layer) implementiert werden können und anderseits Grundlage für unterschiedliche höherwertige Dienste der darüber liegenden Schichten (Collective- und Application-Layer) dienen können. Die Beschränkung auf einige wenige Basis-Protokolle in der Mitte des Modelles schafft eine Abstraktion zwischen den oberen und unteren Schichten, so dass sich diese beliebig erweitern lassen. Durch diese gemeinsame Schnittstelle lassen sich Änderungen an Komponenten der oberen Schichten vornehmen, ohne dass dies Änderungen an Komponenten der unteren Schichten nach sich zieht oder umgekehrt. Die nachfolgende Auflistung (nach [BNS04]) gibt einen Überblick über die einzelnen Schichten und ihre Eigenschaften.

- Fabric-Layer - Die unterste Schicht in der Grid-Architektur bündelt alle Ressourcen (Rechner, Netzwerke, Sensoren, Speichersysteme) innerhalb des Grids, die für den gemeinsamen Zugriff zur Verfügung stehen sollen. Sie besteht aus Protokollen, welche die Kommunikation der Ressourcen

untereinander ermöglichen. Die Komplexität der Ressourcenteilung in der Schicht kann beliebig hoch sein (Co-Scheduling), wird aber aufgrund der einfacheren Realisierbarkeit bewußt niedrig gehalten.

- Connectivity-Layer - Diese Schicht stellt die Netzwerkschicht innerhalb des Grids dar und besteht aus Kommunikations- und Authentifizierungsprotokollen. Die Kommunikationsprotokolle ermöglichen den Austausch von Daten zwischen den Ressourcen des Fabric-Layer. Die Authentifizierungsprotokolle bauen auf den Kommunikationsprotokollen auf und realisieren sichere Verfahren zur Identifikation von Benutzern und Ressourcen. Diese Schicht ist eine der beiden Kernschichten des Modells.

- Resource-Layer - Das Resource-Layer bildet eine der beiden Kernschichten des Modells dar. Hier wird die Kontrolle einer einzelnen Ressource auf Basis der Protokolle der darunter liegenden Schicht erreicht. Diese Schicht besteht aus Informationsprotokollen, die Auskunft über die Struktur und den Zustand einer Ressource geben, und Verwaltungsprotokollen, die die Zugriffskontrolle für einzelne Ressourcen regeln.

- Collective-Layer - In dieser Schicht werden Protokolle für die koordinierte Nutzung mehrerer Ressourcen definiert. Im Gegensatz zur den eher generellen Protokollen aus dem Resource-Layer, weist das Collective-Layer eine hohe Variabilität in Bezug auf die Generalisierung bzw. Spezialisierung seiner Protokolle auf. Die Art der Protokolle richtet sich dabei nach Art und Größe der virtuellen Organisation, die mit diesen Protokollen arbeitet.

- Application-Layer - Die oberste Schicht in der Grid-Architektur fasst die Anwendungen zusammen, die dem Benutzer die Interaktion mit dem Grid ermöglichen. Die Anwendungen werden unter Verwendung der Protokolle der niedrigeren Schichten - mit Ausnahme des Fabric-Layer - implementiert.

Dieses Modell der Grid-Architektur beschreibt eine Infrastruktur, die dazu geeignet ist, die Anforderungen von virtuellen Organisationen (siehe Abschnitt 2.1.2) zu erfüllen. Aufgrund der abstrakten Beschreibung stellt dieses

Schichten-Modell eine Art Gerüst dar, das den grundsätzlichen Aufbau eines Grids beschreibt und dessen Verhalten modelliert. Wie dieses Verhalten jedoch realisiert wird, d.h. auf Grundlage welcher Technologien die Anforderungen an die jeweiligen Protokolle erfüllt werden können, ist nicht Bestandteil dieses Modelles. Um dem Problem verschiedener, inkompatibler Grid-Anwendungen zu begegnen, ist es daher notwendig, sich auf eine gemeinsame Grundlage zur Realisierung von Anwendungen im Grid zu einigen [FKT01]. Einen solchen Standardisierungsvorschlag stellt die im nächsten Abschnitt beschriebene Open Grid Service Architecture dar.

2.2.2 Open Grid Service Architecture

Ausgehend von der zuvor beschriebenen Grid-Architektur wurde 2002 von einer Arbeitsgruppe um Ian Foster [F+02] ein erster Entwurf für die Open Grid Service Architecture (OGSA) vorgestellt. Ziel war es eine standardisierte Architektur zu schaffen, die als Grundlage für die Entwicklung von Grid-Anwendungen dienen sollte. Die OGSA knüpft an die Beschreibung des Schichten-Modells der Grid-Architektur an, indem sie beschreibt, wie das Verhalten der Protokolle innerhalb der einzelnen Schichten realisiert werden kann. Basierend auf einer Analyse verschiedener Anwendungsfälle aus unterschiedlichen Bereichen, definiert die OGSA funktionale und nichtfunktionale Anforderungen, die charakteristisch für Grid-Anwendungen sind. Um diese Anforderungen zu erfüllen zu können, entwirft die OGSA eine serviceorientierte Architektur (SOA) und beschreibt grundlegende Dienste, die bei der Konzeption von Grid-Systemen eine wichtige Rolle spielen. Auf konkrete technische Details wird dabei verzichtet. Im folgenden Abschnitt wird auf das Modell der OGSA eingegangen.

Anforderungen in OGSA

Die Entwicklung der Anforderungen in OGSA stützt sich auf zahlreiche Anwendungsfälle sowohl aus dem wissenschaftlichen als auch kommerziellen Umfeld [F+04b]. Die Analyse der Anwendungsfälle ergab, dass sich allgemeine, vom Einsatzzweck unabhängige, funktionale und nicht funktionale Anforderungen

für Grid-Umgebungen bzw. Grid-Anwendungen identifizieren lassen. Folgende Anforderungen konnten identifiziert werden (nach [FKS06]).

- Unterstützung von heterogenen Ressourcen (Virtualisierung, Identifizierung, standardisierte Protokolle)
- Koordinierte Nutzung von Ressourcen (Lokale Autonomie, Verfahrensrichtlinien, Nutzungsdaten)
- Job-Ausführung (Scheduling, Überwachung, Workflow-Management)
- Verwaltung von verteilten Daten (Zugriff, Aktualisierung, Verwaltung)
- Sicherheit (Authentifizierung, Autorisierung, Delegation)
- Verfügbarkeit (Fehlerbehandlung, Notfallwiederherstellung)

Für diese Aspekte definiert die OGSA generische Schnittstellen, die den beschriebenen Anforderungen genügen. Die konkreten Implementierungen dieser Schnittstellen sind austauschbar. Für die Realisierung der Schnittstellen entwirft die OGSA eine dienstorientierte Architektur, die im nächsten Abschnitt betrachtet wird.

Das OGSA Framework

Wie der Name *Open Grid Service Architecture* schon sagt, ist der grundlegende Aspekt von OGSA die serviceorientierte Sicht auf die Komponenten innerhalb des Grids. Jede Ressource wird dabei als Dienst repräsentiert. Die Ressourcen werden nicht anhand ihrer physischen Eigenschaften unterschieden, sondern anhand der Funktionen definiert, die sie bereitstellen. Hintergrund für diese Sichtweise ist die dadurch erreichbare Interoperabilität zwischen den einzelnen Komponenten. Die dienstorientierte Sicht erlaubt die Zerteilung des Problems der Interoperabilität in zwei Teilprobleme, die sich leichter lösen lassen [F+02]. Das erste Teilproblem betrifft die Erreichbarkeit eines Dienstes. Damit ein Dienst einen anderen Dienst ansprechen kann, um dessen bereitgestellte Funktionen nutzen zu können, muss er wissen, auf welche Art und Weise er dies tun kann. Dazu ist es nötig, dass alle Dienste über Schnittstellen verfügen, die einer

gemeinsamen Definition zugrunde liegen. Das zweite Teilproblem betrifft die Kommunikation der Dienste untereinander. Damit sie sich gegenseitig austauschen können, muss es einen Weg geben, wie ein Dienst die Schnittstelle eines anderen Dienstes ansprechen kann. An dieser Stelle werden also Protokolle gebraucht, die den Diensten die Kommunikation ermöglichen.

Ein weiterer Vorteil, der sich aus der serviceorientierten Sicht ergibt, ist die einfache Möglichkeit der Virtualisierung der Komponenten bzw. ihrer Funktionalitäten. Durch die Definition einer einheitlichen Schnittstelle eines Dienstes lassen sich konkrete Aspekte wie die Implementierung des Dienstes oder das verwendete Protokoll, hinter dieser Schnittstelle kapseln, d.h. nach außen hin ist nur das Verhalten des Dienstes sichtbar, ohne dass man weiß, wie dieses Verhalten realisiert wird. Dadurch wird die Benutzung des Dienstes transparent. So kann ein Dienst verschiedene Protokolle verwenden abhängig von der Aufgabe, die er erfüllen soll. Bei eine Anfrage an den Prozess des Dienstes vom selben Rechner aus würde etwa ein für die interne Kommunikation optimiertes Protokoll zum Einsatz kommen, z.B. lokale Interprozesskommunikation, während bei einer Anfrage von einem entfernten Rechner ein Protokoll für verteilte Kommunikation, z.B. HTTP, verwendet werden würde. Ebenfalls kann ein Dienst mehrere Implementierungen besitzen, die hinter der Schnittstelle verborgen liegen. So ist es denkbar, dass es eine Referenzimplementierung des Dienstes gibt, die auf Portabilität abzielt und so auf möglichst vielen verschiedenen Plattformen eingesetzt werden kann. Daneben kann es Umsetzungen des Dienstes geben, die bestimmte Systemeigenschaften einer einzelnen Plattform ausnutzen, wodurch diese effizienter werden als die plattformunabhängige Implementierung. Solche plattformspezifischen Dienste können dann bevorzugt genutzt werden, wenn sie existieren. Es ist also entscheidend, dass innerhalb des Grids bekannt ist, welcher Dienst plattformspezifische Funktionalitäten anbietet, damit andere Dienste davon Gebrauch machen können. Ein Beispiel für so eine plattformspezifischen Funktion ist z.B. ein Mechanismus zur Reservierung von Rechenzeit auf einer Ressource. Ein für die Ressourcenverwaltung verantwortlicher Dienst könnte diese Funktion dann ausnutzen, um z.B. eine bessere Lastverteilung zu erreichen.

Ein weiterer Punkt, der mit der Virtualisierung einher geht, ist die einfache Möglichkeit höherwertige Dienste durch die Komposition von einzelnen grundlegenden Diensten zu schaffen. [F+02]

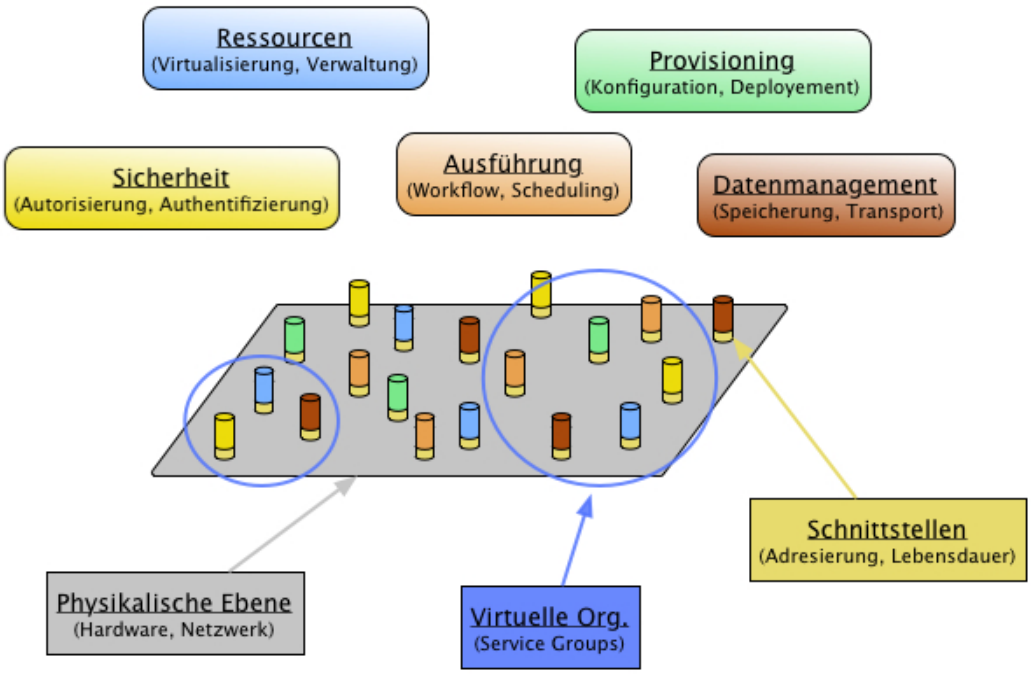

Abbildung 2.3: OGSA Framework, nach [FKS06, S. 14]

Als Technologie für die dienstorientierte Architektur setzt OGSA auf Web Services. Web Services sind eine verbreite und erprobte Technologie zur Umsetzung von verteilten Systeme mit autonomen Teilnehmern und bieten viele Eigenschaften, die bei der Realisierung einer Grid-Umgebung benötigt werden. Die Dienste von OGSA setzen auf Web Services auf, indem sie vorhandene Web Services semantisch erweitern und für das Grid relevante Modifikationen vornehmen. [TT05, S. 85]

Die folgenden Abbildungen illustrieren das Framework von OGSA (siehe Abbildung 2.3) und das Zusammenspiel der einzelnen Dienste innerhalb des Frameworks (siehe Abbildung 2.4).

Die physikalische Umgebung entspricht dem Fabric-Layer des Grids. Hier sind physikalische Ressourcen, wie Netzwerkkomponenten, Rechencluster und

Sensoren angesiedelt. Auf dieser Schicht setzen die Dienste von OGSA auf. Die Dienste werden durch Zylinder repräsentiert. Gleichfarbige Zylinder stehen für eine Menge von Diensten mit ähnlichen Funktionen und Aufgaben. Alle Zylinder bzw. Dienste befinden sich auf einer Ebene. Das soll andeuten, dass alle Dienste miteinander kommunizieren können und z.B. nicht nur Dienste aus benachbarten Schichten. Dies ist wichtig für die Komposition von einzelnen Diensten. Das Paradigma der Komposition ist ein Grundgedanke von OGSA. Aus einer möglichst kleinen Menge von allgemeinen Diensten sollen höherwertige Dienste zusammengesetzt werden. Damit bleibt das Framework gleichzeitig flexibel und robust gegenüber Änderungen. Dies entspricht der Anforderung von wenigen, grundlegenden Protokollen in den Kernschichten des Grids. Die Dienste sind lose untereinander gekoppelt und erfüllen die Anforderungen von OGSA entweder alleine oder im Zusammenwirken mit anderen Diensten (Service Groups). Die lose Kopplung erlaubt es, dass Grid-Umgebungen nur die benötigen Dienste verwenden. Die gesamte Menge aller in OGSA definierten Dienste muss also nicht immer in einer Grid-Umgebung verfügbar sein. Alle Dienste setzen auf gemeinsamen Schnittstellen auf, welche Anforderungen definieren, die jeder Dienst in OGSA erfüllen muss - hier dargestellt durch den gelben Fuß jedes Zylinders. Diese Anforderungen betreffen infrastrukturelle Eigenschaften in einem verteilen System, u.a. Adressierung, Sicherheit und die Verwaltung der Lebensdauer von Diensten [FKS06].

Das Zusammenwirken der Dienste spielt eine entscheidende Rolle in OGSA. Abbildung 2.4 zeigt, wie Dienste in OGSA miteinander interagieren können, um ein gewünschtes Verhalten zu erzeugen. Alle Zylinder einer Ebene stellen eine Menge von Diensten mit Aufgaben aus dem selben Bereich dar, wie Datenmanagement oder Sicherheit. Ein Dienst kann einen anderen Dienst auf derselben Ebene oder in einer anderen Ebene nutzen, um dem Nutzer die gewünschte Funktion anbieten zu können. Ein Dienst kann sich aus anderen Diensten zusammensetzen, um ein komplexeres Verhalten zu erreichen. Ein Dienst kann eine Anfrage an einen anderen Dienst delegieren. Ein Dienst kann sich auf andere Dienst beziehen, um eine Anfrage zu validieren.

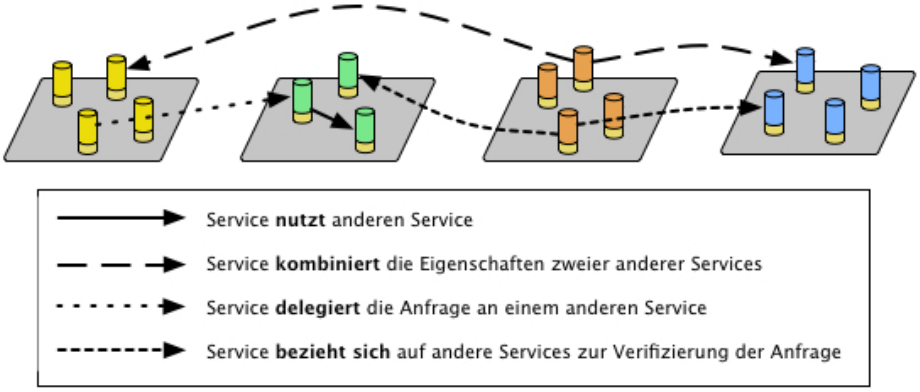

Abbildung 2.4: Beziehungen zwischen OGSA Services, nach [FKS06, S. 15]

Damit die verschiedenen Beziehungen zwischen den Diensten etabliert werden können, müssen alle Dienste grundlegende Anforderungen erfüllen. OGSA definiert diese Anforderungen, um eine Ebene gemeinsamer, grundlegender Funktionalität zu schaffen, die als Basis für alle Grid-Anwendungen bzw. Dienste dienen soll. Die grundlegenden Anforderungen an Dienste im Grid sind [FKS06]:

- Zustandsinformationen - Dienste müssen in der Lage sein den Zustand ihrer Interaktion mit anderen Diensten speichern zu können, wenn diese über mehr als einen Anfrage-Antwort-Zyklus hinausgeht. Eine Komposition von mehreren Diensten, bei der ein Job aus mehreren zusammenhängenden Anfragen an verschiedene Dienste besteht, kann ohne Zustandsinformationen nicht realisiert werden.
- Transiente Instanzen - Dienste müssen dynamisch erzeugt und wieder beendet werden können. Häufig werden Dienste in einer Grid-Umgebung nur für einen bestimmten Zeitraum benötigt, z.B. bei einer Videokonferenz für Punkt-zu-Punkt-Verbindungen, und können dann vernichtet werden [F+02]. Die Handhabung transienter Dienste muss sichergestellt sein.
- Identifizierung und Metadaten - Dienste müssen eindeutig identifiziert und gefunden werden können. Dazu müssen die Informationen, die den

Dienst beschreiben und der Mechanismus zum Auffinden des Dienstes standardisiert werden, um Dienste anhand von bestimmten Merkmalen, wie Lebenszeit oder angebotenen Funktionen, unterscheiden und miteinander vergleichen zu können.

- Verwaltung der Lebenszeit - Dynamisch erzeugte Dienste müssen vernichtet werden können. Dies stellt eine Herausforderung in einer verteilten Umgebung dar. So kann z.B. die Nachricht zur Zerstörung des Dienstes beim Transport verloren gehen oder ein Dienst kann aufgrund eines Fehlers unerwartet beendet werden, ohne dass andere Dienste, die auf ein Ergebnis dieses Dienstes warten, dies merken. Die explizite Zerstörung eines Dienstes muss also sichergestellt sein, um z.B. zu verhindern, dass Ressourcen dauerhaft beansprucht werden. [GCS02]
- Benachrichtigungen - Dienste müssen sich gegenseitig über Zustandsänderungen informieren können. Dies ist z.B. im Fall einer Fehlerbehandlung wichtig.

2.3 Grid-Technologien

Dieser Abschnitt geht auf das Modell der serviceorientierten Architektur und die Technologie der Web Services ein, um zu verdeutlichen, welche Rolle die Verbindung dieser beiden Ansätze bei der Realisierung einer Grid-Umgebung spielt. Zunächst folgt eine Begriffsabgrenzung zwischen SOA und Web Services, da beide Begriffe in der Literatur häufig synonym verwendet werden bzw. der Eindruck entsteht, dass beide Begriffe für dasselbe stehen. Mit SOA wird ein fachliches Architekturmuster beschrieben, das technologieunabhängig verwendet werden kann. Im Gegensatz dazu sind Web Services eine konkrete Technologie, die dieses Muster umsetzen. Prinzipiell läßt sich SOA z.B. auch mittels CORBA oder DCOM realisieren. Der Grund für die häufige Gleichsetzung der beiden Begriffe ist in der weiten Verbreitung von Web Services zu sehen. [RS05]

2.3.1 Service Oriented Architecture

Grundgedanke der SOA ist die Aufteilung einer Anwendung bzw. eines Geschäftsprozesses in atomare Komponenten mit einer eine klar umrissenen, fachlichen Aufgabe. Die Komponenten sind lose miteinander gekoppelt und bieten ihre Funktionalitäten in Form von Services an. Ein Service verfügt über eine fest definierte Leistung, im Idealfall eine unteilbare Anwendungsfunktion, die er entweder unabhängig oder in Verbindung mit anderen Services (Komposition) erbringen kann. Services kommunizieren untereinander durch den Austausch von Nachrichten über standardisierte Schnittstellen. Damit abstrahiert ein Service die Aufgaben der fachlichen Komponente und verbirgt alle Implementierungsdetails. [HR06, S. 231] Die Interaktion zwischen Services erfolgt über einen einheitlichen Aufrufmechanismus, der in der folgenden Abbildung vereinfacht dargestellt ist.

Ein Verzeichnisdienst (Registry) verwaltet alle in einer SOA existierenden Services. Kommt ein neuer Service hinzu, teilt dieser dem Verzeichnisdienst seine Spezifikation (Funktionen, Schnittstellen, unterstützte Protokolle) mit.

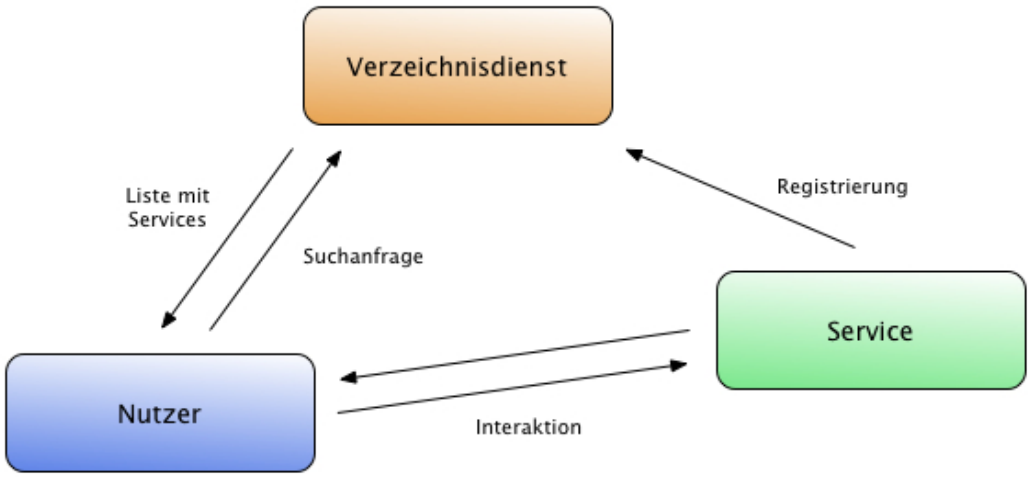

Abbildung 2.5: Interaktionsmodell in SOA, nach [ST05, S. 3]

Ein Nutzer - dies kann ein anderer Service sein - stellt eine Suchanfrage an den Verzeichnisdienst, um einen Service mit den gewünschten Eigenschaften zu finden. Vom Verzeichnisdienst erhält der Nutzer eine Liste mit geeigneten Services. Aus dieser Liste wählt der Nutzer einen Service aus und stellt seine Anfrage an diesen Service über ein gemeinsam genutztes Kommunikationsprotokoll. Der Service beantwortet die Anfrage - entweder mit einem Ergebnis auf die Anfrage oder einer Fehlermeldung.

Ein wichtiger Aspekt der SOA ist die lose Kopplung der Services. Kopplung beschreibt in diesem Zusammenhang wie eng die einzelnen Services miteinander verbunden sind. Die Kapselung aller spezifischen Details macht Services unabhängig voneinander, d.h. die Services bauen nicht aufeinander auf und treffen keine Annahmen über andere Services und deren Verfügbarkeit. Dadurch entsteht eine lose Kopplung zwischen den Services, da sie erst zur Laufzeit entscheiden, welche Daten sie brauchen und wo sie diese anfordern können. Entscheidende Vorteile dabei sind Flexibilität, Skalierbarkeit, Austauschbarkeit und Fehlertoleranz [RS05]. Ein Service kann auf einem beliebigen Server laufen und auch auf andere Server verschoben werden. Solange der Eintrag für den Service im Verzeichnisdienst aktuell gehalten wird, kann jeder Nutzer diesen Service weiterhin verwenden. Services können jederzeit hinzugefügt

bzw. entfernt werden abhängig von den Anforderungen. Vorausgesetzt, dass die Schnittstellen eines Services unverändert bleiben, läßt sich die Implementierung verändern bzw. austauschen, ohne dass Nutzer dieses Dienstes davon betroffen sind. Wenn ein Service oder die damit verbundene Ressource ausfällt, kann der Nutzer mit Hilfe des Verzeichnisdienstes ähnliche Services finden und seine Arbeit fortsetzen.

Den Vorteilen der losen Kopplung stehen Nachteilen gegenüber, die man mit enger Kopplung der Komponenten vermeidet. Hier sind an erster Stelle Performance-Aspekte zu nennen, die man bei der engen Kopplung durch exakte Annahmen und Vorgaben zwischen den Komponenten erreichen kann [ST05]. Da dies aber nicht der Kerngedanke der SOA ist, sondern die flexible Komposition von einfachen Services zu höherwertigen Services, tritt dieser Nachteil in den Hintergrund. Aus diesem Grund eignet sich eine SOA für die Realisierung einer Grid-Umgebung. Dynamische, verteilte Systeme, wie es virtuelle Organisation sind, schließen Annahmen über die Verfügbarkeit von Komponenten von vornherein aus. Schon aufgrund der unterschiedlichen administrativen Bereiche kann sich ein Teilnehmer des Grids nie sicher sein, dass eine bestimmte Ressource zu einem bestimmten Zeitpunkt verfügbar ist. An dieser Stelle trägt die lose Kopplung der Ressourcen innerhalb des Grids dazu bei, dass andere, mit ähnlichen Funktionen ausgestattete Ressourcen bereitgestellt werden können [FKS06].

Eng verbunden mit der losen Kopplung ist der Aspekt des Zustandes eines Services. Ein Service kann zustandslos bzw. zustandsbehaftet sein, was je nach Anforderung gewisse Vor- und Nachteile mit sich bringt. Ein zustandsloser Service behandelt jede Anfrage als unabhängige Transaktion. Dabei spielt es keine Rolle, ob die Anfrage vom selben Nutzer kommt oder nicht. Jede Anfrage wird isoliert von anderen Anfragen betrachtet, so dass alle nötigen Daten jedes Mal übermittelt werden, da der Service keinerlei Daten, z.B. Sitzungsinformationen, zwischenspeichert. Als Beispiel hierfür kann ein Service für die Abfrage von Aktiendaten dienen. Die Abfrage eines Aktienwertes besteht aus der Anfrage des Nutzers und der Rückgabe des Ergebnisses. Danach besteht keinerlei Beziehung mehr zwischen Nutzer und Service. Sie sind also so gesehen lose

miteinander gekoppelt. Die Einfachheit dieses Konzeptes ist ein großer Vorteil von zustandslosen Services. Ein Service kann jederzeit ersetzt oder neugestartet werden. Da er keine Anfragen bezogenen Daten speichert, skaliert er gut und lässt sich leicht überwachen. [ST05]

Für einfache Aufgaben, wie die Abfrage von Aktienkursen, eignen sich zustandslose Services. Komplexere Aufgaben, die aus mehreren Schritten bestehen, erfordern jedoch häufig, dass Zwischenergebnisse gespeichert werden. Dabei speichert der Service Daten aus vergangenen Anfragen ab und greift auf diese in anschließenden Anfragen wieder zu. Nun ist der Nutzer gezwungen seine Anfragen an denselben Service zu übermitteln, da nur dieser die Daten aus den vorherigen Anfragen kennt. Ein Beispiel für die Nutzung eines zustandsbehafteten Service kann die Aufrechterhaltung einer Sitzung sein. Aus Effizienzgründen kann man so den aufwendigen Austausch eines Sicherheitszertifikates bei jedem Anfrage-Antwort-Zyklus durch das Verwenden eines Tokens, das nur Nutzer und Service kennen, vereinfachen. Durch diese gemeinsame Information besteht die Beziehung zwischen Nutzer und Service über mehrere Anfragen hinweg. Beide sind auf diese Weise eng miteinander gekoppelt. Der Vorteil hierbei ist in erster Linie in der Performance zu sehen. Anstatt bei jeder Anfrage ein Sicherheitszertifikat verarbeiten zu müssen, reicht der Austausch eines Tokens. Nachteile dieser Lösung sind zum einen gerade diese enge Verbindung, d.h. das Ersetzen des Services ist gar nicht oder nur schwer möglich, und zum anderen die schlechtere Skalierbarkeit, wenn mehrere Nutzer denselben Service verwenden und deren Nutzungsdaten zwischengespeichert werden müssen. [ST05]

Auch wenn das Konzept der Zustandslosigkeit in einer serviceorientierten Architektur viele Vorteile bietet, lässt es sich häufig nicht einsetzen, da viele Anwendungen den Zugriff auf zustandsbehaftete Daten, wie Nutzerkontexte, voraussetzen [RS05]. Dies trifft auch auf die Dienste in einem Grid zu, die meist kurzlebig und zustandsbehaftet sind [F+02], um die Dynamik und Variabilität in einer solchen Umgebung zu gewährleisten. Ein möglicher Ansatz Zustandsinformationen speichern zu können und dennoch eine lose Kopplung zwischen Nutzer und Service zu erreichen ist in der folgenden Abbildung dargestellt.

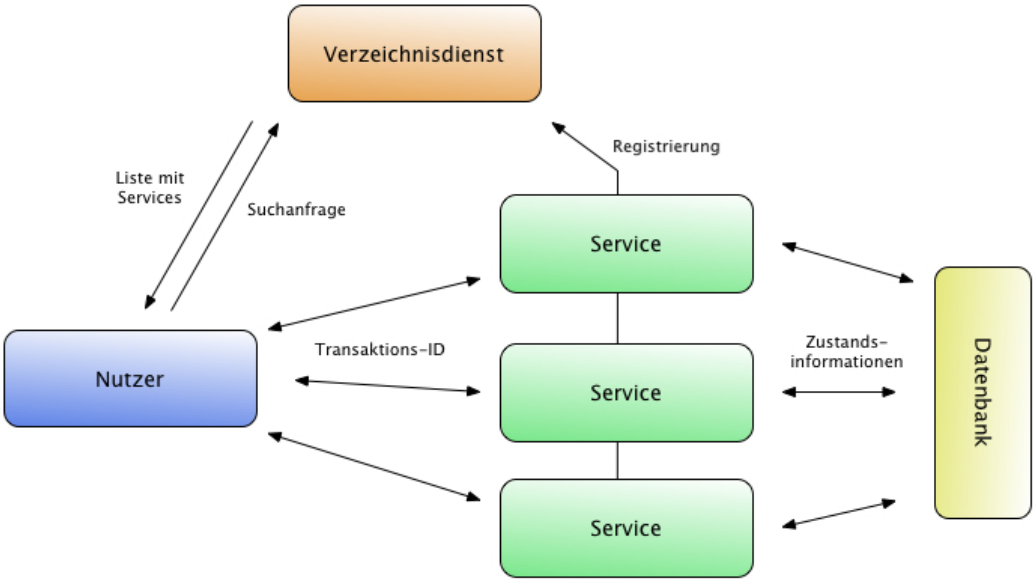

Abbildung 2.6: Mehrstufige Transaktion in SOA, nach [ST05, S. 5]

Um eine aus mehreren Schritten bestehende Transaktion durchführen zu können, muss eine entsprechende Zustandsinformation nach jedem Schritt an den Nutzer zurückgesendet werden. Diese Information muss so aufgebaut sein, dass jeder andere Service, der für diese Aufgabe in Frage käme, mit ihrer Hilfe die laufende Transaktion identifizieren und an dieser Stelle fortsetzen kann. Der Nutzer muss die Information an den Service senden, der den nächsten Schritt der Transaktion bearbeiten soll und dieser Service muss die Information verarbeiten können, unabhängig davon, ob er in vorangegangen Schritten in die Bearbeitung involviert war. In dem Beispiel sind drei verschiedene Services an der Transaktion beteiligt. Die Ergebnisse aus den einzelnen Schritten werden jeweils zusammen mit einer eindeutigen Transaktions-ID in einer Datenbank gespeichert. Die Services und der Nutzer tauschen nun immer diese ID zwischen den einzelnen Schritten aus und jeder Service findet mit Hilfe der ID die aktuellen Daten für seinen Teil der Transaktion. Voraussetzung hierbei ist natürlich, dass alle involvierten Services Zugriff auf die Datenbank haben. Dadurch werden die Zustandsinformationen und die jeweiligen Services, die sie bearbeiten, entkoppelt und die Datenmenge, die zwischen Nutzer und Service

übertragen werden muss, reduziert.

Die lose Kopplung der Komponenten ist eine Eigenschaft der SOA, die den Anforderungen an eine Grid-Umgebung entgegenkommt, gerade im Hinblick auf die Kommunikation zwischen verteilten Services. Die gleichzeitige Forderung nach zustandsbehafteten Services schwächt den Vorteil dieser Eigenschaft ab bzw. macht es notwendig geeignete Konzepte zu entwickeln, um beide Eigenschaften vernünftig zu vereinen. Ansätze für diese Vereinigung werden in Abschnitt 2.3.3 vorgestellt.

2.3.2 Web Services

OGSA setzt Web Services als Technologie für die Umsetzung seines Frameworks ein. Der folgende Abschnitt geht auf die grundlegenden Aspekte von Web Services ein und zeigt die Gründe für ihre Verwendung bei OGSA auf. Web Services stellen eine häufig verwendete Technologie zur Realisierung einer serviceorientierten Architektur dar. Der Erfolg von Web Services basiert auf der Nutzung von standardisierten Internet- und Kommunikationsprotokollen. Die weite Verbreitung im wissenschaftlichen und vorallem auch im kommerziellen Anwendungsbereich haben eine Vielzahl von Anwendungen mit Schnittstellen für Web Services entstehen lassen. Damit haben sich Web Services als eine Art Standard für über das Netzwerk verteilte Anwendungen etabliert. [HR06, S. 177]

Der Grundgedanke bei Web Services ist die gemeinsame Nutzung und Interaktion von Diensten auf geographisch verteilten Rechnern. [TT05, S. 85] In diesem Zusammenhang wichtig sind die Beschreibung der Schnittstellen der Dienste, die Kommunikation zwischen den Diensten und das Veröffentlichen von Diensten. Für diese Zwecke greifen Web Services auf Standards zurück, die im Folgenden beschrieben werden.

- WSDL (Web Service Description Language) - WSDL [C+01] ist eine XML-basierte Beschreibung der Schnittstelle eines Web Service. Solch ein XML-Dokument enthält die zur öffentlichen Schnittstelle gehörenden Operationen mit ihren Signaturen, d.h. die Ein- und Ausgabeparameter.

Die dabei verwendeten Datentypen, *Types*, werden in Form von XML-Schema-Definitionen angegeben. Zur weiteren Beschreibung gehören die *Messages*, eine Auflistung der send- und empfangbaren Nachrichten und die *Bindings*, die die möglichen Zugriffsarten, also konkrete Protokolle und Datenformate, auf den Dienst definieren. Um eine Referenz auf einen Dienst erhalten zu können, werden *Ports* festgelegt, also Adressen für ein Binding, meist in Form von URIs . Mit Hilfe dieser beschreibenden Informationen kann ein Dienst eindeutig beschrieben, identifiziert und lokalisiert werden. [Tay05, S. 222 ff.]

- SOAP - SOAP [G+07b] ist ein XML-basiertes Netzwerkprotokoll für den Nachrichtenaustausch und entfernte Methodenaufrufe (RPC) in verteilten, denzentralisierten Umgebungen. SOAP realisiert dies durch den Entwurf von Regeln für den Nachrichtenaufbau, d.h. die Abbildung und Interpretation der Daten, ohne dabei einschränkend auf die Art der übertragenen Daten zu wirken. Der Mechanismus, den SOAP dabei verwendet, gleicht einem Umschlag für die eigentlichen Daten. Eine SOAP-Nachricht besteht aus zwei Teilen, *Header* und *Body*, die in einem *Envelope* genannten XML-Element enthalten sind. Die anwendungsspezifischen Daten sind im *Body* enthalten, während der *Header* optionale Kontrollinformationen für die Anwendung beinhalten kann. Das zugrundeliegende Transportprotokoll ist von SOAP nicht vorgegeben, so dass theoretisch jedes beliebige Transportprotokoll zum Einsatz kommen kann. Allerdings hat sich hier in dem meisten Fällen HTTP bzw. HTTPS durchgesetzt, nicht zuletzt aufgrund seiner Verbreitung und der einfachen Handhabung mit Firewalls. [Tay05, S. 217 ff.]

- REST (REpresentational State Transfer Architecture) - Obwohl REST [Fie00] eine Architektur beschreibt und keine Technologie darstellt, soll es an dieser Stelle aufgeführt werden, da es interessante Ansätze für die Realisierung von Web Services liefert. REST definiert die Architektur für ein verteiltes Hypermedia-System, in dem jeder Bestandteil als eindeutig identifizierbare Ressource betrachtet wird und Zustandsänderungen über den Transfer von Repräsentationen erfolgen. Am Beispiel des

WWW wird dieses Prinzip deutlich. Websites, Bilder, Skripte sind über einen URI erreichbare Ressourcen.Stellt ein Client eine Anfrage an einen Server, schickt dieser ein HTML-Dokument (die Repräsentation) an den Client, der daraufhin auf eine neue Seite navigiert und so seinen Zustand verändert. Dieses Bild beschreibt drei grundlegende Prinzipien von REST: identifizierbare Ressourcen, Verwendung von Hypermedia und Ressourcen-Repräsentation. Das vierte wichtige Prinzip ist die zustandslose Kommunikation zwischen Client und Server durch selbstbeschreibende Nachrichten. Alle Ressourcen in REST kommunizieren über eine generische Schnittstelle, mit der alle Anwendungsfälle abgedeckt werden können. Im WWW entspricht dies den HTTP-Operationen GET, POST, PUT und DELETE. REST kann in Bezug auf Web Services die Kommunikation ohne eine zusätzliche Schicht wie SOAP realisieren und damit den Overhead deutlich senken. [Bay02]

- UDDI (Universal Description, Discovery and Integration) - UDDI [OAS07b] stellt eine Menge von SOAP-Schnittstellen und Informationsmodellen zur Verfügung, die das Veröffentlichen, Suchen und Abonnieren von konkreten Diensten und Informationen zu Dienstanbietern ermöglichen. Der so realisierte Verzeichnisdienst kann mit einem Telefonbuch verglichen werden, in das neue Web Services per WSDL-Dokument eingestellt werden und bestehende Web Services gefunden werden können. UDDI unterscheidet dabei drei Arten von Informationen - den Anbieter des Dienstes (Unternehmen mit Kontaktdaten), den Dienst selber (das WSDL-Dokument) und technische Spezifikationen des Dienstes (z.B. Kommunikationsprotokoll). Durch die konkrete Zuordnung von Dienstname zu Dienstimplementierung und einem Suchmechanismus für bestimmte Dienste ermöglicht UDDI das eindeutige Auffinden und Abrufen jedes gespeicherten Dienstes. Das Konzept von UDDI setzt auf wenige, zentralisierte Verzeichnisse, in denen viele verschiedene Anbieter ihre Dienste publizieren können. Nachteilig wirkt sich dabei die aufwendige Pflege solcher Verzeichnisse auf die Güte der Suchergebnisse aus, wenn ein Verzeichnis schlecht moderiert wird. Generell ist der Aufbau einer UDDI-Umgebung als komplex zu bewerten und kommt deshalb für viele kleinere Unter-

nehmen nicht in Frage. Aus diesem Grund hat sich UDDI bis heute noch nicht als Standard für Verzeichnisdienste bei Web Services durchsetzen können. [Tay05, S. 228 ff.]

- WS-Inspection - Einen anderen Weg geht das Konzept des WS-Inspection [IBM07]. Im Gegensatz zu UDDI arbeitet WS-Inspection mit vielen kleinen, dezentralen Verzeichnissen, in denen nur ein oder wenige Anbieter ihre Dienste veröffentlichen. Damit soll dem Problem der aufwendigen Pflege der Verzeichnisse bei UDDI begegnet werden. WS-Inspection arbeitet dokumentenbasiert. Auf der Website des Anbieters liegt ein in WSIL (WS-Inspection Language) gehaltenes Dokument, in dem Informationen über die angebotenen Dienste beschrieben sind. Diese Informationen können aus direkten Verweisen auf einzelne Dienste (URL zu einem WSDL-Dokument) oder aus Verweisen auf andere WSIL-Dokumente oder auf UDDI-Verzeichnisse bestehen. WSIL-Dokumente lassen sich strukturiert in Verzeichnissen ablegen, was eine bessere Übersicht über die angebotenen Dienste erlaubt. Der Zugriff auf WS-Inspection Verzeichnisse erfolgt über standardisierte Internetprotokolle, wie HTTP. Ein entscheidender Nachteil gegenüber UDDI ist bei WS-I die Tatsache, dass dem Nutzer die URL zu den Verzeichnissen der Anbieter bekannt sein muss. Einen vergleichbaren, globalen Suchmechanismus gibt es hier nicht. Vorteilhaft gegenüber UDDI ist jedoch die geringere Komplexität des Ansatzes, so dass sich ein WS-I basiertes Verzeichnis einfacher realisieren lässt. [BN01]

Mit Hilfe dieser und weiterer Standards ermöglichen Web Services die Virtualisierung von Ressourcen [ST05]. Die einheitliche Schnittstellenbeschreibung eines Services und der standardisierte Nachrichtenaustausch lassen unterschiedliche Ressourcen miteinander kommunizieren - eine Eigenschaft, die in einer Grid-Umgebung unbedingt vorhanden sein muss und von OGSA gefordert wird. Abbildung 2.7 illustriert das Konzept der Virtualisierung mit Web Services.

Ein Administrator verwaltetet eine Menge unterschiedlicher Ressourcen durch eine Virtualisierungsebene bestehend aus Web Services, die alle eine gemein-

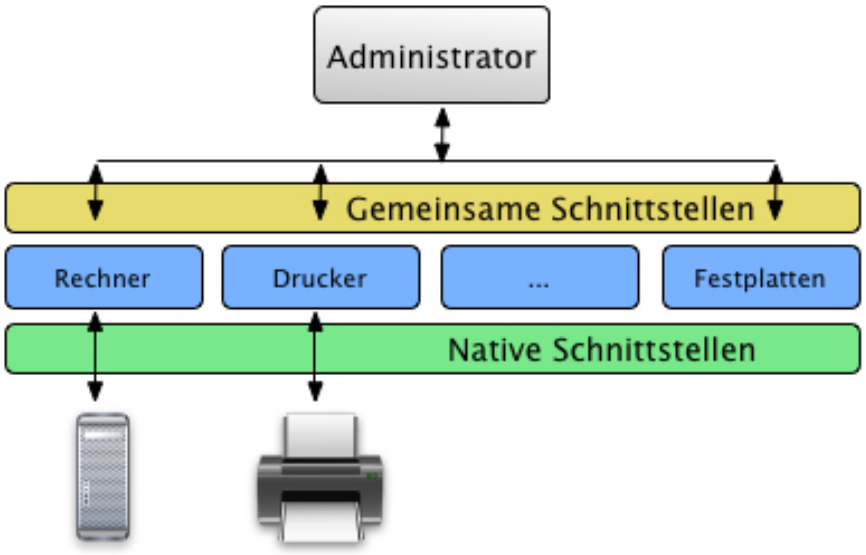

Abbildung 2.7: Zugriff auf Ressourcen durch virtualisierte Schnittstellen, nach [ST05, S. 7]

same Schnittstelle für die dahinter liegende Ressource anbieten. Hinter dieser Virtualisierungsebene kommunizieren die Web Services über die native Schnittstelle mit der eigentlichen Ressource. Das Resultat der Anfrage wird wieder über die gemeinsame Schnittstelle in einem standardisierten Format an den Administrator geschickt.

Neben der Virtualisierung bieten Web Services zwei weitere Eigenschaften, die bei der Entwicklung von OGSA dazu geführt haben, auf diese technologische Basis zu setzen. In einer Grid-Umgebung müssen Dienste dynamisch erstellt und gefunden werden können, d.h. es werden Mechanismen für das Veröffentlichen und Auffinden von Schnittstellendefinitionen und Beschreibungen von Dienstimplementierungen zur dynamischen Generierung von Dienstinstanzen benötigt. Mit WSDL und UDDI bzw. WS-I stehen Standards zur Verfügung, die diese Anforderungen erfüllen [F+02]. Die weite Verbreitung und der Standardisierungsgrad von Web Services eröffnet einem auf dieser Technologie basierenden Framework die Möglichkeit viele schon bestehende Anwendungen und Dienste zu nutzen. So stehen z.B. mit der Web Services Flow Language

(WSFL) oder dem Web Service Invocation Framework (WSIF) Standards für die Komposition und den Aufruf von Web Services und mit Microsoft .NET und Apache Axis bekannte Laufzeitumgebungen für Web Services zur Verfügung. [Tay05, S. 243]

2.3.3 Die Spezifikationen OGSI und WSRF

Die Open Grid Service Architecture stellt einen allgemeinen Rahmen für die Realisierung einer Grid-Umgebung bereit. Sie beschreibt die notwendigen Komponenten einer Grid-Umgebung und klassifiziert grundlegende Dienste, die in einem Grid vorhanden sein müssen. OGSA setzt eine dienstorientierte Architektur ein und adaptiert dabei die Vorteile von Web Services. Im Unterschied zu Web Services, die häufig als persistent und zustandslos betrachtet werden [RS04], sind Dienste in OGSA meist transient und zustandsbehaftet ausgelegt. Daher bestehen wesentliche Unterschiede zwischen Web Services und den in OGSA definierten Diensten in den jeweiligen Methoden zur Adressierung, zur Verwaltung der Lebenszeit und zur Handhabung von Zustandsinformationen. An dieser Stelle bleibt OGSA jedoch auf einer abstrakten Ebene und liefert keine konkreten Vorschläge zur Handhabung dieser Unterschiede. Zu diesem Zweck wurden die beiden Standards Open Grid Services Infrastructure (OGSI) und Web Service Resource Framework (WSRF) entworfen, die eine Spezifizierung von OGSA darstellen.

OGSI [C+03] stellt die erste Spezifizierung von OGSA dar und wurde erstmals im Jahr 2002 von einer Gruppe um Ian Foster veröffentlicht. In OGSI werden alle Ressourcen in einem Grid als sogenannte Grid Services betrachtet. Grid Services sind eine syntaktische Erweiterung von Web Services und stellen Schnittstellen für Basisfunktionen wie Adressierung, dynamische Instanziierung oder Benachrichtigung bereit. Wesentliche Punkte bei OGSI sind die Behandlung der Services als zustandsbehaftete Instanzen, die dynamische Erzeugung von Instanzen durch eine Factory, ein zweischichtiges Referenzierungssystem mit persistenten und temporären Netzwerkzeigern sowie die Strukturveränderung in der WSDL Syntax [Tay05, S. 246-248]. Obwohl im Jahr 2003 mit dem Globus Toolkit Version 3 eine Referenzimplementierung

erschien, konnte sich OGSI nicht durchsetzen. Kritik wurde vor allem laut in Bezug auf die zu objektorientierte Sichtweise auf die Services, die unsaubere Trennung der Einzelteile der Spezifikation und die tiefgreifende Änderung an der WSDL-Struktur [C+04a]. Die erhoffte Konvergenz zwischen den Bereichen Web und Grid blieb aus, so dass eine neue Spezifikation entworfen wurde.

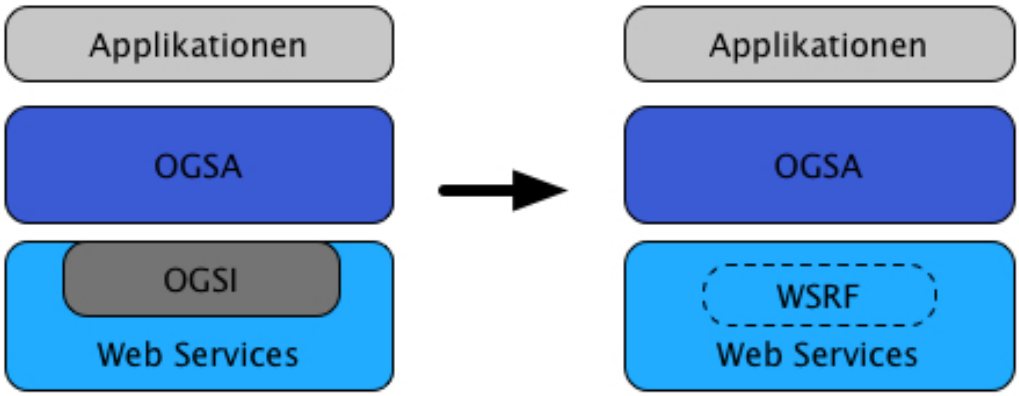

Abbildung 2.8: Vergleich von OGSI und WSRF, nach [Sot04]

Das im Jahr 2004 veröffentlichte Web Service Resource Framework (WSRF) [C+04b] stellt die Weiterentwicklung von OGSI dar. WSRF stellt im wesentlichen ein Refactoring von OGSI dar, das die Bestandteile von OGSI neu strukturiert und in eigenständige Module kapselt. Ein entscheidender Unterschied zu OGSI ist die Vermeidung von Strukturveränderungen in der WSDL-Syntax und die damit verbundene Kompatibilität zu restlichen WS-Familie. Die weiteren grundlegenden Unterschiede sind in der Trennung von Web Service und Zustand durch das sogenannte *Implied Resource Pattern* [F+04a] und der Verwendung eines standardisierten Referenzierungskonzeptes auf Basis der WS-Addressing Spezifikation [B+04] zu sehen [JF03]. Durch die direkte Adressierung der Probleme von OGSI hat sich WSRF zu einem Standard auf diesem Gebiet entwickelt und stellt einen vorläufigen Konvergenzpunkt der Standardisierungbemühungen der Grid- und WS-Gemeinschaft dar [B+05].

3 Integration von Desktop-Anwendungen

Dieses Kapitel beschreibt die Entwicklung eines Ansatzes für die Bewertung der Eigenschaften einer Desktop-Anwendung im Hinblick auf ihre Auswirkungen auf die Integration in ein Grid. Zunächst werden die Gründe für die Integration einer Anwendung im Allgemeinen in ein Grid beleuchtet. Anschließend folgt eine Definition von Desktop-Anwendungen. Darauf basierend werden Kriterien für die Bewertung der Komplexität des Portierungsprozesses definiert. Die Kriterien werden beschrieben und auf ihre Eigenschaften in Bezug auf Desktop- und Grid-Anwendungen untersucht.

3.1 Gründe für eine Integration

Wie in der Einleitung der vorliegenden Arbeit (siehe Abschnitt 1) erwähnt, kann eine Grid-Infrastruktur dazu genutzt werden, vorhandene Ressourcen in das Grid zu integrieren. Dabei kann ein Ziel der Integration Verfügbarkeit sein. Im Folgenden wird erläutert, welche weiteren Ziele eine Integration noch verfolgen kann. Es wird zum einen auf die möglichen Vorteile einer Integration in ein Grid eingegangen und zum anderen auf die Gründe der Integration, also warum ein Interesse besteht, eine bestehende Anwendung in ein neues System zu überführen.

Wenn bereits eine Grid-Infrastruktur innerhalb des Unternehmens besteht, bietet sich die Überlegung einer Integration an. In diesem Fall kann auf ein

schon existierendes System zurückgegriffen werden. Der Nutzen der Einführung einer Grid-Infrastruktur zum Zweck der Integration kann nicht generell bewertet werden. Es muss von Einzelfall zu Einzelfall entschieden werden, wie der Aufwand einer Realisierung einer Grid-Infrastruktur im Verhältnis zum Nutzen einer in diese Infrastruktur integrierten Anwendung steht. Hier liefert die vorliegende Arbeit erste Hinweise in Bezug auf den Nutzen einer Integration, so dass sie als Ausgangspunkt für eine derartige Abschätzung dienen kann.

Ein Ziel bei der Integration kann die verbesserte Performance einer Anwendung sein. Wenn die Anwendung schon einen parallelen Prozessfluss aufweist bzw. wenn es gelingt, die Anwendung in einzelne unabhängige Teile zu trennen, kann sie auf verschiedenen Ressourcen im Grid verteilt und die Anwendung insgesamt beschleunigt werden. Die Integration vieler wissenschaftlicher Anwendungen macht sich dieses Prinzip zu Nutze [JF03].

Ein damit verbundenes Ziel ist die Erhöhung der Verfügbarkeit einer Anwendung durch die Integration, wie zuvor erwähnt. Unabhängig davon, ob die Anwendung parallel oder sequenziell ausgeführt wird, führt eine Aufteilung der Anwendung in einzelne Teile dazu, dass diese Teile theoretisch auf jeder geeigneten Ressource im Grid bearbeitet werden können. Der Ausfall einer Ressource würde so nicht gleich einen Ausfall der Anwendung zur Folge haben. Ein weiteres durch diese Aufteilung realisierbares Ziel ist die Steigerung der Skalierbarkeit. Sobald leistungsstärkere Ressourcen im Grid verfügbar sind, können Teile der Anwendung diese ausnutzen, um wachsende Eingabenmengen weiterhin effizient bearbeiten zu können.

Ein wirtschaftliches Ziel für die Integration ist der reduzierte Aufwand für die Wartung und Aufrechterhaltung der Anwendung. In einem großen Unternehmen mit einer Vielzahl von Arbeitsplatzrechnern bedeutet die Pflege einer lokalen Anwendung auf diesen Rechnern einen hohen Aufwand. Der Installationsprozess und die sich anschließenden Updates müssen auf jedem Rechner vollzogen werden, der diese Anwendung unterstützen soll. Neben der großen Menge kann auch die Heterogenität der Rechner zusätzlichen Aufwand verursachen. So gilt es z.B. Versions- oder Hardwarekonflikte in Bezug auf die Anwendung zu beachten. Dieser Verwaltungsaufwand wird durch die Integra-

tion ins Grid reduziert, da sich die Wartung und Pflege nur noch auf die Teile der Anwendung innerhalb des Grids bezieht.

Wird als Benutzerschnittstelle für das Grid ein Grid-Portal (siehe 4.2.3) verwendet, verringert sich der Aufwand zusätzlich, da dann nur noch ein Browser für den Zugriff auf die Anwendung benötigt wird. Zudem vereinfacht sich die Handhabung der Sicherheitsrichtlinien bei den Arbeitsplatzrechnern, da die Kommunikation über Ports läuft, die in einer Firewall standardmäßig geöffnet sind. Ein positiver Nebeneffekt bei der Verwendung von Grid-Portalen ist die potenziell erhöhte Verfügbarkeit der Anwendung in Bezug auf die Vielfalt der Endgeräte, die Zugriff auf die Anwendung haben.

Neben den technischen Gründen für eine Integration einer bestehenden Anwendung gibt es generelle Gründe, die unabhänig sind von der Anwendung und der Zielplattform, in die integriert werden soll. In erster Linie stellt die Integration einer bestehenden Anwendung einen Investitionsschutz für das Unternehmen dar. Häufig ist die Anwendung der einzige Ort, an dem die Geschäftslogik dokumentiert ist. Wenn sich die Entwickler der Anwendung nicht mehr innerhalb des Unternehmens befinden, gibt es somit kein anderes Wissen über die Anwendung. Die Anwendung ist meist ein wichtiger Bestandteil im Produktivbetrieb und muss auch im neuen System eingesetzt werden können. Der Betrieb der Anwendung kann dabei nicht für die Zeit des Übergangs unterbrochen werden. Eine parallele Neuentwicklung der Anwendung bedeutet mehrfache Kosten, da neben den Kosten für die Neuentwicklung gleichzeitig die Kosten für die Aufrechterhaltung der alten Anwendung aufgebracht werden müssen. Aus diesen Gründen besteht häufig das Interesse der Integration einer bestehenden Anwendung in in neues System. [HR06, S. 172]

3.2 Definition von Desktop-Anwendung

Eine Suchanfrage nach dem Begriff *Desktop-Anwendung* in einer WWW liefert eine Fülle von verschiedenen Ergebnissen. Auch in der Literatur findet man die Verwendung dieses Begriffes in unterschiedlichen Kontexten. Die Bedeutung des Begriffes variiert dabei stark und wird je nach Thematik anders aufgefasst. Allgemein ausgedrückt kann man eine Desktop-Anwendung als Anwendung definieren, die in einer Desktop-Umgebung läuft. Desktop bezeichnet dabei - in Anlehnung an das Bild eines Schreibtisches - die graphische Arbeitsfläche eines Betriebssystems.

Mit dieser Definition wird zugleich eine mögliche Eigenschaft von Desktop-Anwendungen formuliert - deren graphische Benutzeroberfläche. Da es aber auch Anwendungen gibt, die in einer Desktop-Umgebung gestartet werden können, jedoch über keine eigene Benutzeroberfläche bzw. nur eine Kommandozeile verfügen, stellt sich die Frage nach der Allgemeingültig dieser Eigenschaft.

Da der Desktop prinzipiell die Schnittstelle ist, über die ein Benutzer mit einer Anwendung interagiert, stellt sie sich ihm zunächst immer als Desktop-Anwendung dar. Legt man diese Art des Zugriffes als eine Eigenschaft für Desktop-Anwendungen fest, fallen so gut wie alle Anwendungen in diese Kategorie.

Das Beispiel einer Software für Textverarbeitung und einer verteilten Buchhaltungssoftware, die beide über den Desktop bedient werden, verdeutlicht gleichzeitig die Problematik dieser Definition und zeigt eine weitere, mögliche Charakteristik von Desktop-Anwendungen auf: Während die Logik einer Textverarbeitungsanwendung üblicherweise auf dem Rechner liegt, auf dem die Desktop-Umgebung läuft, kann die Geschäftslogik der Buchhaltungssoftware auf entfernten Applikationsservern laufen, so dass lediglich das Frontend zur Bedienung dem Desktop zuzuordnen ist. Die Buchhaltungssoftware wegen dieses Zugriffes als Desktop-Anwendung zu beschreiben, ist fraglich.

Häufig findet man in der Literatur die Abgrenzung zwischen Desktop-Anwendungen und webbasierten Anwendungen, die genau diesen Unterschied in der Lokalität der Geschäftslogik betont. Abgesehen von dieser Abgrenzung und trotz der

häufigen Verwendung des Begriffes, finden sich in der Literatur nur sehr wenige Versuche einer genauen Definition einer Desktop-Anwendung. In [L+98] sind die Eigenschaften von Desktop-Anwendungen charakterisiert. Auf diese Charakterisierung stützt sich die vorliegende Arbeit, um den Begriff der Desktop-Anwendung einzugrenzen. Er wird nachfolgend in dieser Bedeutung verwendet. Eine Desktop-Anwendung zeichnet sich durch folgende Eigenschaften [L+98] aus, die je nach Anwendung unterschiedlich stark ausgeprägt sein können:

- Lokale Installation und Ausführung - Eine Desktop-Anwendung ist lokal auf einem Rechner installiert, d.h. die Gesamte bzw. der Hauptteil der Geschäftslogik liegt auf diesem Rechner und wird dort ausgeführt.
- Graphische Benutzeroberfläche - Eine Desktop-Anwendung besitzt eine graphische Benutzeroberfläche (GUI), die über die Funktionalität eines Terminal-Fensters hinausgeht und über die der Benutzer die Anwendung bedient.
- Interaktives Verhalten - Eine Desktop-Anwendung verhält sich interaktiv, d.h. der Arbeitsablauf der Anwendung wird durch Eingaben des Benutzers gesteuert und kann möglichst jederzeit durch Interaktion des Benutzers in eine andere Richtung gelenkt bzw. abgebrochen werden.
- Hohe Funktionsvielfalt - Eine Desktop-Anwendung unterstützt häufig eine Vielzahl an Funktionen, um verschiedene Aufgaben zu erledigen. (Abgrenzung zu einem Batch-Programm, das oft nur eine einzige Aufgabe bearbeitet.)
- Abhängigkeit von Bibliotheken - Eine Desktop-Anwendung setzt meistens auf einer Menge von plattformspezifischen Bibliotheken auf, um die Eigenschaften des zugrundeliegenden Betriebssystems auszunutzen.
- Nebenläufigkeit - Eine Desktop-Anwendung zeichnet sich häufig durch einen nebenläufigen Programmfluss aus, um u.a. interaktives Verhalten zu realisieren.

3.3 Architekturmuster bei Desktop-Anwendungen

Die Entwicklung genereller Vorgehenweisen für die Portierung von Desktop-Anwendungen in Grid-Architekturen darf sich nicht auf spezifische Eigenschaften einer Anwendung, wie die Implementierungssprache, stützen, sondern muss auf allgemeinen Merkmalen basieren, um auf jede Art von Desktop-Anwendungen anwendbar zu sein. Die Beschreibung allgemeiner Merkmale einer Software wird durch das Konzept der Softwarearchitektur und den damit verbundenen Architekturmustern erreicht. Softwarearchitektur beschreibt Software auf einer hohen Abstraktionsebene und definiert ihre essenziellen Bestandteile und die Beziehung zwischen diesen. Damit bildet sie grundlegende und globale Eigenschaften der gesamten Anwendung ab, ohne sich auf lokale Aspekte zu beziehen. In der Literatur lassen sich viele Definitionen für den Begriff Softwarearchitektur finden. An dieser Stelle sei die folgende erwähnt:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." [BCK05, S. 19]

Hier werden die drei wichtigsten Eigenschaften deutlich, die von der Softwarearchitektur einer Anwendung beleuchtet werden: Die Komponenten, aus denen die Software besteht, die Beziehungen zwischen diesen Komponenten und das durch diese Beziehungen erreichte Verhalten. Damit eignet sich Softwarearchitektur für die Beschreibung allgemeiner Merkmale einer Desktop-Anwendung.

In Anlehnung an die Entwurfsmuster [QC96] aus dem Software Engineering werden Architekturmuster für die Abbildung einer Softwarearchitektur genutzt. Sie bilden also typische Fälle bei einer Architekturentscheidung ab, die sich in der Praxis bewährt haben [Has06]. Die in Teil 2.3.1 erläuterte serviceorientierte Architektur ist z.B. ein häufig verwendetes Architekturmuster, das bei der Entwicklung eines verteilten Systems wie einem Grid Anwendung findet. Je nach Anwendungsbereich leisten Architekturmuster eine andere Aufgabe (Strukturierung der Software, Erweiterbarkeit der Software) und lassen

sich in übergeordneten Kategorien zusammenfassen.

- Strukturierungsmuster: Diese Muster helfen bei der Organisation einer Software, die aus vielen einzelnen Komponenten besteht. Das Muster hilft die Komponenten in verschiedene Subsysteme zu gliedern, die untereinander kommunizieren, und damit die Komplexität der Software zu reduzieren. Zu diesen Muster zählen u.a. die Schichtenarchitektur, das sogenannte Blackboard Muster oder das Client-Server Modell.
- Verteilungsmuster: Diese Muster helfen bei der Verwendung verteilter Ressourcen und Dienste in Netzwerken. Hier sind vor allem SOA und das Broker-Muster zu nennen.
- Interaktionsmuster: Diese Muster helfen bei der Strukturierung der Interaktion des Benutzers mit der Anwendung. Ein bekanntes Beispiel an dieser Stelle ist das MVC-Muster .
- Erweiterungsmuster: Diese Muster helfen, die Fähigkeit der Erweiterbarkeit und Anpassungsfähigkeit eine Software zu erhalten bzw. zu unterstützen. Beispiele hierfür sind die Muster Reflexion und Mikrokernel.

Diese Kategorisierung nach [B+98] erhebt keinen Anspruch auf Vollständigkeit und stellt nur eine mögliche Einteilung der Architekturmuster dar. Es wird deutlich, dass Architekturmuster immer ein bestimmtes Qualitätsmerkmal beschreiben, je nach dem in welchem Kontext sie eingesetzt werden. Das Modell der SOA könnte z.B. auch den Erweiterungsmustern zugeteilt sein, da SOA aufgrund seiner Eigenschaften (lose Kopplung) auch in diesem Kontext eingesetzt werden kann. In diesem Fall würde das Architekturmuster das Merkmal der Erweiterbarkeit von SOA betonen. Die verschiedenen Qualitätsmerkmale variieren von Mustern, die Performance-Probleme adressieren, bis hin zu Mustern, die die Verfügbarkeit einer Anwendung unterstützen wollen. Mit einem Architekturmuster ist also immer eine charakteristische Eigenschaft einer Anwendung verknüpft. Daraus wird deutlich, dass eine Anwendung sich nie auf ein einziges Architekturmuster reduzieren lässt. Einen Hinweis auf diese Tatsache gibt die schon zitierte Definition von [BCK05], die von den “structures” in einer Software spricht.

Die vorangegangenen Überlegungen haben gezeigt, dass eine Klassifizierung von Anwendungen anhand ihrer Architektur stets zu einer eindimensionalen Abbildung der Anwendung auf eine bestimmende Eigenschaft führt. Da eine Anwendung mehrere bestimmende Eigenschaften besitzt, wird sie auch durch mehrere Muster klassifiziert. Zwei verschiedene Desktop-Anwendungen verdeutlichen diese Aussage.

Ein Beispiel für eine Desktop-Anwendung stellt ein Email-Client dar. Zunächst kann man diese Anwendung dem Verteilungsmuster Client-Server zuordnen. Die Darstellung sowie die Anwendungslogik liegen beim Client und er bezieht seine Daten vom Server. Eine weitere Charakterisierung würde z.B. das MVC-Muster darstellen, das bei der Umsetzung der Darstellung zum Einsatz gekommen sein könnte. Ebenso ließe sich bei der Anwendungslogik eine Schichtenarchitektur vorstellen, die damit die Anwendung innerhalb der Kategorie der Strukturierungsmuster positionieren würde.

Ein weiteres Beispiel stellt eine Anwendung zur Bearbeitung von mutlimedialen Daten dar. Diese Anwendung kann sicherlich einer Mutlimedia-Architektur [HR06, S. 423] zugeordnet werden. Für die interne Verarbeitung kann sie das Strukturierungsmuster Pipes and Filters [B$^+$98, S. 54] verwenden. Zudem wäre das Interaktionsmuster MVC auch hier eine Möglichkeit für die Realisierung der Benutzerschnittstelle. Das Beispiel zeigt, dass beide Anwendungen ihre Eigenschaften durch teils identische, teils unterschiedliche Architekturmuster realisieren.

Eine weitere Schwierigkeit der eindeutigen Zuordnung einer Anwendung zu einem Architekturmuster zeigt sich in der Tatsache, dass die Literatur diese Muster auf unterschiedlichen Abstraktionsebenen beschreibt. Es lassen sich unter dem Begriff Architekturmuster die Muster *Implicit Invocation* [SGH03, S. 87] und *Peer-to-Peer* finden. Während Peer-to-Peer eine hohe Abstraktionsebene einnimmt und eine Gesamtsicht auf diese Netzwerk-Architektur darstellt, ist das Muster *Implicit Invocation* auf einer niedrigeren, in Richtung Entwurfsmuster weisenden, Ebene angesiedelt. Dieser Unterschied im Abstraktionsniveau lässt einen eindeutigen Vergleich der Muster nicht zu und erschwert dadurch die Zuordnung einer Anwendung zu genau einem Muster.

3.4 Abgeleitete Kriterien

Die vorangegangen Überlegungen haben gezeigt, dass eine vollständige Beschreibung der Eigenschaften einer Anwendung immer nur durch die Gesamtheit der ihr zugehörigen Architekturmuster erfolgen kann. Deswegen werden für den Zweck der vorliegenden Arbeit die jeweils charakteristischen nicht funktionalen Eigenschaften aus den Kategorien der Architekturmuster extrahiert und als Kriterien definiert.

Verteilungsmuster unterstützen in erster Linie die Verwendung verteilter Ressourcen und Dienste in einem Netzwerk und adressieren damit eine Anforderung, die irrelavant für Desktop-Anwendungen ist. Diese Aussage ist auf sogenannte "Fat Clients" begrenzt, bei denen höchstens die Datenhaltung extern realisiert ist. Daher liefert die Kategorie der Verteilungsmuster an dieser Stelle keine Eigenschaft als Kriterium. Aus der Kategorie der Interaktionsmuster kann die charakteristische Eigenschaft der Usability als Kriterium bestimmt werden. Die Erweiterungsmuster liefern als Kriterium die Modifizierbarkeit. Aus den Strukturierungsmusterm kann das Kriterium der Performance abgeleitet werden.

Diese Kriterien lassen sich in unterschiedlicher Ausprägung bei jeder Desktop-Anwendung finden und beeinflussen den Prozess der Portierung verschieden stark. Nachfolgend werden die Kriterien erläutert und es wird versucht, ihren Einfluss auf die Portierung von Desktop-Anwendungen auf Grid-Architekturen abzuschätzen.

3.4.1 Kriterium Performance

Bedeutung

Der Begriff der Performance einer Anwendung lässt sich in verschiedenen Bedeutungen auffassen. Zunächst kann man ihn als die Länge der Bearbeitungsdauer einer bestimmten Aufgabe verstehen. Je kürzer diese Dauer ist, desto performanter ist die Anwendung. Hier spricht man auch vom Durchsatz einer Anwendung, d.h. wieviele identische Aufgaben innerhalb eines festgelegten

Zeitraumes bearbeitet werden können. Mit Performance kann ebenfalls die Leistung in Bezug auf die Schnelligkeit der Reaktion einer Anwendung gemeint sein, also wie schnell die Anwendung auf Benutzerinteraktion reagiert. In diesem Fall charakterisiert Performance das zeitliche Verhalten bzgl. des Benutzers und man spricht deshalb auch von Antwortzeitverhalten. Eine weitere Bedeutung von Performance kann die Funktionsvielfalt einer Anwendung betreffen. Die Leistungfähigkeit würde in diesem Fall am Funktionsumfang der Anwendung fest gemacht, d.h. eine Anwendung, die eher weniger Funktionen anbietet, wäre nicht so perfomant, wie eine Anwendung mit vielen Funktionen. Für die weitere Betrachtung der Eigenschaft Performance stützt sich diese Arbeit auf die Bedeutungen Durchsatz und Antwortzeitverhalten. [B+95]

Performance bei Desktop-Anwendungen

Die Performance bei einer Desktop-Anwendung hängt von verschiedenen Faktoren ab. Das zugrundeliegende System, also der Rechner, auf dem die Anwendung installiert ist, darf dabei jedoch nicht betrachtet werden, da dies keinen objektiven Vergleich zulässt. Auf exakt identischen Systemen stimmt die Performance einer Desktop-Anwendung übereinstimmen. Eine Eigenschaft, die die Performance bedingt, ist die Implementierungssprache, in der die Desktop-Anwendung entwickelt wurde. Hardwarenahe bzw. Compiler-Sprachen, wie C/C++, weisen hier eine höhere Performance in Bezug auf den Durchsatz auf, als z.B. Interpreter-Sprachen, wie Python. Eine weitere wichtige Eigenschaft der Leistungsfähigkeit einer Anwendung ist in den verwendeten Algorithmen zu sehen. Die Komplexität der verwendeten Algorithmen bestimmt maßgeblich das Laufzeitverhalten einer Desktop-Anwendung. In diesem Zusammenhang spielt auch der Prozessfluß einer Anwendung eine Rolle. Dieser kann eher sequenziell oder parallel ausgeprägt sein. Generelle Aussagen über die Auswirkungen auf die Performance in Bezug auf den Durchsatz lässt der Prozessfluß jedoch nicht zu, da eine Verallgemeinerung, dass eine parallele Verarbeitung immer performanter als eine sequenzielle ist, nicht zulässig ist. Hier spielen Faktoren wie die Kommunikation der Threads untereinander und der damit verbundene Overhead eine entscheidende Rolle. Der Prozessfluß kann

allerdings für das Antwortzeitverhalten entscheidend sein. So kann bspw. bei einer nebenläufigen Desktop-Anwendung, die Kontrolle sofort wieder an den Benutzer zurückgegeben werden und die Bearbeitung läuft in einem Thread im Hintergrund ab, während eine Desktop-Anwendung, die blockierende Aufrufe bzw. nur einen Thread verwendet, Eingaben des Benutzers blockieren würde, bis eine Aufgabe bearbeitet wurde.

Einfluss auf die Portierung

Die Performance einer Desktop-Anwendung beeinflusst den Portierungsprozess in Bezug auf den Durchsatz nicht negativ. Die Bearbeitungsdauer einer Aufgabe kann auf einer Ressource im Grid sicherlich genauso schnell erfolgen, wie auf dem Rechner, auf dem die Desktop-Anwendung installiert ist. Betrachtet man den Prozessfluß einer Desktop-Anwendung, kann dieser eine Portierung u.U. begünstigen. Liegen Teile einer Desktop-Anwendung als paralleler Arbeitsablauf vor, können sie sich eventuell dafür eignen den Durchsatz der Anwendung im Grid zu steigern. Wenn es z.B. gelingt die Desktop-Anwendung so aufzuteilen, dass verschiedene Jobs auf unterschiedlichen Knoten des Grids zur gleichen Zeit ausgeführt werden, kann eine Steigerung der Performance im Vergleich zur Desktop-Anwendung erreicht werden. Wie im vorherigen Absatz erwähnt, lässt sich diese Aussage nicht verallgemeinern und ist für jeden Einzelfall einer Desktop-Anwendung zu betrachten. Als ersten Hinweis auf die Parallelisierbarkeit einer Desktop-Anwendung kann das Maß an Kommunikation zwischen den einzelnen Jobs dienen. In Bezug auf das Antwortzeitverhalten lässt sich kein Einfluss auf den Prozess der Portierung feststellen. Allerdings wird sich die Qualität dieser Eigenschaft aufgrund der Latenz innerhalb des Grids verschlechtern.

3.4.2 Kriterium Modifizierbarkeit

Bedeutung

Die Modifizierbarkeit einer Anwendung bezieht sich auf den Aufwand und die Möglichkeiten eine Software nach ihrer Entwicklung an neue Bedingungen, z.B. eine andere Zielplattform, anzupassen. Der Aspekt der Modifizierbarkeit kann sich dabei auf verschiedene Ebenen der Software beziehen. Zunächst ist eine Erweiterung bzw. Veränderung auf Ebene der Implementierung möglich. Kriterien für eine gute Modifizierbarkeit auf dieser Ebene sind u.a. ein modularer Aufbau oder eine detaillierte Dokumentation. Eine Ebene höher ist die Modifizierbarkeit des Designs einer Software anzusiedeln. Das Design lässt sich z.B. leichter modifizieren, wenn standardisierte Entwurfsmuster verwendet wurden. Auf der obersten Ebene steht die Modifizierbarkeit der Architektur. Eine wichtiges Kriterium für die Modifizierbarkeit einer Architektur ist die logische Trennung der einzelnen Komponenten und der zugrundeliegenden Plattform. [Bar04]

Modifizierbarkeit bei Desktop-Anwendungen

Die Modifizierbarkeit einer Desktop-Anwendung wird demzufolge im Wesentlichen durch diese drei Aspekte bestimmt. Da diese Aspekte bei jeder Desktop-Anwendung unterschiedlich stark ausgeprägt sein können, lassen sich keine generellen Aussagen über die Modifizierbarkeit treffen. Die Architektur einer Desktop-Anwendung kann nicht auf ein Architekturmuster reduziert werden, wie der vorangegangene Abschnitt gezeigt hat. Somit sind generelle Aussagen die Architektur betreffend nicht möglich bzw. sinnvoll. Das Gleiche gilt für die Modifizierbarkeit des Designs bzw. der Implementierung einer Desktop-Anwendung. Die Anwendung kann durch die Verwendung von Entwurfsmustern eine hohe Modularität aufweisen und damit eine leichtere Modifizierbarkeit. Ebenso kann sich die Anwendung jedoch durch eine eher monolithische Struktur auszeichnen, die die Modifizierbarkeit auf dieser Ebene stark einschränkt.

Einfluss auf die Portierung

Für den Prozess der Portierung ins Grid hat die Modifizierbarkeit einer Anwendung die Bedeutung, dass eine höhere Modifizierbarkeit den Portierungsprozess begünstigt bzw. vereinfacht. Eine modular aufgebaute Software erlaubt die einfachere Positionierung der einzelnen Komponenten innerhalb des Grids. Dies lässt sich sowohl auf die Architekturebene als auch auf die Designebene beziehen. Bei einer Anwendung bei der beispielsweise die Datenhaltung von den anderen Teilen getrennt ist, wie es in einem Client-Server Modell der Fall sein kann, könnte der Server auf einer beliebigen Ressource, die den Anforderungen des Servers entspricht, in der Fabric-Schicht (siehe Abschnitt 2.2.1) des Grids plaziert werden. Somit wäre es möglich mehrere redundante Datenquellen für die Anwendung zu haben, während die Anwendung an sich nur auf einer spezifischen Ressource im Grid verfügbar wäre. Ohne diese Trennung von der Datenhaltung müsste jede Ressource, die die Anwendung ausführen soll, stets die Anforderungen der gesamten Anwendung erfüllen können. Damit ist sie nicht mehr so flexibel. Auf die Designebene bezogen unterstützt eine bessere Modifizierbarkeit der Anwendung den Portierungsprozess, da sich die Anwendung leichter in einzelne Komponenten zerlegen lässt, die im Grid verteilt werden können. In Bezug auf die serviceorientierte Architektur eines Grids (siehe Abschnitt 2.2.2), kann der modulare Aufbau einer Anwendung die einfachere Kapselung der grundlegenden Funktionalität in Services ermöglichen. Diese Services können dann im Grid verfügbar gemacht werden und erlauben so zum einen eine Steigerung der Skalierbarkeit der Anwendung und zum anderen eine nochmals erhöhte Modifizierbarkeit durch die Möglichkeit der Komposition mit schon im Grid vorhandenen Services.

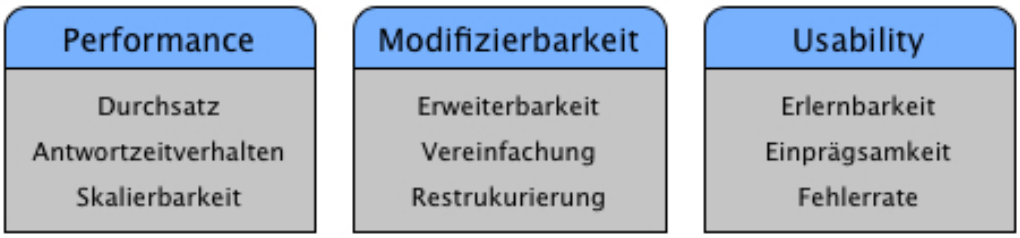

Abbildung 3.1: Abgeleitete Bewertungskriterien

3.4.3 Kriterium Usability

Bedeutung

Mit der Usability bezeichnet man die Gebrauchstauglichkeit einer Anwendung, d.h. sie bezeichnet "das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und mit Zufriedenheit zu erreichen." [ISO07]. Die Eigenschaft Usability setzt sich aus mehreren Faktoren zusammen. Dazu gehören u.a. die Erlernbarkeit, die Fehlerrate und die Einprägsamkeit einer Anwendung. Diese Faktoren sind unterschiedlich zu bewerten, je nach dem in welchem Kontext eine Anwendung eingesetzt werden soll. So steht bspw. bei einem Fahrkartenautomaten die Erlernbarkeit eher im Vordergrund, während bei sicherheitskritischen Systemen mehr Wert auf eine niedrige Fehlerrate gelegt wird. In ihrer Gesamtheit bestimmen diese Faktoren ein Maß für die Qualität der Benuzterinteraktion. [Bar04]

Usability bei Desktop-Anwendungen

Die Usability von Desktop-Anwendungen kann aufgrund der vielen technischen Möglichkeiten eine graphische Benutzerschnittstelle zu realisieren sehr unterschiedlich ausgeprägt sein. Generell von einer hohen Usability bei Desktop-Anwendungen zu sprechen, ist sicherlich falsch, da sich genügend Beispiele für schlecht konzipierte und umgesetzte Benutzerschnittstellen finden lassen. Dennoch bieten Desktop-Anwendungen eine Basis für die Entwicklung

von Benutzerschnittstellen mit hoher Komplexität und hohem Funktionsumfang. Die Gründe hierfür sind in den generellen Eigenschaften von Desktop-Anwendungen zu sehen (siehe Abschnitt 3.2), insbesondere in der lokalen Installation. Die lokale Installation erlaubt Desktop-Anwendungen den direkten bzw. indirekten Zugriff auf die Grafikkarte und den damit verbundenen Hardware-Ressourcen. Im Zusammenspiel mit verschiedenen Grafik-Bibliotheken ist so eine hardwarenahe Implementierung der Benutzerschnittstelle möglich, die sich sowohl im Funktionsumfang als auch im Antwortzeitverhalten positiv bemerkbar macht. Auf diese Weise lassen sich sehr komplexe Oberflächen realisieren, die eine Vielzahl von Bedienelementen (Schaltflächen, Schieberegler, usw.) beinhalten und sich häufig von Benutzern beliebig positionieren lassen, um eine Anpassung an die eigenen Bedürfnisse zu erreichen. Obwohl die Usability einer Desktop-Anwendung immer von Konzeption und Umsetzung abhängt, sind bei dieser Plattform beste Voraussetzungen für das Erreichen einer hohen Usability geben.

Einfluss auf die Portierung

Die Usability einer Desktop-Anwendung hat entscheidenden Einfluss auf den Portierungsprozess in dem Sinn, dass sie ihn mitunter sehr aufwendig macht. Der Grund dafür ist in der Verwendung eines Grid-Portals als Standard für die Benutzerschnittstelle mit dem Grid zu sehen. Abschnitt 4.2.3 bezieht sich auf diese Thematik. Da ein Grid-Portal eine webbasierte Anwendung darstellt, beruht die Umsetzung der Benutzerschnittstelle auf webbasierten Technologien. Ausgehend von dieser Basis bedeutet dies zunächst einmal den Einsatz von HTML. Mit Hilfe dieser Technologie ist eine Umsetzung komplexer Desktop-Oberflächen nicht möglich. Aufgrund der Entwicklung im Bereich der webbasierten Technologien stehen inzwischen viele Ansätze, wie Flash, zur Verfügung, mit denen eine Annäherung an die Qualität von Desktop-Benutzerschnittstellen erreicht werden kann. Der Einsatz dieser Technologien muss aus zwei Gründen für jeden Einzelfall kritisch bewertet werden. Zum einen schränkt der Einsatz die Interoperabilität mit möglichen Zielpattformen ein, was jedoch ein Ziel bei der Verwendung von Grid-Portalen ist. Zum an-

deren vertragen sich die noch relativ jungen Standards im Bereich der Grid-Portale oftmals nicht mit diesen Technologien, so dass ein Einsatz entweder nicht möglich ist oder zusätzlichen Aufwand mit sich bringt. In Bezug auf das Antwortzeitverhalten determiniert die Kommunikation zwischen Grid-Portal und Browser die Qualität dieser Eigenschaft. Hier kann mit Konzepten wie AJAX (Asynchronous JavaScript and XML) eine vergleichbare Qualität zu Desktop-Anwendungen erreicht werden.

Weitere wichtige nicht funktionale Eigenschaften sind z.B. Sicherheit, Verfügbarkeit, Skalierbarkeit und Ausfallsicherheit. Diese Eigenschaften haben keinen relevanten Einfluss auf den Prozess der Portierung einer Desktop-Anwendung. Nach einer Integration in das Grid können diese Eigenschaften für die Anwendung jedoch von Bedeutung sein. Entweder hat sich die Qualität der Eigenschaft verändert oder die Eigenschaft erfährt durch die Integration in das Grid eine Relevanz, die bei der Desktop-Anwendung nicht gegeben war. Die Verfügbarkeit bei einer Desktop-Anwendung hängt allein von der Verfügbarkeit des Rechners ab, auf dem sie installiert ist. Dagegen kann die Verfügbarkeit der integrierten Anwendung innerhalb des Grids von mehreren Ressourcen abhängen. Da die Verfügbarkeit nun anstatt von einer kritischen von mehreren kritischen Komponenten abhängt, ändert sich die Qualität dieser Eigenschaft.

4 Portierung der Anwendung DataFinder

Dieses Kapitel beschreibt den Prozess der Portierung der Anwendung DataFinder. Einleitend wird der Kontext beschrieben, in dem dieses Projekt stattgefunden hat. Eine Schilderung der Vorgehensweise erläutert, wie sich die Eigenschaften der Anwendung auf die Integration ins das Grid ausgewirkt haben. Abschließend wird der Portierungsprozess bewertet und die daraus gewonnenen Kenntnisse dargestellt.

4.1 Ausgangsituation & Aufgabenstellung

Zielsetzung seitens des Auftraggebers Deutsches Zentrum für Luft- und Raumfahrt (DLR) war die Entwicklung einer webbasierten Benutzerschnittstelle für die Anwendung DataFinder. Als Zielplattform wurde das GridSphere Portal Framework vorgegeben. Auf Basis eines Portlets sollten wesentliche Teile der Benutzerschnittstelle sowie eine Teilmenge der Funktionalitäten als Prototyp realisiert werden. Hintergrund für die Auswahl des GridSphere Frameworks als zugrundeliegende Technologie war das schon vorhandene Portal der D-Grid-Initiative [DGI07], einer Forschungsinitiative des Bundesministerium für Bildung und Forschung (BMBF) . In dieses Portal sollte das Portlet integriert werden. Der entwickelte Prototyp sollte als Anhaltspunkt und zur Generierung von Wissen in der Abteilung dienen, im Hinblick auf das langfristige Ziel, die Anwendung DataFinder als Grid-Anwendung zu betreiben.

Die vom DLR in der Abteilung Simulations- und Softwaretechnik SISTEC entwickelte Software DataFinder [DLR07] ist eine Anwendung zur Verwaltung

von wissenschaftlich-technischen Daten. Aufgabe des DataFinder ist es, vorhandene und neu erzeugte, zum Teil ungeordnete, wissenschaftliche Daten in geeigneter Weise mit Metadaten zu kennzeichnen, zu strukturieren und so Ordnung zu schaffen. Das Datenmodell zur Strukturierung der Daten ist dabei vom Administrator frei wählbar, so dass die Art der Daten, die vom DataFinder verwaltet werden kann, nicht vorgegeben ist. Die Metadaten werden dabei auf einem WebDAV-fähigen Server abgelegt. Die eigentlichen Daten können entweder direkt auf diesem Server oder mittels verschiedener Speicherprotokolle wie z.B. NFS oder GridFTP auf verteilten Servern gespeichert werden. Eine Suchfunktion in den Metadaten der Datenobjekte ermöglicht ein Wiederauffinden der gespeicherten Daten.

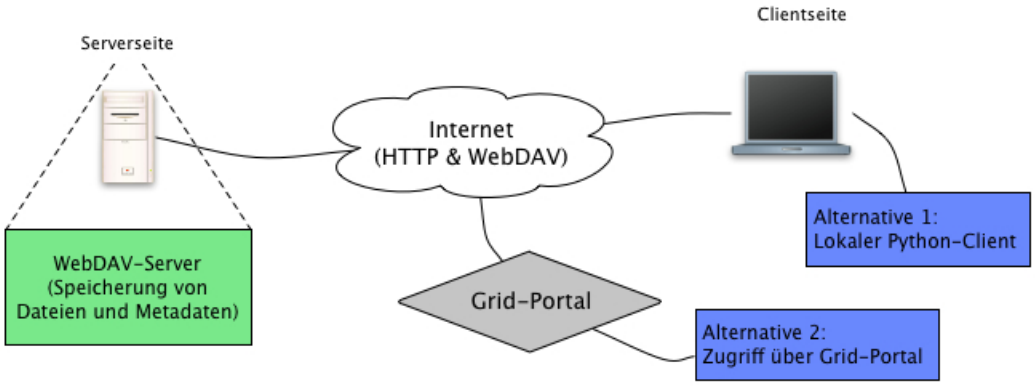

Abbildung 4.1: Grundlegende Architektur des DataFinder

Zur Verwaltung der Daten nutzt der DataFinder das WebDAV-Protokoll (Web-based Distributed Authoring and Versioning) [IET07]. WebDAV ist eine auf XML basierende Erweiterung des HTTP-Protokolls. Es realisiert ein Dateisystem über ein Netzwerk und ermöglicht durch Funktionen wie das Erstellen und Löschen von Dateien, Synchronisation von Änderungen, Versionierung und Zugriffskontrolle das gemeinsame Arbeiten an den Daten in diesem Netzwerk.

Implementiert wurde die Software, d.h. die Client-Anwendung, mit Python [PSF07] und Qt [Tro07]. Die Client-Anwendung setzt sich aus zwei Benutzeroberflächen zusammen, zum einen der Anwendersicht, die die Verwaltung der Daten ermöglicht, und zum anderen der Administratorsicht, die die Datenmo-

dellierung und Konfiguration der Datenhaltung ermöglicht. Auf der Serverseite ist grundsätzlich jeder WebDAV-konforme Server einsetzbar.

Zum Zeitpunkt der vorliegenden Arbeit existierte die oben angesprochene auf Python und Qt basierende Desktop-Anwendung des DataFinder, die auch intern im DLR eingesetzt wird. Zudem wurde an einer Integration des DataFinder für Microsofts SharePoint Server [Mic07] gearbeitet. Noch nicht vorhanden war eine Web-Umsetzung des DataFinder, d.h. eine Benutzeroberfläche, die über einen Webbrowser die Bedienung des DataFinder ermöglicht. An dieser Stelle setzt die vorliegende Arbeit an. Der Fokus der Arbeit liegt auf den clientseitigen Teil der Anwendung, insbesondere die Funktionalität der Anwendersicht des DataFinder. Im Vordergrund stehen die Umsetzung der Benutzerschnittstelle sowie eine Teilmenge von Funktionen der Anwendersicht.

4.2 Vorgehensweise

Dieser Abschnitt beschreibt die generelle Vorgehensweise bei der Portierung der Anwendung DataFinder.

4.2.1 Analyse

Zu Beginn der Konzeption stand das Verstehen und Kennenlernen der Anwendung. Dies bezüglich war die Situation in der Abteilung nicht zufriedenstellend, da die Hauptentwickler der Anwendung sich nicht mehr innerhalb des Unternehmens befanden und eine technische Dokumentation, die den Entwicklungsprozess der Anwendung festhält, nur in sehr geringem Umfang vorhanden war. Das Wissen über die Anwendung wurde hauptsächlich über Code-Review bezogen.

Zunächst lässt sich die Anwendung DataFinder als klassische Client-Server-Anwendung charakterisieren und reiht sich damit bei den Verteilungsmustern ein. In dieser zweischichtigen Architektur liegen die Geschäftslogik und die Darstellung der Benutzerschnittstelle im Aufgabenbereich des Clients, während der Server die Anfragen des Clients bearbeitet. Im Fall der Anwendung DataFinder kann man von einem sogenannten “Fat Client” sprechen, da ein Großteil der Verarbeitung auf der Clientseite ausgeführt wird. Die Aufgabe des WebDAV-Servers besteht hauptsächlich in der Bearbeitung von Anfragen in Bezug auf das Metadatenmodell. Die strenge Einteilung in eine zweischichtige Architektur wird durch die Funktionalität des DataFinder aufgeweicht, Daten auch auf externen Servern speichern zu können, statt nur auf dem WebDAV-Server. In diesem Fall erhält man eine dreischichtige Architektur, in der die Datenhaltung auf die genannten externen Server ausgelagert ist. Klassisch ist diese Architektur jedoch nicht, da hier der Kern der Anwendungslogik nicht in der mittleren Schicht (WebDAV-Server) liegt, sondern immer noch in der obersten Schicht (Client) lokalisiert ist.

Eine genauere Betrachtung der Clientseite lieferte weitere Anhaltspunkte über die Eigenschaften der Anwendung in Bezug auf ihre Architektur. Die auf

den ersten Blick ersichtliche Trennung von Präsentations- und Geschäftslogik legt den Schluß nahe, dass das MVC Architekturmuster verwendet wurde. Tatsächlich ließ sich dieses Interaktionsmuster wiederfinden. Im Verlaufe des Code-Review wurde festgestellt, dass das Muster nicht sauber angewendet worden war, d.h. es hatte eine Vermischung von Präsentationslogik und Anwendungslogik stattgefunden. Dies konnte z.B. bei der Funktion der Suche innerhalb der Metadaten beobachtet werden. Die Funktionalität dafür war innerhalb der Komponenten umgesetzt worden, die für die Präsentation zuständig sind. Diese enge Verzahnung ließ sich noch an weiteren Stellen innerhalb der Anwendung beobachten. Sie verhinderte so die einfache Verwendung einer anderen Präsentationsart bzw. die separate Verwendung der Funktionalität des DataFinder. In diesem speziellen Fall hieß das, dass für die Ausführung des DataFinder auch immer die Grafik-Bibliotheken von Qt benötigt wurden.

Als weiteres Merkmal der Anwendung ließ sich eine Schichtenarchitektur identifizieren, die die einzelnen Komponenten in Beziehung zueinander setzt und dabei fachlich gegeneinander abgrenzt. Ziel bei der Verwendung dieses Musters ist die Reduzierung der Abhängigkeiten zwischen den einzelnen Bestandteilen der Anwendung, d.h. es ist den Strukturierungsmustern zu zuordnen. Im Idealfall lassen sich so Schichten austauschen, ohne eine Veränderung in anderen Schichten nach sich zu ziehen. Bei der Anwendung DataFinder konnten sich fünf Schichten bestimmen lassen, die nachfolgend dargestellt sind.

Die unterste Schicht in dem Modell bildet die sogenannte Persistenzschicht. In dieser Schicht sind die Funktionen für die Kommunikation mit dem WebDAV-Server implementiert. Sie stellt die grundlegenden Operationen für die Abfrage und Veränderung des Metadatenmodells zur Verfügung. Aufbauend darauf ist die Metadatenschicht realisiert. Diese Schicht kapselt die Low-Level-Funktionalität der unter ihr liegenden Persistenzschicht und bietet höherwertige Funktionen für die Manipulation der Metadaten an. Sie stellt z.B. zwei Objekte *DFResourceStorer* und *DFCollectionStorer* bereit, die die logische Sicht auf eine Datei bzw. ein Verzeichnis auf dem WebDAV-Server repräsentieren. Die nächste Schicht ist die DataStore-Schicht. In dieser Schicht ist die Datenhaltung realisiert, d.h. hier werden die Operationen für die Zugriffe auf die

Abbildung 4.2: Schichtenmodell des DataFinder

verschiedenen Speicherserver implementiert. Die sogenannte Fassaden-Schicht setzt darauf auf und stellt eine vereinfachte Schnittstelle für den Zugriff auf die darunter liegenden Schichten dar. Aus der Fassaden-Schicht werden Aufrufe an die unteren Schichten delegiert. An dieser Stelle zeigt sich besonders deutlich, dass die Schichtenarchitektur bei der Anwendung DataFinder nicht linear ist. Die oberste Schicht realisiert die Benutzerschnittstelle. In ihr sind die Anwendersicht und Administratorsicht implementiert.

Die Analyse der Struktur der Anwendung zeigte, dass eine Trennung der Komponenten nicht in dem Maße gegeben war, dass eine einfache Wiederverwendung einzelner Komponenten möglich gewesen wäre. Als Gründe sind vor allem die enge Kopplung verschiedener Schichten durch viele direkte Zugriffe sowie die Verwendung des Singleton-Entwurfsmusters bei der Klasse *Configuration* zu nennen. Ein weiterer Punkt ist die fragwürdige Einordnung der Datenhaltungsschicht in diesem Modell in Bezug auf die darunterliegende Metadatenschicht. Ein besser abstrahierendes Konzept auf Ebene der Persistenzschicht scheint hier eine elegantere Lösung zu sein.

Die Erkenntnisse aus diesen Beobachtungen konnten u.a. für ein zum Zeitpunkt der vorliegenden Arbeit geplantes Refactoring der Anwendung genutzt

werden. Eine weitere Erkenntnis aus der Analyse bestätigt die in Abschnitt 3.3 getroffene Aussage, dass sich eine Anwendung nicht anhand eines einzigen Architekturmusters beschreiben lässt.

4.2.2 Anwendungslogik

Die Vorgabe des auf Java basierenden GridSphere Portal Frameworks als Zielplattform wirft die Fragestellung nach der Wiederverwendbarkeit des Python-Quellcodes des DataFinder auf. Hier lassen sich generelle Ansätze zur Migration einer Anwendung finden, die von der Verwendung von Wrappern bis hin zur kompletten Neuentwicklung reichen. Jeder dieser Ansätze weist unterschiedliche Ausprägungen in Bezug auf Kosten und Komplexität auf.

Die Option einer kompletten Neuentwicklung des DataFinder in Java wurde nicht in Betracht gezogen, da dies nicht der Zielvorgabe des Auftraggebers entsprach. Die Möglichkeit der Verwendung von Wrappern wurde untersucht, zum einen da hier schon einige Ansätze bzw. Tools existieren, welche die Kommunikation zwischen Java und Python erlauben, zum anderen da so zwei unterschiedliche Implementierungen derselben Anwendung und dem damit verbundenen doppelten Wartungsaufwand vermieden werden können.

Eine Analyse der bestehenden Tools, Jython [G+07a], JPype [JPy07] und Jepp [Joh07], zeigte jedoch, dass damit eine einfache Verwendung des bestehenden Python-Codes nicht möglich ist. Die Gründe hierfür sind auf der einen Seite in der häufig schlechten bis nicht vorhandenen Dokumentation oder der nie über den Beta-Status hinausgekommenen Entwicklung einiger Tools zu sehen. Auf der anderen Seite spricht die Architektur des DataFinder gegen eine Verwendung bestimmter Tools - so z.B. bei Jython, einer vollständigen Implementierung von Python in Java. Bei Jython ist es nicht möglich, die in C geschriebenen Erweiterungen für den DataFinder zu nutzen.

Aus diesen Gründen wurde auf eine Verwendung von Wrappern verzichtet und eine Entscheidung zu einer Portierung in Java getroffen. Da nur eine Teilmenge der Funktionen zur Verfügung stehen sollte, konnten bestimmte Teile

von der Portierung ausgeschlossen werden (z.B. ein großer Teil der DataStore-Schicht). So war eine schnelle Realisierung des Java-Prototypen gewährleistet. Der Java-Prototyp konnte dann im Applikationsserver des Grid-Portals laufen und die geforderten Funktionalitäten zur Verfügung stellen.

4.2.3 Benutzerschnittstelle

Grid-Portal

An dieser Stelle soll kurz auf die Technologie der Portale und ihre Bedeutung im Hinblick auf das Grid beschrieben werden. Mit der Durchsetzung der Idee des Grids im wissenschaftlichen und kommerziellen Bereich bekam die Frage des Zugangs zu dieser Technologie mehr Bedeutung. Die Weiterentwicklung auf dem Gebiet der Grid-Middleware erlaubte einer immer größeren Nutzergemeinde den Zugang zur Grid-Technologie und in Folge dessen nahm die Anzahl der Grid-Anwendungen stetig zu. Die daraus entstandene Nutzergemeinde war nicht mehr nur auf Grid-Entwickler beschränkt, sondern erstreckte sich auch auf Mitarbeiter aus wissenschaftlichen Institutionen und Unternehmen. Für diese neue Nutzergruppe, die nicht über Expertenwissen im Umgang mit Grid-Middleware verfügte, mußten neue Konzepte für den Zugang zum Grid entwickelt werden. Die erforderlichen Lernprozesse für diese Konzepte sollten dabei möglichst kurz und einfach sein, um einen Großteil der Nutzergruppe zu erreichen. Innerhalb der Grid-Community wurden zwei Konzepte vorgeschlagen, zum einen clientbasierte Desktoplösungen und zum anderen ein webbasierte Portale, mit denen die Nutzergruppe im Umgang vertraut war. Das Konzept des Portals setzte sich schnell durch und ist inzwischen zum Standard für die Schnittstelle zwischen Benutzer und Grid avanciert. Während eine clientbasierte Desktoplösung in Bezug auf Funktionsumfang und graphische Benutzerschnittstelle eine höhere Komplexität als webbasierte Portale aufweist, ergeben sich bei diesem Konzept Nachteile bzgl. Flexibilität und Administration. Die Unterstützung von Grid spezifischen Protokollen, z.B. zur Datenübertragung mit GridFTP, erfordert häufig die Öffnung von bestimmten Ports in der Firewall und stellt damit Anforderungen an die Sicherheitsrichtlinien der jeweili-

gen Clients. Diese sind in einem begrenzten Umfeld, wie z.B. einem Intra-Grid (siehe Abschnitt 2.1.1), durchaus erfüllbar, erfordern aber einen hohen administrativen Aufwand. Zudem setzt dieses Konzept voraus, dass jeder Rechner, auf dem der Client arbeiten soll, die entsprechenden Voraussetzungen bzgl. Laufzeitumgebung und Bibliotheken erfüllt. Dem gegenüber steht das Konzept des webbasierten Portals, das lediglich einen Web-Browser als Laufzeitumgebung benötigt. Dieser ist inzwischen Standard auf heutigen Rechnern. Damit entfällt die lokale Installation einer Clientsoftware und der Zugang zum Grid weitet sich so auf alle Geräte aus, die einen Browser unterstützen, z.B. auch Mobiltelefone und PDAs. [AWY07]

Wie zu Beginn des Grid-Computing mangelte es bei den ersten Versionen von Grid-Portalen am Willen zur Standardisierung, so dass die entstandenen Portale sehr spezifisch und nicht wiederverwendbar für andere Anwendungen waren [AWY07]. Dies änderte sich erst durch die Entwicklung der Standards auf diesem Gebiet, JSR 168 (Java Specification Request) [AH03] und WSRP (Web Services for Remote Portlets) [OAS07a]. Diese beiden Spezifikationen bilden die Grundlage, auf der heutige Grid-Portale basieren. Grid-Portale adaptieren das Konzept des benutzerbezogenen Kontextes von Web-Portalen, um dem Nutzer einen personalisierten Zugangspunkt zum Grid zur Verfügung zu stellen. Grundsätzlich lassen sich drei Bestandteile eines Grid-Portals identifizieren: das Portal, der Portlet Container und die Portlets.

Das Portal ist eine webbasierte Anwendung, die die Präsentationsschicht für den Nutzer bereitstellt und Funktionalitäten wie Session Management, Single Sign-On, rollenbasierte Nutzerverwaltung und Personalisierung unterstützt. Portlets sind auf Java basierende Komponenten, die dynamisch Fragmente einer Portalseite visualisieren und Anfragen des Benutzers entgegennehmen. Die Portlets sind dabei unabhängig voneinander und bieten jedes eine eigene Funktionalität an. Die Aggregation mehrerer Portlets bildet die gesamte Portalseite. Ein Portlet Container stellt die Laufzeitumgebung für Portlets dar, verwaltet die Lebenszeit der Portlets und delegiert die Benutzeranfragen an die entsprechenden Portlets. Durch das Zusammenspiel dieser Komponenten erreicht ein Grid-Portal die Bereitstellung von benutzerbezogenem “Grid-Kontext” für je-

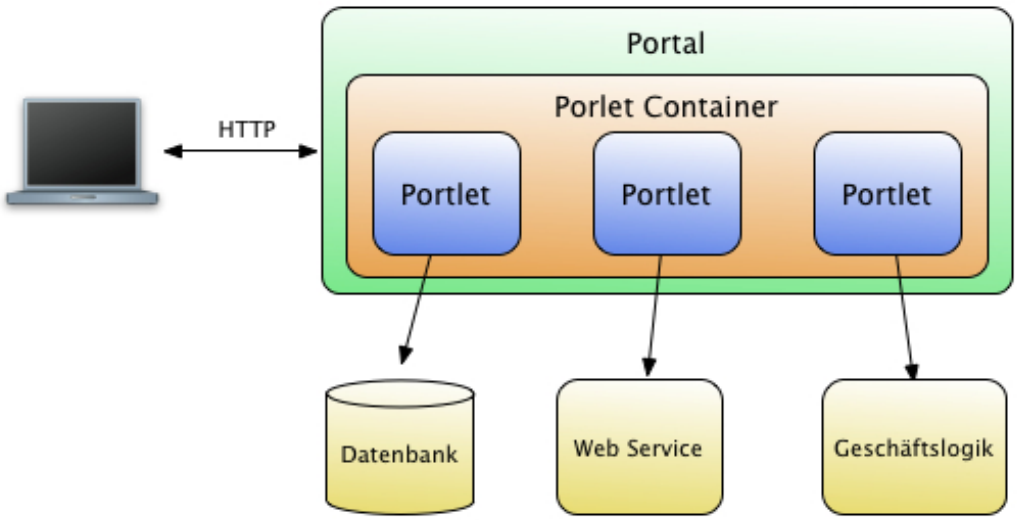

Abbildung 4.3: Aufbau eines Portals, nach [AWY06]

den einzelnen Anwender. Diese Eigenschaft ist im Umgang mit einem Grid besonders wichtig, da sich hier der Kontext eines Nutzers über das Grid verteilt. Ein Grid-Portal bündelt die einzelnen Teile des Kontextes und stellt sie dem Benutzer transparent in einem Zugriffspunkt dar. Der Grid-Kontext eines Nutzers kann sich über entfernte Datenquellen, verschiedene Dienstverzeichnisse, Logeinträge über vergangene Jobs, Listen über aktuell laufende Jobs und Beschreibungen von Arbeitsabläufen erstrecken. Kurz gesagt, der Grid-Kontext repräsentiert alle Objekte, mit denen der Benutzer im Grid interagiert hat. Das Portal bietet dem Benutzer die Möglichkeit seinen Grid-Kontext zu konfigurieren. [AWY07]

Umsetzung

Die Umsetzung der Benutzerschnittstelle setzte zunächst ein Verständnis des bisherigen Ansatzes voraus. Zudem war es wichtig die Benutzergruppe zu kennen, die später mit dieser Benutzerschnittstelle arbeiten sollten. Die vom Auftraggeber angegebene Gruppe setzte sich sowohl aus im Umgang mit der Desktop-Anwendung erfahrenen Benutzern als auch aus neuen Benutzern zu-

sammen. Aus diesem Grund wurde bei der Konzeption der Benutzerschnittstelle darauf geachtet, den bestehenden Ansatz aus der Desktop-Anwendung nach Möglichkeit zu übernehmen. Aufgrund der Verwendung eines Grid-Portals ist die grundlegende Technologie für die Darstellung HTML. Die in dem Portal angezeigten Portlets generieren HTML-Fragmente, die zusammengesetzt die gesamte Darstellung der Portalseite ergeben. Die Möglichkeiten der Präsentation mit HTML sind beschränkt und es muss daher überlegt werden, welche Bestandteile der bestehenden Benutzerschnittstelle mit diesem Mittel umgesetzt werden können und welche sich nicht dafür eignen. Die Benutzerschnittstelle des DataFinder besteht überwiegend aus Elementen, die sich alle mit HTML umsetzen lassen. Dazu zählen u.a. eine Menüstruktur, Buttons und verschiedene Anzeigebereiche. Einige Eigenschaften, wie z.B. das dynamische Ändern der Größe der Anzeigebereiche, lassen sich mit Standard-HTML nicht umsetzen und wurden bewußt weggelassen, da sie keine erhebliche Verbesserung der Bedienbarkeit bedeuten. Auf weitere Technologien (z.B. Applets oder Flash), die in diesem Zusammenhang die Benutzerfreundlichkeit steigern können, wurde aus Gründen der Portabilität mit Endgeräten, wie Mobiltelefonen oder PDAs, verzichtet.

Für die Entwicklung der Benutzerschnittstelle im Grid-Portal wurde ein auf JSR 168 basierendes Portlet konzipiert. Als unterstützende Technologie wurde das Framework JSF (Java Server Faces) [SM07] verwendet. JSF bietet zum einen mehr Funktionalität bei der Erzeugung der Darstellung als das in GridSphere verwendete UI-Framework und bietet zum anderen durch die strikte Einhaltung des MVC-Musters eine gute Möglichkeit zur Strukturierung der Anwendung. Dies war besonders wichtig im Hinblick auf die Wiederverwendung des Portlets für spätere Projekte. Es wurden einheitliche Schnittstellen definiert, die das Portlet vom Java-Prototypen des DataFinder trennen und ein Austauschen der Anwendungslogik erlauben. An dieser Stelle könnten dann in der weiteren Entwicklung Client-Stubs ansetzen, wenn die Integration in das Grid vorangeschritten ist und die Funktionalitäten des DataFinder in Web Services gekapselt wurden. Die Abbildungen 4.4 und 4.5 zeigen die Desktop-Anwendung und das neu entwickelte Portlet.

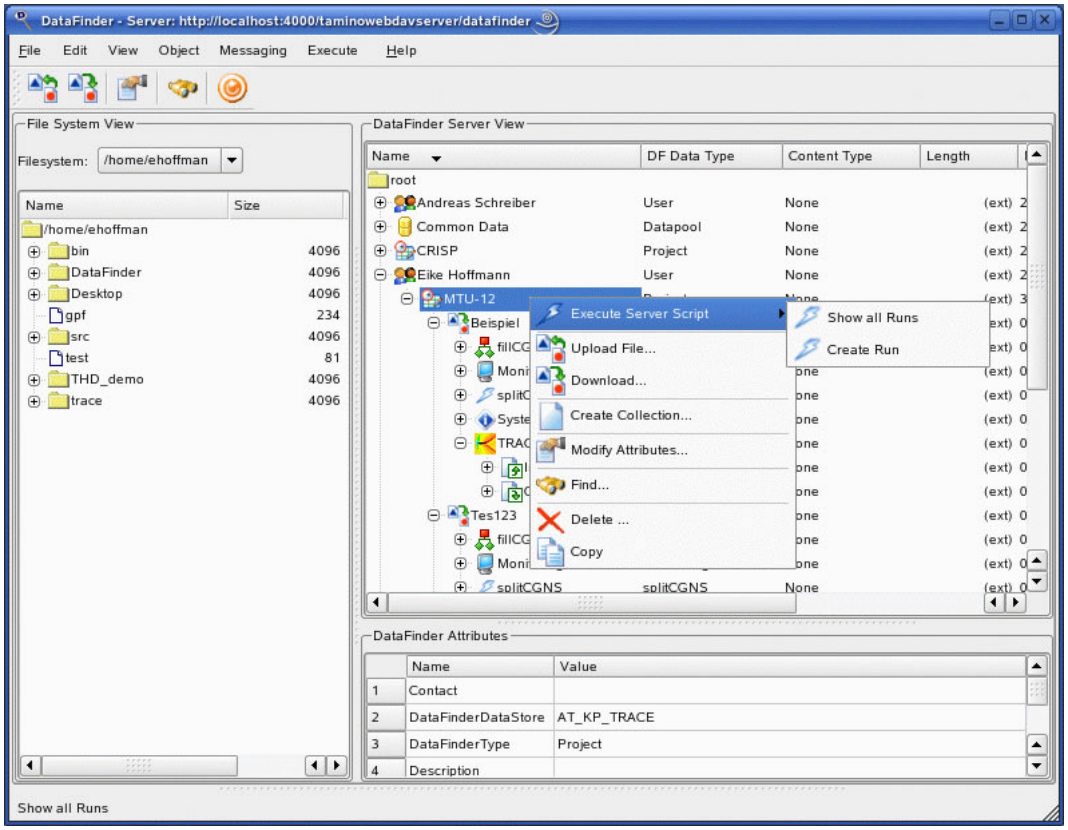

Abbildung 4.4: Screenshot: DataFinder Client

Mit dem DataFinder Portlet ist ein erster Schritt in Richtung Grid erreicht wurden. Durch das Einbinden in das Grid-Portal steht auf der obersten Ebene der Grid-Architektur eine Schnittstelle zur Verfügung, die spätere Entwicklungen ausnutzen können.

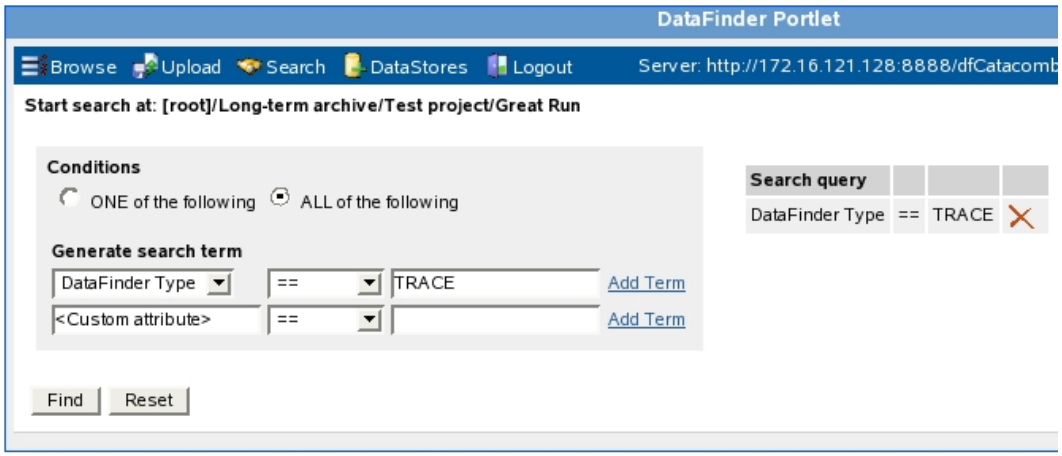

Abbildung 4.5: Screenshot: DataFinder Portlet

4.3 Bewertung

In diesem Abschnitt wird die Integration des DataFinder anhand der in Abschnitt 3.4 definierten Kriterien beleuchtet. Es wird gezeigt, wie sich die einzelnen Eigenschaften auf die Portierung ausgewirkt haben.

4.3.1 Performance

In Bezug auf die Aspekte der Performance eignet sich der DataFinder für eine Portierung und Integration in ein Grid. Der Prozessfluß der Anwendung ist streng sequenziell. Es konnten keine parallelen Bestandteile identifiziert werden. Eine mögliche Steigerung der Performance durch eine Parallelisierung bedarf einer genaueren Analyse der Anwendungsstruktur, die nicht Teil der Arbeit war. Der Wechsel der Implementierungssprache durch die Portierung von Python zu Java hatte keinen Einfluß auf die Performance in Bezug auf den Durchsatz. Die Struktur der Anwendung wurde eins zu eins übernommen, so dass es keine Veränderungen hinsichtlich der implementierten Algorithmen gab. In Bezug auf die Ausführungsgeschwindigkeit wurde durch die Portierung eine Steigerung erzielt, da Java eine Compiler-Sprache ist. Das Antwortzeitverhalten wurde allein durch die Kommunikation zwischen Grid-Portal und

Browser bestimmt. Zwischen Portlet und Java-Prototyp sind keine negativen Auswirkungen zu beobachten, da sie auf demselben Applikationsserver laufen.

4.3.2 Modifizierbarkeit

Die generelle Client-Server Architektur des DataFinder unterstützte den Portierungsprozess dahingehend, dass sie eine klare Trennung der einzelnen Komponenten der Anwendung ermöglichte. So konnte die Client-Anwendung für sich betrachtet werden, ohne die Faktoren bzgl. des Servers zu beachten. Das Konzept des WebDAV-Servers wurde unverändert übernommen, so dass das Portlet direkt gegen den Server getestet werden konnte. In einem späteren Schritt kann der Server in das Grid integriert werden und im Fabric-Layer (siehe Abschnitt 2.2.1) angesiedelt werden. Die Verwendung des Interaktionsmusters MVC begünstigte den Portierungsprozess in Bezug auf die Portierung der Anwendungslogik. Große Teile des Datenmodells konnten problemlos übernommen werden. Die erwähnte unsaubere Verwendung an einigen Stellen führte dagegen zu zusätzlichem Aufwand. Die Schichtenarchitektur des DataFinder, die zum Teil nicht linear ist und dadurch eine enge Kopplung zwischen einzelnen Komponenten herstellt, verhindert ein hohes Maß an Modifizierbarkeit. Für die Übernahme des Python-Codes in den Java-Prototypen hat dies keine Bedeutung, da die bestehende Struktur übernommen wird. Im Hinblick auf die Integration ins Grid hat diese Eigenschaft jedoch einen erschwerenden Einfluß auf den Portierungsprozess. Eine Aufteilung der grundlegenden Eigenschaften des DataFinder in Web Services gestaltet sich aufgrund der erwähnten engen Verzahnung innerhalb der Anwendung als aufwendig.

4.3.3 Usability

Im Fall des DataFinder erweist sich die Auswirkung der Usability auf den Portierungsprozess als moderat. Dies liegt u.a. an der wissenschaftlichen Ausprägung der Anwendung, die die Funktionalität stärker priorisiert als die Usability. Dies ist ein Unterschied zu kommerziell ausgerichteten Desktop-Anwendungen.

Ein weiterer Grund für den moderaten Aufwand ist in der einfach strukturierten Benutzerschnittstelle der Desktop-Anwendung zu sehen. Die grundlegenden Bedienelemente setzen sich aus Komponenten zusammen, die mit dem verwendeten JSF Framework realisiert werden können. Die Aufteilung der Benutzeroberfläche in drei Teilbereiche in der Desktop-Anwendung wurde bei der Umsetzung zum Portlet zu einer einzigen Anzeigefläche vereinfacht. Diese Vereinfachung war zum einen durch den Wegfall des Teilbereiches für das lokale Dateiverzeichnis möglich, das es bei einer webbasierten Anwendung nicht geben kann und zum anderen durch die Zusammenfassung der Ansicht des Serververzeichnisses mit der Ansicht für die Suchergebnisse und Logmeldungen.

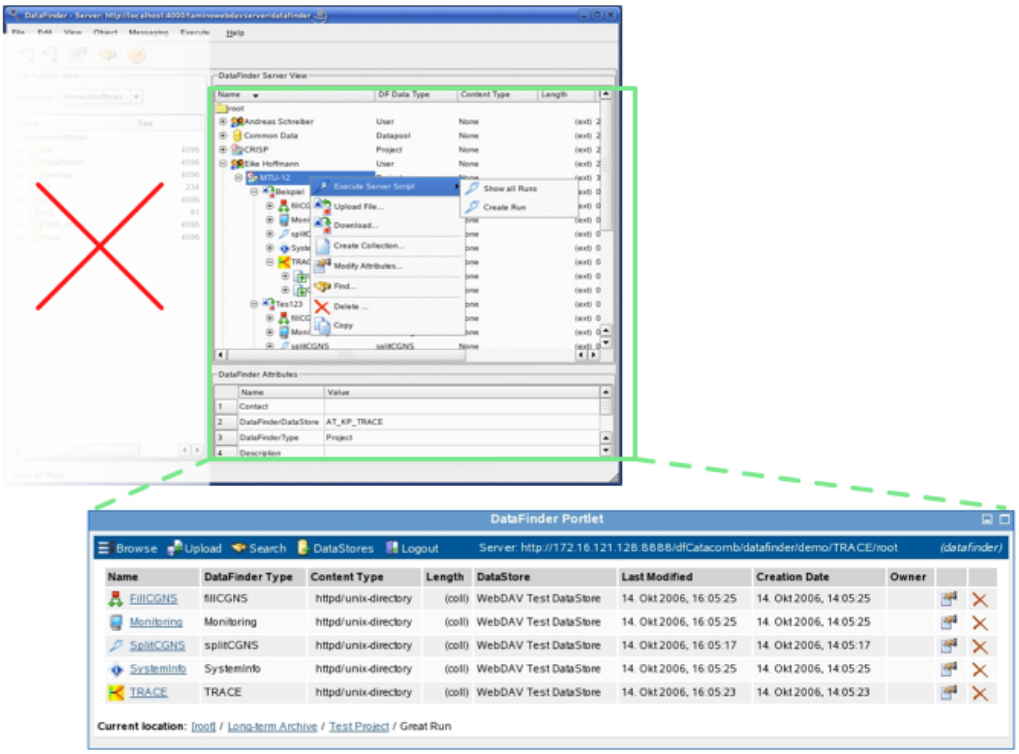

Abbildung 4.6: Umsetzung der Benutzerschnittstelle

Die daraus entstandene Präsentationsform ist einfach strukturiert, um eine leichte Erlernbarkeit für neue Benutzer zu gewährleisten, und bewahrt gleichzeitig die wesentlichen Aspekte der Benutzerschnittstelle der Desktop-Anwendung, um von erfahrenen Benutzer wiedererkannt zu werden. Eine de-

taillierte Überprüfung anhand eines Usability-Tests zur Validierung der Ergebnisse war nicht Teil dieser Arbeit.

4.3.4 Ergebnis

Das Beispiel DataFinder zeigt, dass bei der Portierung einer Desktop-Anwendung in ein Grid eine Wechselwirkung zwischen dem Portierungsprozess und den nicht funktionalen Eigenschaften der Desktop-Anwendungen entsteht. Die Ausprägung der nicht funktionalen Eigenschaften beeinflusst den Portierungsaufwand. Die Stärke des Einflusses hängt dabei maßgeblich davon ab, ob und wie weit vorhandene Eigenschaften bei der Portierung erhalten bleiben sollen. So folgt z.B. aus der Anforderung nach der Erhaltung der Usability einer komplexen Benutzerschnittstelle ein erhöhter Aufwand beim Portierungsprozess.

Die in Kapitel 3.4 definierten Kriterien zur Analyse der Herausforderungen bei der Integration in ein Grid können daher nicht für jede Desktop-Anwendung im gleichen Maß Anwendung finden. Hier muss von Einzelfall zu Einzelfall unterschieden werden, welche Ziele mit der Integration erreicht werden sollen bzw. welche Ziele niedriger priorisiert werden können. Damit folgt, dass für jede Desktop-Anwendung, die in ein Grid integriert werden soll, eine Gewichtung der nicht funktionalen Eigenschaften erfolgen muss, um abschätzen zu können, wie schwierig der Portierungsprozess sein wird. Dabei gilt es, einen Kompromiss zu finden zwischen der Priorität der Erhaltung der Eigenschaften und dem Aufwand für die Portierung. Je mehr Eigenschaften erhalten bleiben sollen, desto schwieriger gestaltet sich unter Umständen die Integration. Diese Gewichtung kann einen ersten Hinweis liefern. Jede nicht funktionale Eigenschaft untergliedert sich in weitere Aspekte, die für jeden Einzelfall gesondert betrachtet werden müssen.

5 Fazit & Ausblick

Im Rahmen der vorliegenden Arbeit wurde der Fragestellung nachgegangen, wie die Integration einer bestehenden Desktop-Anwendung in ein Grid gelingen kann und wie eine geeignete Vorgehensweise hierfür aussehen kann. Anhand der Anwendung *DataFinder* der Abteilung Simulations- und Softwaretechnik des DLR wurde eine beispielhafte Portierung durchgeführt.

Zunächst wurde auf den Begriff *Grid* eingegangen und eine Definition vorgelegt, auf die sich diese Arbeit stützt. Anschließend wurden die Architektur und die Technologien in Bezug auf das Grid erläutert. Im Zusammenhang damit wurde die Bedeutung von SOA und Web Services als technologische Grundlage für das Grid beschrieben. Die Gründe für eine Integration von Desktop-Anwendungen in ein Grid wurden ausführlich erläutert. Aus dem Versuch einer Klassifizierung von Desktop-Anwendungen anhand ihrer Architektur wurden Kriterien abgeleitet, die den Prozess der Portierung in ein Grid beeinflussen können. Diese Kriterien wurden in Bezug auf ihre Ausprägung bei Desktop-Anwendungen und ihre damit verbundene Auswirkung auf die Portierung untersucht. Die beispielhafte Integration der Anwendung *DataFinder* wurde im anschließenden Abschnitt der Arbeit beschrieben. Nach einer Vorstellung der Anwendung und der Vorgehensweise bei der Portierung wurde der Portierungsprozess anhand der zuvor erörterten Kriterien bewertet. Die Bewertung ergab, dass es keine allgemeinen Vorgehensweisen für eine Integration von Desktop-Anwendungen in ein Grid gibt und Desktop-Anwendungen nur bedingt für die Zielplattform Grid geeignet sind.

Wie die beschriebene Arbeit gezeigt hat, gestaltet sich die Entwicklung von allgemeinen Vorgehensweisen für eine Integration von Desktop-Anwendungen in ein Grid schwierig. Die Gründe hierfür sind zunächst in der Schwierigkeit

der Abgrenzung einer Desktop-Anwendung zu anderen Arten von Software zu sehen. Die Vielschichtigkeit dieser Anwendungen lässt nur eine sehr allgemeine Definition von Merkmalen (z.B. lokale Installation) zu, die alle Anwendungen hinreichend charakterisiert. Auf Basis dieser Merkmale ist eine Entwicklung von generellen Strategien nicht möglich. Der Versuch einer Klassifizierung von Desktop-Anwendungen anhand ihrer Architektur zur Präzisierung der Merkmale scheitert vor allem aus zwei Gründen: Zum einen lassen sich Anwendungen beliebiger Art nicht auf eine Architektur bzw. ein Architekturmuster reduzieren. Zum anderen verhindern die unterschiedlichen Abstraktionsebenen vieler Architekturmuster sinnvollen Vergleich dieser Muster.

Als kleinster gemeinsamer Nenner können nicht funktionale Eigenschaften dazu dienen charakteristische Merkmale von Desktop-Anwendungen zu bestimmen. Dabei zeigt sich, dass sich einige nicht funktionale Eigenschaften und der Portierungsprozess gegenseitig beeinflussen. Eine Priorisierung der einzelnen Eigenschaften wird notwendig, um den Portierungsaufwand bestimmen zu können. Daraus folgt, dass eine Entwicklung allgemeiner Vorgehensweisen für eine Integration von Desktop-Anwendungen in ein Grid auf dieser Grundlage nicht sinnvoll ist.

Die Integration der Anwendung *DataFinder* war auf die Benutzerschnittstelle und die Entwicklung eines Portlets im GridSphere Portal Framework ausgerichtet. Damit sollte ein erster Schritt in Richtung Grid unternommen werden, mit der langfristigen Zielsetzung die komplette Anwendung zu integrieren. Zudem stand die Generierung von Expertenwissen in Bezug auf die Portal-Technologie im Vordergrund. Das entwickelte Portlet arbeitet zusammen mit einem Java-Prototyp, der eine bestimmte Teilmenge der Funktionalität der Desktop-Anwendung unterstützt, in einem Grid-Portal und stellt damit die Integration auf der obersten Schicht des Grids dar. Ausgehend von diesem "Frontend" kann die tiefere Integration des *DataFinder* in das Grid realisiert werden. Dem Portierungsprozess bei dieser Aufgabe lag eine klare Gewichtung auf Seiten der Usability zugrunde, was in der Bewertung deutlich wird. Insgesamt gesehen kann man sagen, dass die Integration des *DataFinder* in ein Grid einen Mehrwert für die Anwendung darstellen kann. Dieser besteht jedoch

vordergründig nicht in der Eigenschaft der verteilten Rechenleistung in einem Grid, sondern vielmehr in der Möglichkeit der Erweiterung der Funktionalität durch die Nutzung der verteilten Ressourcen in Bezug auf weitere Dienste und weitere Systeme für die Datenhaltung.

Auch wenn es mit dem *DataFinder* ein Beispiel für eine Desktop-Anwendung gibt, bei der sich eine Integration lohnt, kann man feststellen, dass Desktop-Anwendungen nur bedingt geeignet für ein Grid sind. Die Eignung der Anwendung *DataFinder* hängt vor allem mit der Ausrichtung zusammen, die sie seit ihrer Entwicklung hatte - das Datenmanagement auf verteilten Servern mit grid-spezifischen Protokollen, wie z.B. GridFTP. Eine Integration war aus diesem Grund der nächste logische Schritt.

Zusammenfassend lässt sich daher feststellen, dass eindeutige Aussagen über die Eignung von Desktop-Anwendungen für die Integration in ein Grid nicht möglich sind und auf diesem Gebiet weiter geforscht werden muss. Führt man sich den Begriff Desktop-Anwendung vor Augen, wird deutlich, dass diese Anwendungen für eine Zielplattform, nämlich den Desktop, entwickelt und optimiert werden. In den meisten Fällen erfüllt ein Desktop damit die Anforderungen der Anwendung an z.B. Rechenleistung und Arbeitsspeicher.

Bestimmte Desktop-Anwendungen aus dem graphischen (CAD-Anwendungen) oder mathematischen (Statistikprogramme) Bereich, die eine hohe Rechenleistung erfordern und mit großen Datenmenge umgehen müssen, können sich dagegen eher für die Integration in ein Grid eignen. An diesen Punkt anküpfend muss die Forschung vertieft werden, um diese Unterschiede genauer zu untersuchen. Eine offene Frage ist in diesem Zusammenhang, ob es Arten von Desktop-Anwendungen gibt, bei denen eine Integration immer sinnvoll ist und durch welche Eigenschaften diese sich auszeichnen. Mit Hilfe dieser Eigenschaften könnten Rückschlüsse auf alle anderen Desktop-Anwendungen gezogen werden.

Literaturverzeichnis

[AH03] Abdelnur, A.; Hepper, S.: JSR 168 Portlet Specification, http://jcp.org/aboutJava/communityprocess/review/jsr168/, o.A. 2003. (Seiten 58.)

[AWY06] Allan, R., Wang, X.; Yang, X.: JSR 168 AND WSRP 1.0 - How mature are portal standards?, http://epubs.cclrc.ac.uk/bitstream/1082/paper06A.pdf, Warrington (UK) 2006. (Seiten iii, 59.)

[AWY07] Allan, R., Wang, X.; Yang, X.: Development of standards-based grid portals: Review of grid portals, http://www.ibm.com/developerworks/webservices/library/ws-wsiluddi.html?dwzone=webservices, o.A. 2007. (Seiten 58, 59.)

[B+95] Barbacci, M. et al.: Ouality Attributes, http://www.cert.org/archive/pdf/tr021.95.pdf, Pittsburgh (USA) 1995. (Seiten 43.)

[B+98] Buschmann, F. et al.: Pattern-orientierte Softwarearchitektur. Ein Pattern-System. Addison-Wesley, Bonn 1998. (Seiten 40, 41.)

[B+04] Box, D. et al.: Web Services Addressing (WS-Addressing), http://www.w3.org/Submission/ws-addressing/, o.A. 2004. (Seiten 33.)

[B+05] Baker, M. et al.: Emerging Grid Standards, Computer, 38(4), 43–50, Portsmouth (UK) 2005. (Seiten 33.)

[Bar04] Barbacci, M.: Software Quality Attributes: Modifiability and Usability, http://www.cert.org/archive/pdf/tr021.95.pdf, Pittsburgh (USA) 2004. (Seiten 45, 47.)

[Bay02] Bayer, T.: REST Web Services, http://www.oio.de/public/xml/rest-webservices.htm, o.A. 2002. (Seiten 29.)

[BCK05] Bass, L., Clements, P.; Kazman, R.: Cluster Computing. Springer, Berlin 2005. (Seiten 39, 40.)

[BM05] Bauke, H.; Mertens, S.: Cluster Computing. Springer, Berlin 2005. (Seiten 4.)

[BN01] Ballinger, K.; Nagy, W.: The WS-Inspection and UDDI Relationship, http://www.ibm.com/developerworks/webservices/library/ws-wsiluddi.html?dwzone=webservices, o.A. 2001. (Seiten 30.)

[BNS04] Bry, F., Nagel, W.; Schroeder, M.: Gird-Computing, Informatik-Spektrum, 27(6), 542–545, München 2004. (Seiten 9, 10, 11, 13.)

[C+01] Christensen, E. et al.: Web Services Description Language, http://www.w3.org/TR/wsdl, Redmond (USA) 2001. (Seiten 27.)

[C+03] Czajkowski, K. et al.: Open Grid Services Infrastructure (OGSI), http://www.globus.org/toolkit/draft-ggf-ogsi-gridservice-33-2003-06-27.pdf, o.A. 2003. (Seiten 32.)

[C+04a] Czajkowski, K. et al.: From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution, http://www.globus.org/wsrf/specs/ogsi-to-wsrf-1.0.pdf, o.A. 2004. (Seiten 33.)

[C+04b] Czajkowski, K. et al.: The WS-Resource Framework,

http://www.globus.org/wsrf/specs/ws-wsrf.pdf, o.A. 2004. (Seiten 33.)

[CER07] CERN, Hrsg.: What is the Grid? - Power grid analogy, http://gridcafe.web.cern.ch/gridcafe/whatisgrid/dream/powergrid.html, Genf 2007. (Seiten 4.)

[D+96] DeFanti, T. et al.: Overview of the I-WAY: Wide Area Visual Supercomputing, International Journal of Supercomputer Applications, 10(2), 123–130, Chicago (USA) 1996. (Seiten 5.)

[DGI07] D-Grid Initiative, Hrsg.: D-Grid Initiative, http://www.d-grid.de/, o.A. 2007. (Seiten 50.)

[DLR07] DLR, Hrsg.: DataFinder, http://www.dlr.de/sc/datafinder, Braunschweig 2007. (Seiten 50.)

[F+02] Foster, I. et al.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, http://www.globus.org/alliance/publications/papers/ogsa.pdf, Chicago (USA) 2002. (Seiten 15, 16, 18, 20, 25, 31.)

[F+04a] Foster, I. et al.: Modelling Stateful Resources with Web Services, http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf, Chicago (USA) 2004. (Seiten 33.)

[F+04b] Foster, I. et al.: Open Grid Services Architecture Use Cases, http://www.ogf.org/documents/GFD.29.pdf, Argonne (USA) 2004. (Seiten 15.)

[Fie00] Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. Doktorarbeit, University of California, Irvine (USA), 2000. (Seiten 28.)

[FK99] Foster, I.; Kesselmann, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, San Francisco (USA) 1999. (Seiten 5.)

[FKS06] Foster, I., Kishimoto, H.; Savva, A.: The Open Grid Services Architecture Version 1.5, http://www.ogf.org/documents/GFD.80.pdf, Argonne (USA) 2006. (Seiten iii, 16, 18, 19, 20, 24.)

[FKT01] Foster, I., Kesselman, C.; Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organization, The International Journal of Supercomputing Applications, http://www.globus.org/alliance/publications/papers/anatomy.pdf, 15(3), 200–222, Chicago (USA) 2001. (Seiten iii, 8, 9, 10, 11, 12, 13, 15.)

[Fos02] Foster, I.: What is the Grid? A Three Point Checklist, GRIDtoday, http://www.gridtoday.com/02/0722/100136.html, 1(6), Chicago (USA) 2002. (Seiten 5, 6, 7.)

[G+07a] Charlie Groves and others: The Jython Project, http://www.jython.org/Project/index.html, o.A. 2007. (Seiten 56.)

[G+07b] Gudgin, M. et al.: SOAP Version 1.2, http://www.w3.org/TR/soap12-part1/, Redmond (USA) 2007. (Seiten 28.)

[GCS02] Gannon, D., Chiu, K.; Slominski, A.: A Revised Analysis of the Open Grid Services Infrastructure, http://www.extreme.indiana.edu/ aslom/papers/ogsa-analysis4.pdf, Bloomington (USA) 2002. (Seiten 21.)

[Has06] Hasselbring, W.: Software-Architektur, Informatik Spektrum, 29(1), 48–52 2006. (Seiten 39.)

[HR06] Hasselbring, W.; Reussner, R.: Handbuch der Software-Architektur. dpunkt.verlag, Heidelberg 2006. (Seiten 5, 22, 27, 36, 41.)

[IBM07] IBM, Hrsg.: Web Services Inspection Language, http://www-

128.ibm.com/developerworks/library/specification/ws-wsilspec/, o.A. 2007. (Seiten 30.)

[IET07] IETF, Hrsg.: HTTP Extensions for Web Distributed Authoring and Versioning, http://tools.ietf.org/html/rfc4918, o.A. 2007. (Seiten 51.)

[ISO07] ISO, Hrsg.: Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability, http://www.iso.org/iso/iso-catalogue/catalogue-tc/catalogue-detail.htm?csnumber=16883, o.A. 2007. (Seiten 47.)

[JEF04] Joseph, J., Ernst, M.; Fellenstein, C.: Evolution of Grid Computing Architecture and Grid Adoption Models, IBM Systems Journal, 43(04), 624–645, Yorktown Heights (USA) 2004. (Seiten 7.)

[JF03] Joseph, J.; Fellenstein, C.: Grid Computing. Prentice Hall, Upper Saddle River (USA) 2003. (Seiten 7, 33, 35.)

[Joh07] Mike Johnson: Jepp - Java Embedded Python, http://jepp.sourceforge.net/, o.A. 2007. (Seiten 56.)

[JPy07] JPype, Hrsg.: JPype - Java to Python integration, http://jpype.sourceforge.net/, o.A. 2007. (Seiten 56.)

[L+98] Lee, D. et al.: Execution characteristics of desktop applications on Windows NT, ACM SIGARCH Computer Architecture News, 26(3), 27–38, New York (USA) 1998. (Seiten 38.)

[Mic07] Microsoft, Hrsg.: SharePoint Server-Homepage, http://office.microsoft.com/de-de/sharepointserver/default.aspx, Redmond (USA) 2007. (Seiten 52.)

[OAS07a] OASIS, Hrsg.: OASIS Web Services for Remote Portlets (WSRP) TC, http://www.oasis-open.org/committees/tc-home.php?wg-abbrev=wsrp, o. A. 2007. (Seiten 58.)

[OAS07b] OASIS, Hrsg.: UDDI Specifiactions, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm, o. A. 2007. (Seiten 29.)

[PSF07] Python Software Foundation, Hrsg.: Python Programming Language, http://www.python.org/, o.A. 2007. (Seiten 51.)

[QC96] Quibeldey-Cirkel, K.: Entwurfsmuster – Das aktuelle Schlagwort, Informatik Spektrum, 19(6), 326–327, o.A. 1996. (Seiten 39.)

[RS02] Reinefeld, A.; Schintk, F.: Concepts and Technologies for a Worldwide Grid Infrastructure, Lecture Notes in Computer Science, 2400, 62–71, Berlin 2002. (Seiten 8.)

[RS04] Reinefeld, A.; Schintke, F.: Dienste und Standards für das Grid Computing, Lecture Notes in Informatics, 55(15), 293–304, Düsseldorf 2004. (Seiten 5, 32.)

[RS05] Richter, J.-P.; Schrey, P.: Serviceorientierte Architektur, Informatik-Spektrum, 28(5), 413–416, München 2005. (Seiten 22, 23, 25.)

[SGH03] Schahram, D., Gall, H.; Hauswirth, M.: Software-architekturen für verteilte Systeme. Springer Verlag, Berlin 2003. (Seiten 41.)

[SM07] Sun Microsystems, Inc. (Hrsg.): JavaServer Faces Technology, http://java.sun.com/javaee/javaserverfaces/, Santa Clara (USA) 2007. (Seiten 60.)

[Sot04] Sotomayor, B.: The Globus Toolkit 4 Programmer's Tutorial, http://gdp.globus.org/gt4-tutorial/multiplehtml/ch01s01.html, Chicago (USA) 2004. (Seiten iii, 33.)

[ST05] Srinivasan, L.; Treadwell, J.: An Overview of Service-oriented Architecture, Web Services and Grid Computing, http://devresource.hp.com/drc/technical-soa/SOA-Grid-HP.pdf, Palo Alto (USA) 2005. (Seiten iii, 23, 24, 25, 26, 30, 31.)

[Tay05] Taylor, I.: From P2P to Web Services and Grids. Springer, London 2005. (Seiten 5, 28, 30, 32.)

[TGA07] The Globus Alliance, Hrsg.: Globus: The Globus Alliance, http://globus.org/, Chicago (USA) 2007. (Seiten 5.)

[TOP07] TOP500.org, Hrsg.: TOP500 Supercomputing Sites, http://www.top500.org/, o.A. 2007. (Seiten 4.)

[Tro07] Trolltech, Hrsg.: Qt, http://trolltech.com/products/qt, Oslo (Norwegen) 2007. (Seiten 51.)

[TT05] Talia, D.; Trunfio, P.: Peet-to-Peer Protocols and Grid Services for Resource Discovery on Grids. In: Grid Computing - The New Frontier of High Performance Computing, Band 14 der Reihe *Advances in Parallel Computing*, Rende (Italien) 2005. (Seiten 18, 27.)

[UoC07] University of California, Hrsg.: SETI@home, http://setiathome.ssl.berkeley.edu/, Berkeley (USA) 2007. (Seiten 4.)