INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

DAGA – Active Probing zur Bestimmung der Verfügbarkeit von Grid-Ressourcen

Christian Straube

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

DAGA – Active Probing zur Bestimmung der Verfügbarkeit von Grid-Ressourcen

Christian Straube

Aufgabensteller:	Prof. Dr. Kranzlmüller
Betreuer:	Dr. Michael Schiffers
Abgabetermin:	03. November 2011

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

München, den 03. November 2011

.....

(Unterschrift des Kandidaten)

Abstract

In den vergangenen Jahren hat sich Grid-Computing aufgrund seiner Eigenschaften und den daraus resultierenden Vorteilen für die Lösung komplexer Berechnungsprobleme etabliert. Für den Betrieb eines Grids sind Verfügbarkeits-Informationen zu den dabei involvierten Grid-Ressourcen unabdingbar, da sie in allen Betriebsbereichen sowie bei der Ausführung der Berechnungsjobs eine zentrale Rolle spielen, beispielsweise beim Scheduling oder Checkpointing. Der Gewinnung von Verfügbarkeits-Informationen muss daher eine hohe Bedeutung beigemessen werden. In dieser Arbeit wird gezeigt, dass bestehende Ansätze die dabei entstehenden Anforderungen nicht ausreichend erfüllen und daher für die Gewinnung von Verfügbarkeits-Informationen in Grid-Infrastrukturen nicht geeignet sind. Aufbauend auf den dabei gefundenen Fehlstellen wird mit DAGA ein neuer Ansatz für die Gewinnung von Verfügbarkeits-Informationen entwickelt, der die Dienst-Orientierung und die Verwendung von Active Probing als Kernkonzepte einsetzt. Das in dieser Arbeit entwickelte technische Konzept von DAGA wird prototypisch implementiert.

Inhaltsverzeichnis

1 Einführung

Um die immer komplexer und anspruchsvoller werdenden Berechnungsaufgaben aus Forschung, Wissenschaft und Industrie bearbeiten zu können, hat sich *Grid-Computing* als geeignetes und zukunftsweisendes Vorgehen erwiesen [FKT01; BMA09; NPF08].

Die beim Grid-Computing entwickelten Anwendungen werden auf einer *Grid-Infrastruktur*, einer Spezialform einer verteilten Berechungs-Infrastruktur, ausgeführt. Unter einer Grid-Infrastruktur bzw. einem *Grid* versteht man die Koordination und lose Kopplung von zum Teil stark heterogenen Ressourcen, die nicht zentralisiert verwaltet werden und Entscheidungs- sowie Verwaltungsautonomie besitzen (Kapitel 2.1.1). Für die Kommunikation zwischen den Ressourcen werden standardisierte, offene und allgemeine Protokolle verwendet. Innerhalb eines Grids bieten Services unter Verwendung dieser Ressourcen nicht-triviale Dienste an [FKT01] (Kapitel 2.1), beispielsweise Daten-Transfers, Simulationen oder Rechenleistung aus Cluster-Strukturen. Ein Grid wird hauptsächlich für das koordinierte Problem-Lösen in dynamischen, multi-institutionalen *Virtuellen Organisationen* (Kapitel 2.1.3) verwendet [FKT01].

Wegen der hohen Komplexität und den umfangreichen Aufgaben beim Betrieb eines Grids haben sich mehrere spezialisierte Betriebsbereiche entwickelt (Kapitel 2.2): Information Services, Grid-Scheduler, Grid-Job–Ausführung und Daten-Management. Daneben gibt es weitere, übergreifende Betriebsbereiche wie Performance-Analyse, Performance-Optimierung oder Sicherheitsmechanismen (Kapitel 2.2). Sowohl in den spezialisierten als auch übergreifenden Betriebsbereichen sind Verfügbarkeits-Informationen über einzelne Ressourcen grundlegend für den Betrieb. Sie werden beim Grid-Scheduling beispielsweise benötigt, da für die Zuweisung von Ressourcen zu Grid-Jobs bekannt sein muss, welche Ressourcen verfügbar sind oder sein werden [BMA09]. Bei der Ausführung eines Grid-Jobs kann mit Verfügbarkeits-Informationen besser auf Veränderungen in der Ressourcen-Landschaft reagiert werden, wenn beispielsweise bekannt ist, ob eine Ressource wegen Wartungsarbeiten nur wenige Minuten ausfällt oder für längere, unbestimmte Zeit nicht verfügbar ist. Beim Daten-Management können mit Verfügbarkeits-Informationen bessere Entscheidungen bezüglich der Replikation getroffen werden, was einen positiven Einfluss auf die Performance des Grids hat. Doch nicht nur die Aussage über die Verfügbarkeit oder Nicht-Verfügbarkeit einzelner Ressourcen ist von großer Bedeutung, sondern ebenso deren Ursachen, da unterschiedliche Ursachen unterschiedliche Auswirkungen auf Anwendungen im Grid haben, je nach deren Laufzeitumgebung und Fähigkeit zur Anpassung an Veränderungen in der Ressourcen-Landschaft [RL08] (Kapitel 2.4). Zudem ist die Verfügbarkeit von Ressourcen eine zentrale Voraussetzung für den zuverlässigen und sicheren Betrieb eines Grids [Avi+04].

Die beispielhaft genannten Anwendungsgebiete von Verfügbarkeits-Informationen verdeutlichen deren zentrale Rolle für den erfolgreichen und performanten (Kapitel 2.2.5) Betrieb eines Grids und zeigen, dass der Gewinnung der Verfügbarkeits-Informationen hohe Bedeutung beigemessen werden muss. Dabei ergeben sich aufgrund der Eigenschaften eines Grids – beispielsweise der dezentralen Verwaltung und der starken Heterogenität (Kapitel 2.1) – und den vielseitigen Anwendungsfällen umfassende und zum Teil schwierige Anforderungen (Kapitel 3.2.5) sowohl an die Gewinnung von Verfügbarkeits-Informationen als auch an die

Verfügbarkeits-Informationen selbst. Werden diese Anforderungen nicht oder nur unzureichend erfüllt, können die Verfügbarkeits-Informationen in den Betriebsbereichen nicht ausreichend genutzt werden, was zu negativen Auswirkungen auf die Performance bis hin zum Erliegen des Grids führen kann.

Die bestehenden Konzepte und Implementierungen für die Gewinnung von Verfügbarkeits-Informationen in Clustern und klassischen Computernetzen wie beispielsweise Monitoring (Kapitel 4.1.1) können nicht ohne weiteres in Grid-Umgebungen portiert werden, da sie wegen ihrer Granularität, Zielsetzung oder fehlenden Flexibilität gegenüber sich häufig ändernden heterogenen Ressourcen-Landschaften die Anforderungen (Kapitel 3.2.5) nicht ausreichend erfüllen und daher für den Einsatz in Grids nur sehr bedingt geeignet sind (Kapitel 4). Hinzu kommen deren Positionierung sehr weit unten in der Grid-Protokoll–Schicht-Architektur (Abbildung 2.3, Seite 6) sowie deren Konzentration auf einzelne Schichten, was eine ganzheitliche Sichtweise auf die Verfügbarkeits-Situation erschwert und eine (autonome) Ursachenforschung fast unmöglich macht.

Wesentlich besser eignet sich eine Top-Down-Herangehensweise (Abbildung 5.1, Seite 67), bei der proaktiv einzelne Ressourcen getestet und je nach Testergebnis die Testmethoden autonom adaptiert werden, um die Problemursache zu ermitteln. Dieses Vorgehen bietet neben einem höheren Informationsgehalt auch eine geringere Intrusiveness und kann zudem dynamisch auf die häufigen Veränderungen im Grid reagieren (Kapitel 5). Dieses Vorgehen kann unter Verwendung des Active Probings realisiert werden (Kapitel 4.2.2). Dabei werden kleine Test-Pakete, sogenannte Probes, an die zu überprüfenden Entitäten geschickt und iterativ Informationen über System-Komponenten ermittelt [Ris+04]. Trotz der sehr guten Eignung des Active Probings für den Einsatz in Grid-Umgebungen gibt es bisher keine Anwendungen oder Implementierungen des Konzeptes, es wurden lediglich verschiedene Algorithmen für die Auswahl von Probes entwickelt ([Bro+03; Bro+02; BRM01]).

Vor diesem Hintergrund analysiert diese Arbeit die Anforderungen an die Gewinnung von Verfügbarkeits-Informationen und untersucht bestehende Ansätze für die Gewinnung von Verfügbarkeits-Informationen auf die jeweilige Erfüllung der analysierten Anforderungen im Grid-Kontext. Aus den dabei gefundenen Fehlstellen wird anschließend mit DAGA ein neuer Ansatz für die Gewinnung von Verfügbarkeits-Informationen in einem Grid unter Verwendung des Active Probings entwickelt. Dabei werden auftretende technische und sicherheitsrelevante Fragestellungen diskutiert, wie beispielsweise die Anwendung über administrative Domänengrenzen hinweg oder die Autorisierung der Probes. Um möglichst einfach in bestehende Grid-Infrastrukturen und Middlewares wie dem Globus Toolkit integriert werden zu können, werden bei der Entwicklung der Architektur möglichst viele, bereits etablierte Methoden und Konzepte verwendet.

Die Arbeit gliedert sich wie folgt: zunächst wird in Kapitel 2 eine detaillierte Einführung in die relevanten Begrifflichkeiten und die allgemeine Problematik bei der Informationsgewinnung in Grid-Infrastrukturen gegeben. In Kapitel 3 werden die Anforderungen an eine Informationssammlung analysiert und bestehende Ansätze in Kapitel 4 auf die Erfüllung dieser Anforderungen evaluiert. Kapitel 5 erläutert die Ziele des neuen Ansatzes DAGA und dessen Kernkonzepte, die Dienst-Orientierung und das Active Probing. Die Architektur von DAGA wird in Kapitel 6 konzipiert und in Kapitel 7 prototypisch implementiert. Die Ergebnisse und weiterführende Problemstellungen werden in Kapitel 8 zusammengefasst.

2 Grid-Computing und die zentrale Rolle von Verfügbarkeits-Informationen

2.1 Grid-Computing

Für die Lösung komplexer Berechnungsprobleme, wie beispielsweise Wetterprognosen oder wissenschaftliche Simulationen, wurden in den vergangenen Jahren hauptsächlich Cluster-Infrastrukturen eingesetzt [BMA09]. Ein Cluster ist ein Verbund aus homogenen Arbeitsstationen oder PCs (Rechenknoten), auf denen jeweils das gleiche Betriebssystem verwendet wird und die über ein lokales Hochgeschwindigkeitsnetz miteinander verbunden sind [TS08; Wil09]. Die Rechenknoten werden über einen einzigen, dedizierten Master-Knoten, der gleichzeitig als Schnittstelle für den Benutzer fungiert, gesteuert und verwendet. Eine typische Cluster-Infrastruktur ist in Abbildung 2.1 dargestellt.

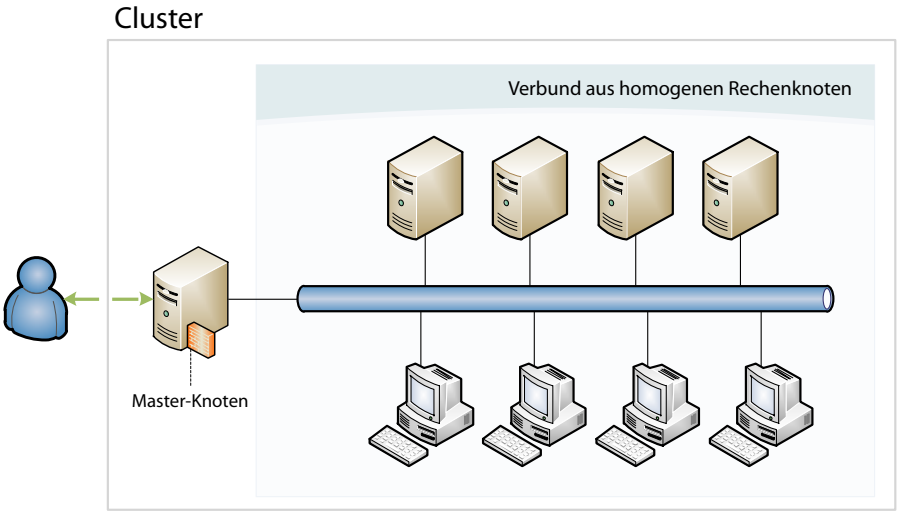


Abbildung 2.1: Typische Elemente, Aufbau und Funktionen einer Cluster-Infrastruktur

Aufgrund der stetig wachsenden Komplexität der Berechnungsprobleme und Datenmengen [BKS05] sowie den immer neuen Anforderungen an die errechneten Ergebnisse reichen diese Cluster-Infrastrukturen inzwischen aber nur noch selten aus, um zufriedenstellende Lösungen zu erzielen [FKT01]. Neben reinen Rechenkapazitäten in Form von *Central Processing Units* (CPU) sind immer mehr verschiedenartige Typen von Ressourcen notwendig – beispielsweise große Datenspeicher, Software-Lizenzen, Sensoren, spezielle Versuchsvorrichtungen oder mehrere kooperierende Cluster –, um die vielfältigen Facetten der Berechnungs-Problematiken zu bewältigen [LB05]. Bei der Wetterprognose sollen beispielsweise nicht nur die Sensoren eines Landes, sondern eines ganzen Kontinents verwendet werden; bei wissenschaftlichen Simulatio-

nen sollen mehrere Datenquellen verschiedener Institutionen mit unterschiedlichen Software-Produkten bearbeitet werden.

Die Ressourcen, die für die Problemlösung verwendet werden sollen, sind also geographisch (stark) verteilt und werden von unterschiedlichen Institutionen in verschiedenen Ländern betrieben [BKS05]. Eine geographische Bündelung und zentrale Administration der verschiedenen Ressourcen, wie es bei Cluster-Infrastrukturen der Fall ist [TS08], ist dabei aus mehreren Gründen nicht möglich: 1) die Ressourcen sind für eine korrekte Funktionsweise an einen spezifischen Ort gebunden und können nur in diesem Installationskontext sinnvoll verwendet werden. 2) Teilweise existieren manche Ressourcen-Typen sogar nur an einem Ort und können nicht beliebig oft hergestellt und an unterschiedlichen Orten installiert werden, wie der Large Hadron Collider (LHC), ein 27 km langer Teilchenbeschleuniger [Eurb] der European Organization for Nuclear Research (CERN) [Eurc]. 3) Auch die Finanzierung und Unterstützung durch den Staat hängt maßgeblich vom Standort ab [BKS05]. 4) Zudem ist der Betrieb aller Ressourcen durch eine einzige Institution nicht realistisch, da häufig mehrere (zehn)tausend heterogene Ressourcen eingesetzt werden sollen [Ios+07], die stark in ihrer Anzahl und Verfügbarkeit variieren [NPF08] und deren Betrieb daher für eine einzelne Institution viel zu komplex und kostenintensiv wäre.

2.1.1 Eigenschaften eines Grids

Um die geographisch wie administrativ stark verteilten Ressourcen dennoch für die koordinierte Problemlösung nutzen zu können, werden diese über eine Grid-Infrastruktur oder kurz ein Grid verbunden und betrieben. Ein Grid ist eine Infrastruktur, die sich durch die in der „Grid-Checkliste“ definierten Eigenschaften auszeichnet [Fos02]:

- Ein Grid ist eine Spezialform einer „Distributed Computing Infrastructure“.
- Die eingesetzten Ressourcen werden nicht zentral verwaltet. Dies bedeutet im Umkehrschluss, dass alle Ressourcen autonom sind und auch autonom verwaltet werden.
- Für die Kommunikation werden standardisierte, offene und allgemeine Protokolle verwendet, um die lokale Kontrolle zu bewahren und Änderungen an lokalen Policies überflüssig zu machen [FKT01].
- Es werden nicht-triviale Quality-of-Services zugrunde gelegt, z.B. Response-Zeit oder Durchsatz, wobei die Kombination der Einzelkomponenten wesentlich größeren Nutzen hat als die Einzelnutzung.
- Die eingesetzten Ressourcen sind stark heterogen und können Datenspeicherung, High-Performance-Cluster, Supercomputer, Software-Lizenzen, Sensoren, Instrumente oder einzelne Notebooks und PDAs umfassen [RL07].
- Die Ressourcen sind lose gekoppelt.
- Ein Grid ist meist als Service-orientierte Architektur (SOA) aufgebaut. Ein Service ist dabei nur durch seine Schnittstellen für Anfragen und Antworten definiert, wobei die Implementierung für den Anfrager keine Rolle spielt.

Anhand der Eigenschaften eines Grids wird deutlich, dass Grids die zu Beginn erwähnten bestehenden Cluster-Infrastrukturen nicht ersetzen, sondern vielmehr in wesentlich größere,

heterogenere und stärker verteilte Berechnungsstrukturen integrieren [FKT01]. Dies macht auch den zuvor genannten Nachteil, dass nicht alle Ressourcen von einer einzigen Institution betrieben werden können, zu einem Vorteil, da so die Kompetenzen verschiedenster Institutionen zusammenarbeiten können. Die Eigenschaften erklären ebenfalls die hohe Dynamik in einem Grid. Besonders durch die Verwaltungs-Autonomie der einzelnen Ressourcen und die lose Kopplung kann die Volatilität theoretisch beliebig steigen, und Ressourcen können in sehr kurzen Abständen Teil des Grids sein oder auch nicht.

2.1.2 Grid-Ressourcen – Definition und Verwendung

Im Grid-Kontext bezeichnet eine Ressource bzw. *Grid-Ressource* nicht nur eine physische Entität, sondern vielmehr „any capability that may be shared and exploited in a networked environment“ [CFK04], also jegliche Art von Dienst bzw. Service, wobei ein Service eine physische Ressource wie beispielsweise Rechenleistung oder Speicherplatz, aber auch echte Dienste wie Datenbanken, Daten-Transfers (Kapitel 2.2.1) oder Simulationen anbieten kann [CFK04; FK04b]. In dieser Arbeit werden die Begriffe Grid-Ressource und Ressource synonym verwendet und bezeichnen Dienste gemäß der Definition von [CFK04].

2.1.3 Virtuelle Organisationen

Administrativ genutzt und organisiert wird der im Grid betriebene Verbund aus Ressourcen über *Virtuelle Organisationen* (VO), einer konzeptionellen Grundlage für den Betrieb eines Grids [Fos+98]. Eine VO ist eine Organisationsform, die zu einem bestimmten Zweck [Wil09] mit einer festgelegten und begrenzten Lebensdauer gegründet wird und aus beliebig vielen realen Entitäten (Ressourcen, Individuen, Institutionen etc.) besteht [MSZ06]. Diese Entitäten können sowohl Ressourcen einbringen und somit als *(Ressource-)Provider* fungieren als auch Ressourcen als *(Ressource-)Consumer* nutzen [FKT01].

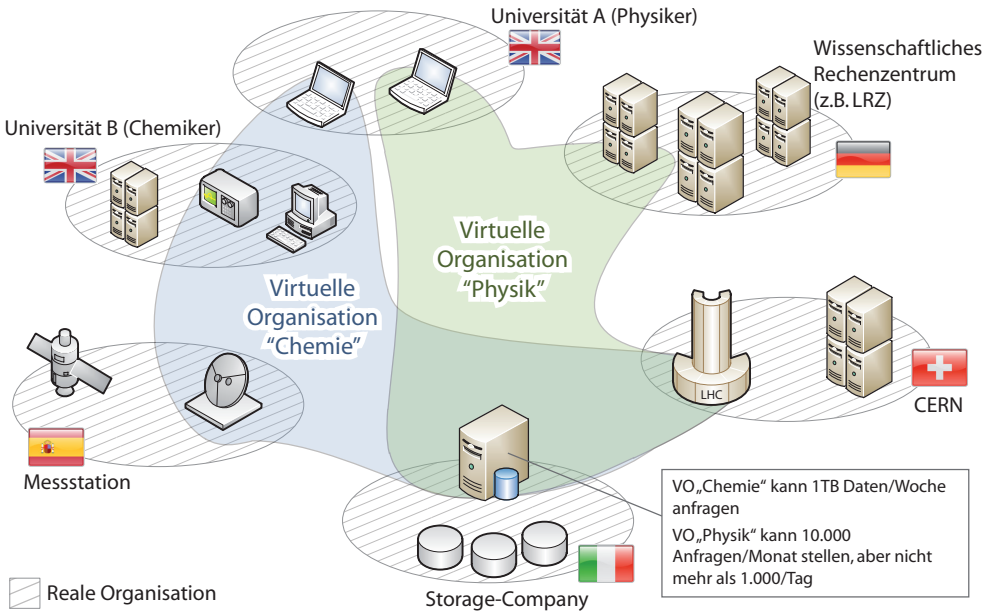


Abbildung 2.2: Beispielhafte Darstellung einer Grid-Infrastruktur und verschiedener VOs

Eine VO ermöglicht es also ganz verschiedenen realen Organisationen und Individuen, Ressourcen auf eine kontrollierte Art und Weise zu teilen, so dass die Mitglieder der VO zusammenarbeiten können, um ein gemeinsames Ziel zu erreichen [FKT01; WH03]. Zu diesem Zweck besitzen VOs immer einen (umfassenden) Regelsatz in Form von Policies, in denen die Nutzungs- und Einbringungsregeln für Ressourcen definiert werden [FKT01; CFK04] und mittels derer sichergestellt wird, dass Benutzer nur die Ressourcen verwenden können, die der VO freigegeben wurden, zu der sie gehören [BKS05]. Diese Policies enthalten beispielsweise Angaben, wie viele seiner insgesamt verfügbaren Ressourcen ein Ressource-Provider einzelnen VOs zur Verfügung stellt und zu welchen (virtuellen) Kosten dies geschieht [KB06]. Das Management virtueller Organisationen in Grids ist wegen deren Dynamik nicht trivial, wobei diese Fragestellung, mit der sich beispielsweise [Sch07] intensiv auseinandersetzt, im Rahmen dieser Arbeit vernachlässigt werden kann, auch wenn Verfügbarkeits-Informationen immer VO-gefärbt sind.

Abbildung 2.2 (Seite 5) zeigt beispielhaft eine Grid-Landschaft mit sechs verschiedenen realen Organisationen – den Universitäten A und B, einem wissenschaftlichem Rechenzentrum, dem CERN, einer Storage Company und einer Messstation – sowie zwei virtuelle Organisationen „Physik“ und „Chemie“. Die Überschneidungen der realen und virtuellen Organisationen zeigen, dass eine Ressource für die jeweilige VO zur Verfügung gestellt wird. Universität B stellt beispielsweise eine Arbeitsstation sowie ein Messinstrument der VO „Chemie“ zur Verfügung, nicht aber den universitären Cluster. Zudem sind die beiden genannten Ressourcen nur der VO „Chemie“ zugänglich, nicht aber der VO „Physik“. Die Abbildung zeigt auch, dass reale Organisationen in mehreren VOs teilnehmen können, indem sie alle oder nur einen Teil ihrer Ressourcen der VO zur Verfügung stellen [CFK04], und dass sie geographisch (stark) verteilt sein können. Am Datenbankserver der „Storage Company“ ist eine mögliche Nutzungspolicy angegeben. Die reale Organisation „CERN“ stellt mit dem „LHC“ eine Ressource bereit, die an ihren realen Ort gebunden ist, da sie wegen der immensen Installationskosten nicht mehrfach vorhanden ist. Das in Abbildung 2.2 dargestellte Grid ist ein multi-institutionales Grid. Daneben gibt es noch Cluster-Grids und Enterprise-Grids, die in dieser Arbeit aber nicht speziell behandelt werden, da die Anforderungen eines multi-institutionalen Grids am komplexesten sind und damit die Anforderungen der beiden anderen Typen abgedeckt werden.

Um die hohe Komplexität eines Grids bewältigen zu können, wurde von [FKT01] das *Layer-Pattern* [Bus+96] auf das Grid-Konzept angewandt: dabei entstanden die in Abbildung 2.3 dargestellten Abstraktionsschichten eines Grids, wobei jeweils eine Schicht auf die darunterliegenden Schichten, wie mit den Pfeilen dargestellt, zugreifen kann. Für ein besseres Verständnis gibt Abbildung 2.3 für jede Schicht ein Beispiel aus den soeben beschriebenen VOs.

Abbildung 2.3: Die verschiedenen Schichten des Grid-Layer-Stacks nach [FKT01]

2.1.4 Grid-Jobs

Der Bereich der Applikationsentwicklung für Grids sowie deren Anwendung in einem Grid wird *Grid-Computing* genannt, einer Spezialform des verteilten Rechnens [Du+04]. Beim Grid-Computing werden *Grid-Jobs* auf dem Grid ausgeführt, wobei Grid-Job jedes vom Benutzer initiierte Programm bezeichnet, das im Grid ausgeführt wird [Wil09] und dabei Ressourcen benötigt [Sch04], häufig mehrere Ressourcen parallel [CFK04; FK04b]. Diese Programme können in C, C++, Java oder jeder anderen Sprache entwickelt worden sein [Wil09]. Ein Grid-Job kann eine Laufzeit von wenigen Minuten bis zu mehreren Tagen haben [Arp95], mehrere hundert Ressourcen benötigen und sich in beliebig viele kleinere Einzeltasks aufsplitten [LB05].

Die Intention eines Grids sowie die administrativen und sicherheitsrelevanten Herausforderungen werden im „Grid-Problem“ zusammengefasst, das die flexible, sichere und koordinierte Verteilung von Ressourcen über dynamische Mengen von Individuen, Institutionen und Ressourcen beschreibt [FKT01]. Aus den Eigenschaften eines Grids und dem Grid-Problem ergeben sich die „Five Big Ideas“ des Grid-Computings: „Resource Sharing“, „Secure Access“, „Resource Use“, „Death of Distance“ und „Open Standards“ [Gri].

2.2 Betriebsbereiche eines Grids und die Rolle von Verfügbarkeits-Informationen

Der technische Betrieb eines Grids und insbesondere die technische Koordination der Ressourcen für die Ausführung eines Grid-Jobs sind so umfassend, dass sich mehrere spezialisierte Betriebsbereiche gebildet haben.

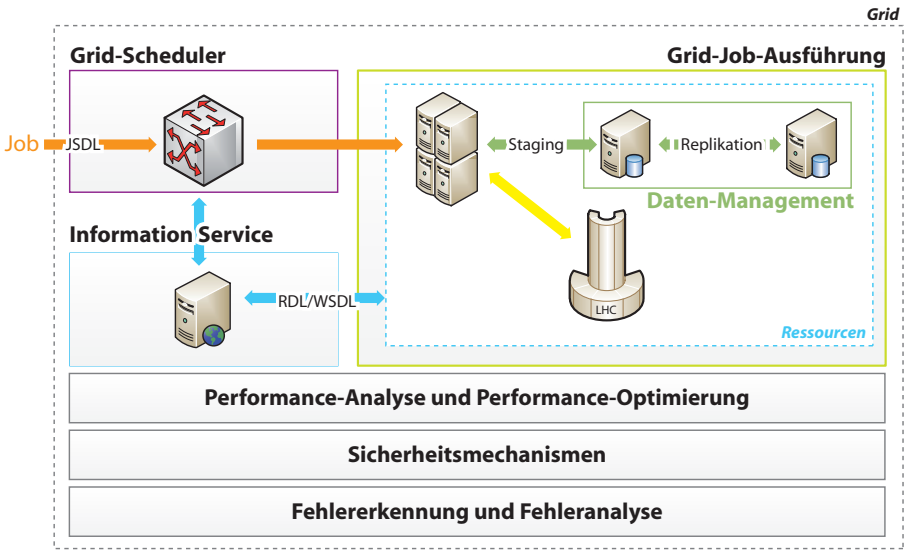


Abbildung 2.4: Die verschiedenen Betriebsbereiche eines Grids und deren Zusammenhänge bei der Ausführung eines Grid-Jobs

Abbildung 2.4 (entwickelt aus [HT04; BKS05; Joe+01]) stellt die einzelnen Betriebsbereiche sowie deren Zusammenhänge bei der Ausführung eines Grid-Jobs dar. Die Betriebsbereiche sind entsprechend der Verwendung in dieser Arbeit strukturiert, können aber ebenso

anders gruppiert und beschrieben werden. [FK04b] definiert beispielsweise den Bereich des Ressourcen-Managements als „all aspects of the process of locating various types of capability, arranging for their use, utilizing them, and monitoring their state“. Das Ressource-Management fasst also entsprechend [FK04b] das hier getrennt genannte Scheduling (Kapitel 2.2.2) mit der Instrumentierung und Überwachung von Ressourcen bzw. der Job-Ausführung (Kapitel 2.2.3) zusammen. Neben einer Beschreibung der einzelnen Betriebsbereiche wird in diesem Unterkapitel auch die jeweilige Bedeutung von Verfügbarkeits-Informationen aufgezeigt und erläutert, dass für eine korrekte Funktionsweise jeder dieser Bereiche möglichst aktuelle und umfassende Informationen über die Verfügbarkeit der einzelnen Ressourcen benötigt werden [ZS05; BS02; Bal+01], da nur mit diesen Informationen richtige, also der Intention des jeweiligen Bereichs entsprechende Entscheidungen getroffen werden können.

2.2.1 Dienst-Orientierung und Information Services

In Kapitel 2.1.2 wurden Ressourcen im Grid-Kontext als Services definiert, die physische Ressourcen oder erweiterte Dienste anbieten. Ebenso wie die Definition von Ressourcen folgt auch die allgemeine Struktur eines Grids dem Service-Paradigma und ist dementsprechend als *Service-orientierte Architektur* (SOA) [Boo+04] aufgebaut, eine Eigenschaft, die schon in Kapitel 2.1.1 in der Grid-Checkliste von [Fos02] erwähnt wurde. In einer Service-orientierten Architektur gibt es *Service Provider*, die die Services anbieten, es gibt *Service Consumer*, die die Services nutzen, und eine *Service Registry*, die die Beschreibungen der einzelnen Services bzw. Ressourcen verwaltet [Bur07]. Die Ressourcen beschreiben ihre Eigenschaften und Fähigkeiten in Meta-Daten in einem *Resource Description Language (RDL)*–Dokument. Diese Meta-Daten werden von der Service Registry, im Grid-Kontext *Information Service* oder *Directory Service* genannt, verwaltet [Cer02] und enthalten beispielsweise die Adresse, über die eine Ressource angefragt werden kann, von welchem Typ die Ressource ist oder wie sie verwendet wird. Über einen Information Service können die Teilnehmer einer VO die Existenz sowie die genannten Attribute eines Dienstes abfragen [Cza+01] und dadurch passende Ressourcen finden und verfolgen [ZS05; Fit+97; Cza+01]. Die drei Rollen Service Provider, Service Consumer und Information Service sind in Abbildung 2.4 (Seite 7) dargestellt: Der mit LHC beschriftete Large Hadron Collider bietet gemäß seiner Rolle als Service Provider einen Dienst an, nämlich die Messdaten von physikalischen Experimenten. Die in einer RDL verfasste Dienstbeschreibung wird im Information Service hinterlegt, wo entweder der Grid-Scheduler oder der Rechencluster nach solch einem Service suchen können und ihn anschließend anhand der Informationen aus dem RDL-Dokument kontaktieren und einsetzen.

Die genannten Rollen Service Provider, Service Consumer und Service Registry werden in einem Grid mittels der Technologien und Protokolle von *Web Services* implementiert. Web Services basieren auf den Konzepten einer SOA und verwenden offene und anerkannte Standards wie XML oder HTTP. Ein Web Service verwendet als RDL die *Web Service Definition Language (WSDL)* [Chr+01], der Information Service wird durch eine *Universal Description, Discovery, and Integration (UDDI)* implementiert, und SOAP [Gud+07] wird für die Kommunikation zwischen den drei genannten Rollen eingesetzt [LB05]. Ein Web Service ist eine „loosely coupled, encapsulated, platform and programming language neutral, composable server-side component that can be described, published, discovered and invoked over an internal network or on the Internet“ [LB05].

2.2.2 Grid-Scheduler

Der Grid-Scheduler ist beim Betrieb eines Grids für zwei Aufgabenbereiche zuständig.

Der **erste Aufgabenbereich** umfasst die Auswahl passender Ressourcen für die Ausführung eines Grid-Jobs sowie dessen anschließende Ausführung (Kapitel 2.2.3). Dieser Bereich ist in mehrere Schritte gegliedert [LB05]: 1) zuerst werden beim *Ressource Discovery*, was [CFK04] als „the process of querying the distributed state of the Grid to identify those resources whose characteristics and state match those desired by the resource consumer“ definiert ist, diejenigen Ressourcen gesammelt, bei denen die Verwendungsvorgaben und die Vorgaben des Grid-Jobs erfüllt werden [Wil09; LB05] und die zum Ausführungszeitpunkt verfügbar sind. 2) Aus dieser Menge an Ressourcen werden in einem zweiten Schritt bei der *Ressource Selection* die am besten passenden [LB05] Ressourcen ausgewählt [CFK04], damit eine möglichst hohe Performance erreicht wird [RL08]. 3) Nach der Generierung eines Schedules, bei der die Ressourcen einem Grid-Job zugewiesen werden [RL07], wird der Grid-Job ausgeführt (Kapitel 2.2.3). Zusammengefasst ist Grid-Scheduling „the process of mapping Grid jobs to resources over multiple administrative domains“ [LB05]. Die einzelnen Ressourcen können dabei auch Rechenzentren bzw. Cluster darstellen (Kapitel 2.1), die wiederum eigene, lokale Scheduler besitzen. Daher werden Grid-Scheduler auch *Meta-Scheduler* [LB05; Wil09] oder *Ressource-Manager* genannt [LB05].

Der **zweite Aufgabenbereich** umfasst die Verfügbarkeits-Vorhersage von Ressourcen [RL07] mittels stochastischer Methoden [BMA09; NPF08]. Die Verfügbarkeits-Vorhersage ist von besonderer Bedeutung, da Grid-Jobs mehrere Tage Laufzeit haben können (Kapitel 2.1.4) und vornehmlich Ressourcen ausgewählt werden sollten, die während der gesamten Dauer, in der der Grid-Job ausgeführt wird, verfügbar sind [RL08], um so die Effektivität des Grid-Schedulers zu verbessern und Ausführungsfehler eines Grid-Jobs zu vermeiden. Außerdem können Grid-Jobs erst zu einem späteren Zeitpunkt ausgeführt werden, beispielsweise wenn benötigte Messdaten vorliegen. In diesem Fall werden Ressourcen beim *Advance-Reservation* im Voraus für den Grid-Job reserviert, und es werden Vereinbarungen über das *wann* und *wie lange* einer Ressourcennutzung getroffen [CFK04]. Wegen der starken Volatilität der Ressourcenlandschaft (Kapitel 2.1) ist zum Zeitpunkt des Schedulings bzw. der Reservierung aber häufig unklar, welche Ressourcen in (naher) Zukunft verfügbar sein werden. Für die Erzeugung von Vorhersagen werden von bestehenden Ansätzen, wie beispielsweise [WSH99], [Pie04] oder [RL08], statistische Auswertungen und Modelle aus historischen Daten erzeugt: „prediction algorithm examines a resource's historical availability data“ [RL08].

Die Beschreibung der Aufgaben eines Grid-Schedulers macht deutlich, dass Verfügbarkeits-Informationen für eine korrekte Funktionsweise und Entscheidungsfindung von grundlegender Bedeutung sind, da ohne sie weder entschieden werden kann, welche Ressourcen für einen Grid-Job eingesetzt werden können [BMA09; HT04; WWW10; ZS05] noch Vorhersagen über die Verfügbarkeit von Ressourcen erstellt und somit für die zukünftige Verwendung und Reservierung ausgewählt werden können [BS02]. Es muss bekannt sein, welche Ressourcen verfügbar sind bzw. sein werden [BMA09]. Es wird aber nicht nur die reine Existenz von Verfügbarkeits-Informationen gefordert, sondern ebenso eine hohe Qualität der Daten. Hat der Grid-Scheduler nur unsaubere Informationen, kommt es wegen falscher Annahmen über die Verfügbarkeiten von Ressourcen zu „Job Submission Failures“, bei denen ein Grid-Job Ressourcen zugeteilt wird, die gar nicht oder nur teilweise geeignet sind [Ios+07]. Dass diese Fehler sehr häufig vorkommen – 10 bis 15 mal mehr als „Job Execution Failures“, also Fehlern während der Ausführung eines Grid-Jobs – hat [Ios+07] empirisch gezeigt. Interessante

Werte sind hierbei freie CPUs, freie Rechenknoten oder bestehende Reservierungen zu einem bestimmten Zeitpunkt [IRD05].

2.2.3 Grid-Job–Ausführung

Die Grid-Job–Ausführung enthält alle Aspekte, die während der Ausführung eines Grid-Jobs von Bedeutung sind, wie das *Staging*, ein Teilbereich des Daten-Managements (Kapitel 2.2.4), oder die Adaption laufender Programme an Veränderungen im Ausführungskontext. Beim Staging werden die benötigten Daten als Input-Daten an die Stelle transferiert, an der der Grid-Job ausgeführt wird. Die Ergebnisse der Berechnungen werden wieder als Output-Daten an die entsprechenden Stellen zurückgeschrieben. Staging kann somit als „arranging that complete files are moved to where they are needed“ [Wil09] definiert werden und ist unerlässlich für die Ausführung eines Grid-Jobs [Wil09]. Die Adaption laufender Programme bezieht sich insbesondere auf sich verändernde Ressourcen-Landschaften, da während der Ausführung eines Grid-Jobs – besonders bei langer Laufzeit – der Fall eintreten kann, dass sich die benötigten (Berechnungs-)Ressourcen während der Laufzeit verändern bzw. nicht mehr verfügbar sind [HT04; Gou+06] und sich der Grid-Job an diese Situation anpassen können muss [LB05].

Auch im Bereich der Grid-Job–Ausführung sind Verfügbarkeits-Informationen von Bedeutung, da mit ihnen die Adaption an veränderte Verfügbarkeiten der Ressourcen besser möglich ist [HT04], je nach deren Fähigkeit, auf solche Veränderungen zu reagieren [RL08]. So unterstützen beispielsweise einige Grid-Jobs Checkpointing [Conb], also das Setzen von Sicherungspunkten nach bestimmten Berechnungs- oder Ausführungsschritten. Sind Informationen bekannt, warum und wie lange eine Ressource nicht verfügbar ist, können diese Sicherungspunkte wesentlich besser bzw. überhaupt gesetzt werden, und der Grid-Job muss nicht von Beginn neu gestartet werden, sondern kann beim letzten Sicherungspunkt fortgesetzt werden [RL08].

2.2.4 Daten-Management

Die digitalen Daten, die bei der Ausführung von Grid-Jobs verarbeitet werden, haben Volumina im Terabyte- bzw. inzwischen Petabyte-Bereich [Atk+04] und sind meist über mehrere Datenbanken und Dateisysteme verteilt [Atk+04]. Das Daten-Management ist für das Daten-Discovery, das Staging (Kapitel 2.2.3), die Verlässlichkeit sowie die Performance (Kapitel 2.2.5) zuständig. Die Verlässlichkeit wird über *Replikation* erreicht [TS08; Hos+00; Joe+01]. Dabei werden mehrere Kopien von einem Datum angelegt und automatisch konsistent gehalten [ANF11]. Durch die verschiedenen Kopien wird der Datenverlust minimiert, da bei einer beschädigten Kopie oder einem fehlgeschlagenen Schreibvorgang immer der Zustand als konsistent gilt, den die meisten Kopien aufweisen [TS08]. Für den Transfer der Daten werden meist dedizierte *Grid Data Transfer Services* verwendet (Kapitel 2.2.1), wie beispielsweise *GridFTP* oder *Reliable File Transfer* (RFT) [Wil09], da die alternativen verteilten Netz-Dateisysteme nur selten zum Einsatz kommen [Wil09].

Hierbei werden Verfügbarkeits-Informationen benötigt, um abhängig von der Verfügbarkeit oder der Netz-Anbindung zu entscheiden, von welchem Replikat die Daten am besten kopiert werden sollen [HT04; ZS05]. Wichtige Werte sind beispielsweise die Latenz der Datenverbindung, die (theoretisch) verfügbare Bandbreite [IRD05] oder die Lesekapazitäten der Ressource, die die Daten speichert.

2.2.5 Performance-Analyse und Performance-Optimierung

Die Analyse und Optimierung der Performance ist Teil aller Betriebsbereiche eines Grids.

Grid-Scheduler – Die Performance eines Grid-Schedulers kann auf mehrere Arten bewertet werden, im Folgenden werden zwei davon vorgestellt. Die erste Möglichkeit, die Performance eines Grid-Schedulers zu bewerten, ist die Betrachtung der über alle Grid-Jobs gemittelten Zeitspanne zwischen der Übermittlung (Submission) und der Fertigstellung (Completion) eines Grid-Jobs, dem *Makespan* [RL08]. Je kleiner der Wert ist, desto schneller werden Grid-Jobs ausgeführt. Die zweite Möglichkeit ist die Betrachtung der Anzahl der abgebrochenen Grid-Jobs wegen Nicht-Verfügbarkeit von Ressourcen, auch *Eviction* genannt [RL08]. Hier bedeutet ein möglichst kleiner Wert ebenfalls eine gute Performance, da wenige Grid-Jobs abgebrochen werden mussten. Beide Möglichkeiten zeigen, dass die Performance eines Grid-Schedulers stark von den Verfügbarkeits-Informationen abhängt [Ios+07]. Ein kleiner Makespan-Wert ergibt sich aus der Auswahl besonders gut geeigneter Ressourcen, was umfangreiche Verfügbarkeits-Informationen voraussetzt (Kapitel 2.2.2). Ein kleiner Eviction-Wert bedeutet, dass Ressourcen ausgewählt wurden, die während der Laufzeiten der Grid-Jobs verfügbar geblieben sind. Es wurden somit gute Vorhersagen getroffen, was ebenfalls umfassende und insbesondere hochwertige Verfügbarkeits-Informationen voraussetzt (Kapitel 2.2.2). Durch die Verfügbarkeits-Vorhersage kann also die Performance vieler Grid-Anwendungen verbessert werden [MN06].

Grid-Job–Ausführung – Da die Grid-Anwendungen ihre Leistung mit Verfügbarkeits-Informationen besser anpassen können, tragen Verfügbarkeits-Informationen auch bei der Grid-Job–Ausführung zur Performance-Optimierung bei [ZS05]. Für die Analyse der häufig und unerwartet auftretenden Performance-Engpässe bei der Ausführung von Grid-Jobs sind umfangreiche Verfügbarkeits-Informationen ebenfalls wichtig [HT04; WWW10; BS02].

Daten-Management – Die Performance beim Daten-Management wird über die in Kapitel 2.2.4 beschriebene Replikation erreicht [TS08; Hos+00; Joe+01], da immer diejenige Kopie bzw. dasjenige Replikat verwendet werden kann, das am nächsten am ausgeführten Grid-Job platziert ist bzw. das die beste Leistung verspricht. Das passende Replikat kann aber nur mit ausreichenden Verfügbarkeits-Informationen ermittelt und verwendet werden.

Zusammenfassend kann festgestellt werden, dass die Gesamt-Performance eines Grids – messbar in Grad der Verwendung, Wait-and-Response-Time oder Durchsatz [Ios+07] – erheblich gesteigert werden kann, wenn die Verfügbarkeit von Ressourcen aktiv einbezogen wird [Ios+07] und dabei möglichst viele Verfügbarkeits-Informationen vorhanden sind, wie beispielsweise [Ios+07] empirisch gezeigt hat.

2.2.6 Sicherheitsmechanismen

Durch die geographische Verteilung der Ressourcen, die vielen teilnehmenden Entitäten aus verschiedenen realen Organisationen (Kapitel 2.1) und die vielfältigen Angriffsmöglichkeiten ergeben sich umfassende Sicherheitsaspekte, die beim Grid-Computing berücksichtigt werden müssen [LB05]. So muss ein Grid beispielsweise die Eigenschaften Vertraulichkeit, Integrität, Verfügbarkeit und Zusicherung erfüllen [LB05; Fos+98] und Funktionalitäten für die Authentifizierung [FKT01] und Autorisierung von Entitäten und Aktionen bereitstellen. Wie wichtig die genannten Punkte sind, zeigt sich in der Etablierung einer eigenen Infrastruktur, der *Grid Security Infrastructure* (GSI) [But+00], die ein Public-Key System, gegenseitige Authentifizierung mittels digitaler Zertifikate und Credential-Delegation mit Single Sign-On in Form

von Tools, Bibliotheken und Protokollen bündelt [LB05].

Um alle notwendigen Sicherheits- und Account-Dienste zur Verfügung stellen und sicher betreiben zu können, sind Verfügbarkeits-Informationen notwendig [HT04]: ist beispielsweise bei der Verwendung von Proxies das entfernte System kompromittiert, ist der private Schlüssel des Proxies nicht mehr geheim [LB05]. Liegen nun aber Verfügbarkeits-Informationen vor, die das entfernte System als „im Wartungsmodus für die nächsten 10 Stunden“ auszeichnen, das entfernte System aber dennoch normal reagiert, kann durch diese Information eine Überprüfung bezüglich einer Kompromittierung angestoßen werden. Auch bei der Ressourcen-Allokation durch einen Ressource-Proxy sind Verfügbarkeits-Informationen notwendig, da bei einer Ressourcen-Anfrage an eine nicht verfügbare Ressource ein *Allocation Failure* eintritt [Fos+98], der mit ausreichenden Verfügbarkeits-Informationen hätte vermieden werden können.

2.2.7 Fehlererkennung und Fehleranalyse

Die Erkennung und Analyse von Fehlern ist weniger für den direkten Betrieb eines Grids, sondern vielmehr für die Optimierung seiner Funktionsweise von Bedeutung. Je weniger Fehler auftreten, beispielsweise bei der Zuweisung von Ressourcen oder der Speicherung von Daten, desto stabiler und desto zuverlässiger können die einzelnen Grid-Jobs ausgeführt werden.

Um Fehler analysieren zu können, sind Verfügbarkeits-Informationen eine wichtige Datenbasis [BS02; IRD05; WWW10], die zu einem beliebigen Zeitpunkt ausgewertet werden kann, um Rückschlüsse zu ziehen und Verbesserungspotentiale zu identifizieren. Je detaillierter und umfassender die Verfügbarkeits-Informationen sind, desto besser können die Fehler beschrieben werden: um beispielsweise einen Fehler zu ermitteln, der in unregelmäßigen Abständen nur sehr kurz und bei einem bestimmten Grid-Job–Typ auftritt, müssen die Verfügbarkeits-Informationen sowohl eine hohe Quantität [ZS05] als auch Qualität aufweisen.

2.3 Definition von Verfügbarkeit

Die Verfügbarkeit von Ressourcen kann auf verschiedene Weise und auf unterschiedlichen Abstraktionsniveaus definiert werden. So kann sie an konkrete technische Zustände und Werte gebunden sein, wie beispielsweise in [NPF08], wo eine Ressource verfügbar ist, „if it is turned on and accessible remotely“, oder in [RL07], wo die Verfügbarkeit einer Ressource als „currently running with network connectivity, non-zero idle time, and a local CPU load of less than the CPU threshold“ definiert wird. Andere verwenden abstraktere Definitionen, beispielsweise [BSV03]: „the quality of being present or ready for immediate use“. Eine abstraktere Definition hat zwei Vorteile: 1) sie schließt die konkreteren Definitionen mit ein, denn wenn eine Ressource verwendet werden kann, muss sie auch eingeschaltet und remote erreichbar sein, 2) durch eine abstraktere Betrachtungsweise kann eine größere Bandbreite an unterschiedlichen Aspekten der Nicht-Verfügbarkeit untersucht werden. Auch wenn eine Ressource nach [NPF08] verfügbar ist – also eingeschaltet und remote erreichbar –, kann dennoch ein zu hoher CPU-Load dazu führen, dass die Ressource keine Berechnungsaufgaben mehr übernehmen kann. Eine zu spezifische Definition grenzt die Ursachenuntersuchung durch eine implizite Selektion, die bereits vor der Auswertung stattfindet, stark ein, und es können nicht alle Aspekte, die für die Ausführung eines Grid-Jobs eine Rolle spielen, betrachtet werden. Aufgrund der genannten Vorteile einer abstrakten Definition, der Service-Orientierung von Grids (Kapitel 2.1.4, 2.2.1) und der zentralen Rolle von Grid-Jobs sowie deren Ausführung

(Kapitel 2.1) wird in dieser Arbeit eine Ressource als verfügbar definiert, wenn sie „eine von einem Grid-Job geforderte Aufgabe übernehmen kann“. Es findet also eine Konzentration auf die Ausführung eines Grid-Jobs statt und nicht auf einzelne technische Werte. Neben den genannten Vorteilen einer abstrakten Definition im Rahmen dieser Arbeit entspricht eine allgemeine Betrachtung auch dem Vorgehen der Grundlagen-Literatur zum Thema der Verfügbarkeit, beispielsweise [Avi+04], wo Verfügbarkeit noch allgemeiner als „readiness for correct service“ definiert wird.

Obwohl die verwendete Definition von Verfügbarkeit recht einfach erscheint, verbirgt sich dahinter eine enorme Komplexität und Vielfältigkeit [BSV03; HT04]: eine Ressource kann aufbauend auf den in Kapitel 2.1.1 vorgestellten Eigenschaften eines Grids entweder aus *technischen* oder aus *organisatorischen* Gründen nicht verfügbar sein. **Technische Gründe** sind beispielsweise eine fehlerhafte Netzverbindung, eine fehlgeschlagene Datenbank-Anfrage oder eine zu hohe CPU-Auslastung. **Organisatorische Gründe** hängen mit den unterschiedlichen Anwendungs-Policies, dem Verhalten einzelner Individuen, den Scheduling-Mechanismen der Grid-Middlewares sowie den Wartungsarbeiten der einzelnen Ressource-Provider und VOs zusammen [Ios+07; NPF08]: so kann eine Ressource zwar technisch verfügbar sein, vom Ressource-Provider aber nur für das Grid bereitgestellt werden, wenn er die Ressource nicht selbst verwenden möchte. In diesem Fall kann der Ressource-Provider die Ressource (kurzfristig) aus dem Grid entfernen, um die gesamte Kapazität selbst nutzen zu können [NPF08]. Die organisatorischen Gründe sind jedoch nicht nur auf Grid-Ebene, sondern ebenso auf VO-Ebene anzutreffen: so kann ein Ressource-Provider entscheiden, eine Ressource nur bestimmten VOs oder sogar nur einigen VO-Teilnehmern zur Verfügung zu stellen [Sch07], nicht aber dem gesamten Grid, wobei diese Bereitstellung an verschiedene Kriterien wie Zeiträume oder bereits genutzte Kapazitäten gekoppelt sein kann. Die Ressource ist also wiederum technisch verfügbar, sie ist aber aus organisatorischen Gründen nicht für alle Teilnehmer des Grids nutzbar und daher für diese Teilnehmer trotz der technischen Verwendbarkeit als nicht verfügbar zu betrachten.

Die Ziele der Ressource-Consumer und Ressource-Provider können also sehr unterschiedlich sein [CFK04; Ios+07]. Für solche Fälle bieten verschiedene in Grids eingesetzte Systeme umfassende Möglichkeiten für die Definition sehr feingranularer Policies an. So kann beispielsweise in Condor [Cona] eine Zeitspanne definiert werden, nach der eine Ressource im Leerlauf (idle) als verfügbar gilt und ab welcher prozentualen Prozessor-Auslastung sie nicht mehr verfügbar ist [Conb; Tha+03]. Andere Ressourcen werden wiederum dediziert für das Grid bereitgestellt und betrieben und sind damit unabhängig von den genannten Einschränkungen. Da als Ressourcen neben großen Clustern zunehmend auch einzelne Notebooks oder Desktop-PCs verwendet werden, die nachts typischerweise ausgeschaltet werden, oder für Software-Installationen ein Neustart durchgeführt werden muss, verändern sich die Intervalle, in denen Ressourcen zwischen Verfügbarkeit und Nicht-Verfügbarkeit wechseln, und werden zunehmend kleiner [RL07].

2.4 Informationen über die Ursachen einer Nicht-Verfügbarkeit

Neben der binären Information über die Verfügbarkeit oder Nicht-Verfügbarkeit einer Ressource sind auch erweiterte Informationen über die Ursachen einer eventuellen Nicht-Verfügbarkeit von großer Bedeutung [SG06]. Werden neben Informationen über das *wann* auch weitere Informationen über die Ursachen der Nicht-Verfügbarkeit zur Verfügung gestellt, können daraus

Aussagen über das *wie* und *warum* abgeleitet werden [NPF08], mit denen Anwendungen für den Betrieb des Grids und ebenso Grid-Jobs besser funktionieren können.

So können diese erweiterten Informationen von Grid-Schedulern verwendet werden, um bei der Auswahl von Ressourcen (Kapitel 2.2.2) auch die Eigenschaften der Anwendungen in die Entscheidung einzubeziehen. Dies führt zu einer besseren Performance des Grids, da bessere Entscheidungen darüber getroffen werden können, welche Ressourcen verwendet werden sollen. Ist bei einer Nicht-Verfügbarkeit beispielsweise bekannt, dass diese von einem Fehler bei der Autorisierung verursacht wird, kann die Ressource für Grid-Jobs verwendet werden, die Checkpointing unterstützen: sobald ein durch eine fehlerhafte Autorisierung verursachtes Problem auftritt, kann ein Sicherungspunkt gesetzt werden, von dem aus die Berechnung fortgesetzt werden kann, sobald das Problem der fehlerhaften Autorisierung behoben wurde. Ohne die zusätzlichen Informationen wäre die Ressource für keinen Grid-Job verwendbar [RL08], und es würde unnötiger Leerlauf der Ressource entstehen. Außerdem helfen Ursachen-Informationen Grid-Schedulern bei der Vorhersage der Ressourcen-Verfügbarkeiten, die ohne diese erweiterten Informationen nur wesentlich schlechtere Entscheidungen treffen können: Grid-Scheduler, die wissen *wann*, *warum* und *wie* es zu der Nicht-Verfügbarkeit gekommen ist, können wesentlich effektiver und besser arbeiten [RL07] und beispielsweise kritische Jobs eher Ressourcen zuordnen, die zuverlässiger sind als andere [SG06]. Die Ursachen-Informationen stellen somit die Grundlage für „failure-aware predictive grid-scheduling“ dar [RL07].

Auch Anwendungen können besser reagieren, da mit den Informationen entschieden werden kann, wie der Job bei einer Nicht-Verfügbarkeit weiter agieren soll: ist eine Ressource wegen fehlerhafter Berechtigungen nicht verfügbar, kann der Grid-Job mit der Ausführung warten, bis gültige Berechtigungen eingerichtet wurden. Ohne dieses Wissen muss der Grid-Job davon ausgehen, dass die Ressource fortwährend nicht verfügbar ist, eine alternative Ressource suchen und die Berechnung von Neuem beginnen. Die Nicht-Verfügbarkeit ein und derselben Ressource kann sich auf unterschiedliche Grid-Anwendungen also unterschiedlich auswirken, abhängig von der Laufzeitumgebung und den Fähigkeiten der jeweiligen Anwendung [RL08].

Ebenso wichtig sind die Ursachen-Informationen bei der (autonomen) Fehlerbehebung und bei der Konzeption zuverlässiger Systeme [SG06], da meist nicht nur eine einzelne Ressource nicht verfügbar ist, sondern meistens mehrere, da die (unmittelbaren) Nachbarn ebenfalls in Mitleidenschaft gezogen werden [Ios+07]. Liegen erweiterte Informationen zu den Ursachen einer Nicht-Verfügbarkeit vor, muss bei den indirekt betroffenen Ressourcen keine Fehlersuche durchgeführt werden, da der Fehler einzig bei der einzelnen, als erstes ausgefallenen Ressource zu suchen ist, es kann also aufbauend auf den erweiterten Informationen bestmöglich reagiert werden.

3 Anforderungs-Analyse an die Gewinnung von Verfügbarkeits-Informationen in Grids

Aufbauend auf den in Kapitel 2 eingeführten Konzepten und Begrifflichkeiten des Grid-Computings werden in diesem Kapitel die funktionalen und nicht-funktionalen Anforderungen an die Gewinnung von Verfügbarkeits-Informationen entwickelt. Das Kapitel ist wie folgt gegliedert: zunächst werden in Kapitel 3.1 die Methodik und das allgemeine Vorgehen bei der Analyse der Anforderungen an die Gewinnung von Verfügbarkeits-Informationen beschrieben. Dabei werden der Betrachtungsgegenstand bzw. die Systemgrenzen, die Notationen und der Analyse-Prozess definiert. Anschließend wird die eingeführte Methodik in Kapitel 3.2 durchgeführt. Dabei werden Anforderungen formuliert sowie Use Cases erstellt.

In Kapitel 4 werden bestehende Ansätze sowie die in der Literatur am häufigsten genannten Implementierungen auf die Erfüllung der entwickelten Anforderungen analysiert. Die Ergebnisse der Evaluation, die in Kapitel 4 zusammengefasst werden, dienen als Ausgangsbasis für die Entwicklung des neuen Ansatzes in Kapitel 5.

3.1 Methodik der Anforderungsanalyse

3.1.1 Notationen und Begrifflichkeiten

Anforderung

Die Anforderungsanalyse stützt sich auf die Definition einer Anforderung von [Rup09], nach der eine Anforderung „eine Aussage über eine Eigenschaft oder Leistung eines Produktes, eines Prozesses oder der am Prozess beteiligten Personen“ ist. Damit sind sowohl funktionale als auch nicht-funktionale Anforderungen abgedeckt. Funktionale Anforderungen beschreiben eine konkrete, implementierbare Funktion, nicht-funktionale Anforderungen sind „alle Anforderungen, die nicht funktional sind“ [Rup09] bzw. Randbedingungen oder Qualitätsmerkmale des zu entwickelnden Systems. Eine Anforderung muss eine sogenannte „Cause-Effect“–Kette besitzen, bestehend aus einer Begründung, einer Definition und einer Messbarkeit. So wird sichergestellt, dass nur begründete Anforderungen angegeben werden und dass jede Anforderung aufgrund ihrer Messbarkeit auch auf ihre Erfüllung überprüft werden kann. Für eine einheitliche Form der Beschreibungen werden die in [Rup09] vorgestellten Anforderungs-Templates verwendet. Durch die „Cause-Effect“–Kette und die Verwendung der Templates werden die von [Rup09] und [Pre98] definierten Qualitätskriterien für Anforderungen, wie beispielsweise Korrektheit, Verständlichkeit und Konsistenz, eingehalten. Die Schlüsselworte „muss“, „sollte“ und „kann“ sollten gemäß der RFC 2119 [Bra97] interpretiert werden.

UML

Die analysierten Anwendungsfälle bzw. Use Cases werden mittels Konzepten und Notationen der *Unified Modeling Language* (UML) beschrieben, einer Sprache zur „Modellierung, Dokumentation, Spezifizierung und Visualisierung komplexer Softwaresysteme, unabhängig von deren Fach- und Realisierungsgebiet“ [RQZ07]. Als Notationssprache wird die UML gewählt, da sie umfangreiche und ausreichende Mittel zur Verfügung stellt, um die gesammelten Anforderungen zu modellieren und adäquat zu dokumentieren.

Use Case–Diagramm

Für die Beschreibung von Softwaresystemen bietet die UML eine Vielzahl von Diagrammarten, beispielsweise Klassendiagramme oder Aktivitätsdiagramme [RQZ07]. Für die Anforderungsanalyse werden jedoch nur *Use Case*–Diagramme und deren Stereotypen verwendet, da sie „die Frage beantworten, was das geplante System leisten soll“ [RQZ07], sie ein hohes Abstraktionsniveau besitzen und für die Kontextabgrenzung geeignet sind [RQZ07]. Ein Use Case–Diagramm besteht aus den im Folgenden erläuterten Elementen *Use Case*, *System* und *Akteur*.

Sowohl bei der Beschreibung der Use Cases als auch der Akteure wird das Konzept der Generalisierung bzw. Spezialisierung verwendet, wobei zwei Elemente so in Beziehung gesetzt werden, dass das eine Element eine Verallgemeinerung des anderen darstellt und das spezialisierende Element alle Eigenschaften und Beziehungen des allgemeinen Elements erbt [RQZ07].

Use Case: „A Use Case is the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system.“ [OMG07]

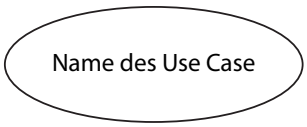


Abbildung 3.1: Darstellung eines Use Case

Ein Use Case fasst also eine Reihe von Aktionen zusammen, die nach einer schrittweisen Ausführung ein bestimmtes, „fachliches“ Ergebnis liefern [RQZ07]. Ein Use Case kann auch als „Hülle“ interpretiert werden, die den Standardfall sowie Sonder- und Fehlerfälle beinhaltet. Dargestellt wird ein Use Case in Form einer Ellipse mit dem Namen des Use Case in der Mitte (Abbildung 3.1 [RQZ07]). Weitere Eigenschaften sind in [RQZ07] dargestellt und in der UML–Spezifikation zu finden [OMG07].

Bei der späteren Definition der Use Cases wird die *include*-Beziehung verwendet. Die include-Beziehung besagt, dass ein Use Case A das Verhalten eines anderen Use Case B importiert, wobei die Einbindung nicht optional ist und der inkludierte Use Case für den aufrufenden Use Case für eine korrekte Ausführung notwendig ist [RQZ07].

Bei der Anforderungsanalyse in Kapitel 3.2.2 werden die einzelnen Use Cases entsprechend dem in Tabelle 3.1 (Seite 17) beschriebenen Schema [Kle08] angegeben.

Feld	Inhalt
ID	Eindeutige ID des Use Case, bestehend aus einer Nummerierung und einem Titel
Kurzbeschreibung	1 bis 2 Sätze zur Erläuterung
Vorbedingungen	Optionale Vorbedingungen, die vor der Ausführung des Use Case gelten müssen
Primärszenario	Abfolge der im Use Case durchgeführten Schritte
Nachbedingungen	Optionale Nachbedingungen, die nach der Ausführung des Use Case gelten müssen
Hergeleitet aus	Liste der informellen Anforderungen, aus denen der Use Case hervorgeht

Tabelle 3.1: Felder, die einen Use Case spezifizieren

System (Betrachtungsgegenstand): „Extending a classifier with the capability to own use cases.“ [OMG07]

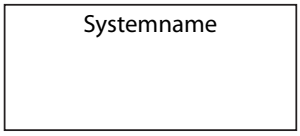


Abbildung 3.2: Darstellung des Systems

Anders als die Definitionen eines Use Case und Akteurs ist die Definition eines Systems in der UML–Spezifikation nicht ohne weitere Auseinandersetzung mit den Konzepten der UML ersichtlich. Daher wird die ebenfalls ausreichende Definition von [RQZ07] verwendet: „Das System ist diejenige Einheit, die das Verhalten, welches durch die Use Cases beschrieben wird, realisiert und anbietet“, wobei das System „durch seine zur Verfügung gestellten Anwendungsfälle und durch seine Grenzen beschrieben wird“ [Hit+05]. Das System wird als Rechteck mit dem Namen in der oberen Mitte dargestellt (Abbildung 3.2 [RQZ07]). Weitere Eigenschaften sind in [RQZ07] beschrieben und in der UML–Spezifikation zu finden [OMG07].

Akteur: „An actor specifies a role played by a user or any other system that interacts with the subject.“ [OMG07]

Abbildung 3.3: Übliche Darstellung eines Akteurs

Ein Akteur modelliert eine *Rolle*, die mit dem System interagiert. Daher muss ein Akteur nicht zwangsläufig eine natürliche Person darstellen, sondern kann ebenso einen Sensor oder ein technisches Gerät modellieren [RQZ07]. Ein Akteur kann das System benutzen, indem

er Anwendungsfälle initiiert, oder er kann vom System benutzt werden, indem er Funktionalitäten für die Realisierung von Use Cases zur Verfügung stellt [Hit+05]. Ein Akteur wird in einem Use Case–Diagramm üblicherweise als Strichmännchen dargestellt (Abbildung 3.3 [RQZ07]), es können aber auch eigene Symbole verwendet werden [RQZ07]. Weitere Eigenschaften sind in [RQZ07] dargestellt und in der UML–Spezifikation zu finden [OMG07].

3.1.2 Prozess der Anforderungsanalyse

Für die Entwicklung der funktionalen und nicht-funktionalen Anforderungen an die Gewinnung von Verfügbarkeits-Informationen (Kapitel 3.1.1) wird der in Abbildung 3.4 dargestellte Prozess (entwickelt aus [Hen10b; RQZ07; Rup09]) durchgeführt, der als Kombination aus Vorgehensmodellen von [Rup09] und der *Use Case–Analyse* [Jac92] entwickelt wurde. Beide Modelle stammen aus dem Requirements-Engineering, einem Bereich der Software-Entwicklung.

Zu Beginn des Prozesses werden in Schritt 1 die Ziele definiert, die mit dem zu entwickelnden System erreicht werden sollen, da eine klare Zieldefinition maßgeblichen Einfluss auf den Erfolg des Entwicklungsprozesses hat [Rup09] und Grundlage für die Erhebung von Anforderungen ist [Rup09]. Dabei werden sowohl eine Systemabgrenzung als auch eine Kontextabgrenzung durchgeführt, es wird also definiert, „welche Aspekte durch das geplante System abgedeckt werden sollen und welche Aspekte Teil der Umgebung dieses Systems sind“ und „welche Aspekte in der Umgebung eine Beziehung zu dem geplanten System haben“ [Rup09; RP09].

Anschließend werden in Schritt 2 in einer informellen, textuellen Beschreibung die Anforderungen an das System zur Gewinnung von Verfügbarkeits-Informationen gesammelt, die sich aus einer umfassenden Literatur-Recherche und den Eigenschaften eines Grids ergeben.

Aufbauend auf den formulierten Anforderungen werden in Schritt 3 die Akteure und Use Cases (Kapitel 3.1.1) nach den Methodiken der Use Case–Analyse ermittelt. Dabei werden verschiedene Use Case–Diagramme und detaillierte Beschreibungen erstellt, die eine genaue Definition der gewünschten Funktionalitäten in dem Use Case–Modell darstellen. Schritt 3 fasst die Schritte „Bestimmung der beteiligten Akteure“, „Bestimmung der vorhandenen Use Cases“, „Use Case Diagramme erzeugen“ und „Use Cases detailliert beschreiben und verfeinern“ aus der Use Case–Analyse [Hen10b] zusammen.

Die nicht-funktionalen Anforderungen, also jene Anforderungen, die Qualitätsmerkmale des gesamten zu entwickelnden Systems betreffen, werden in Schritt 4 erstellt.

Abschließend wird in Schritt 5 ein Evaluations-Framework entwickelt, das für die Evaluation der bestehenden Ansätze und von *DAGA* (Kapitel 5) verwendet wird. Mit dem Evaluations-Framework kann ein Ansatz auf die Erfüllung konkreter Anforderungen überprüft werden.

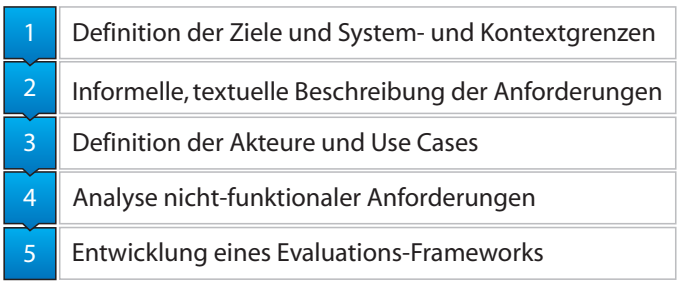


Abbildung 3.4: Durchzuführende Prozessschritte bei der Analyse und Entwicklung von Anforderungen an die Gewinnung von Verfügbarkeits-Informationen

Dieses Vorgehen wird gewählt, da es alle für die Gewinnung von Anforderungen und insbesondere für die Evaluation bestehender Ansätze wichtigen Fragen beantwortet [RQZ07]: welche Benutzer (Akteure) sind an der Gewinnung von Verfügbarkeits-Informationen beteiligt, was sind die wichtigsten Funktionalitäten und wie stehen die Akteure mit den Funktionalitäten in Beziehung.

3.2 Durchführung

3.2.1 Schritt 1 – Definition der Ziele und System- und Kontextgrenzen

Ziel der Entwicklung ist es, ein System für die Gewinnung von Verfügbarkeits-Informationen zu schaffen, das die Eigenschaften eines Grids (Kapitel 2.1) und die hohe Komplexität der Verfügbarkeiten von Grid-Ressourcen (Kapitel 2.2) berücksichtigt und somit optimale Ergebnisse bei der Generierung von Aussagen über die Verfügbarkeit bzw. deren Grad von einzelnen Grid-Ressourcen erzielen kann. Das System soll Funktionalitäten bzw. Schnittstellen bereitstellen, mit denen Zustände von Grid-Ressourcen ermittelt, publiziert und dargestellt werden können [BFR07]. Da der Grid-Job das zentrale Element bei der Lösung von Berechnungsproblemen darstellt (Kapitel 2.1), sollen die Verfügbarkeits-Informationen so erhoben und verwendbar gemacht werden, dass der Grid-Job auch hier im Mittelpunkt steht.

Die Verfügbarkeits-Informationen, die vom System generiert werden, sollen von Support-Mitarbeitern, Endbenutzern, Ressourcen–Administratoren, Ressourcen–Anbietern und Grid-Administratoren [BFR07] verwendet werden können und den Grid-Betriebsbereichen sowie Grid-Anwendungen zur Verfügung gestellt werden.

Der Kontext des zu entwickelnden Systems sind eine Grid-Infrastruktur (Kapitel 2.1), die darin vorkommende Ressourcenlandschaft sowie die (Nicht-)Verfügbarkeit einzelner Entitäten und deren Ursachen. Er umfasst dabei die in Grids üblicherweise eingesetzten Mechanismen und Technologien für das Ressourcen-Discovery, die Authentifizierung und Autorisierung von Entitäten sowie die Ausführung von Grid-Jobs. All diese Bereiche sind daher nicht Teil des zu entwickelnden Systems, sondern sie werden nur von ihm verwendet bzw. beeinflussen es.

3.2.2 Schritt 2 – Informelle, textuelle Beschreibung der Anforderungen

In diesem Kapitel wird die informelle Beschreibung der Anforderungen an die Gewinnung von Verfügbarkeits-Informationen vorgestellt, die sich aus einer umfassenden Literaturrecherche zu Verfügbarkeits-Informationen in Grids und verteilten Systemen und den in Kapitel 2.1 beschriebenen Eigenschaften eines Grids ergeben haben, wobei die in Prozessschritt 1 festgelegten Ziele verfolgt werden. Die einzelnen Anforderungen sind von IA01 bis IA11 durchnummeriert und nach dem Grad ihrer Abstraktion sortiert: Begonnen wird mit den methodischen Anforderungen, anschließend werden die Anforderungen an die Implementierung festgelegt und abschließend Anforderungen zum Betrieb angegeben. Dabei haben alle Anforderungen die in Kapitel 3.1.1 vorgegebene Form: zunächst wird eine Begründung für die Anforderung gegeben, anschließend werden daraus eine oder mehrere spezifische Anforderungsdefinitionen abgeleitet, die wiederum entsprechend der Anforderung durchnummeriert sind. Jede Anforderungsdefinition enthält eine Messbarkeit, und es wird festgelegt, wann diese Anforderung erfüllt ist.

IA01 – Dienst-Orientierung und Verwendung der Sicherheits-Mechanismen

Aufgrund der Dienst-Orientierung bei der Definition (Kapitel 2.1.2) und Verwendung (Kapitel 2.2.1) von Ressourcen, bei der Ausführung von Grid-Jobs sowie wegen der Zieldefinition der Systementwicklung (Kapitel 3.2.1) sollte sich auch das System zur Gewinnung von Verfügbarkeits-Informationen am Dienst-Paradigma orientieren, um möglichst aussagekräftige Informationen sammeln zu können. Dies bedeutet, dass Ressourcen als Dienst statt als physische Entitäten betrachtet werden und infolgedessen deren nach außen zur Verfügung gestellte Funktionalität und nicht einzelne (technische) Werte überwacht werden müssen. Dabei ist zu beachten, dass Ressourcen ihre Funktionalitäten nur authentifizierten und autorisierten Entitäten anbieten (Kapitel 2.2.6), das System also in der Lage sein muss, die Sicherheitsmechanismen eines Grids anzuwenden [BFR07; Bau+06].

Durch die Dienst-Orientierung des Systems ist auch die Betrachtung eines breiteren Spektrums an Gründen für die Nicht-Verfügbarkeit von Ressourcen möglich. Wie in Kapitel 2.3 bereits eingeführt, gibt es technische und organisatorische Gründe für die Nicht-Verfügbarkeit von Ressourcen. Da bei der Zuweisung von Grid-Jobs zu Ressourcen und der Ausführung eines Grids-Jobs nur verfügbare Ressourcen verwendet werden können (Kapitel 2.2.2), müssen beide Ursachen-Bereiche für Nicht-Verfügbarkeiten erfasst werden, da sonst keine vollständige Aussage über die Verfügbarkeit (Kapitel 2.3) einer Ressource möglich ist. Die Konzentration auf rein technische oder organisatorische Gründe reicht nicht aus, wie das Beispiel in Kapitel 2.3 bereits erläutert hat. Durch die Dienst-Orientierung ist es möglich, sowohl technische als auch organisatorische Ursachen zu ermitteln.

Ein weiterer Grund für die Dienst-Orientierung ist die Tatsache, dass bei Grid-Jobs häufig viele verschiedene Ressourcen verwendet werden: werden nur einzelne, vorher festgelegte Ressourcen überwacht, kann der Grid-Job fehlschlagen, obwohl keine der überwachten Ressourcen einen Fehler gemeldet hat. Wird dagegen der Dienst betrachtet, der möglicherweise weitere Ressourcen aggregiert oder abstrahiert, werden keine Ressourcen bei der Überwachung übersehen. Als Analogie zum Software-Engineering kann gesagt werden, dass bei der Dienst-Orientierung die legalen Abläufe betrachtet werden und nicht die illegalen.

Durch die Dienst-Orientierung kann bei der (Ressourcen-)Analyse die technische Struktur eines Grids berücksichtigt werden: die technischen Details eines Dienstes werden in Grids verschattet, d.h. sie sind für den Verwender von außen nicht sichtbar. Wird beispielsweise ein Speicher-Dienst verwendet, wird nur dessen Dienst-Adresse in Form einer URI publiziert, nicht aber, wo die vom Speicher-Dienst eingesetzten physischen Speichermedien liegen oder welche Konfiguration sie besitzen. Dadurch können einzelne physische Speichermedien zur Laufzeit ausgetauscht oder neu konfiguriert werden, ohne dass sich der Verwender der Ressource daran anpassen muss, da er durch die Verschattung von diesen Änderungen nicht betroffen ist. Dieses Szenario macht deutlich, dass bei der Ressourcen-Analyse der Dienst und nicht einzelne technische Entitäten untersucht werden müssen, um ein korrektes Abbild aus Sicht des Ressourcen-Verwenders zu erhalten.

IA01.1 – *Das System muss die analysierten Ressourcen als Dienste betrachten und damit die Ressourcen über ihr Anfrage-Interface ansprechen, um Informationen von der Ressource zu erhalten.* Die Anforderung ist erfüllt, wenn das System für die Ermittlung von Verfügbarkeits-Informationen das von der Ressource zur Verfügung gestellte Dienst-Interface verwendet und nicht invasiv einzelne technische Werte ermittelt.

IA01.2 – *Das System muss sich für das Stellen von Anfragen an Ressourcen authentifizieren und autorisieren können.* Die Anforderung ist erfüllt, wenn sich das System mit den Sicherheitsmechanismen, die im Grid verwendet werden, selbstständig authentifizieren und autorisieren kann. Die initiale Konfiguration, beispielsweise das Anlegen von notwendigen Zertifikaten, kann manuell ausgeführt werden.

IA01.3 – *Das System muss technische Gründe für die Nicht-Verfügbarkeit von Ressourcen ermitteln können.* Die Anforderung ist erfüllt, wenn das System Gründe wie beispielsweise „Keine Netzverbindung“ oder „Festplattenfehler“ erkennen kann.

IA01.4 – *Das System muss organisatorische Gründe für die Nicht-Verfügbarkeit von Ressourcen ermitteln können.* Die Anforderung ist erfüllt, wenn das System die folgenden Gründe für eine Nicht-Verfügbarkeit erkennen kann: „Überschreitung des Kontingents für VO“, „Keine Berechtigung zur Nutzung aufgrund der Nutzungs-Policy“, „Ungenügende Autorisierung“und „Wartungsarbeiten“. Die Erfüllungsbedingung beschränkt sich auf die vier genannten Gründe, um eine Messbarkeit der Anforderung zu ermöglichen. Beim Betrieb und der Verwendung eines Grids können wesentlich mehr Gründe auftreten.

IA02 – Aussagen über die Ursachen und den Grad von Nicht-Verfügbarkeiten

Wie in Kapitel 2.4 ausgeführt ist nicht nur das *wann* einer Nicht-Verfügbarkeit von Bedeutung, sondern ebenso das *wie* (z.B. in welchem Umfang) und *warum* (z.B. wegen einer Überschreitung der Nutzungs-Policy). Bei der Gewinnung von Verfügbarkeits-Informationen müssen daher Einträge der Form *Ressource A war zum Zeitpunkt t wegen einer Überschreitung der Nutzungspolicy für 30 Minuten nicht erreichbar* generiert werden, um die einzelnen Betriebsbereiche eines Grids bestmöglich unterstützen zu können.

Außerdem sollen auch der Grad einer partiellen Nicht-Verfügbarkeit und eventuell angebotene Alternativen ermittelt werden können. Wird beispielsweise angegeben, dass bei einer Speicher-Ressource nicht 100 Terabyte (TB) zur Verfügung gestellt werden können, soll ebenfalls enthalten sein, dass stattdessen 2x50TB verfügbar wären. Der Grad hängt allerdings stark vom Typ der Ressource ab und ist nicht bei allen Ressourcen-Typen möglich. So kann bei einer Berechnungs-Ressource angegeben werden, dass nur 50% der CPU-Kapazitäten zur Verfügung stehen, bei einer Datenbank-Ressource ist solch eine Aussage nicht sinnvoll.

IA02.1 – *Das System muss den Zeitpunkt einer Nicht-Verfügbarkeit ermitteln können.* Die Anforderung ist erfüllt, wenn bei jeder Nicht-Verfügbarkeit ein eindeutiger Zeitstempel angegeben werden kann.

IA02.2 – *Das System muss die technischen und organisatorischen Gründe einer Nicht-Verfügbarkeit ermitteln können.* Diese Anforderung fasst die beiden Anforderungen *IA01.3* und *IA01.4* zusammen und ist inhaltlich identisch. Sie wird dennoch angegeben, um zu verdeutlichen, dass sie sich ebenfalls aus den Erläuterungen zu *IA02* ergibt.

IA02.3 – *Das System sollte den Grad einer Nicht-Verfügbarkeit einer Ressource ermitteln können.* Die Anforderung ist erfüllt, wenn für die Ressourcen-Typen „Berechnungs-Ressource“ und „Speicher-Ressource“ (Kapitel 2.1.2) prozentual angegeben werden kann, wieviel der ursprünglich angegebenen Leistung verfügbar ist oder wenn ermittelt werden kann, in welchem Ausmaß ein Dienst nicht korrekt funktioniert, beispielsweise ob ein Berechnungsdienst das falsche Ergebnis liefert oder gar nicht erreichbar ist.

IA03 – Vollständiges und aktuelles Abbild der Verfügbarkeits-Situation zu einem bestimmten Zeitpunkt t

Das System soll die Situation der zum Zeitpunkt t im Grid vorhandenen Ressourcen möglichst vollständig und aktuell abbilden, wie es auch von [BFR07; Bau+06; Bau+09b] gefordert wird.

Die **Vollständigkeit** ist von Bedeutung, da nicht-verfügbare Ressourcen meist nicht nur lokale, sondern ebenso globale Auswirkungen im gesamten Grid haben [Ios+07]: tritt ein Fehler auf, sind durchschnittlich zehn weitere Ressourcen betroffen [Ios+07], und die Ursachen für eine Nicht-Verfügbarkeit sind breit gestreut [SG06]. Wird nur ein Teil der vorhandenen Ressourcen analysiert, führt dies zu einer unvollständigen Datenbasis, aus der falsche oder nicht optimale Entscheidungen resultieren wie beispielsweise beim Grid-Scheduling (Kapitel 2.2.2) [LB05]. Aus vollständigen Daten mit verschiedenen (Nicht-)Verfügbarkeits-Informationen kann ein Grid-Scheduler hingegen optimierte Selektionsentscheidungen ableiten [NPF08]. Zudem hilft ein vollständiges Abbild, die Dynamik in (weltweiten) Grids besser zu verstehen, bessere Langzeit-Kapazitäts-Planungen durchzuführen und zu ermitteln, welche Grid-Bereiche am zuverlässigsten sind [ZS05].

Die **Aktualität** der Daten zu einem Zeitpunkt t ist erforderlich [LB05], da es durch die dezentrale Verwaltung und hohe Volatilität häufig zu Änderungen in der Ressourcen-Landschaft kommt [ZS05; FKT01] und Ressourcen recht oft zwischen den Zuständen *verfügbar* und *nicht-verfügbar* wechseln [NPF08; Ios+07]: je nach Typ sind Ressourcen teilweise nur 1000 Minuten durchgängig verfügbar [NPF08], und die durchschnittliche Zeit zwischen zwei Nicht-Verfügbarkeiten innerhalb des Grids beträgt 12 Minuten [Ios+07]. Eine hohe Aktualität liefert die notwendige Unterstützung beim Betrieb eines Grids bezüglich der hohen Dynamik [BFR07; Bau+06], denn je aktueller die Verfügbarkeits-Informationen sind, desto wahrscheinlicher ist es, dass sie den tatsächlichen aktuellen Zustand des Grids widerspiegeln und den Kontext eines ausgeführten Grid-Jobs oder einer einzelnen Ressource korrekt wiedergeben. Eine hohe Aktualität bzw. die Erfassung in Echtzeit von Verfügbarkeits-Informationen ist essentiell für ein effektives Management, für die Fehlererkennung [ZS05] und für intelligente Scheduling-Entscheidungen [Sme+04].

Neben dem verbesserten Betrieb eines Grids unterstützt ein vollständiges und aktuelles Abbild, also die „perfekte Information“ [Ios+07], auch die Ableitung von Zuverlässigkeitsaussagen (Kapitel 8.2.2) und die Vorhersage von Ressourcen-Verfügbarkeiten. Auch geringe Verbesserungen der Vorhersage-Qualität können umfassende Auswirkungen auf die Performance haben, wie [RL08] empirisch analysiert hat. Das System soll allerdings keine Vorhersagen treffen können, sondern es soll umfassende Informationen zur Verfügung stellen, mit denen bestehende Vorhersage-Ansätze unterstützt werden.

Die Aktualität wird durch eine geringe Latenz beeinflusst, die Vollständigkeit des Abbildes durch eine Zeitsynchronisation: dauert die Übertragung der Messergebnisse von der Datener-

fassung zur Datenspeicherung aufgrund einer hohen Latenz sehr lange, wirkt sich das negativ auf die Aktualität aus. Das System soll also eine möglichst geringe Latenz bei der Übertragung der Messergebnisse von der Datenerfassung zur Datenspeicherung [Rib+98] aufweisen. Die Vollständigkeit des Abbildes wird zu einem diskreten Zeitpunkt t gemessen. Sind alle Daten vorhanden, aber mit unterschiedlichen Zeitstempeln versehen, ergibt sich daraus eine Unvollständigkeit zum Zeitpunkt t, obwohl alle Daten vorhanden sind. Daher ist für eine sinnvolle Auswertung der Daten eine Zeitsynchronisation notwendig [ZS05], um Verzögerungen bei der Datenübertragung und durch die geographische Verteilung auszugleichen und einen möglichst global geltenden Zustand zu ermitteln [MsS93].

Das Ressource-Discovery, das für die dynamische Anpassung an Veränderungen in der Ressourcen-Landschaften notwendig ist [BFR07; Bau+06], soll durch den Zugriff auf die Directory Services (Kapitel 2.2.1) realisiert werden anstatt durch die Verwendung (proprietärer) Ressourcen-Discovery–Systeme. Werden die Ressourcen aus einem allgemeinen Directory Service ermittelt, werden die Vollständigkeit und Aktualität unterstützt, und es ist kein gesonderter Entwicklungs- und Betriebsaufwand für Ressource-Discovery notwendig.

IA03.1 – *Das System muss die Menge der überwachten Ressourcen von einem im Grid verwendeten Information Service beziehen.* Die Anforderung ist erfüllt, wenn keine proprietären Systeme zur Ermittlung der vorhandenen und damit zu überwachenden Ressourcen verwendet werden, sondern ein Information Service abgefragt wird, den auch andere Grid-Anwendungen verwenden.

IA03.2 – *Die gesammelten Verfügbarkeits-Informationen müssen möglichst aktuell sein.* Die Anforderung ist erfüllt, wenn die Informationen nicht älter als fünf Minuten sind (Wert ergibt sich aus der im Text dargestellten Volatilität der Ressourcen-Verfügbarkeit). Die Anforderung wird durch eine geringe Latenz unterstützt.

IA03.3 – *Das System muss eine Zeitsynchronisation zwischen der Erfassung und den Consumern der gesammelten Verfügbarkeits-Informationen unterstützen.* Die Anforderung ist erfüllt, wenn eine systeminterne Zeitsynchronisation durchgeführt sowie eine Schnittstelle für die Zeitsynchronisation zwischen den Consumern und dem System angeboten wird. Dabei muss eine höhere Genauigkeit als beim Network Time Protocol [Mil92] erzielt werden können [ZS05].

IA04 – Autonome Suche nach Ursachen und Zusammenhängen von Nicht-Verfügbarkeiten

Wegen der vielen beteiligten Ressourcen, deren Unterschiedlichkeit und deren geographischer Verteilung (Kapitel 2.1.1) in Grid-Infrastrukturen können die Ursachen für Nicht-Verfügbarkeiten sehr komplex und versteckt sein. Um hier möglichst schnelle und zuverlässige Hilfe zu bieten, soll das System bei auftretenden Nicht-Verfügbarkeiten nicht nur die Symptome melden, sondern autonom deren Ursachen durch schrittweise Annäherung analysieren und ermitteln. Dadurch sollen zwei Vorteile erzielt werden: wegen der enormen Komplexität von Nicht-Verfügbarkeiten (Kapitel 2.3) bedeutet eine autonome Ursachenforschung eine Zeit- und Kostenersparnis [PB08], da es bei den immer komplexer werdenden Systemen schwierig ist, (Performance-)Probleme und ihre Ursachen nachzuvollziehen [Bro+02] und autonome

Systeme durch ihre Testmethodiken bereits viele Ursachen ausschließen können [Tie+98]. Zudem wird die Fehleranfälligkeit reduziert, da bei einer manuellen Ursachenanalyse die Ergebnisse stark von den Fähigkeiten des zuständigen Administrators abhängen [SG06] und dies bei einer autonomen Fehleranalyse nicht der Fall ist. Den Zusammenhang zwischen manuellen Eingriffen und der Fehleranfälligkeit hat [Ios+07] empirisch untersucht: werden Verfügbarkeits-Informationen den Grid-Schedulern nicht autonom, sondern von Menschen manuell zur Verfügung gestellt, führt das zu mehr „Job Submission Failures“ und insgesamt zehnmal mehr Fehlern als bei autonomer Intervention [Ios+07].

Die einzelnen Schritte, die bei der Analyse von Nicht-Verfügbarkeiten durchgeführt werden, sollen als Traces abrufbar sein, um den Analyse-Prozess ggf. verbessern zu können und Zusammenhänge erkennbar zu machen. Die Zusammenhänge – beispielsweise eine höhere Latenz bei Anfragen aus einem bestimmten Teil der VO – sollen bei der Performance-Optimierung des Grids verwendet werden können.

IA04.1 – *Das System muss Ursachen für Nicht-Verfügbarkeiten (partiell) autonom ermitteln können.* Die Anforderung ist erfüllt, wenn das System einen schrittweise operierenden Prozess besitzt, mit dem es sich ausgehend von einem Symptom an die Ursache des Problems annähern kann. Eine vollständige Lösung durch das autonome Vorgehen ist nicht gefordert.

IA04.2 – *Das System muss Kausalitätsketten für Nicht-Verfügbarkeiten einer Ressource erstellen können.* Die Anforderung ist erfüllt, wenn bei einer Nicht-Verfügbarkeit neben der betroffenen Ressource auch der Hergang der Nicht-Verfügbarkeit ermittelt werden kann, ähnlich dem Stack-Trace einer Java-Exception.

IA05 – Einheitlicher Ansatz zur Erhebung und Publikation der Verfügbarkeits-Informationen

Die starke Heterogenität und die geographische Verteilung der Ressourcen in einem Grid (Kapitel 2.1.1) sind eine besondere Herausforderung bei der Gewinnung von Verfügbarkeits-Informationen [Bau+09b] und der Generierung eines vollständigen und aktuellen Abbildes (Anforderung *IA03*). Dieser Herausforderung kann auf zwei Arten begegnet werden: es können lokal eingesetzte und für den jeweiligen Einsatzort gesondert konfigurierte Systeme eingesetzt werden, deren Ergebnisse kombiniert und für eine Gesamtaussage aggregiert und konsolidiert werden; alternativ kann ein globales Gesamtsystem verwendet werden, das dediziert für den Einsatz in einem Grid und die dabei entstehenden Anforderungen entwickelt wurde und bei dem eine Zusammenführung der Daten nicht notwendig ist.

Auch wenn der erste Ansatz den Quasi-Standard darstellt, ergeben sich doch verschiedene, zum Teil gravierende Nachteile. Zunächst können sich aus den einzelnen Teilsystemen Inseln bilden, die isoliert nebeneinander bestehen [BFR07; Bau+09b] und für eine gemeinsame Nutzung – beispielsweise für eine einheitliche Publikation und Lesbarkeit von Meta-Informationen über heterogene Ressourcen [BFR07] – eine Migration, Transformation und Konsolidierung der Daten erfordern, beispielsweise durch Übersetzungs-Gateways [BFR07; HAN99] oder Konsolidierungsschichten [BFR07]. Dabei entstehen jedoch systemimmanent Datenverluste, beispielsweise weil Interoperabilitätsmatrizen verwendet werden [Fie] und Felder nicht ausgefüllt

sind, weil die Vereinheitlichung teilweise gar nicht möglich ist [BFR07] oder weil unterschiedliche Ressourcen-Beschreibungen verwendet werden, was sehr komplexe Anpassungen nach sich zieht, oder Beschreibungen nur lokal verwendet werden können [BFR07].

Die Vermeidung der genannten Probleme sowie eine Vereinheitlichung der Daten ist nicht durch eine (einfache) aufgesetzte Oberflächen-Integration zu erreichen. Stattdessen muss bei der Entwicklung des Systems ein grundlegender [BFR07] Bottom-Up–Ansatz verfolgt werden, bei dem bereits bei der Planung der Architektur auf ein einziges, spezialisiertes System hingearbeitet wird. Neben der Vermeidung der oben genannten Nachteile ermöglicht solch ein einziges System, dass die gesammelten Verfügbarkeits-Informationen von allen Consumern einheitlich verstanden, verwendet und ausgewertet werden können [BFR07] und von allen Endpunkten aus darauf zugegriffen werden kann [BFR07]. Zudem entfällt der Schritt der Standardisierung vieler verschiedener Systeme und Protokolle, der für eine Vereinheitlichung bei Verwendung vieler Systeme notwendig wäre [BFR07]. Auch die Forderung nach einheitlichen Schnittstellen [BFR07; Bau+06], Metriken und Ressourcenbeschreibungen [BFR07; Bau+06] ist erfüllt bzw. hinfällig, da keine Schnittstellen mehr benötigt werden und die Metriken und Ressourcenbeschreibungen nur von dem System gestellt werden.

IA05.1 – *Das System muss eine Architektur verwenden, bei der Daten-Migrationen und -Transformationen vermieden werden.* Die Anforderung ist erfüllt, wenn die Verfügbarkeits-Situation sowohl einzelner Ressourcen als auch des gesamten Grids mittels des entwickelten Systems betrachtet werden kann und dafür keine Migration oder Transformation der Daten notwendig ist.

IA06 – Unterstützung umfangreicher Verwendungsmöglichkeiten der Daten

Die gewonnenen Verfügbarkeits-Informationen sollen vom System so zur Verfügung gestellt werden, dass sie auf möglichst vielfältige Weise verwendet werden können, dass sie von anderen Grid-Diensten, Nutzerportalen, Grid-Applikationen und Grid-Usern verwendet und weiterverarbeitet werden können [BFR07] und dass die Verwendung nicht durch systemimmanente Prozesse eingeschränkt wird. Diese Vorgaben hängen von fünf Themengebieten ab: dem *Preprocessing*, einer *Anfragesprache*, den *Datenverteilungs-Modellen*, der *Interoperabilität* und einem *Standard-UI*.

Preprocessing – Beim Preprocessing bzw. der Vorverarbeitung werden die Verfügbarkeits-Informationen verändert, bevor sie den Consumern zur Verfügung gestellt werden. Dieses Preprocessing soll in dem System weder den Datenumfang, die Datenqualität noch die Datenvielfalt negativ beeinflussen, damit es nicht zu den eingangs genannten systemimmanenten Einschränkungen kommt. Die systemimmanenten Einschränkungen können bei der Erfassung und Übertragung der Daten von der Datenerfassung zur Datenspeicherung, beim Zugriff auf die gewonnenen Daten sowie bei der Administration der Daten auftreten. Ein Beispiel für den ersten Fall wäre, dass bereits bei der Erhebung der Verfügbarkeits-Informationen Daten entfernt werden, um technische Einschränkungen bei der Übertragung der Daten von der Datenerfassung zur Datenspeicherung zu erfüllen. Diese Daten stehen den Consumern dann natürlich nicht mehr zur Verfügung, obwohl der Consumer sie eventuell hätte verwenden wollen. Ein Beispiel für den zweiten Fall wäre, dass verschiedene Sichten auf den Rohdatenbestand gebildet und dem Consumer nur diese Sichten zur Verfügung gestellt werden, wobei es vor-

kommen kann, dass manche Ressourcen mehrfach enthalten sind [BFR07] und der Consumer damit gezwungen wird, die Daten vor dem eigentlichen Einsatz zu vereinheitlichen. Besser wäre es hier, dem Consumer die Rohdaten mittels einer Anfragesprache zugänglich zu machen. Ein Beispiel für den dritten Fall wäre, dass historische Daten, die beispielsweise für statistische Auswertungen beim Scheduling (Kapitel 2.2.2) abgefragt werden können müssen und daher gespeichert werden sollten [BFR07; Bau+06; ZS05], aus technischen Gründen gelöscht werden müssen und daher den Consumern nicht mehr zur Verfügung stehen. Ist dies der Fall, kann der Scheduler weniger gute Entscheidungen treffen. Das System soll also möglichst kein Preprocessing der Daten durchführen, sondern die Verwertung der Daten den Consumern überlassen.

Anfragesprache – Um eine flexible und vielfältige Verwendung zu ermöglichen, soll das System eine Anfragesprache anbieten, mit der Consumer Verfügbarkeits-Informationen abfragen können und komplexe Anfrageoperationen durchführbar sind [BFR07; Bau+06]. Durch eine Anfragesprache werden drei Vorteile erzielt: zunächst kann der Consumer die für ihn relevanten Verfügbarkeits-Informationen sehr feingranular [BFR07; Bau+06] und individuell auswählen, beispielsweise nach einzelnen VOs [BFR07], und ist somit nicht an die meist eingeschränkten Auswahl-Optionen eines Web-Interface gebunden; des Weiteren liegen die Daten in einer maschinenlesbaren Form vor und können daher in Betriebsbereichen wie dem Grid-Scheduling oder der Grid-Job-Ausführung, aber auch in Grid-Anwendungen auf vielfältige Weise direkt verwendet werden, ohne sie in einem vorher notwendigen Schritt für die maschinelle Verarbeitung verfügbar machen zu müssen. Außerdem kann die Anfrage-Funktionalität als Ersatz eines einheitlichen Bussystems betrachtet werden, das für eine einheitliche Nutzung der Informationen ansonsten notwendig wäre [BFR07].

Datenverteilungs-Modelle – Neben der genannten Anfragesprache spielen auch die vom System angebotenen Datenverteilungs-Modelle eine wichtige Rolle für die Flexibilität und Einfachheit bei der Verwendung der Daten. Um alle Anwendungsfälle optimal und performant abdecken zu können, muss das System sowohl ein Push- als auch ein Pull-Modell anbieten [BFR07; Bau+06; ZS05] und der Open Grid Service Architecture (OGSA) [Fos+06] folgen [BFR07], da dadurch Kompatibilität zu kommenden Neuentwicklungen bei den Grid-Middlewares gLite und UNICORE sichergestellt wird [BFR07]. Beim Push-Modell werden die Daten an alle registrierten Observer geschickt, beim Pull-Modell kann der Consumer eine einzelne Anfrage an das System stellen. Wird nur eines der beiden Modelle angeboten, sind einige Funktionalitäten gar nicht möglich, beispielsweise die Benachrichtigung von Consumern, oder die Funktionalitäten haben einen hohen Overhead: möchte ein Consumer beispielsweise nur eine einzelne Anfrage stellen und das System bietet nur ein Push-Modell an, muss sich der Consumer für diese eine Anfrage registrieren. Eine einfache Pull-Anfrage wäre an dieser Stelle wesentlich effizienter [ZS05]. Außerdem können durch das Anbieten beider Modelle die verschiedenen Consumer-Verhaltensmuster, die von wenigen Interaktionen bis hin zu langlebigen Subscriptions für einen konstanten Datenstrom reichen können [ZS05], abgedeckt werden.

Interoperabilität – Da die Verfügbarkeits-Informationen, die vom System erzeugt werden, von unterschiedlichsten Grid-Anwendungen verwendet werden, müssen das System [Tie+02] und das Datenschema interoperabel sein. Dass das System interoperabel ist, wird durch die bereits genannten Anforderungen *Anfragesprache* und *Datenverteilungs-Modelle* sichergestellt. Doch auch das Datenschema muss interoperabel sein, damit es von allen Consumern gleich verstanden wird. Daher sollte vom System ein im Grid etabliertes und verbreitetes Datenschema verwendet werden, wie beispielsweise GLUE oder CIM. Wird statt eines verbreiteten ein proprietäres Datenschema verwendet, müssen alle Consumer darauf angepasst werden.

Standard-UI – Um zu vermeiden, dass für einzelne Anfragen eigene Anwendungen entwickelt werden müssen, muss das System eine Standard-Oberfläche anbieten, über die Benutzer die Daten anschauen können, beispielsweise ein Web Interface [IRD05]. Dies steht nicht im Gegensatz zu den Begründungen für die Anfragesprache, da die Standard-UI nur einen möglichen Weg bietet, den Bezug und die Verwendung der Daten aber nicht einschränkt. Neben einer reinen Darstellung der Daten sollte die Standard-UI auch Funktionalitäten anbieten, mit denen z.B. Daten angefordert [BS02] oder Anfragen in einem Anfrage-Fenster abgesetzt werden können.

Werden eine Anfragesprache und verschiedene Datenverteilungs-Modelle angeboten, wird damit auch implizit die für Grid-Anwendungen allgemein geforderte Interoperabilität [BFR07; Bau+06] ermöglicht, da andere Grid-Anwendungen und User unabhängig von deren technischem Umfeld über die generische Schnittstelle auf die Daten zugreifen können.

IA06.1 – *Das System muss die gewonnenen Verfügbarkeits-Informationen dem Consumer möglichst ohne Preprocessing zur Verfügung stellen.* Die Anforderung ist erfüllt, wenn kein Preprocessing für den Betrieb des Systems notwendig ist oder wenn das durchgeführte Preprocessing die Daten nicht negativ beeinflusst.

IA06.2 – *Das System muss eine Anfragesprache zur Verfügung stellen.* Die Anforderung ist erfüllt, wenn mittels einer Anfragesprache komplexe Anfragen formuliert werden können.

IA06.3 – *Das System muss ein Push-Modell anbieten.* Die Anforderung ist erfüllt, wenn das System die Möglichkeit bietet, einen Consumer als Observer zu registrieren, und diesen automatisch benachrichtigt, wenn sich entsprechend seinen Vorgaben Änderungen am Datenbestand ergeben. Für die Erfüllung der Anforderung ist unerheblich, ob die neuen Daten in der Benachrichtigung enthalten sind oder ob der Consumer nach der Benachrichtigung eine Pull-Anfrage stellt.

IA06.4 – *Das System muss ein Pull-Modell anbieten.* Die Anforderung ist erfüllt, wenn eine Schnittstelle vorhanden ist, über die der Consumer Anfragen auf den Datenbestand absetzen kann und die Daten maschinenlesbar zurückgeliefert werden.

IA06.5 – *Das System muss ein Standard–User-Interface zur Verfügung stellen.* Die Anforderung ist erfüllt, wenn es eine allgemein zugängliche Oberfläche gibt, über die Consumer Anfragen stellen und die Verfügbarkeits-Informationen einsehen können. Außerdem muss es einen geschützten Bereich geben, über den authentifizierte und autorisierte User administrative Aufgaben erledigen können.

IA07 – Einfache und zentrale Skalierbarkeit, Erweiterbarkeit sowie Portierbarkeit

Durch die dezentrale Verwaltung, die hohe Dynamik, mit der Ressourcen im Grid hinzugefügt und entfernt werden können, sowie die starke Heterogenität der Ressourcen (Kapitel 2.1) kommt es zu häufigen Veränderungen in der Ressourcenlandschaft eines Grids [ZS05; FKT01; Fos02; IRD05]. Bedingt durch die Verknüpfung von Grids, Mobile-Grids, Desktop-Grids und

P2P-Systemen sowie die Integration mobiler Endgeräte [ZP10] wird die Heterogenität in Grid-Infrastrukturen in Zukunft noch weiter steigen [RL07]. Um auf diese Veränderungen reagieren zu können, muss das System *skalierbar*, *erweiterbar* und *portierbar* sein.

Skalierbarkeit – Die Skalierbarkeit ist eine wichtige Anforderung an Anwendungen in einem Grid und wird sowohl in der *Grid Monitoring Architecture* (GMA) [BS02] (Kapitel 4.1.1) als auch in der Literatur, wie beispielsweise in [BS02; BFR07; ZS05], gefordert. Die Skalierbarkeit wird im Grid-Kontext in die drei Skalierungsdimensionen *administrative Skalierbarkeit*, *geographische Skalierbarkeit* und *Größenskalierbarkeit* unterteilt [KS10], wobei jede der drei Dimensionen für den Einsatz im Grid unterstützt werden muss [ZS04]. Die **administrative Skalierbarkeit** bezieht sich auf die Anzahl administrativer Domänen, die vom System gehandhabt werden können, bzw. allgemeiner auf die Fähigkeit, mehrere VOs und Mehrdomänenfähigkeit zu unterstützen. Da ein Grid aus mehreren VOs bestehen kann (Kapitel 2.1.3), muss das System, wie auch von [BFR07; Bau+06] gefordert, administrativ skalierbar sein. Die **geographische Skalierbarkeit** beschreibt die Fähigkeit, mit verschiedenen Distanzen zwischen den einzelnen Knoten umgehen zu können. Die **Größenskalierbarkeit** bezieht sich auf die Anzahl der Benutzer bzw. Consumer sowie die Anzahl der überwachten Ressourcen [IRD05; ZS05]. Sie wird durch die Vermeidung eines Single-Point-of-Failure unterstützt, wobei kein zentraler Server, sondern unabhängige, verteilte Komponenten verwendet werden [BS02].

Erweiterbarkeit – Die in der GMA [BS02] und Literatur [BFR07; Bau+06; ZS05] geforderte Erweiterbarkeit beschreibt die Flexibilität, mit der neue Ressourcen, Ressourcentypen [ZS04] und Fehlertypen sowie semantische Anpassungen wie beispielsweise neue Datenfelder sowohl statisch als auch dynamisch hinzugefügt werden können, und wie einfach diese Erweiterung durchgeführt werden kann. Besonders die dynamische Erweiterbarkeit, also die Erweiterbarkeit während des Betriebs, ist wegen der zu Beginn erwähnten hohen Dynamik von großer Bedeutung, da ohne sie nur die zum Zeitpunkt des Deployments bekannten Ressourcen-Typen überwacht und über neue Ressourcen-Typen keine Aussagen über deren Verfügbarkeit erstellt werden können [ZS05]. Die betrachteten Ressourcen-Typen können sehr vielfältig sein [BS02] (siehe „Grid-Checkliste“ Kapitel 2.1.1). Um trotz dieser Vielfalt eine möglichst gute Erweiterbarkeit erreichen zu können, müssen die für die Überwachung des Systems notwendigen Änderungsmaßnahmen, auch *Instrumentierung* genannt, möglichst gering gehalten werden [ZS05]. Anders formuliert ist Instrumentierung „the process of adding probes to generate monitoring event data“ [HT04]. Um die Erweiterbarkeit um neue Fehler- und Ressourcentypen möglichst einfach zu halten, sollen die Erweiterungen auch remote durchgeführt werden können [IRD05].

Portierbarkeit – Die Portierbarkeit definiert, wie leicht ein System in einer anderen Umgebung installiert und betrieben werden kann, beispielsweise, ob zusätzliche Treiber oder Third-Party–Software installiert werden müssen. Die Portierbarkeit, insbesondere der Datenerfassung, ist von größter Wichtigkeit, da ansonsten bestimmte Ressourcentypen nicht überwacht werden können [Wol+04; ZS05] und somit nicht zu einem vollständigen Abbild (IA03) beitragen. Für eine möglichst gute Portierbarkeit soll das System auf Software aufsetzen, die in Grids bereits eingesetzt wird, standardisierte und offene Komponenten verwenden [BS02] und mit Programmiersprachen entwickelt werden, die von den wichtigsten Grid-Middlewares unterstützt werden wie beispielsweise Java oder C++. Zusammenfassend sollen existierende Standards verwendet werden [BFR07; Bau+06], denn nur so kann nachhaltig auf die fortwährende Entwicklung der Grid-Middlewares [ZS05] reagiert werden.

Alle drei Eigenschaften sind eine wichtige Voraussetzung, um lokale Eigenentwicklungen

von Administratoren, denen die bestehenden Daten und Funktionalitäten nicht ausreichen [BFR07], zu vermeiden. Es ist wichtig, diese Eigenentwicklungen möglichst zu unterbinden, da diese wegen ihres begrenzten und individuellen Entwicklungsziels und den daraus resultierenden unzureichenden Schnittstellen [BFR07] nur lokal einsetzbar sind [BFR07]. Ein erster Schritt zur Vermeidung ist die Verwendung von einheitlichen Konventionen und einem Directory Service [BFR07]. Des Weiteren unterstützt eine auf Standards basierende Architektur die drei genannten Bereiche [FKT01].

IA07.1 – *Das System muss administrativ skalierbar sein.* Die Anforderung ist erfüllt, wenn das System in mehr als einer VO gleichzeitig Informationen über die Verfügbarkeit von Ressourcen ermitteln kann.

IA07.2 – *Das System muss geographisch skalierbar sein.* Die Anforderung ist erfüllt, wenn das System Verfügbarkeits-Informationen unabhängig von der geographischen Lage einer Ressource ermitteln kann. Nicht erfüllt wäre die Anforderung, wenn das System nur die Verfügbarkeiten eines lokalen Clusters ermitteln kann.

IA07.3 – *Das System muss größenskalierbar sein.* Die Anforderung ist erfüllt, wenn dem System während des Betriebs neue zu überwachende Ressourcen und Consumer hinzugefügt werden können.

IA07.4 – *Das System muss hinsichtlich neuer Ressourcen, Ressourcentypen, Fehlertypen und Datenfelder erweiterbar sein.* Die Anforderung ist erfüllt, wenn allen vier Bereichen während der Laufzeit neue Entitäten bzw. Einträge hinzugefügt werden können.

IA07.5 – *Das System muss eine geringe Intrusiveness haben.* Diese Anforderung wird in *IA08* ausführlich erörtert.

IA07.6 – *Das System muss portierbar sein.* Die Anforderung ist erfüllt, wenn der Systemkern ohne umfangreichen Aufwand auf eine andere Systemumgebung übertragen werden kann.

IA07.7 – *Das System muss die Menge der überwachten Ressourcen von einem im Grid verwendeten Information Service beziehen.* Identisch mit *IA03.1*.

IA07.8 – *Das System muss mit gängigen Grid-Middlewares interoperabel sein.* Die Anforderung ist erfüllt, wenn das System ohne größeren Aufwand auf den Grid-Middlewares Globus Toolkit, UNICORE und gLite installiert und eingesetzt werden kann.

IA08 – Geringe Intrusiveness

Intrusiveness bezeichnet den Grad, mit dem eine Messmethode die untersuchte Entität bzw. das beobachtete System negativ beeinflusst [MsS93]. Diese Beeinflussung umfasst sowohl die für die Messung notwendige initiale Anpassung der untersuchten Entitäten durch Einfüge- und Installationsoperationen, was auch als Instrumentierung (siehe *IA07*) bezeichnet wird

[MsS93], als auch die Last, die während der Messung auf den untersuchten Entitäten sowie im Gesamt-System, z.B. durch Netzlast, erzeugt wird.

Bei der Gewinnung von Verfügbarkeits-Informationen soll das System eine möglichst geringe Intrusiveness haben [BFR07; Bau+06; ZS05], d.h., der Mess-Prozess soll das überwachte System möglichst wenig beeinflussen [IRD05]. Das bedeutet laut der gegebenen Begriffserklärung zum Einen, dass keine Veränderung am Sourcecode oder zusätzliche Software notwendig sein dürfen [BS02] und zum Anderen, dass eine möglichst geringe Last während einer Messung erzeugt wird, damit die im Grid zur Verfügung stehenden Ressourcen möglichst vollumfänglich für die Bearbeitung von Grid-Jobs verwendet werden können. Dies kann beispielsweise durch verschiedene Mess-Policies wie „periodisch“ oder „on-demand“ unterstützt werden [ZS05], ebenso durch ein effizientes und dennoch aussagekräftiges und flexibles Datenformat [ZS05]. Eine geringe Last-Erzeugung ist nicht nur für eine sinnvolle Verwendung der vorhandenen Ressourcen notwendig, sondern ebenso für die Aussagekraft der Messungen. Konkurriert das Mess-System mit bestehenden Aufgaben um die vorhandenen Ressourcen, kann es zu Verzerrungen bei den Aussagen kommen [MsS93].

Die Bewertung der Intrusiveness wird für die spätere Evaluation entsprechend der eingangs angeführten Beschreibung in zwei Teile aufgeteilt. Der erste Teil beschreibt die notwendige Instrumentierung, wobei die folgende Skala verwendet wird: *0* für keine Änderungen an der überwachten Ressource notwendig, *1* für die Installation von Software und *2* für die Integration von zusätzlichem Sourcecode in bestehende Software. Der zweite Teil, der die erzeugte Last bzw. die notwendige Ressourcennutzung bei der Messung beschreibt, ist wesentlich schwieriger anzugeben, da dieser Wert relativiert werden muss: so ist ein zwei Megabyte (MB) Footprint auf einer Festplatte mit einer Kapazität von mehreren TB weniger relevant als bei einem Arbeitsspeicher mit wenigen MB [Tie+02]. Für die Beurteilung spielen daher die Menge der Daten, die Häufigkeit der Interaktionen zwischen der untersuchten Entität und dem Mess-System sowie die Rechenkomplexität, die bei einer Untersuchung auf der Entität entsteht, eine Rolle. All diese Eigenschaften werden zu einem Nutzungswert zusammengefasst, der zum Instrumentierungswert addiert wird. Der Nutzungswert wird wie folgt berechnet: zunächst werden die Größenordnungen in ein Verhältnis gesetzt. Am Beispiel der Festplatte würde das die Gegenüberstellung von Megabyte zu Terabyte in einem Verhältnis von 0,00001:1 bedeuten. Anschließend wird die benötigte Kapazität mit dem Verhältnis multipliziert, also $2 \cdot 0,00001$. Die Einzelergebnisse werden anschließend im Nutzungswert summiert.

Neben den hier genannten Gründen ist eine geringe Intrusiveness eine Haupt-Voraussetzung für eine möglichst gute Skalierbarkeit [ZS05] (*IA07.5*).

IA08.1 – *Das System muss bei der Gewinnung von Verfügbarkeits-Informationen einen geringen Intrusiveness-Wert besitzen.* Die Anforderung ist erfüllt, wenn der in der Begründung gegebene Bewertungsindex kleiner als 1.0 ist.

IA08.2 – *Das System muss die beiden Mess-Policies „periodisch“ und „on-demand“ unterstützen.* Die Anforderung ist erfüllt, wenn das System beide Mess-Policies anbietet.

IA08.3 – *Das System muss ein effizientes und flexibles Datenformat für den Datenaustausch verwenden.* Die Bedingung ist erfüllt, wenn weder ein proprietäres noch ein rechenintensives Datenformat für den Datenaustausch verwendet wird.

IA09 – Zuverlässigkeit

Wie bereits erläutert, sind Verfügbarkeits-Informationen unabdingbar für den (performanten) Betrieb eines Grids (Kapitel 2.2) und grundlegend für die Funktionalität von Sicherheitsmechanismen (Kapitel 2.2.6). Daraus ergibt sich eine zentrale Rolle für das System zur Gewinnung von Verfügbarkeits-Informationen: fällt es aus oder funktioniert es nicht korrekt, kann das Grid nicht weiter betrieben werden. Somit ist eine hohe Zuverlässigkeit des Systems gefordert [BFR07; Bau+06]. Zuverlässigkeit wird häufig als Kombination aus *Fehlertoleranz* und *Sicherheit* beschrieben [BGN02].

Fehlertoleranz – Dass Fehler in einem System auftreten und es kaum möglich ist, ein vollständig fehlerfreies System zu entwickeln, hat [Ios+07] empirisch gezeigt. Daher muss das System bei einem auftretenden Fehler angemessen reagieren, um den Betrieb des Grids nicht zu gefährden oder negativ zu beeinflussen. Beispielsweise soll die Situation, dass für eine Ressource keine Daten mehr zur Verfügung stehen, weil die (einzige) Datenerfassung ausgefallen ist, vermieden werden. Im Hinblick auf die geforderte geringe Intrusiveness (*IA08*) sollte die Fehlertoleranz nicht durch redundante Installationen abgesichert, sondern von Anfang an berücksichtigt [Tie+02] und durch darauf ausgelegte Architekturentscheidungen sichergestellt werden, wobei ein Single-Point-of-Failure vermieden werden muss [Tie+02].

Sicherheit – Die Zuverlässigkeit eines Systems hängt neben der technischen Verfügbarkeit, die durch eine hohe Fehlertoleranz unterstützt wird, auch von der Verlässlichkeit der ermittelten Daten ab. Werden beispielsweise Daten bei der Übertragung unerlaubterweise manipuliert, sind zwar Daten vorhanden, diese sind aber nicht korrekt. Um diese Situation zu verhindern, müssen im Grid geltende Sicherheitsstandards bei der Daten-Gewinnung, Daten-Übertragung und Daten-Abfrage eingehalten werden [Tie+02].

IA09.1 – *Das System muss eine hohe Fehlertoleranz gewährleisten.* Die Anforderung ist erfüllt, wenn es keinen (technischen) Single-Point-of-Failure gibt, wobei dies im Hinblick auf eine geringe Intrusiveness (*IA08*) nicht durch redundante Installationen, sondern durch darauf ausgelegte Architekturentscheidungen sichergestellt werden muss. Zudem müssen bei einem auftretenden Fehler innerhalb des Systems automatisch passende Gegenmaßnahmen eingeleitet werden, wie beispielsweise eine Email an den System-Administrator.

IA09.2 – *Das System muss bei der Daten-Gewinnung, -Übertragung und -Abfrage im Grid geltende Sicherheitsstandards einhalten.* Diese Anforderung ist erfüllt, wenn die üblichen Sicherheitsmechanismen der GSI bei den drei genannten Bereichen angewandt werden.

IA10 – Selbsttest des Systems

Das System muss die Fähigkeit besitzen, die eigene Funktionsweise aus technischer und inhaltlicher Sicht autonom zu überwachen.

Technischer Selbsttest – Der technische Selbsttest soll dazu verwendet werden können, die hohe Ausfallsicherheit (*IA09*) zu unterstützen bzw. zu ermöglichen. Zeichnet sich bei solch einem Selbsttest ein Problem ab, kann rechtzeitig korrigierend eingegriffen werden. Die gegensteuernden Maßnahmen können autonom durchgeführt werden, müssen aber nicht.

Inhaltlicher Selbsttest – Da bei verteilten Systemen durch die Übermittlung von Daten inhärent Datenschutzprobleme auftreten [Tan07] und die Zusicherung ein wichtiger Aspekt bei der Sicherheit in Grids ist (Kapitel 2.2.6), müssen die ermittelten Daten auf ihre Integrität überprüft werden können.

IA10.1 – *Das System muss technische Selbsttests durchführen können.* Die Anforderung ist erfüllt, wenn das System Funktionalitäten anbietet, mit denen technische Selbsttests durchgeführt werden können. Die Tests betreffen beispielsweise die Erreichbarkeit einzelner Messstationen oder die Speicherung der gewonnenen Verfügbarkeits-Informationen.

IA10.2 – *Das System muss inhaltliche Selbsttests durchführen können.* Die Anforderung ist erfüllt, wenn das System Funktionalitäten anbietet, mit denen inhaltliche Selbsttests durchgeführt werden können. Die Tests betreffen beispielsweise die Integrität und Plausibilität der gewonnenen Verfügbarkeits-Informationen.

IA11 – Überprüfung der Sicherheits-Mechanismen

Aufgrund der Eigenschaften eines verteilten Systems (Kapitel 2.1) und den sich daraus ergebenden Sicherheitsanforderungen (Kapitel 2.2.6) ist die Sicherheit beim Betrieb eines Grids von zentraler Bedeutung. Deswegen soll es mit dem System möglich sein, die korrekte Funktionsweise der Sicherheitsmechanismen zu überwachen. So soll beispielsweise überprüft werden können, ob ein Autorisierungsmechanismus korrekt funktioniert und nur den richtigen Entitäten Zugriff gewährt wird. Dafür muss das System unterschiedliche Identitäten annehmen können, da aufgrund der Identität entschieden wird, ob eine bestimmte Funktionalität ausgeführt wird oder nicht [LB05].

IA11.1 – *Das System muss die in Grids verwendeten Sicherheitsmechanismen überprüfen können.* Die Anforderung ist erfüllt, wenn die in Kapitel 2.2.6 genannten Sicherheitsmechanismen auf ihre korrekte Funktionalität überprüft werden können.

3.2.3 Schritt 3 – Definition der Akteure und Use Cases

In diesem Schritt werden aus den Zielen aus Prozessschritt 1 (Kapitel 3.2.1) und den informellen Anforderungen aus Prozessschritt 2 (Kapitel 3.2.2) die Use Cases, die mit dem System realisierbar sein sollen, sowie die beteiligten Akteure abgeleitet und beschrieben.

Akteure

Abbildung 3.5 (Seite 33) zeigt alle Akteure bzw. Rollen, die mit dem System interagieren können. Deren Definition wird entsprechend der in Kapitel 3.1.1 gegebenen Notation erstellt und verwendet für eine bessere Übersichtlichkeit die bereits erläuterte Generalisierung.

Informationsnutzer – Der Informationsnutzer ist ein abstrakter Akteur, der die vom System gewonnenen Verfügbarkeits-Informationen in beliebiger Form nutzt, sei es in maschinenlesbarer Form oder als natürlichsprachlicher Text. Die von diesem Akteur ableitenden Akteure haben entweder besondere Verwendungen für die Verfügbarkeits-Informationen oder

bestimmte Aufgaben innerhalb des Grids. Um die Verfügbarkeits-Informationen nutzen zu können, muss sich der Informationsnutzer authentifizieren und autorisieren.

Grid-Betriebsbereich – Dieser Akteur leitet vom Informationsnutzer ab, steht für die in Kapitel 2.2 vorgestellten Betriebsbereiche und verwendet die Verfügbarkeits-Informationen für die dort erläuterten Aufgaben. Die Verfügbarkeits-Informationen müssen immer in maschinenlesbarer Form vorliegen, da ein Grid-Betriebsbereich immer durch Software, aber nie durch eine natürlich Person realisiert wird.

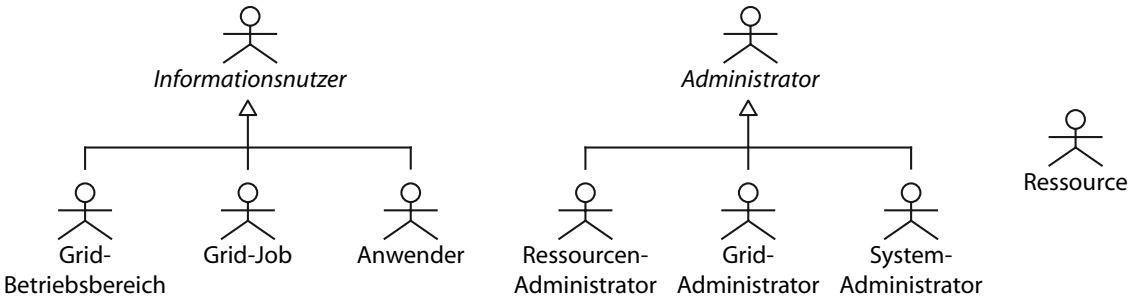


Abbildung 3.5: Alle involvierten Akteure der Use Case Analyse

Grid-Job – Der Grid-Job leitet vom Informationsnutzer ab und steht für alle Berechnungen, die im Grid ausgeführt werden (Kapitel 2.1.4). Die Verfügbarkeits-Informationen müssen ebenfalls in maschinenlesbarer Form vorliegen, werden aber für andere Dinge, wie beispielsweise das optimierte Setzen von Checkpoints, verwendet.

Anwender – Anders als der Grid-Betriebsbereich und Grid-Job beschreibt dieser Akteur einen natürlichen Anwender, der die Verfügbarkeits-Informationen nicht in maschinenlesbarer Form, sondern in natürlicher Sprache verwendet. Ein Anwender kann beispielsweise ein Support-Mitarbeiter oder ein Angestellter im Accounting-Bereich sein.

Administrator – Der Administrator ist ein abstrakter Akteur und zeichnet sich durch besondere Rechte und Pflichten innerhalb seines Aufgabengebietes aus.

Ressourcen-Administrator – Der Akteur leitet vom Administrator ab und ist für die Administration einer oder mehrerer Ressourcen (Kapitel 2.1.2) zuständig. Die Administration umfasst Aufgaben wie Wartungsarbeiten, Änderungen an den Nutzungs-Policies oder den Austausch von Hardware. In der Modellierung wird davon ausgegangen, dass der Ressourcen-Administrator auch gleichzeitig der Anbieter der Ressource ist, was in der Realität natürlich nicht immer so sein muss, für die Anforderungs-Analyse allerdings unerheblich ist.

Grid-Administrator – Der Grid-Administrator, dessen Administrationsbereich das gesamte Grid umfasst, leitet ebenfalls vom Administrator ab. Der Akteur fasst sehr viele Aufgaben zusammen, die tatsächlich von unterschiedlichen Personen und Institutionen wahrgenommen werden. Für die Analyse der Anforderungen ist dieser Sachverhalt jedoch unerheblich, da bei der Modellierung durch die Abstraktion keine wichtigen Informationen verloren gehen. Eine beispielhafte Tätigkeit ist das Hinzufügen oder Entfernen von VOs in einem Grid.

System-Administrator – Auch der System-Administrator leitet vom Administrator ab. Er ist aber für die Administration des Systems zuständig und hat damit keine Aufgaben innerhalb des Grids. Eine beispielhafte Tätigkeit ist das Hinzufügen neuer zu überwachender Ressourcen oder Ressourcentypen.

Ressource – Dieser Akteur stellt eine zu überwachende Entität entsprechend der Definition in Kapitel 2.1.2 dar.

Use Cases

Nach der Definition der verschiedenen Akteure werden nun die Use Cases mit den in Kapitel 3.1.1 angegebenen Feldern festgelegt. Für eine bessere Übersichtlichkeit sind die Use Cases in Subsysteme gruppiert, in denen jeweils verwandte oder ähnliche Use Cases enthalten sind. Die folgende Liste gibt einen Überblick über die auf den nächsten Seiten erläuterten Use Cases:

A – Gewinnung von Verfügbarkeits–Informationen
Use Cases, die die Gewinnung von Verfügbarkeits-Informationen betreffen (Abbildung 3.6, Seite 35).

UC1 – Ressource überprüfen

UC2 – Nicht-Verfügbarkeit erkennen (abstrakt)

UC2.1 – Technische Nicht-Verfügbarkeit ermitteln

UC2.2 – Organisatorische Nicht-Verfügbarkeit ermitteln

UC2.3 – Grad der Nicht-Verfügbarkeit ermitteln

UC3.1 – Ursachen für Nicht-Verfügbarkeit ermitteln

UC3.2 – Trace für die Ursachen einer Nicht-Verfügbarkeit erstellen

UC4 – Sicherheitsmechanismen überprüfen

B – Verwendung von Verfügbarkeits-Informationen
Use Cases, die die Verwendung der gewonnenen Verfügbarkeits-Informationen betreffen (Abbildung 3.7, Seite 38).

UC5.1 – Anfrage in Anfragesprache formulieren

UC5.2 – Pull-Anfrage senden

UC5.3 – Push-Anfrage senden

UC5.4 – Einzel-Anfrage über Standard-UI senden

C – System-Administration
Use Cases, die die Administration des Systems betreffen (Abbildung 3.8, Seite 40).

UC6 – Administration der gespeicherten Daten

UC7.1 – Neue zu überwachende Ressource hinzufügen

UC7.2 – Neue VO hinzufügen

UC7.3 – Neuen Fehlertyp hinzufügen

UC7.4 – Neuen Ressourcentyp hinzufügen

UC8.1 – Technischen Selbsttest durchführen

UC8.2 – Inhaltlichen Selbsttest durchführen

A – Gewinnung von Verfügbarkeits–Informationen Für die Gewinnung von Verfügbarkeits-Informationen werden die Use Cases dieses Subsystems (Abbildung 3.6) ausgeführt. Die Use Cases *UC2* und *UC2.1* bis *UC2.3* betreffen dabei die Identifikation einer Nicht-Verfügbarkeit, die Use Cases *UC3.1* und *UC3.2* die Ursachen-Ermittlung der Nicht-Verfügbarkeit.

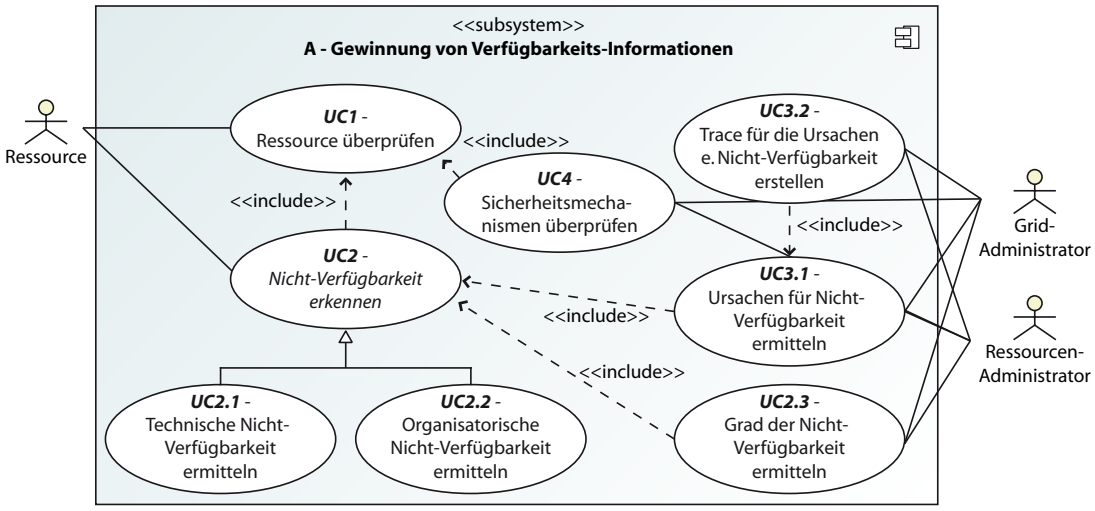


Abbildung 3.6: Use Cases im Subsystem „A – Gewinnung von Verfügbarkeits–Informationen“

ID	UC1 – Ressource überprüfen
Kurzbeschreibung	Der Use Case lagert Aktionen aus, die von mehreren anderen Use Cases mittels einer include-Beziehung (Kapitel 3.1.1) bei der Gewinnung von Verfügbarkeits-Informationen verwendet werden
Vorbedingungen	Es ist eine zu überwachende Ressource aus *UC7.1* vorhanden
Primärszenario	1) Zu überwachende Ressource aus der in *UC7.1* erstellten Liste auswählen 2) Authentifizierung und Autorisierung für die gewünschte Aktion durchführen 3) Das Dienst-Interface für die konkrete Aktion vorbereiten
Nachbedingungen	Das Dienst-Interface der zu analysierenden Ressource ist für die Verwendung vorbereitet
Hergeleitet aus	*IA01.1, IA01.2*

ID	UC2 – Nicht-Verfügbarkeit erkennen
Kurzbeschreibung	Abstrakter Use Case, der das Auffinden einer Nicht-Verfügbarkeit allgemein beschreibt und verschiedene Eigenschaften und Beziehungen für die Spezialisierung durch die Use Cases *UC2.1* und *UC2.2* zusammenfasst
Vorbedingungen	Die zu analysierende Ressource wurde in *UC1* vorbereitet
Primärszenario	1) Aktivität auf der Ressource ausführen 2) Überprüfen, ob eine Nicht-Verfügbarkeit vorliegt 3) Wenn ja, genauen Zeitpunkt erfassen
Nachbedingungen	Die allgemeinen Daten der Nicht-Verfügbarkeit wurden für die weitere Verwendung in *UC2.1* und *UC2.2* vorbereitet
Hergeleitet aus	*IA01.1, IA01.2, IA02.1 und IA02.2*

ID	UC2.1 – Technische Nicht-Verfügbarkeit ermitteln
Kurzbeschreibung	Die in *UC2* erkannte Nicht-Verfügbarkeit wird als technische Nicht-Verfügbarkeit identifiziert
Vorbedingungen	Es liegt eine Nicht-Verfügbarkeit aus *UC2* vor
Primärszenario	1) Ressource auf die in *IA01.3* definierten Gründe einer technischen Nicht-Verfügbarkeit überprüfen 2) Analyse-Ergebnisse speichern
Nachbedingungen	Die technischen Gründe einer Nicht-Verfügbarkeit wurden dokumentiert
Hergeleitet aus	*IA01.3, IA02.2*

ID	UC2.2 – Organisatorische Nicht-Verfügbarkeit ermitteln
Kurzbeschreibung	Die in *UC2* erkannte Nicht-Verfügbarkeit wird als organisatorische Nicht-Verfügbarkeit identifiziert
Vorbedingungen	Es liegt eine Nicht-Verfügbarkeit aus *UC2* vor
Primärszenario	1) Ressource auf die in *IA01.4* definierten Gründe einer organisatorischen Nicht-Verfügbarkeit überprüfen 2) Analyse-Ergebnisse speichern
Nachbedingungen	Die organisatorischen Gründe einer Nicht-Verfügbarkeit wurden dokumentiert
Hergeleitet aus	*IA01.4, IA02.2*

ID	UC2.3 – Grad der Nicht-Verfügbarkeit ermitteln
Kurzbeschreibung	Der Use Case wird ausgeführt, um den Grad einer Nicht-Verfügbarkeit, sei sie technischer oder organisatorischer Natur, zu ermitteln.
Vorbedingungen	Es liegt eine Nicht-Verfügbarkeit aus *UC2* vor und wurde mittels *UC2.1* bzw. *UC2.2* genauer identifiziert. Die überprüfte Ressource muss die in *IA02.3* angegebenen Eigenschaften besitzen
Primärszenario	1) Analysieren, wie viele der angeforderten Kapazitäten der Ressource zur Verfügung stehen bzw. in welchem Umfang die Analyse-Aktion korrekt ausgeführt wurde, beispielsweise, ob ein Berechnungsergebnis korrekt ist 2) Das Analyse-Ergebnis wird gespeichert
Nachbedingungen	Der Grad einer einer Nicht-Verfügbarkeit wurde dokumentiert
Hergeleitet aus	*IA02.3*

ID	UC3.1 – Ursachen für Nicht-Verfügbarkeit ermitteln
Kurzbeschreibung	Durch einen schrittweise operierenden Prozess wird die Ursache einer Nicht-Verfügbarkeit ermittelt
Vorbedingungen	Es liegt eine Nicht-Verfügbarkeit aus *UC2* vor
Primärszenario	1) Schrittweise Annäherung an die Ursache einer Nicht-Verfügbarkeit
Nachbedingungen	Die Ursache einer Nicht-Verfügbarkeit wurde identifiziert
Hergeleitet aus	*IA04.1*

ID	UC3.2 – Trace für die Ursachen einer Nicht-Verfügbarkeit erstellen
Kurzbeschreibung	Aus vorhandenen Teil-Ursachen für eine Nicht-Verfügbarkeit wird ein Ursachen-Trace erstellt
Vorbedingungen	Es liegt eine Nicht-Verfügbarkeit aus *UC2* vor, zu der in *UC3.1* Teil-Ursachen ermittelt wurden
Primärszenario	1) Ausgabe der in *UC3.1* ermittelten Teil-Ursachen in Form eines maschinenlesbaren Traces
Nachbedingungen	Es wurde ein Trace aus Teil-Ursachen für eine Nicht-Verfügbarkeit erstellt
Hergeleitet aus	*IA04.2*

ID	UC4 – Sicherheitsmechanismen überprüfen
Kurzbeschreibung	Der Use Case überprüft die korrekte Funktionsweise der Sicherheitsmechanismen in einem Grid an einer konkreten Ressource
Vorbedingungen	Die zu überprüfende Ressource wurde mittels *UC1* für die Analyse vorbereitet
Primärszenario	1) Die korrekte Funktionalität der Ressource überprüfen 2) Erneute Authentifizierung und Autorisierung am Dienst-Interface der überprüften Ressource mit einer anderen Identität 3) Erneute Überprüfung der korrekten Funktionalität der Ressource 4) Ergebnis der Analyse protokollieren
Nachbedingungen	Die Feststellung, ob die Sicherheitsmechanismen einer Ressource bzw. ihres Dienst-Interfaces korrekt funktionieren, wurde protokolliert
Hergeleitet aus	*IA11.1*

B – Verwendung von Verfügbarkeits-Informationen Die Verfügbarkeits-Informationen, die im Subsystem A ermittelt wurden, werden in diesem Subsystem in den verschiedenen Use Cases verwendet (Abbildung 3.7).

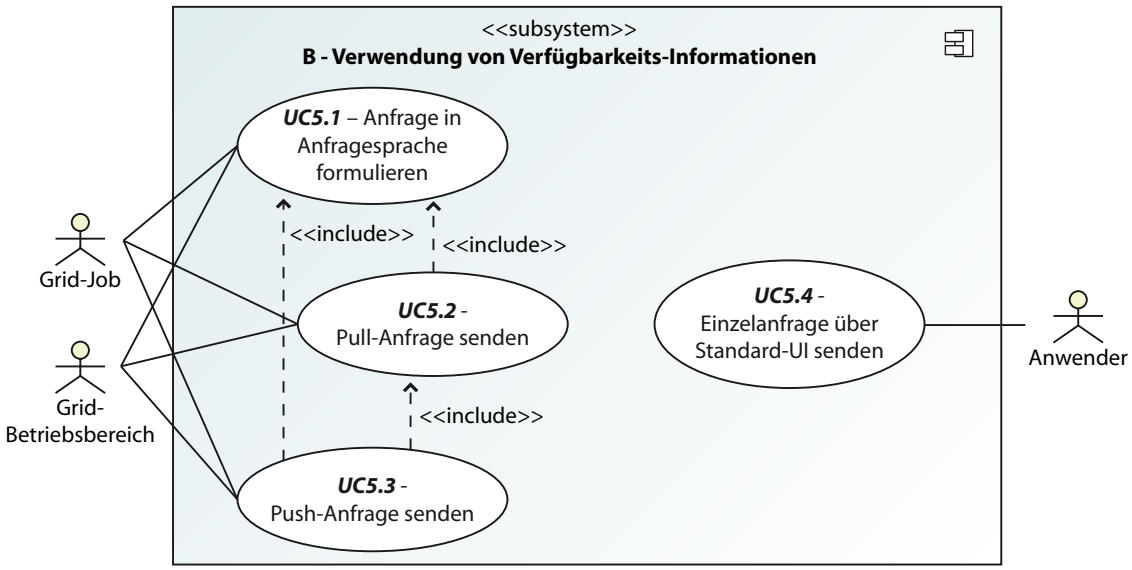


Abbildung 3.7: Use Cases im Subsystem „B – Verwendung von Verfügbarkeits-Informationen“

ID	UC5.1 – Anfrage in Anfragesprache formulieren
Kurzbeschreibung	Für die Abfrage der gewonnenen Verfügbarkeits-Informationen wird eine Anfrage formuliert
Vorbedingungen	*keine*
Primärszenario	1) Formulierung einer Anfrage, mit der eine Untermenge an relevanten Daten aus den vorhandenen Verfügbarkeits-Informationen ausgewählt werden können
Nachbedingungen	Es steht ein generischer Anfrage-Mechanismus zur Verfügung, der in anderen Use Cases verwendet werden kann
Hergeleitet aus	*IA06.2*

ID	UC5.2 – Pull-Anfrage senden
Kurzbeschreibung	Ein Informationsnutzer stellt eine Pull-Anfrage, d.h. eine Query/Response-Anfrage an das System, das die Anfrage mit den gewünschten Verfügbarkeits-Informationen beantwortet
Vorbedingungen	Es steht der Anfrage-Mechanismus aus *UC5.1* zur Verfügung
Primärszenario	1) Der Informationsnutzer formuliert eine Anfrage mit einschränkenden Kriterien für die Auswahl der relevanten Verfügbarkeits-Informationen 2) Der Informationsnutzer sendet die Anfrage an das System 3) Das System liefert die zur Anfrage passenden Daten in maschinenlesbarer Form zurück, da die beteiligten Akteure *Grid-Job* und *Grid-Betriebsbereich* die Daten nur in dieser Form verarbeiten können
Nachbedingungen	Der anfragende Informationsnutzer kann die angeforderten Verfügbarkeits-Informationen verwenden
Hergeleitet aus	*IA06.4*

ID	UC5.3 – Push-Anfrage senden
Kurzbeschreibung	Ein Informationsnutzer registriert sich über eine Push-Anfrage als Observer beim System
Vorbedingungen	*keine*
Primärszenario	1) Formulierung einer Anfrage mit einschränkenden Kriterien für die Auswahl der relevanten Verfügbarkeits-Informationen 2) Der Informationsnutzer verwendet die vom System bereitgestellte Schnittstelle, um sich als Observer für die formulierte Anfrage zu registrieren 3) Bei einer Änderung des Datenbestandes, die eine Ergebnismenge für die bei der Registrierung angegebene Anfrage enthält, informiert das System den Informationsnutzer 4) Der Informationsnutzer führt daraufhin *UC5.1* aus
Nachbedingungen	Bei jeder Datenänderung, die zu den Kriterien des Observers passt, wird dieser darüber benachrichtigt.
Hergeleitet aus	*IA06.3*

ID	UC5.4 – Einzel-Anfrage über Standard-UI senden
Kurzbeschreibung	Ein Anwender verwendet das vom System bereitgestellte Standard–User-Interface (UI), um auf die gewonnenen Verfügbarkeits-Informationen zuzugreifen
Vorbedingungen	Es existiert ein Standard-UI
Primärszenario	1) Der Anwender startet die Standard-Benutzeroberfläche, d.h. das Standard-UI 2) Innerhalb der Standard-UI wählt der Anwender mit den dort vorhandenen Eingabe-Elementen die anzuzeigenden Daten aus 3) Die Auswahl wird an das System gesendet 4) Das System liefert die passenden Daten an das Standard-UI, das die Daten für den Anwender grafisch aufbereitet darstellt
Nachbedingungen	*keine*
Hergeleitet aus	*IA06.5*

C – System-Administration Die Use Cases innerhalb dieses Subsystems enthalten die Aktionen für die Administration des Systems und betreffen die Administration der gewonnenen Verfügbarkeits-Informationen, die Erweiterung des Systems sowie verschiedene Selbsttests (Abbildung 3.8).

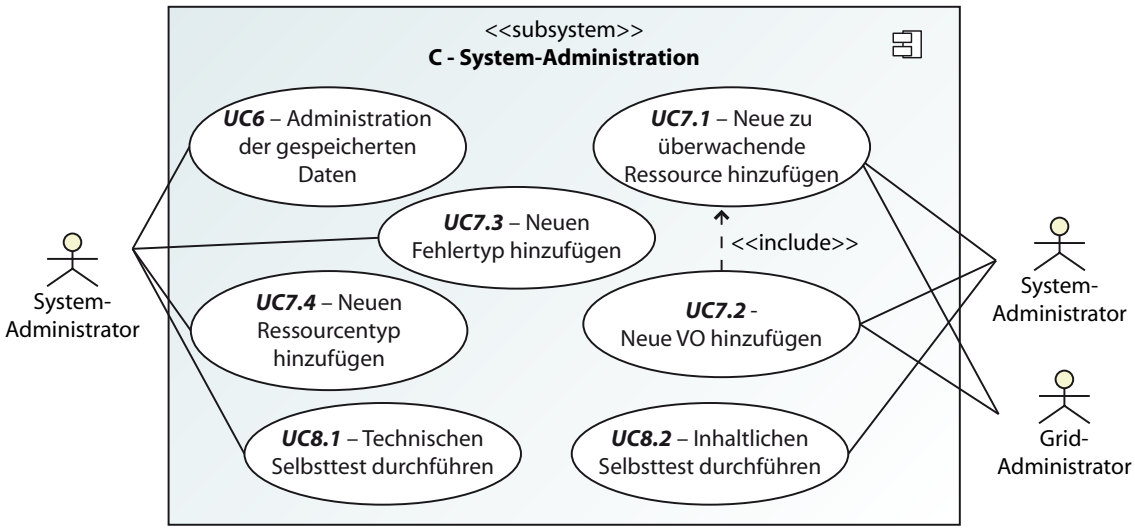


Abbildung 3.8: Use Cases im Subsystem „C – System-Administration“

ID	UC6 – Administration der gespeicherten Daten
Kurzbeschreibung	Der System-Administrator verwaltet die gespeicherten Verfügbarkeits-Informationen
Vorbedingungen	Es existiert ein Administrations-Interface
Primärszenario	1) Der System-Administrator authentifiziert und autorisiert sich beim Administrations-Interface 2) Der System-Administrator löscht einen Test-Datensatz
Nachbedingungen	Die Änderungen des System-Administrators wurden persistent im System gespeichert
Hergeleitet aus	*IA06.5*

ID	UC7.1 – Neue zu überwachende Ressource hinzufügen
Kurzbeschreibung	Es wird eine neue Ressource hinzugefügt, die vom System überwacht werden soll. Der Use Case kann sowohl während des Betriebs als auch während einer initialen Konfiguration ausgeführt werden
Vorbedingungen	*keine*
Primärszenario	1) Das System fragt für jede überwachte VO den entsprechenden Directory Service an 2) Der jeweilige Directory Service sendet eine aktuelle Liste mit registrierten Ressourcen bzw. Diensten 3) Die Liste wird für die Aktualisierung der internen Liste verwendet
Nachbedingungen	Die interne Liste der durch das System zu überwachenden Ressourcen wurde aktualisiert und enthält die zum Abfragezeitpunkt geltende Ressourcenlandschaft
Hergeleitet aus	*IA03.1, IA07.7, IA07.3*

ID	UC7.2 – Neue VO hinzufügen
Kurzbeschreibung	Es wird eine neue VO hinzugefügt, deren Ressourcen überwacht werden sollen. Der Use Case kann sowohl während des Betriebs als auch während einer initialen Konfiguration ausgeführt werden
Vorbedingungen	*keine*
Primärszenario	1) Die neue VO wird der Liste der zu verwaltenden VOs hinzugefügt 2) Die interne Liste der zu überwachenden Ressourcen wird mit *UC7.1* aktualisiert
Nachbedingungen	Die Ressourcen der neu hinzugefügten VO werden überwacht
Hergeleitet aus	*IA07.1*

ID	UC7.3 – Neuen Fehlertyp hinzufügen
Kurzbeschreibung	Während des Betriebs wird ein neuer Fehlertyp hinzugefügt, der eine Nicht-Verfügbarkeit als solche identifiziert
Vorbedingungen	*keine*
Primärszenario	1) Der System-Administrator authentifiziert und autorisiert sich beim Administrations-Interface 2) Die notwendigen Daten werden aktualisiert bzw. eingefügt
Nachbedingungen	Der eingetragene Fehler wird ab sofort als Ursache für eine Nicht-Verfügbarkeit erkannt und kann in den Use Cases *UC2.** verwendet werden
Hergeleitet aus	*IA07.4*

ID	UC7.4 – Neuen Ressourcentyp hinzufügen
Kurzbeschreibung	Während des Betriebs wird ein neuer Ressourcentyp hinzugefügt
Vorbedingungen	*keine*
Primärszenario	1) Der System-Administrator authentifiziert und autorisiert sich beim Administrations-Interface 2) Die notwendigen Daten werden aktualisiert bzw. eingefügt
Nachbedingungen	Der hinzugefügte Ressourcentyp kann ab sofort in den Use Cases *UC1* und *UC2.** analysiert werden
Hergeleitet aus	*IA07.4*

ID	UC8.1 – Technischen Selbsttest durchführen
Kurzbeschreibung	Das System führt einen technischen Selbsttest durch, um Fehlfunktionen zu erkennen und ggf. zu reagieren
Vorbedingungen	*keine*
Primärszenario	1) Generierung künstlicher Verfügbarkeits-Informationen 2) Speicherung der generierten Daten im System über den in den Use Cases *UC2.** verwendeten Protokollierungsmechanismus 3) Überprüfen, ob die Speicherung erfolgreich war, indem die gespeicherten Daten mittels *UC5.4* abgefragt wird 4) Bei einem Fehlverhalten wird eine konfigurierte Aktion ausgeführt
Nachbedingungen	*keine*
Hergeleitet aus	*IA10.1*

ID	UC8.2 – Inhaltlichen Selbsttest durchführen
Kurzbeschreibung	Das System führt einen inhaltlichen Selbsttest durch, um Fehlfunktionen und Kontaminationen zu erkennen und ggf. zu reagieren
Vorbedingungen	*keine*
Primärszenario	1) Generierung künstlicher Verfügbarkeits-Informationen 2) Manipulation der generierten Daten 3) Speicherung der generierten Daten im System über den in den Use Cases *UC2.** verwendeten Protokollierungsmechanismus 4) Die generierten und manipulierten Daten mittels *UC5.4* auslesen 5) Es wird überprüft, ob die Manipulation erkannt wird 6) Wenn nicht, wird eine konfigurierte Aktion ausgeführt
Nachbedingungen	*keine*
Hergeleitet aus	*IA10.2*

3.2.4 Schritt 4 – Analyse nicht-funktionaler Anforderungen

Da nicht-funktionale Anforderungen (NFA) Randbedingungen oder Qualitätsmerkmale des gesamten Systems darstellen (Kapitel 3.1.1), können sie nicht in Use Cases abgebildet werden und werden daher nachfolgend gesondert aufgelistet. Alle nicht-funktionalen Anforderungen ergeben sich aus der informellen, textuellen Beschreibung in Prozessschritt 2. Jede Anforderung hat eine eindeutige Bezeichnung (NFA{1–6}), eine Erläuterung und eine Angabe, aus welchen informellen Anforderungen sie abgeleitet wurde.

NFA1 - Verwendung eines einheitlichen und etablierten Datenschemas
Um eine hohe Interoperabilität zu gewährleisten und (rechenintensive) Datentransformationen zu vermeiden, muss das System ein in Grids verbreitetes Standard-Datenschema verwenden. Das Datenschema muss bei der Gewinnung, der Speicherung und der Übertragung der Verfügbarkeits-Informationen eingesetzt werden.
→ *abgeleitet aus IA06, IA07, IA08*

NFA2 - Geringe Intrusiveness
Während der gesamten Entwicklung des Systems muss darauf geachtet werden, einen möglichst geringen Intrusiveness-Wert zu erreichen. Die Berechnung des Wertes ist in *IA08* erläutert.
→ *abgeleitet aus IA07.5, IA08.1*

NFA3 - Einhaltung von Sicherheitsstandards
In allen Funktionsbereichen des Systems, d.h. bei der Gewinnung, Verteilung, Speicherung und Nutzung der Daten, müssen die in Grids üblichen Sicherheitsmechanismen, die wie in Kapitel 2.2.6 beschrieben in der GSI zusammengefasst sind, verwendet werden [FKT01], um Vertraulichkeit, Integrität und Verfügbarkeit zu erzielen.
→ *abgeleitet aus IA09.2*

NFA4 - Hohe Aktualität
Das System muss geeignete Vorkehrungen für eine möglichst hohe Aktualität der gewonnenen Daten in allen Funktionsbereichen treffen, beispielsweise eine effiziente Kommunikation und eine hohe Performance, die durch einen geringen Intrusiveness-Wert unterstützt werden (*NFA2*). Das Alter der Daten darf nicht über fünf Minuten liegen.
→ *abgeleitet aus IA03.2*

NFA5 - Zeitsynchronisationsmechanismus
Das System muss einen globalen Zeitsynchronisationsmechanismus verwenden, der bei der Erfassung und Speicherung der gewonnenen Verfügbarkeits-Informationen eingesetzt und den Informationsnutzern als Schnittstelle angeboten wird.
→ *IA03.3*

NFA6 - Keine Daten-Migration und kein Daten-Preprocessing
In keinem der Funktionsbereiche Gewinnung, Verteilung, Speicherung oder Nutzung darf eine systembedingte Datenmigration oder ein Daten-Preprocessing durchgeführt werden, durch die für den Informationsnutzer interessante Daten verloren gehen könnten und er dadurch bei der Auswahl von Daten eingeschränkt wird.
→ *abgeleitet aus IA05.1, IA06.1*

3.2.5 Schritt 5 – Entwicklung eines Evaluations-Frameworks

Im letzten Schritt des in Kapitel 3.1.2 vorgestellten Analyseprozesses wird ein Evaluations-Framework, bestehend aus einer Ergebnistabelle und Prozessvorgaben, erstellt, mit dem sowohl die bestehenden Ansätze (Kapitel 4) als auch der neu entwickelte Ansatz DAGA (Kapitel 6.4, 7.5) evaluiert werden. Die Tabelle enthält die im Prozessschritt 3 entwickelten Use Cases und die in Prozessschritt 4 definierten nicht-funktionalen Anforderungen.

Ergebnistabelle

Die Ergebnistabelle (Abbildung 3.9) fasst in einer Übersicht die Details der jeweiligen Evaluation zusammen und ermöglicht so einen schnellen Überblick über die Analyseergebnisse.

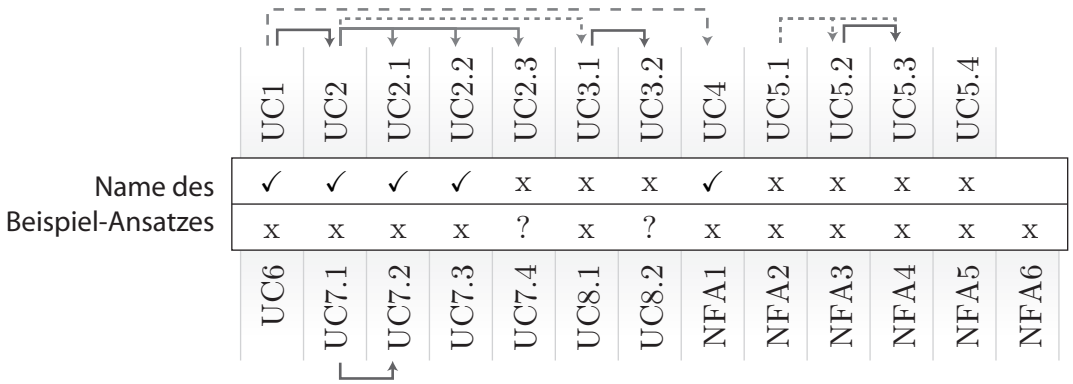


Abbildung 3.9: Ergebnistabelle zur Darstellung der Evaluationsergebnisse

Ein Häkchen (✓) in der Ergebnistabelle bedeutet, dass der entsprechende Ansatz den Use Case wie beschrieben umsetzen kann bzw. die nicht-funktionale Anforderung erfüllt. Ist ein Kreuz (x) gesetzt, ist dies nicht der Fall. Bei einem Fragezeichen (?) ist eine Aussage nicht möglich bzw. hängt die Erfüllung von einer konkreten Implementierung ab.

Die Pfeile zwischen den einzelnen Use Cases stellen deren Abhängigkeiten und Vorbedingungen dar und wie sich die Nicht-Erfüllung einer Anforderung auf andere Anforderungen auswirkt. Ist beispielsweise *UC3.1* nicht erfüllt, kann *UC3.2* ebenfalls nicht erfüllt werden, da dieser *UC3.1* mittels einer include-Beziehung integriert und daher die korrekte Ausführung von *UC3.1* für die korrekte Ausführung von *UC3.2* zwingend notwendig ist (Kapitel 3.1.1). Die Angabe der Kaskadierungspfeile ermöglicht eine wesentlich schlankere Evaluationsbegründung, da bei der Nicht-Erfüllung von *UC3.1* nicht bei jedem untersuchten Ansatz erneut erläutert werden muss, dass wegen der include-Beziehung *UC3.2* ebenfalls nicht erfüllt werden kann. Die Kaskadierung der Ergebnisse gilt neben der include-Beziehung auch für die Generalisierung (Kapitel 3.1.1) und betrifft nur die Nicht-Erfüllung, bei der Erfüllung einer Anforderung können keine Rückschlüsse gezogen werden. Da nicht-funktionale Anforderungen Qualitätsmerkmale darstellen und somit das gesamte System betreffen (Kapitel 3.1.1), gibt es keine konkreten Abhängigkeiten und somit auch keine Kaskadierungspfeile von oder zu einzelnen nicht-funktionalen Anforderungen.

Prozessvorgaben

Um nachvollziehbare, überprüfbare und einheitliche Ergebnisse bei der Evaluation zu erhalten, enthält das Evaluations-Framework neben der eingeführten Ergebnistabelle auch einen festgelegten Prozess (Abbildung 3.10), nach dem die Evaluation der verschiedenen Ansätze durchgeführt werden muss.

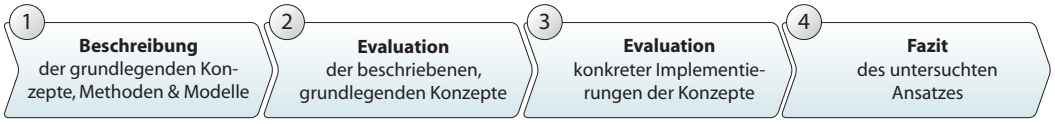


Abbildung 3.10: Prozessschritte, die vom Evaluations-Framework für die Analyse der verschiedenen Ansätze vorgegeben werden

Schritt 1 – Beschreibung der grundlegenden Konzepte, Methoden und Modelle Im ersten Schritt der Evaluation werden die grundlegenden Konzepte und Modelle des Ansatzes beschrieben. Beim Monitoring wären dies beispielsweise die Eigenschaften der Grid Monitoring Architecture (GMA). Diese Beschreibung ist notwendig, um ein gemeinsames Verständnis des Ansatzes zu erhalten und um eine Grundlage für die Evaluation in Schritt 2 zu schaffen.

Schritt 2 – Evaluation der beschriebenen, grundlegenden Konzepte Im zweiten Schritt wird die Ergebnistabelle für die grundlegenden Konzepte und Modelle des Ansatzes erstellt und erläutert, wie sich der Inhalt der einzelnen Zellen der Ergebnistabelle zusammensetzt. Bei jeder Begründung ist in Klammern das Resultat angegeben: (*UC1*/✓) bedeutet beispielsweise, dass aus der Begründung folgt, dass *UC1* erfüllt wird. Die Begründungen sind für eine bessere Übersichtlichkeit entsprechend den Use Cases aus Prozessschritt 3 gegliedert.

Schritt 3 – Evaluation konkreter Implementierungen der Konzepte Im dritten Schritt werden konkrete Implementierungen der grundlegenden Konzepte untersucht, wobei für jede Implementierung eine Ergebnistabelle und eine Begründung angegeben werden.

Es werden nur diejenigen Use Cases und nicht-funktionalen Anforderungen analysiert, die von den zugrunde liegenden Konzepten erfüllt werden (✓) oder deren Erfüllung von der jeweiligen Implementierung abhängt (?). Nicht erfüllte Use Cases und nicht-funktionale Anforderungen müssen nicht für jede Implementierung evaluiert werden, da bereits das zugrunde liegende Konzept diese nicht erfüllt und es daher nicht möglich ist, dass die Implementierung sie erfüllt. Die nicht untersuchten Zellen der jeweiligen Ergebnistabelle sind für eine bessere Übersichtlichkeit grau dargestellt.

Die Auswahl der untersuchten Implementierungen richtet sich nach ihrer Relevanz in der Literatur und ihrer Aktualität. So werden aktuelle, im Einsatz befindliche Implementierungen ebenso untersucht wie Implementierungen, die nicht mehr existieren, beispielsweise wegen der Beendigung des Forschungsprojektes, aber dennoch häufig in der Literatur erwähnt werden. Durch diese Auswahlstrategie ist die Analyse eines möglichst breiten Spektrums sichergestellt.

Schritt 4 – Fazit des untersuchten Ansatzes Im letzten Schritt wird ein Fazit für den untersuchten Ansatz erstellt, das die Ergebnisse aus Schritt 2 und 3 der Evaluation zusammenfasst. Die verschiedenen Ergebnisse werden am Ende der Evaluation aggregiert (Kapitel 4) und dienen als Grundlage für die Entwicklung von DAGA.

4 Evaluation bestehender Ansätze

In diesem Kapitel werden Konzepte und Implementierungen mit dem in Prozessschritt 5 (Kapitel 3.2.5) entwickelten Evaluations-Framework evaluiert, die für die Gewinnung von Verfügbarkeits-Informationen im Allgemeinen und insbesondere in Grids eingesetzt werden bzw. eingesetzt werden können.

Das Kapitel ist in drei Bereiche gegliedert: zunächst werden in Kapitel 4.1 Ansätze untersucht, die einen klaren Bezug zu Grid-Infrastrukturen haben, beispielsweise aufgrund ihrer besonderen Konzepte oder dedizierten Implementierungen. Daran anschließend werden in Kapitel 4.2 Ansätze evaluiert, die nicht explizit für den Einsatz in Grids, sondern allgemeiner für den Einsatz in verteilten Systemen oder Netzen entwickelt wurden. Sie werden behandelt, da Grids eine Sonderform verteilter Systeme sind (Kapitel 2.1) und diese Ansätze daher möglicherweise nur angepasst oder erweitert werden müssen, um in Grid-Infrastrukturen eingesetzt werden zu können. Die Ergebnisse der Evaluation werden in Kapitel 4.3 zusammengefasst und für die weitere Verwendung in dieser Arbeit vorbereitet.

4.1 Grid-spezifische Ansätze

4.1.1 GMA-basierte Ansätze

Schritt 1 – Beschreibung der grundlegenden Konzepte, Methoden und Modelle

Monitoring wird als „the act of collecting information concerning the characteristics and status of resources of interest“ [ZS05] definiert bzw. als „the process of measurement and publication of the state of a grid resource at a particular point in time“ [IRD05; HT04; BFR07]. Die Unterscheidung zwischen traditionellen Monitoring-Ansätzen und Grid-Monitoring war lange Zeit Forschungsthema [ZS05; Car72], wobei sich als wesentliche Unterscheidungsmerkmale die Skalierung über Wide-Area Networks, eine über mehrere reale Organisationen verteilte, enorme Datenmenge und die Betrachtung vieler heterogener Ressourcen herauskristallisiert haben [Tie+02; ZS05]. Bei der Evaluation wird nur das Grid-Monitoring betrachtet. Da einige Aussagen aber für das allgemeine Monitoring sowie das spezielle Grid-Monitoring gelten, werden für eine bessere Verständlichkeit die beiden Begriffe im Folgenden synonym verwendet.

Beim Monitoring überwacht ein Sensor bzw. ein Prozess eine Entität im Fabric-Layer des Grid-Layer-Stacks (Abbildung 2.3) [FKT01] und generiert dabei *Events* [ZS05], d.h. eine Sammlung von typisierten Daten zu einer spezifischen Entität mit einem Zeitstempel [ZS05]. Typische überwachte Entitäten sind Prozessoren, Arbeitsspeicher, Hintergrundspeicher und Prozesse [ZS05]. Eine Entität ist als „any networked resource, which can be considered useful, unique, having a considerable lifetime and general use“ [PDL02] definiert.

Monitoring in verteilten Systemen, und damit auch in Grid-Umgebungen, erfolgt typischerweise in vier Schritten [ZS05; PDL02; MSS93]: 1) Generierung von Events, wobei Sensoren Ressourcen anfragen und die Messergebnisse aufzeichnen, 2) Prozessierung der aufgezeichneten Daten, z.B. Filterung oder Gruppierung der Daten entsprechend zuvor festgelegter

Kriterien, 3) Distribution der Events an alle interessierten Consumer, 4) Präsentation der Events, normalerweise mit weiterer Aufbereitung für eine bessere Verständlichkeit. Im Grid-Umfeld wird der letzte Schritt meistens mit Consumption umschrieben, da die Daten nicht zwangsläufig nur von Anwendern, sondern ebenso von Software verwendet werden können.

Für das Monitoring in Grids hat sich die *Grid Monitoring Architecture* (GMA) etabliert, die ein Interaktionsmodell mit den Rollen Producer, Consumer und Registry definiert und das Daten-Discovery von der Daten-Übertragung trennt [Bau+09b]. Datenschemata oder konkrete Technologien werden entsprechend der Natur eines Interaktionsmodells dabei explizit nicht vorgegeben. Abbildung 4.1 [Gma; Tie+02; ZS05] zeigt die genannten Rollen innerhalb der GMA sowie den im Interaktionsmodell festgelegten Aktions- und Informationsfluss, dessen Reihenfolge durch die Zahlen eins bis fünf angegeben ist.

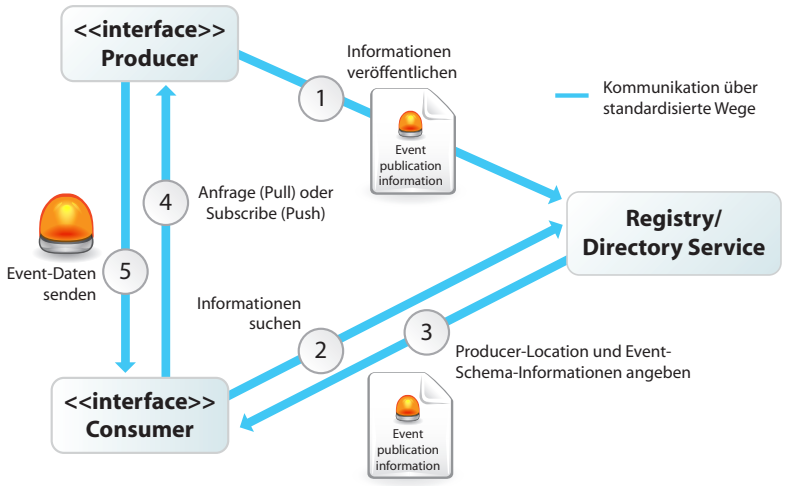


Abbildung 4.1: Architektur, Zusammenhänge und Informationsfluss der Grid Monitoring Architecture (GMA)

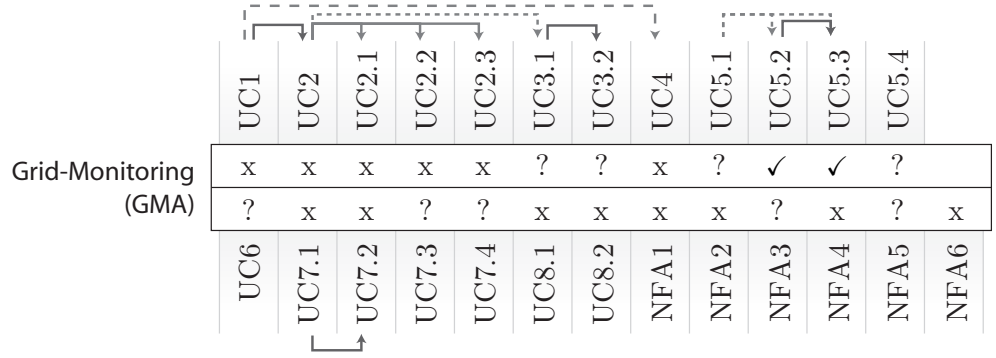
Die GMA beschreibt eine Service-orientierte Architektur (Kapitel 6.2.2) und ist daher leicht erweiterbar, es können also neue zu überwachende Ressourcen, in der GMA die Rolle der Producer, einfach hinzugefügt werden [BS02]. Wie in Abbildung 4.1 zu sehen ist, wird entsprechend den SOA-Eigenschaften innerhalb einer GMA-Infrastruktur ein Directory Service bzw. Information Service (Kapitel 2.2.1) verwendet, über den die Producer ihre Daten(-dienste) publizieren können. Diese Verwendung bedeutet jedoch nur, dass Producer ihr Dienst-Interface veröffentlichen, nicht aber, dass die Dienst-Interfaces der überwachten Ressourcen angesprochen werden.

Sowohl der Producer als auch der Consumer haben ein festgelegtes Interface, das von der Implementierung unabhängig ist. So kann ein Dienst gleichzeitig als Consumer und als Producer auftreten, wenn beide Interfaces realisiert werden. Somit kann durch das Interface-Konzept eine hierarchische Struktur aufgebaut werden, bei der beispielsweise ein Producer die Daten mehrerer anderer Producer aggregiert, gegenüber denen er als Consumer auftritt. Zudem unterstützt die GMA ein Pull- und ein Push-Modell [BS02], über das sowohl eine simple Notification als auch ein konstanter Datenstrom realisiert werden können [ZS05].

Die Verwendung von Grid-spezifischen Sicherheitsmechanismen ist nicht explizit festgelegt [BS02].

Schritt 2 – Evaluation der beschriebenen, grundlegenden Konzepte

Die Evaluation der grundlegenden Konzepte des Monitorings ergibt folgendes Ergebnis:



Evaluation Subsystem „A – Gewinnung von Verfügbarkeits–Informationen“ Wie eingangs erläutert, findet Monitoring im Fabric-Layer des Grid-Layer–Stacks statt. Da das Dienst-Interface einer Grid-Ressource aber im höheren Collective- bzw. Ressource-Layer zu finden ist (Kapitel 2.1), ist es mit oben beschriebenem Monitoring grundsätzlich nicht möglich, das Dienst-Interface zu verwenden (*UC1*/x).

Die Unterscheidung zwischen aktiven und passiven Sensoren bei der Überwachung der verschiedenen Entitäten [ZS05] bezieht sich auf die Aktionsrichtung der Datengewinnung, beispielsweise, ob die Daten zu einem überwachten Arbeitsspeicher vom Sensor gemessen werden (aktiv) oder ob er nur die vom Betriebssystem gelieferten Werte aufzeichnet (passiv) [ZS05]. Eine aktive, schrittweise Annäherung an eine mögliche Problemursache ist damit nicht definiert und hängt somit von der jeweiligen Implementierung ab (*UC3.1*/?). Auch die Möglichkeit zur Erstellung eines Problem-Traces ist nicht durch die GMA definiert (*UC3.2*/?).

Da sich Monitoring auf die Überwachung einzelner Entitäten und technischer Werte konzentriert, wie beispielsweise die Nutzung von CPUs oder den freien Arbeitsspeicher, ist die Überwachung Grid-spezifischer Sicherheitsmaßnahmen nicht möglich (*UC4*/x), da diese nicht auf einzelne Entitäten, sondern oberhalb des Connectivity-Layers auf Grid-Ressourcen angewendet werden (Kapitel 2.1). Die Sicherheitsmaßnahmen innerhalb und unterhalb des Connectivity-Layers umfassen Mechanismen aus Netzen wie beispielsweise Firewalls und haben mit Grid-spezifischen Sicherheitsmechanismen wenig gemein.

Evaluation Subsystem „B – Verwendung von Verfügbarkeits-Informationen“ Anders als bei der Gewinnung werden bei der Verwendung der Verfügbarkeits-Informationen wesentlich mehr Anforderungen erfüllt, da die GMA sowohl ein Pull- (*UC5.2*/✓) als auch ein Push-Modell (*UC5.3*/✓) anbietet, mit dem wiederum Subscriptions und Notifications umgesetzt werden können. Eine Anfragesprache (*UC5.1*/?) sowie eine Standard-UI (*UC5.4*/?) werden in den Konzepten nicht explizit gefordert und hängen damit von der jeweiligen Implementierung ab.

Evaluation Subsystem „C – System-Administration“ Die Use Cases zur Administration des Systems werden von den Konzepten des Monitorings bzw. der GMA nicht erfüllt oder hängen von der jeweiligen Implementierung ab, wie beispielsweise die Möglichkeit, die gespeicherten Daten zu administrieren (*UC6*/?).

Die Verwendung eines im jeweiligen Grid eingesetzten Information Service für die Erzeugung der Liste der überwachten Ressourcen ist nicht möglich, da ein Information Service die Dienst-Beschreibungen der einzelnen Ressourcen enthält (Kapitel 2.2.1) und die damit implizierte Verwendung des Dienst-Interface einer Ressource mit der Fokussierung des Monitorings auf den Fabric-Layer kollidiert (*UC7.1*/x). Stattdessen werden proprietäre, von der jeweiligen Implementierung abhängige Mechanismen eingesetzt, um die Liste der zu überwachenden Ressourcen zu ermitteln.
Organisatorische Anpassungen, wie das Hinzufügen neuer Ressourcentypen oder Fehlertypen, werden nicht explizit durch die GMA festgelegt und hängen somit von der jeweiligen Implementierung ab (*UC7.3*/?,*UC7.4*/?).
Selbsttests sind mit den angegebenen Konzepten nicht zu realisieren (*UC8.1*/x, *UC8.2*/x), da für (Selbst-)Tests Vergleichsdaten notwendig sind. Diese Vergleichsdaten müssten beim Monitoring von redundanten Producern kommen, was negative Auswirkungen auf eine möglichst geringe Intrusiveness bedeuten würde (*NFA2*/x).

Evaluation nicht-funktionaler Anforderungen Da die GMA nur ein Interaktionsmodell vorgibt, nicht aber konkrete Implementierungsregeln oder Kommunikationsprotokolle, hängen Vorgaben zu den verwendeten Datenschemata, die Einhaltung Grid-spezifischer Sicherheitsstandards (*NFA3*/?) und die Verwendung einer Zeitsynchronisation (*NFA5*/?) von den jeweiligen Implementierungen ab. Gerade bei der Verwendung von Datenschemata kann es aber nachteilig sein, die Entscheidungsfreiheit den jeweiligen Implementierungen zu übertragen: da es durch die Struktur und das Interface-Konzept der GMA möglich ist, mit unterschiedlichsten Systemen zu arbeiten, andererseits aber keinerlei Vorgaben zu den eingesetzten Datenschemata existieren, kommt es sehr häufig zu Ad-Hoc–Lösungen mit einer geringen Interoperabilität, und es werden keine einheitlichen Datenformate verwendet. Bestehende, proprietäre Netz- und Host-Monitoring-Lösungen sind nicht offen genug, um interoperabel und anpassbar verwendet zu werden [ZS05]. Je nach Sichtweise kann *NFA1* daher als nicht erfüllt oder von der jeweiligen Implementierung abhängend betrachtet werden. Im Rahmen dieser Arbeit wurde der erste Fall gewählt (*NFA1*/x), da ein einheitliches Datenschema von großer Wichtigkeit ist und daher verbindlich vorgegeben sein sollte.
Da beim Monitoring systemimmanent eine Abwägung zwischen aktuellen Informationen und starker Last-Erzeugung gefunden werden muss [Ios+07], ist es nicht möglich, eine hohe Aktualität der Daten zu erzielen (*NFA4*/x) und gleichzeitig eine geringe Intrusiveness zu garantieren (*NFA2*/x). Die geringe Intrusiveness wird zudem durch das beständige Sammeln von Daten sowie die Notwendigkeit, Sensoren zu installieren, negativ beeinflusst (*NFA2*/x).
Das Interface-Konzept der GMA ermöglicht es, Hierarchien aus Consumern und Producern zu entwickeln und dabei Daten zu aggregieren. Da die Bildung von Hierarchien jedoch nur ermöglicht, nicht aber fest vorgeschrieben wird, hängt die Verwendung von der jeweiligen Implementierung ab (erster Teilaspekt von *NFA6*/?). Durch den beschriebenen Monitoring-Prozess ist allerdings eine allgemeine Prozessierung der Daten vorgeschrieben, was eine Verarbeitung der Daten bedeutet, die nicht vom Consumer initiiert wurde (zweiter Teilaspekt von *NFA6*/x und somit gesamthaft *NFA6*/x).

Schritt 3 – Evaluation konkreter Implementierungen der Konzepte

Da sehr viele Implementierungen für das Monitoring in Grid-Infrastrukturen existieren, wird die von [ZS05] vorgestellte Taxonomie (Abbildung 4.2, Seite 51) von Grid-Monitoring–Sys-

temen verwendet, um möglichst viele Implementierungen in der Evaluation berücksichtigen zu können, ohne dafür jede im Detail analysieren zu müssen.

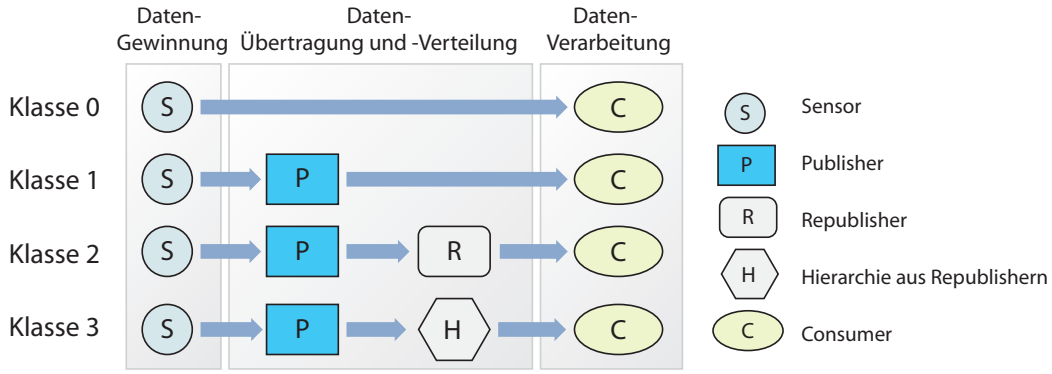
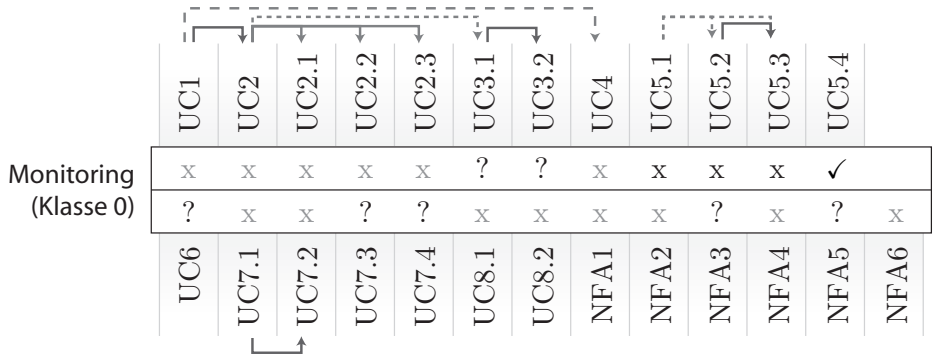


Abbildung 4.2: Die Taxonomie-Klassen aus [ZS05] über Grid-Monitoring–Systemen

Die Taxonomie betrifft hauptsächlich den Datenfluss und weniger einzelne spezifische Funktionalitäten, weswegen die schrittweise Ermittlung von Ursachen von Nicht-Verfügbarkeiten (*UC3.1*/?, *UC3.2*/?), die Administration der gespeicherten Daten (*UC6*/?), das Hinzufügen neuer Fehler- und Ressourcentypen (*UC7.3*/?, *UC7.4*/?), die Einhaltung von Sicherheitsstandards (*NFA3*/?) und die Verwendung einer Zeit-Synchronisation (*NFA5*/?) wiederum von der jeweiligen Implementierung abhängig sind. Obwohl damit knapp die Hälfte der zu überprüfenden Einträge von einer Implementierung abhängen, ist die Verwendung der Taxonomie dennoch hilfreich, da gezeigt werden soll, dass eine Gruppe von Implementierungen verschiedene Use Cases und nicht-funktionale Anforderungen erfüllen bzw. nicht erfüllen kann. Use Cases, die von einer Implementierung abhängen, wie beispielsweise die Administration der gespeicherten Daten (*UC6*), sind dabei sekundär, da bereits wichtige Punkte wie die Verwendung des Dienst-Interfaces (*UC1*), eines Information Services (*UC7.1*) oder eines einheitlichen Datenschemas (*NFA1*) nicht erfüllt sind. Bei der Begründung der Evaluationsergebnisse werden Besonderheiten der einzelnen Implementierungen diskutiert.

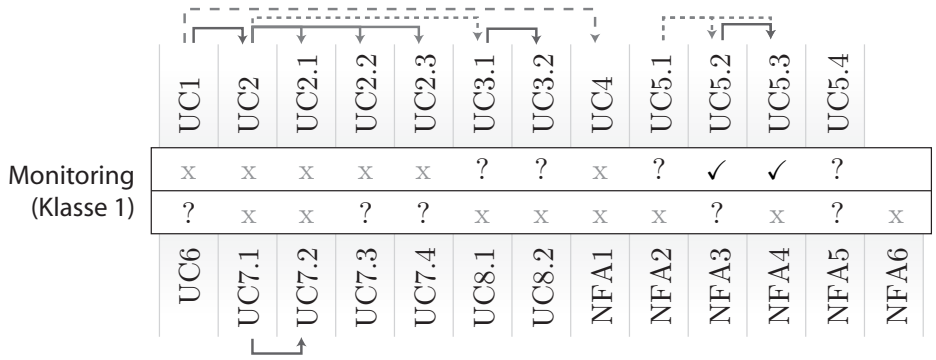
Eine umfangreiche Betrachtung von Grid-Monitoring-Systemen geben [ZS05], [Ger+04], [BS02] und [IRD05]. Detailinformationen sind in den jeweils angegebenen Quellen zu finden.

Klasse 0 – Abgeschlossene Systeme Die Monitoring-Daten bei Implementierungen dieser Klasse werden unmittelbar vom Sensor zum Consumer übertragen (Abbildung 4.2) und können nicht über eine (generische) Producer-Schnittstelle abgerufen werden [ZS05], sondern lediglich über einen proprietären Kanal, meist in Form von HTML-GUIs [ZS05]. Es ist also weder möglich, eine Anfrage zur Datenauswahl mittels einer Anfragesprache zu formulieren (*UC5.1*/x), noch Daten über ein Pull- (*UC5.2*/x) oder Push-Modell (*UC5.3*/x) zu empfangen. Da der proprietäre Kanal aber in den meisten Fällen eine Benutzeroberfläche realisiert [ZS05], können Anwender einzelne Daten abfragen (*UC5.4*/✓).

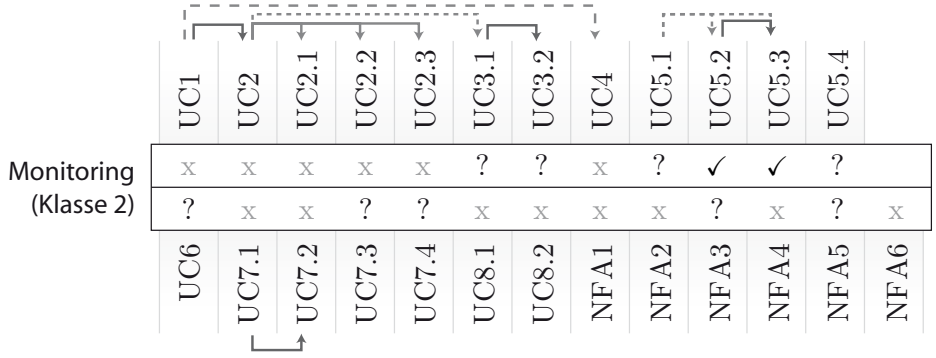


Die Implementierungen dieser Klasse sind *MapCenter* [ZS05; BHP02] und *GridICE* [ZS05; BFR07; And+03]). MapCenter verwendet keine Sicherheitsmechanismen [BHP02].

Klasse 1 – Producer-API Die Implementierungen dieser Klasse verwenden Producer, über die die Monitoring-Daten abgefragt werden können [ZS05]. Diese Producer bieten gemäß der GMA sowohl ein Pull- (*UC5.2*/✓) als auch ein Push-Modell (*UC5.3*/✓) für das Abrufen von Daten an. Die Verwendung einer Anfragesprache sowie die Realisierung einer Standard-UI hängen von den jeweiligen Implementierungen ab (*UC5.1*/?, *UC5.4*/?). Auch wenn in Klasse 1 nur *Autopilot* [ZS05; BS02; Rib+98] enthalten ist, werden die von der jeweiligen Implementierung abhängigen Felder mit einem Fragezeichen belassen, da die Ergebnistabelle die Evaluation der Klasse und nicht die Evaluation von Autopilot darstellen soll. In die Klasse können möglicherweise noch andere Ansätze eingeordnet werden, die neueren Datums sind als Autopilot, das in den 1990ern entwickelt wurde.

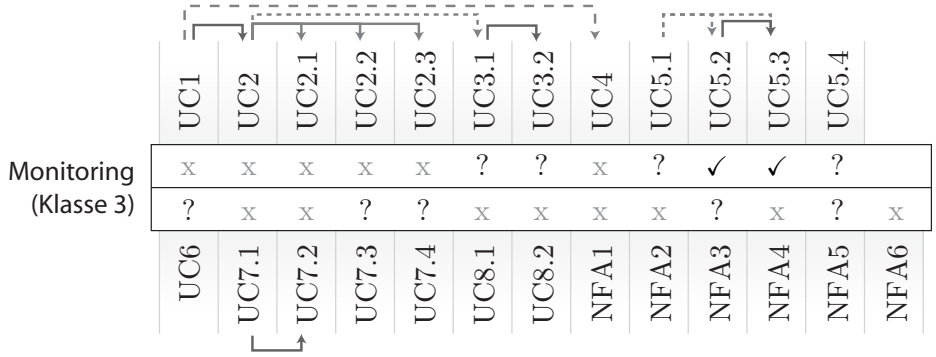


Klasse 2 – Republisher Neben Producern bieten Implementierungen dieser Klasse zusätzlich einen Republisher an [ZS05], und die Distribution der gewonnenen Daten ist auf mehrere Hosts verteilt [ZS05], wobei zwischen zentralisierten, verteilten und replizierenden Systemen unterschieden wird [ZS05]. Auch wenn die Implementierungen dieser Klasse komplexer sind als die der Klasse 1, ergibt sich doch das gleiche Evaluationsergebnis, da die zusätzliche Komplexität nur die Möglichkeiten der Datenverteilung betrifft und diese bereits in Klasse 1 erfüllt wird.



Zu dieser Klasse gehören *GridRM* [ZS05; BS02; BS03], *Hawkeye (Condor)* [ZS05; Liv11], *JAMM* [ZS05; Tie+00], *NetLogger* [ZS05; TG03], *Network Weather Service* [ZS05; WSH99] und *Remos* [ZS05; Dew+97; Din+01].

Klasse 3 – Republisher-Hierarchien Die in dieser Klasse enthaltenen Implementierungen erweitern die Implementierungen aus Klasse 2 konzeptionell um zusätzliche Komplexität: die Republisher können in beliebige Hierarchien strukturiert werden [ZS05] und werden größtenteils generisch definiert [ZS05]. Auch wenn die Implementierungen dieser Klasse komplexer sind als die der Klasse 2, ergibt sich doch das gleiche Evaluationsergebnis.



Zu dieser Klasse gehören *Ganglia* [ZS05; MCC04], *MDS (Globus Toolkit)* [ZS05; Cza+01], *MonaLISA* [ZS05; Cza+01] und *R-GMA* [ZS05; Coo+03].

Schritt 4 – Fazit „GMA-basierte Ansätze“

Zusammenfassend kann festgestellt werden, dass sowohl die grundlegenden Konzepte als auch existierende Implementierungen eine Reihe von Use Cases und nicht-funktionalen Anforderungen erfüllen, aber die im Grid-Kontext besonders wichtigen Anforderungen bereits von den grundlegenden Konzepten nicht erfüllt werden. Dazu zählen beispielsweise die Verwendung des Dienst-Interfaces der untersuchten Ressourcen (*UC1*/x), die Überprüfung von Sicherheitsmechanismen (*UC4*/x), die Verwendung eines Information Service für die Erzeugung der Liste mit zu überwachenden Ressourcen (*UC7.1*/x) oder eine geringe Intrusiveness (*NFA2*/x). Auch wenn viele Ansätze existieren, ist keiner der bestehenden Ansätze für den alleinigen Einsatz in einer Grid-Infrastruktur geeignet, ein Ergebnis, zu dem auch andere Arbeiten, wie beispielsweise [IRD05] oder [Ger+04; ZS05], kommen.

Auch eine Kombination verschiedener Systeme, wie es in Grids häufig praktiziert wird [IRD05], führt nicht zu einem zufriedenstellenden Ergebnis, da bereits die grundlegenden Konzepte die wichtigen Anforderungen nicht erfüllen und die Kombination verschiedener Systeme zudem eine umfassende Datenmigration notwendig machen würde [ZS05].

4.1.2 D-MON

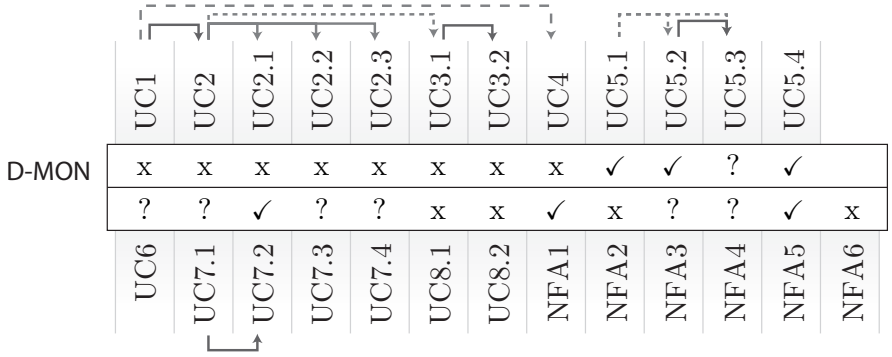
Schritt 1 – Beschreibung der grundlegenden Konzepte, Methoden und Modelle

D-MON bezeichnet bei der Evaluation stellvertretend das in [Bau+09b; Bau+09a] vorgestellte „Interoperable Grid Information System for Integrated Resource Monitoring Based on Virtual Organizations“. D-MON wurde im Rahmen des deutschen e-Science Monitoring Projektes [Gmbb] entwickelt [Bau+09b] und dessen Implementierung in der D-Grid Infrastruktur der deutschen e-Science Initiative [Ale+08] eingesetzt [Bau+09b].

D-MON beschreibt eine verteilte Architektur für ein interoperables Grid-Monitoring-System [Bau+09b], die die Monitoring-Daten von verschiedenen bestehenden Quellsystemen wie beispielsweise MDS4, BDII oder CIS aggregiert [Bau+09b], mittels ETL-Prozessen (Extract-Load-Transform) [CD97] auf das GLUE 2.0 Schema vereinheitlicht [Bau+09b] und anschließend Consumern zur Verfügung stellt. D-MON sammelt somit nicht selbst Verfügbarkeits-Informationen, sondern verwendet die von anderen Systemen bereitgestellten Daten und ist bei der Evaluation deshalb teilweise von deren Evaluationsergebnissen abhängig.

Schritt 2, 3 – Kombination beider Schritte

Da D-MON kein allgemeines Konzept, wie beispielsweise die GMA, zugrunde liegt, sondern eine eigene Infrastruktur entwickelt wurde, fallen die Schritte 2 und 3 bei der Evaluation von D-MON zusammen.



Evaluation Subsystem „A – Gewinnung von Verfügbarkeits–Informationen“ Da D-MON nur bestehende Daten aggregiert und Daten nicht selbst sammelt, müssen bei der Gewinnung von Verfügbarkeits-Informationen die zugrunde liegenden Systeme, die die Daten liefern, betrachtet werden. Da es sich bei diesen Systemen um die Monitoring-Systeme MDS4, BDII und CIS der jeweiligen Grid-Middleware Globus Toolkit, UNICORE 6 und gLite handelt [Bau+09b], die alle die unteren Schichten des Grid-Layer–Stacks betrachten [Bau+08] und damit nicht das Dienst-Interface einer zu untersuchenden Ressource verwenden, kann auch D-MON diese Anforderung nicht erfüllen (*UC1*/x). Durch die zentrale Sammlung der Daten könnte es möglich sein, Rückschlüsse auf Ursachen von Nicht-Verfügbarkeiten zu ziehen,

da aber die zugrunde liegenden Daten für diese Aussagen aus den genannten Monitoring-Systemen und damit aus den unteren Grid-Layer–Schichten stammen, wobei nicht das Dienst-Interface einer Ressource verwendet wurde, ist die Anforderung nach der Ermittlung für die Ursachen von Nicht-Verfügbarkeiten nicht erfüllt (*UC3.1*/x). Die gleiche Begründung gilt für die Nicht-Erfüllung der Bildung von Ursachen-Traces (*UC3.2*/x) und der Überprüfung von Sicherheitsmechanismen (*UC4*/x).

Evaluation Subsystem „B – Verwendung von Verfügbarkeits-Informationen“ Als Speicherlösung wird eine Datenbank mit SQL-Schnittstelle verwendet, wodurch es möglich ist, Anfragen in einer Anfragesprache zu formulieren (*UC5.1*/✓) und Pull-Anfragen zu stellen (*UC5.2*/✓). Ob die Registrierung von Observern möglich ist, also das Stellen von Push-Anfragen, geht aus den Dokumenten nicht hervor (*UC5.3*/?). Ein Standard–User-Interface bietet D-MON in Form eines Portlets an (*UC5.4*/✓), das in einem GridSphere [Pro] Container deployt werden kann.

Evaluation Subsystem „C – System-Administration“ Es wird keine Aussage darüber getroffen, ob es eine Administrations-Oberfläche für die aggregierten Daten gibt. Die für die Administration relevanten Daten werden an verschiedenen Stellen administriert, z.B. in der VO-Verwaltung. Ob aber eine zentrale Stelle in D-MON für die Administration vorhanden ist, ist nicht ersichtlich (*UC6*/?). Auch wie neue zu überwachende Ressourcen hinzugefügt werden kann nicht abschließend beantwortet werden, da die Daten von darunterliegenden Systemen wie MDS4 kommen. Woher diese Systeme ihre zu überwachenden Ressourcen beziehen, ist von dem jeweiligen System bzw. dem jeweiligen Administrator abhängig und daher für D-MON nicht bewertbar (*UC7.1*/?). Neue VOs können hinzugefügt werden, da bei der Entwicklung von D-MON von Beginn an auf eine „VO-awareness“ geachtet wurde und für die Verwaltung der VOs auf bestehende Systeme im Grid zurückgegriffen wird (*UC7.2*/✓), beispielsweise den *Grid Resource Registration Service* [Bau+09b]. Das Hinzufügen neuer Fehler- und Ressourcentypen ist wegen der Verwendung des GLUE 2.0 Datenschemas theoretisch möglich (*UC7.3*/?, *UC7.4*/?), ob die Flexibilität des GLUE 2.0 Datenschemas aber tatsächlich verwendet wird, hängt von den Quellsystemen ab. Da keine Referenzdaten vorhanden sind, sind weder technische (*UC8.1*/x) noch inhaltliche (*UC8.2*/x) Selbsttests möglich.

Evaluation nicht-funktionaler Anforderungen Das verwendete Datenformat entspricht dem GLUE 2.0 Datenschema, womit ein einheitliches und etabliertes Datenschema verwendet wird (*NFA1*/✓). Eine geringe Intrusiveness ist nicht gegeben (*NFA2*/x), da die zugrunde liegenden Quellsysteme übliche Monitoring-Systeme sind, die durch das beständige Sammeln von Daten einen mittleren bis hohen Intrusiveness-Wert besitzen. D-MON aggregiert die Daten zwar nur und hat damit selbst kaum einen Intrusiveness-Wert, als Gesamtsystem eingesetzt ist der Intrusiveness-Wert aber anders zu bewerten und daher wesentlich höher. Es werden zwar bei der Anzeige der Daten Sicherheitsmechanismen wie *Policy Decision Points* (PDP) bei der Abfrage der Daten eingesetzt, ob die Daten aber auch sicher übertragen werden, ist aus der Dokumentation nicht ersichtlich (*NFA3*/?). Die Aktualität der Daten hängt erneut von den zugrunde liegenden Quellsystemen ab (*NFA4*/?), da nur eine Aussage darüber getroffen wird, dass die aggregierten Daten alle fünf Minuten verfügbar gemacht werden [Bau+09b], nicht aber darüber, wie alt die Daten aus den verschiedenen Quellsystemen sind. Sammelt ein Quellsystem beispielsweise nur alle fünf Stunden neue Daten, sind die Daten veraltet,

obwohl sie in D-MON alle fünf Minuten verfügbar gemacht werden. Die gesammelten Daten werden in D-MON in einem ETL-Prozess vereinheitlicht, wobei auch eine Zeitsynchronisation durchgeführt wird (*NFA5*/✓). Innerhalb des ETL-Prozesses findet eine Daten-Migration, Daten-Transformation [Bau+09b] und ein Daten-Preprocessing statt (*NFA6*/x).

Schritt 4 – Fazit „D-MON“

D-MON ermöglicht einen zentralen Zugriff auf die existierenden Monitoring-Daten in einem Grid. Da für den zentralen Zugriff aber lediglich die Daten bestehender Quellsysteme aggregiert werden, die die Daten nur aus den unteren Schichten des Grid-Layer–Stacks beziehen, und D-MON selbst keine weitere Daten-Gewinnung durchführt, ist es nicht möglich, dass das Dienst-Interface einer zu überwachenden Ressource verwendet wird. D-MON kann daher nicht als geeignete Lösung betrachtet werden.

4.1.3 INCA

Schritt 1 – Beschreibung der grundlegenden Konzepte, Methoden und Modelle

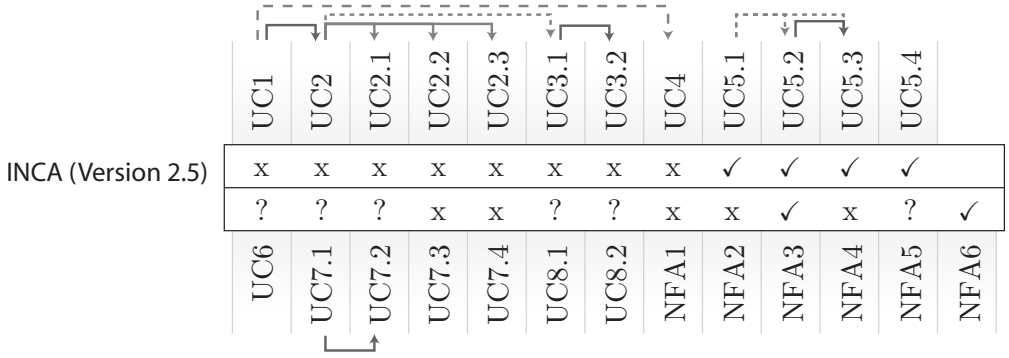
INCA [Sma+07; SEH] baut auf einer ganz eigenen Infrastruktur auf und verwendet keine allgemeine Architektur, wie beispielsweise die GMA. Kernelement bei INCA sind die sogenannten *Reporter*, meist kleine, ausführbare Programme, die auf der jeweiligen zu überprüfenden Ressource von dem dort installierten INCA-*Reporter Manager* ausgeführt werden. Die Konfiguration, die Häufigkeit der Ausführung sowie die entsprechenden Ressourcen werden in einer *Suite* festgelegt, die ein Anwender über eine proprietäre GUI erstellen kann. Die gewonnenen Daten werden in INCA in einem Depot, einem „server that is responsible for storing the data produced by reporters“, gespeichert.

Obwohl aus der allgemeinen Beschreibung, in der „User-Level Tests“ angegeben werden, vermutet werden könnte, dass das Dienst-Interface der zu überprüfenden Ressource verwendet wird, ist dies nicht der Fall. Es ist vielmehr so, dass mit „User-Level Account“ ein UNIX-Benutzer ohne `root`-Rechte gemeint ist und dass die Reporter, also die ausgeführten Tests, damit so Low-Level sind, dass sie auch manuell über die Kommandozeile ausgeführt werden können. Damit diese Low-Level–Reporter auf der zu überprüfenden Ressource ausgeführt werden können, muss in den Konfigurationsvorgaben der jeweiligen Suite daher die verwendete Programmiersprache des Reporters angegeben werden.

Die Daten können über eine eigene Java- oder Perl-API sowie über ein Web Service-Interface verwendet werden. Bei Verwendung des Web Service-Interface können Daten mittels HQL-Statements ausgelesen werden.

Schritt 2, 3 – Kombination beider Schritte

Die Prozessschritte 2 und 3 fallen bei der Evaluation von INCA zusammen, da INCA kein allgemeines Konzept, wie beispielsweise die GMA, verwendet, das gesondert evaluiert werden müsste, sondern eine eigene Infrastruktur entwickelt wurde, die mit der Implementierung deckungsgleich ist.



Evaluation Subsystem „A – Gewinnung von Verfügbarkeits–Informationen“ Wie in der allgemeinen Beschreibung von INCA erläutert, wird nicht das Dienst-Interface der zu überprüfenden Ressource verwendet, sondern es werden kleine, ausführbare Programme verwendet, die mit einem UNIX-Account auf der Ressource ausgeführt werden (*UC1*/x).
Da die Suites, die die verschiedenen Test-Definitionen enthalten, manuell und einzeln erstellt werden müssen, ist es nicht möglich, mittels eines autonomen, schrittweisen Prozesses die Ursachen einer Nicht-Verfügbarkeit zu ermitteln (*UC3.1*/x) und anschließend auszugeben (*UC3.2*/x).
Es kann zwar überprüft werden, ob ein Benutzer einen gültigen Grid-Proxy besitzt, dies bedeutet aber nicht, dass die Sicherheitsmechanismen im Grid überprüft werden können, da nicht die Dienst-Interfaces der entsprechenden Ressourcen für die Überprüfung verwendet werden (*UC4*/x).

Evaluation Subsystem „B – Verwendung von Verfügbarkeits-Informationen“ Durch die verschiedenen APIs sowie die Möglichkeit, HQL-Anfragen zu erstellen, ist es möglich, Anfragen in einer Anfragesprache zu formulieren (*UC5.1*/✓) und ein Pull- (*UC5.2*/✓) sowie ein Push-Modell (*UC5.3*/✓) zu realisieren. INCA bietet zudem ein standardisiertes Web-UI, über das Anwender die Daten abrufen können (*UC5.4*/✓).

Evaluation Subsystem „C – System-Administration“ Da sich INCA bei der Beschreibung der Datenspeicherung auf die Aussage beschränkt, einen (dedizierten) Server für die Datenspeicherung zu verwenden, kann nicht festgestellt werden, wie die Daten administriert werden können (*UC6*/?).
Welche Ressourcen bei INCA überwacht werden, wird in den bereits erwähnten manuell erstellten Suites festgelegt. Wie die Liste der zu überprüfenden Ressourcen dabei generiert wird, ist nicht definiert (*UC7.1*/?, *UC7.2*/?). Neue Fehler- und Ressourcentypen sind in der Architektur von INCA implizit enthalten, beispielsweise durch die Installation der Reporter Manager, und können damit nicht erweitert werden (*UC7.3*/x, *UC7.4*/x).
Aus der INCA-Dokumentation geht nicht hervor, ob technische (*UC8.1*/?) oder inhaltliche Selbsttests (*UC8.2*/?) durchgeführt werden können.

Evaluation nicht-funktionaler Anforderungen INCA speichert die gewonnenen Informationen in einem proprietären Datenschema und verwendet somit kein in Grids etabliertes

Datenschema (*NFA1*/x). Da es notwendig ist, auf jeder überwachten Ressource einen Reporter Manager zu installieren und die Tests periodisch ohne dynamische Anpassung ausgeführt werden, hat INCA eine hohe Intrusiveness (*NFA2*/x). Aus der statischen Ausführung folgt zudem eine suboptimale Aktualität der Daten (*NFA4*/x).
Da INCA für die Kommunikation der verschiedenen Komponenten SSL verwendet und für die Ausführung eines Tests ein gültiges GSI-Credential notwendig ist, werden die in Grids bestehenden Sicherheitsstandards eingehalten (*NFA3*/✓).
Ob ein Zeitsynchronisationsmechanismus verwendet wird, geht aus der Dokumentation nicht hervor (*NFA5*/?). Dafür ist erkennbar, dass keine Migration oder Vorverarbeitung der Daten notwendig ist und diese unmittelbar vom Consumer verwendet werden können (*NFA6*/✓).

Schritt 4 – Fazit „INCA (Version 2.5)“

Obwohl INCA im Vergleich zu GMA-basierten Ansätzen einige Vorteile hat, werden wichtige Punkte wie die Verwendung des Dienst-Interfaces (*UC1*/x), eine schrittweise autonome Ursachenforschung (*UC3.1*/x) oder eine geringe Intrusiveness (*NFA2*/x) nicht erfüllt.

4.2 Ansätze aus Netzen und verteilten Systemen

4.2.1 Probing-basierte Ansätze

Schritt 1 – Beschreibung der grundlegenden Konzepte, Methoden und Modelle

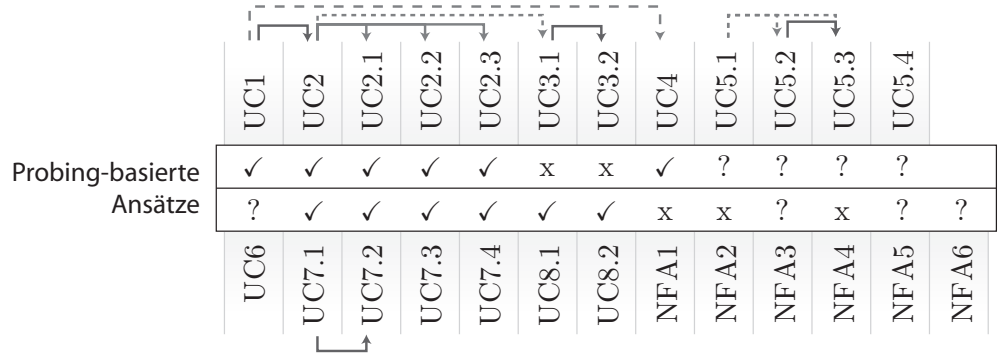
Probing ist ein Konzept, um in Netzen Problem-Ursachen autonom und effektiv zu ermitteln. Dafür werden pro-aktiv Testpakete an die zu überprüfenden Ressourcen geschickt, die jeweiligen Antworten ausgewertet und so deren Zustand analysiert. Probing wurde nicht spezifisch für den Einsatz in Grids, sondern allgemein für den Einsatz in verteilten Systemen entwickelt [Bro+03] und wird dort bereits eingesetzt, beispielsweise für die Messung der Verfügbarkeit von Peer-to-Peer–Netzen [CLL02; BSV03] oder von Workstations im Internet [LMG95; BSV03].

Kernelement des Probings sind die sogenannten *Probes*, die auf einer spezifischen Maschine, der *Probe-Station*, ausgeführt werden. Bei der Ausführung einer Probe, die auch als Test-Programm oder End-to-End–Transaktion interpretiert werden kann, wird ein dem Typ der Probe entsprechender Befehl oder eine entsprechende Transaktion an die zu überprüfende Ressource geschickt und das Ergebnis ausgewertet [BRM01; Bro+03]. Die Definition einer Probe ist sehr allgemein formuliert und umfasst daher neben speziell entwickelten Testprogrammen auch die Ausführung eines Pings [Bro+02] und den Versand einer Test-Email oder die Anfrage an einen Web Service [Bro+02], wie es in IBM's EPP-Technologie [FL99] ausgeführt wird.

Weitere Vorgaben, wie die spätere Verwendung der ermittelten Daten, Datenschemata oder Interaktionen werden durch das Probing nicht vorgegeben.

Schritt 2 – Evaluation der beschriebenen, grundlegenden Konzepte

Bei der Evaluation der grundlegenden Konzepte ergibt sich folgendes Ergebnis mit den dazugehörigen Begründungen. Auffällig dabei ist das wesentlich bessere Abschneiden des Probing-Ansatzes im Vergleich zu GMA-basierten Ansätzen (Kapitel 4.1.1) oder dem D-MON–Ansatz (Kapitel 4.1.2).



Evaluation Subsystem „A – Gewinnung von Verfügbarkeits–Informationen“ Wie in der Beschreibung der grundlegenden Konzepte erläutert, kann aufgrund der allgemeinen Definition einer Probe auch eine Anfrage an einen Web Service gestellt werden, es ist also möglich, das Dienst-Interface einer Ressource zu verwenden, um ihre Funktionstüchtigkeit zu überprüfen (*UC1*/✓). Die allgemeine Formulierung erlaubt es zudem, vielfältige Probe-Typen zu verwenden, wodurch technische und organisatorische Nicht-Verfügbarkeiten identifiziert (*UC2.1*/✓, *UC2.2*/✓), der Grad einer Nicht-Verfügbarkeit analysiert (*UC2.3*/✓) und die Sicherheitsmechanismen in einem Grid (*UC4*/✓) überprüft werden können. Da beim Probing-Prozess zwar eine Analyse der Ergebnisse durchgeführt wird, die jeweils zu versendenden Probes aber bereits vor dem Probing festgelegt werden müssen (siehe dazu auch die Beschreibung des Active Probings in Kapitel 4.2.2), ist es mittels Probing-basierter Ansätze nicht möglich, Ursachen für eine Nicht-Verfügbarkeit autonom zu ermitteln, da eine vorherige Festlegung der Probing-Reihenfolge dafür nicht flexibel genug ist (*UC3.1*/x).

Evaluation Subsystem „B – Verwendung von Verfügbarkeits-Informationen“ Da zur Verwendung der gewonnenen Daten, abgesehen von der Auswertung der Probe-Antworten, keinerlei Angaben vom Probing-Konzept gemacht werden, hängt es von der jeweiligen Implementierung ab, ob es eine Anfragesprache (*UC5.1*/?), ein Pull- (*UC5.2*/?) und Push-Modell (*UC5.3*/?) sowie eine Standard-UI für Einzelanfragen (*UC5.4*/?) gibt.

Evaluation Subsystem „C – System-Administration“ Wie bereits in der Evaluation des Subsystems B erwähnt, macht das Probing-Konzept keine Angaben zur Verwendung und damit zur Administration der gewonnenen Daten (*UC6*/?). Da wegen der allgemeinen Formulierung einer Probe die Dienst-Interfaces der überwachten Ressourcen verwendet werden können (siehe Evaluation Subsystem A), kann die Liste der zu überwachenden Ressourcen samt deren Dienst-Interface aus einem Information Service ausgelesen werden (*UC7.1*/✓). Die allgemeine Formulierung ermöglicht ebenso eine theoretisch beliebige Erweiterung um neue Fehler- (*UC7.3*/✓) und Ressourcentypen (*UC7.4*/✓). Abschließend ermöglicht es die allgemeine Formulierung, Probes zu definieren, die als Ziel-Entität wiederum Probe-Stations haben, die über diesen Weg sowohl technisch (*UC8.1*/✓) als auch inhaltlich (*UC8.2*/✓) überprüft werden können.

Evaluation nicht-funktionaler Anforderungen Wie schon bei der Evaluation der GMA-basierten Ansätze erläutert wurde, ist die Verwendung eines einheitlichen und etablierten

Datenschemas von enormer Wichtigkeit und sollte aus den dort genannten Gründen nicht von einer konkreten Implementierung abhängen, sondern durch das Konzept vorgegeben werden (*NFA1*/x), um zu vermeiden, dass nicht interoperable Ad-Hoc–Lösungen verwendet werden (Kapitel 4.1.1).
Die Probes, die beim Probing-Ansatz ausgeführt werden, werden, wie eingangs erläutert, in einem statisch definierten Ablauf- und Zeitplan ausgeführt. Dadurch werden unter Umständen wesentlich mehr Probes ausgeführt, als notwendig wäre, da es nicht möglich ist, die Zahl und den Typ der Probes dynamisch an eine Situation anzupassen, was zu einer hohen Lasterzeugung und damit Intrusiveness führt (*NFA2*/x). Damit ist es ebenfalls nicht möglich, eine beständig hohe Aktualität der Daten zu gewährleisten, da der statisch definierte Ablaufplan möglicherweise zu lange Abstände enthält, um eine ausreichend hohe Aktualität der Daten zu erzielen (*NFA4*/x).
Da durch das Probing-Konzept außer der Art der Datengewinnung keine weiteren Aussagen getroffen werden, ist die Einhaltung von Sicherheitsstandards (*NFA3*/?), die Verwendung eines Zeitsynchronisationsmechanismus (*NFA5*/?) sowie die Vermeidung von Daten-Migrationen (*NFA6*/?) von den jeweiligen Implementierungen abhängig.

Schritt 3 – Evaluation konkreter Implementierungen der Konzepte

Das Probing-Konzept wird in Netzen und verteilten Systemen bereits umfassend eingesetzt, beispielsweise um entfernte Betriebssysteme mittels des TCP Fingerprints zu überprüfen [GJbXy06], um den Status eines Netzes zu ermitteln [KH04] oder für die Kapazitätsanalyse in Wireless Ad-Hoc–Netzen [Che+05].

Obwohl das Probing-Konzept wesentlich besser für die Gewinnung von Verfügbarkeits-Informationen in Grid-Umgebungen geeignet ist als bestehende Grid-spezifische Ansätze, gibt es im Vergleich nur sehr wenige Implementierungen, die das Probing-Konzept in Grids einsetzen, und es gibt nur einen Ansatz, der es für die Gewinnung von Verfügbarkeits-Informationen verwendet. Diese Implementierungen umfassen beispielsweise die Suche nach der schnellsten und zuverlässigsten Daten-Replikation (Kapitel 2.2.4) in Data Grids [Zho+06], die Entwicklung eines verbesserten Ressourcen-Discovery–Algorithmus [Cui+06; Wu+08], die Deadlock-Erkennung bei der Replikation in Data-Grids [ANF11] oder die Performance-Messung im Low-Level–Bereich [Chu+04], wie beispielsweise der Daten-Übertragung zwischen Grid-Entitäten mittels Funktionalitäten des Globus Toolkit. All diese Ansätze können aufgrund ihres abweichenden Fokus nicht evaluiert werden.

Der in [BS02] erwähnte *Globus Heartbeat Monitor* (GHM) [Ste+98] ist die einzige Implementierung, die für die Gewinnung von Verfügbarkeits-Informationen in Grids das Probing-Konzept einsetzt bzw. eingesetzt hat. Das GHM-Projekt wurde bereits vor längerer Zeit beendet und hat keine nennenswerten Vorteile, weshalb es ebenfalls nicht evaluiert wird.

Schritt 4 – Fazit „Probing-basierte Ansätze“

Wie in Schritt 2 gezeigt wurde, erfüllt der Probing-Ansatz bereits wesentlich mehr Use Cases und nicht-funktionale Anforderungen als die grundlegenden Konzepte und Implementierungen der Grid-spezifischen Ansätze. Dennoch gibt es keine Implementierungen, die das Probing für die Gewinnung von Verfügbarkeits-Informationen in Grid-Infrastrukturen einsetzen.

4.2.2 Active Probing-basierte Ansätze

Schritt 1 – Beschreibung der grundlegenden Konzepte, Methoden und Modelle

Active Probing[1] [BRM01; Bro+02; Ris+04; Bro+03] folgt der Idee, „die richtigen Fragen zur richtigen Zeit zu stellen“ [Ris+04], und ist eine Weiterentwicklung des zuvor beschriebenen Probings. In den Anfängen des Probings wurden Probes meist nach festgelegten Plänen zu bestimmten Zeitpunkten verschickt [BRM01]. Da eine im Voraus festgelegte Reihenfolge und Typdefinition aber naheliegende Nachteile hat, beispielsweise die Notwendigkeit, alle Eventualitäten in der Definition der Reihenfolge zu berücksichtigen [Ris+04], entwickelte sich das Probing zum *Active Probing* weiter. Hierbei werden die Probes on demand während der Ausführung des im nächsten Unterkapitel beschriebenen Active Probing–Algorithmus versandt und ausgewählt. Neben der Ermöglichung einer autonomen Ursachenforschung wird die Lasterzeugung durch die Verwendung des Active Probings stark reduziert, da durch die interaktive Auswahl der Probes deren Menge um bis zu 20% gegenüber dem pre-planned Probing verringert werden kann [Bro+03]. Zusammengefasst ist Active Probing ein Framework aus Algorithmen und Definitionen, mit denen in Netzen in Echtzeit Probleme eingegrenzt, deren Ursachen autonom ermittelt und Fehler lokalisiert werden können [BRM01].

Der Active Probing–Algorithmus

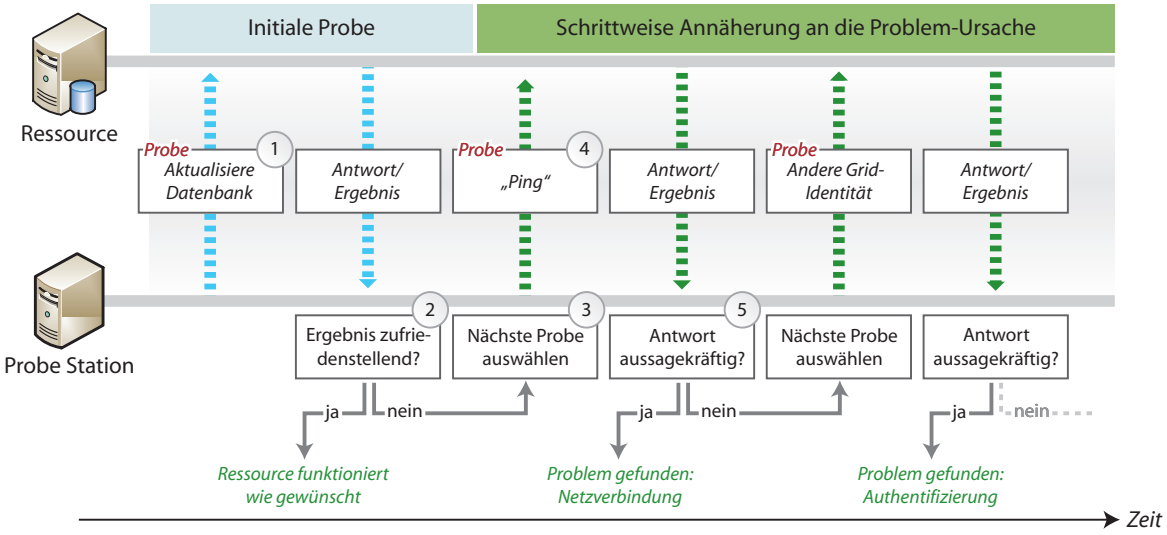


Abbildung 4.3: Beispielhafter Ablauf der Ursachen-Ermittlung einer Nicht-Verfügbarkeit mittels Active Probing (Anlehnung an [Bro+03])

Die Probes werden vom Active Probing–Algorithmus, dem *aktiven* Messen von Ressourcen [BRM01], in den in Abbildung 4.3 dargestellten Schritten verwendet (Abbildung entwickelt aus [BRM01; Bro+02; Ris+04; Bro+03]): 1) Zu Beginn wird eine initiale Probe, die „Initial Probe“, ausgeführt, um eine spezifische Eigenschaft oder Funktion einer Ressource zu überprüfen, beispielsweise die korrekte Funktionsweise einer Datenbank durch die Aktualisierung eines Datenbank-Eintrags. 2) Im zweiten Schritt wird die Antwort der Ressource ausgewertet, die vom Zustand der betrachteten Ressource abhängt [Bro+03]. War die Ausführung

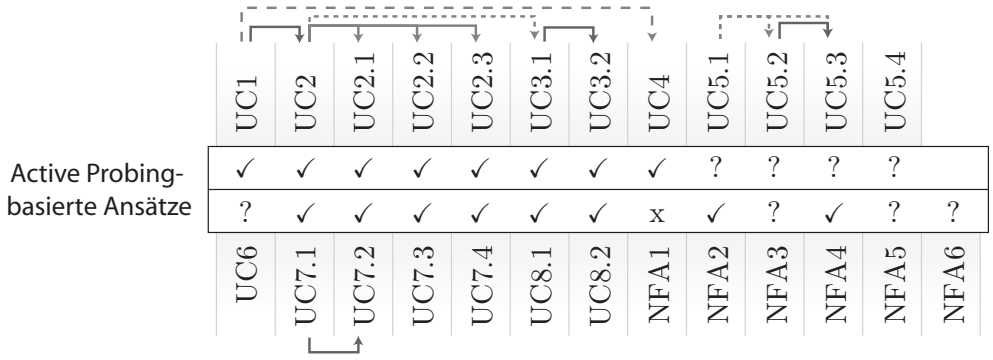
[1] Patent von Brodie (IBM) beantragt: US Patent Application 20080209269 [LLC08]

des Befehls erfolgreich – die Datenbank wurde aktualisiert – wird der Probing-Durchgang beendet und protokolliert. Ist hingegen ein Fehler aufgetreten, wird mit der autonomen und schrittweise operierenden Ursachenanalyse begonnen. 3) Dafür wird in Schritt drei eine neue Probe, eine sogenannte „Investigative Probe“, ausgewählt, um möglichst viele und aussagekräftige Informationen zu der Fehlfunktion zu erhalten, also ein Probe-Typ, der den größten Informationsgewinn verspricht [Ris+04]. Im Datenbank-Beispiel wäre das ein Ping, um zu überprüfen, ob die Ressource über das Netz erreichbar ist. 4) Die neu ausgewählte Probe wird in Schritt vier ausgeführt und 5) in Schritt fünf das erhaltene Ergebnis ausgewertet. Je nach Antwort wurde entweder die Ursache gefunden und der Probing-Prozess wird beendet, oder es wird erneut eine Probe ausgewählt, um die Ursache weiter einzugrenzen. Im Datenbank-Beispiel würde eine Antwort auf den Ping bedeuten, dass die Ressource im Netz verfügbar ist; da das Update der Datenbank aber nicht durchgeführt werden konnte, ist die Problemursache noch nicht gefunden, und es wird erneut eine Probe ausgewählt, wieder mit dem Ziel eines möglichst hohen Informationsgewinns. Im Datenbank-Beispiel könnte das die Verwendung einer anderen Grid-Identität sein, um Fehler in der Authentifizierung und Autorisierung zu analysieren. Dieser Prozess wird so lange fortgeführt, bis die Ursache gefunden ist [Ris+04]. Hätte im Datenbank-Beispiel die Aktualisierung der Datenbank mit einer anderen Grid-Identität funktioniert, würde die Ursache in der Authentifizierung bzw. Autorisierung liegen.

Zusammengefasst besteht der Active Probing–Algorithmus aus der Auswahl von Probes, deren Ausführung und der anschließenden Analyse der erhaltenen Ergebnisse, wobei die Auswahl der nächsten Probes von den Ergebnissen vorheriger Probes abhängt [Bro+03].

Schritt 2 – Evaluation der beschriebenen, grundlegenden Konzepte

Da das Active Probing eine Weiterentwicklung des bereits evaluierten Probings ist, ähneln sich deren Evaluations-Ergebnisse, wobei vom Active Probing Konzept weitere Use Cases und nicht-funktionale Anforderungen erfüllt werden, wie in der Ergebnistabelle zu sehen ist.



Evaluation Subsystem „A – Gewinnung von Verfügbarkeits–Informationen“ Die allgemeine Definition einer Probe wird vom Probing-Konzept übernommen, wodurch es auch beim Active Probing möglich ist, das Dienst-Interface einer Ressource für deren Überprüfung zu verwenden (*UC1*/✓), Probe-Typen für die Identifikation technischer (*UC2.1*/✓) und organisatorischer (*UC2.2*/✓) Nicht-Verfügbarkeiten sowie die Analyse des Grads einer Nicht-Verfügbarkeit (*UC2.3*/✓) zu definieren und Probes für die Überprüfung von Sicherheitsmechanismen einzusetzen (*UC4*/✓). Im Vergleich zum Probing werden beim Active Probing die

zu versendenden Probes nicht statisch festgelegt, sondern dynamisch und automatisch ausgewählt, wodurch es möglich ist, die Ursachen einer Nicht-Verfügbarkeit autonom in einem schrittweisen Prozess zu ermitteln (*UC3.1*/✓) und diese Ergebnisse in einem Ursachen-Trace darzustellen (*UC3.2*/✓).

Evaluation Subsystem „B – Verwendung von Verfügbarkeits-Informationen“ Auch beim Active Probing werden zur Verwendung der gewonnenen Daten keinerlei Angaben gemacht, es hängt somit wieder von der jeweiligen Implementierung ab, ob es eine Anfragesprache (*UC5.1*/?), ein Pull- (*UC5.2*/?) und Push-Modell (*UC5.3*/?) sowie eine Standard-UI für Einzelanfragen (*UC5.4*/?) gibt.

Evaluation Subsystem „C – System-Administration“ Bei der System-Administration entspricht das Active Probing dem Probing ohne zusätzliche Erweiterungen, woraus sich die gleichen Evaluationsergebnisse mit den gleichen Begründungen ergeben, die bereits bei der Evaluation des Probings angegeben wurden (Kapitel 4.2.1).

Evaluation nicht-funktionaler Anforderungen Obwohl die Verwendung eines einheitlichen und etablierten Datenschemas von enormer Wichtigkeit ist, wird sie auch vom Active Probing nicht vorgegeben (*NFA1*/x).
Da die Probes beim Active Probing entsprechend den vorausgegangenen Probe-Ergebnissen mittels dedizierter Auswahl-Algorithmen dynamisch selektiert werden, kann die Anzahl und die Art der Probes optimiert und, wie eingangs beschrieben, um bis zu 20% reduziert werden, woraus sich eine geringe Intrusiveness ergibt (*NFA2*/✓). Ebenso kann das Zeitintervall zwischen zwei Initial Probes bzw. Investigative Probes dynamisch angepasst werden, womit eine hohe Aktualität erzielt werden kann (*NFA4*/✓).
Da durch das Probing-Konzept neben der Art der Datengewinnung keine weiteren Aussagen getroffen werden, ist die Einhaltung von Sicherheitsstandards (*NFA3*/?), die Verwendung eines Zeitsynchronisationsmechanismus (*NFA5*/?) sowie die Vermeidung von Daten-Migrationen (*NFA6*/?) von den jeweiligen Implementierungen abhängig.

Schritt 3 – Evaluation konkreter Implementierungen der Konzepte

Ebenso wie das Probing wird auch das Active Probing außerhalb von Grid-Infrastrukturen bereits in Netzen und verteilten Systemen eingesetzt. Im Grid-Umfeld existieren noch weniger Implementierungen zu Active Probing als es beim Probing-Konzept der Fall ist: so gibt es beispielsweise nur eine Implementierung, die explizit Active Probing einsetzt, um bei der Überwachung von Netz-Aktivitäten zwischen Grid-Anwendungen die Intrusiveness zu verringern [ZL03]. Eine dedizierte Implementierung für die Gewinnung von Verfügbarkeits-Informationen in Grid-Infrastrukturen existiert nicht.

Schritt 4 – Fazit „Active Probing-basierte Ansätze“

Wie in der Ergebnistabelle in Schritt 2 zu sehen ist, erfüllt das Active Probing Konzept nahezu alle wichtigen Use Cases und nicht-funktionalen Anforderungen oder macht sie von der jeweiligen Implementierung abhängig. Dass die nicht-funktionale Anforderung zur Verwendung eines einheitlichen Datenschemas nicht erfüllt ist (*NFA1*), kann durch eine spätere Implementierung behoben werden und ist daher vernachlässigbar.

4.3 Zusammenfassung der Evaluationsergebnisse

Trotz der großen Anzahl an bestehenden Ansätzen hat deren Evaluation gezeigt, dass keine Implementierung existiert, die die in Kapitel 3 analysierten Anforderungen ausreichend erfüllt. Alle existierenden Implementierungen haben individuelle Vor- und Nachteile und sind für den Zweck, für den sie vorrangig entwickelt wurden, geeignet, nicht aber für den Einsatz in Grid-Umgebungen, ein Ergebnis, zu dem auch andere Arbeiten wie [BS02] kommen.

Bei den Grid-spezifischen Ansätzen, deren Evaluationsergebnisse in Abbildung 4.4 zusammengefasst sind, werden bereits von den grundlegenden Konzepten wie der GMA wichtige Anforderungen nicht erfüllt, beispielsweise die Verwendung des Dienst-Interfaces (*UC1*) bei der Ressourcen-Analyse oder die Überprüfung von Sicherheitsmechanismen (*UC4*). Außerdem lassen die einzelnen Implementierungen wichtige Funktionalitäten und Qualitätsmerkmale, wie eine geringe Intrusiveness (*NFA2*), vermissen. Abbildung 4.4 zeigt, dass die Use Cases für die Gewinnung von Verfügbarkeits-Informationen von den Grid-spezifischen Ansätzen erfüllt werden.

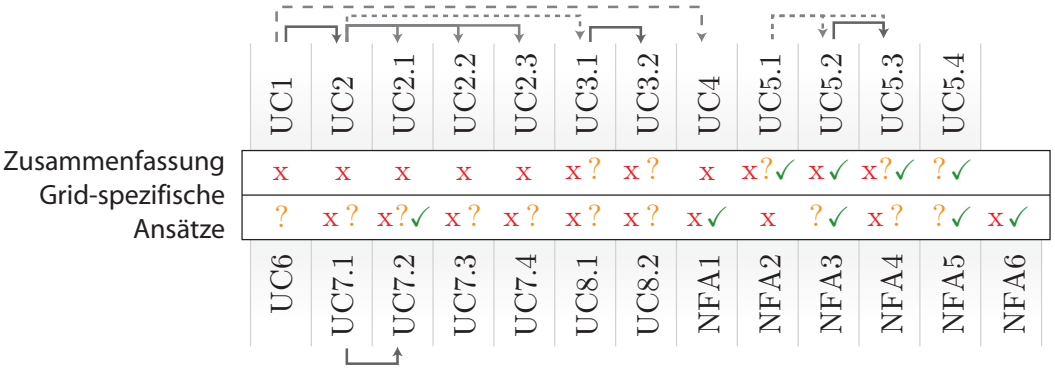


Abbildung 4.4: Evaluationsergebnisse der Grid-spezifischen Ansätze

Bei den Konzepten aus Netzen und verteilten Systemen, deren Evaluationsergebnisse in Abbildung 4.5 zusammengefasst sind, ist das Active Probing ein vielversprechendes Vorgehen, da es alle wichtigen Anforderungen erfüllt bzw. einige der konkreten Implementierung überlasst. Besonders die Erfüllung aller Anforderungen für die Gewinnung von Verfügbarkeits-Informationen wird in Abbildung 4.5 deutlich. Allerdings wird in der Literatur keine Implementierung erwähnt, die das Active Probing in einer Grid-Infrastruktur für die Gewinnung von Verfügbarkeits-Informationen einsetzt.

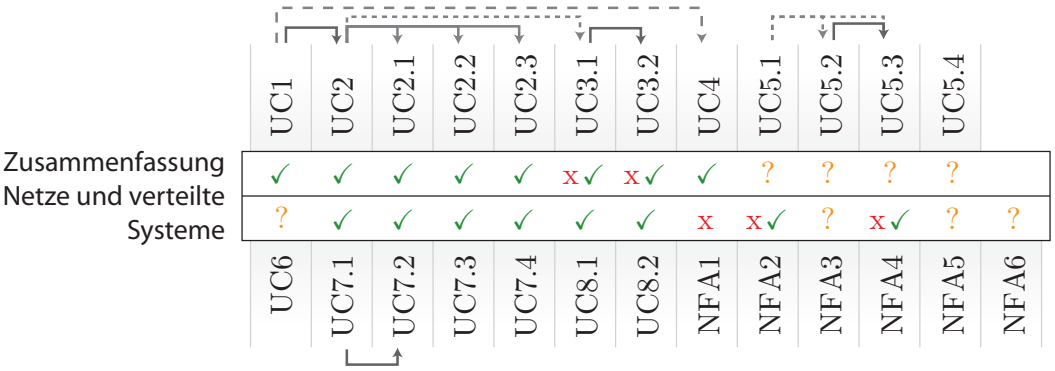


Abbildung 4.5: Evaluationsergebnisse der Ansätze aus Netzen und verteilten Systemen

5 DAGA – Determining Availability of Grid-Resources using Active Probing

Die Evaluation bestehender Ansätze in Kapitel 4 hat gezeigt, dass keine Implementierung zur Gewinnung von Verfügbarkeits-Informationen von Grid-Ressourcen existiert, die alle bestehenden Anforderungen (Kapitel 3.2.5) an ein solches System erfüllt und daher sinnvoll in Grid-Infrastrukturen eingesetzt werden könnte.

Dieser Mangel ist darin begründet, dass vorhandene Implementierungen die bestehenden Anforderungen nicht oder nur teilweise erfüllen und gleichzeitig das vielversprechende Konzept des Active Probings bisher nicht in einer Implementierung für Grid-Infrastrukturen umgesetzt wurde.

Diese Lücke soll mit dem neu entwickelten Ansatz *DAGA* (***D***etermining ***A***vailability of ***G***rid-Resources using ***A***ctive Probing) geschlossen werden. DAGA verwendet Active Probing als grundlegendes Konzept, realisiert die in der Evaluation des Active Probings (Kapitel 4.2.2) von der jeweiligen Implementierung abhängigen Punkte und folgt entsprechend den Eigenschaften eines Grids der Dienst-Orientierung. Als Ergebnis ergibt sich mit DAGA eine neue Implementierung, die alle an ein System zur Gewinnung von Verfügbarkeits-Informationen in Grid-Infrastrukturen analysierten Anforderungen erfüllt.

Nach der bereits durchgeführten Anforderungsanalyse und Evaluation bestehender Ansätze gliedert sich der zweite Teil dieser Arbeit wie folgt: zunächst werden in diesem Kapitel die Ziele aufgezeigt, die bei der Entwicklung von DAGA verfolgt werden und die Kernkonzepte vorgestellt, die DAGA von bestehenden Ansätzen unterscheiden. Aus diesen Grundlagen wird in Kapitel 6 das Entwurfsmodell entwickelt und dieses in Kapitel 7 prototypisch implementiert.

5.1 Ziele bei der Entwicklung von DAGA

Bei der Entwicklung von DAGA werden drei Ziele verfolgt, ein methodisches, ein funktionales und ein technisches Ziel. Jedes der drei Ziele hat maßgeblichen Einfluss auf die Qualität des Entwicklungsergebnisses sowie die Erfüllung der analysierten Anforderungen durch DAGA.

5.1.1 Methodisches Vorgehen im gesamten Entwicklungsprozess

Das erste Ziel bei der Entwicklung von DAGA ist es, einen methodisch sauberen Entwicklungsprozess zu verwenden. Daher werden alle Schritte des Software-Entwicklungs–Prozesses durchlaufen, beginnend bei der bereits durchgeführten Anforderungsanalyse und Anforderungsspezifikation (Kapitel 3.2.5) über die Konzeption (Kapitel 6) und Realisierung (Kapitel 7) bis hin zum Test und Deployment des Systems (Kapitel 7). Nur wenn diese Schritte durchlaufen werden, können alle relevanten Anforderungen analysiert und darauf aufbauend ein System entwickelt werden, das diese Anforderungen auch erfüllt. Auch wenn ein Grid eine

„Distributed Computing Infrastructure“ ist (Kapitel 2.1) und Anforderungen für dieses allgemeine Einsatzgebiet existieren, ergeben sich durch die speziellen Ausprägungen in einem Grid einige zusätzliche Anforderungen, die ebenfalls erfüllt werden müssen (Kapitel 3.2.5). Dass die bestehenden Ansätze und Implementierungen diese Anforderungen nicht erfüllen ist darin begründet, dass sie aus anderen Anwendungsdomänen, z.B. Clustern [Tie+98] oder allgemein verteilten Systemen (NetLogger [Tie+98]), wo sie gut funktionieren, in Grid-Umgebungen nur übertragen und angepasst wurden. Es wurde also kein methodisch sauberer Prozess angewandt, sondern es wurden lediglich Anpassungen vorgenommen, ohne dabei die besonderen Anforderungen innerhalb eines Grids zu berücksichtigen.

5.1.2 Realisierung neuer, in Grids wichtiger Funktionalitäten

Das zweite Ziel bei der Entwicklung ist es, neue, in Grids wichtige Funktionalitäten bei der Gewinnung von Verfügbarkeits-Informationen zu realisieren. Dazu gehören eine *„Global View“*, die *autonome Suche* nach Ursachen für Nicht-Verfügbarkeiten und die konsequente *Berücksichtigung von in Grids relevanten Sicherheitsmechanismen.*

Durch eine **„Global View“** wird ein umfassendes Verständnis der quantitativen und qualitativen Ressourcen-Situation über den Zeitverlauf im gesamten Grid ermöglicht [ZS05; ZS04]. Dadurch unterscheidet sich DAGA von bestehenden Ansätzen, die meist nur in einem (kleinen) festgelegten Spezialbereich angesiedelt sind und für eine Gesamtsicht des Grids verschiedene Systeme aggregieren müssen, woraus die in Kapitel 3.2.2 genannten Nachteile entstehen. Neben der Hauptaufgabe, der Betrachtung von Verfügbarkeiten der Grid-Ressourcen, können die Daten in einem späteren Schritt (Kapitel 8.2.2) durch die globale Sichtweise zusätzlich für eine langfristige Kapazitätenplanung oder das leistungsabhängige Accounting verwendet werden [ZS04].

Mit einer **autonomen Suche** nach den Ursachen einer Nicht-Verfügbarkeit kann wesentlich besser auf die Komplexität, Heterogenität und geographische Verteilung der Ressourcen eines Grids (Kapitel 2.1.1) eingegangen werden, als es mit einer manuellen Ursachenforschung möglich ist: aufgrund der besonderen Eigenschaften eines Grids und der daraus erwachsenden Komplexität von Nicht-Verfügbarkeiten (Kapitel 2.3) kann die manuelle Ursachenforschung, wie sie bei verschiedenen bestehenden Ansätzen durchgeführt wird, sehr zeitintensiv sein [PB08]. Deswegen kann nur wesentlich langsamer adäquat auf eine Nicht-Verfügbarkeit reagiert werden, als dies mit einer autonomen Ursachenforschung möglich ist. Die adäquate Reaktion auf Nicht-Verfügbarkeiten wird außerdem durch eine autonome Fehlersuche unterstützt, da sie im Vergleich zu einer manuellen Fehlersuche eine wesentlich geringere Fehlerrate aufweist [Ios+07] und so unnötige, eventuell kostenintensive Analysen vermieden werden.

Um der in Grid-Infrastrukturen bestehenden **Sicherheitssituation** (Kapitel 2.2.6) gerecht zu werden, werden bei der Entwicklung von DAGA konsequent die in der GSI definierten Sicherheitsmechanismen berücksichtigt. Wie in der Evaluation gezeigt wurde, erfüllen nur wenige bestehende Ansätze diese Anforderung, zumeist wegen ihrer mangelnden Dienst-Orientierung.

5.1.3 Entwicklung einer flexiblen und ausgereiften Implementierung

Das dritte Ziel bei der Entwicklung umfasst verschiedene Aspekte der Implementierung von DAGA. Um ein leicht wart- und erweiterbares System mit geringem Entwicklungs- und

Testaufwand zu erstellen, sollen ausschließlich etablierte Standard[1]-Spezifikationen und -Implementierungen verwendet werden, wie beispielsweise JAX-WS (Kapitel 7.3.3) für die Instanziierung einer SOA. Ebenso sollen für ganze Funktionsbereiche fertige, von einer (großen) Community entwickelte Systeme eingesetzt werden, wie beispielsweise Cassandra für eine verteilte Datenbank, um die Entwicklungs- und Testanforderungen weiter zu verringern [BKS05]. Dieses Vorgehen unterstützt die Erfüllung vieler der im ISO 9126 Standard [SJS01] genannten Punkte zur Erreichung einer hohen Software-Qualität, entspricht dem allgemeinen Vorgehen, durch die Verwendung von bestehenden Standards eine hohe Interoperabilität zu erreichen [Bau+09b; Mar+07] und folgt einer der „Five Big Ideas“ des Grid-Computings (Kapitel 2.1.4), der Verwendung von offenen Standards.

5.2 Kernkonzepte

Um die in Abschnitt 5.1 erläuterten Ziele erreichen zu können, basiert DAGA auf den in diesem Unterkapitel vorgestellten Kernkonzepten: der Dienst-Orientierung und dem Active Probing.

5.2.1 Dienst-Orientierung

Die Dienst-Orientierung (Kapitel 3.2.2) ist eines der wichtigsten Kernkonzepte von DAGA und gleichzeitig das größte Unterscheidungsmerkmal zu bestehenden Ansätzen. Durch die Dienst-Orientierung werden Ressourcen nicht als technische Entitäten, beispielsweise Festplatten, betrachtet, sondern als Dienste (Kapitel 2.1.2), deren angebotene Funktionalität über deren jeweiliges Dienst-Interface verwendet wird.

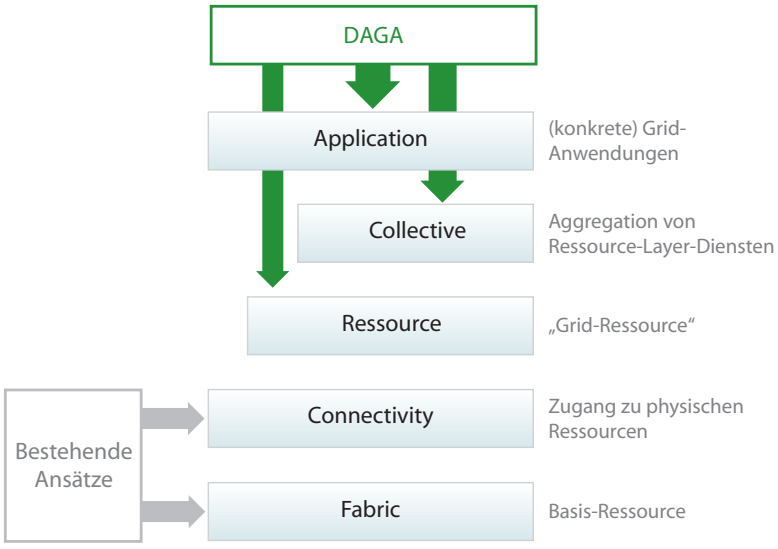


Abbildung 5.1: Dienst-Orientierung in DAGA

Der Unterschied ist in Abbildung 5.1 dargestellt: statt einzelne (technische) Werte in niederen Schichten des Grid-Layer–Stacks (Kapitel 2.1.3) zu sammeln, werden die Ressourcen über ihre nach außen zur Verfügung gestellte Dienst-Schnittstelle angesprochen und so aus

[1] Sachverhalt, der als allgemeingültig und überprüfbar angesehen wird [HH10]

Sicht eines Grid-Jobs (Kapitel 2.1.4), dem zentralen Element bei der Verwendung von Grids, betrachtet. Es werden also nicht einzelne Lese-/Schreibzugriffe oder die Übertragung einzelner Pakete analysiert [Tie+98], sondern die Funktionsweise eines Dienstes entsprechend seiner Dienstbeschreibung.

Da die Ressourcen aus Grid-Job–Sicht verwendet werden, sind auch die üblichen GSI-Credentials für die Verwendung einer Ressource notwendig, und es ist damit möglich, die Sicherheitsmechanismen eines Grids zu überprüfen, da die Analyse der Ressourcen im gleichen Umfeld stattfindet wie deren Nutzung. Unter einem *Credential* versteht man eine Information, die dazu genutzt werden kann, die Identität eines Subjekts nachzuweisen, wie beispielsweise Passwörter oder Zertifikate [Fos+98].

5.2.2 Active-Probing

Für das in Kapitel 4.2.2 beschriebene Active Probing wurde bei der Evaluation bestehender Ansätze gezeigt, dass es sich für den Einsatz in Grid-Umgebungen zur Gewinnung von Verfügbarkeits-Informationen am besten eignet. Gleichzeitig wurde aber auch verdeutlicht, dass keine Implementierung existiert, die Active Probing in Grid-Infrastrukturen einsetzt.

Diese Lücke soll mit DAGA geschlossen werden, indem das überaus positiv evaluierte Active Probing in eine für Grids zugeschnittene Lösung integriert wird. Die Anforderungen, die in der Ergebnistabelle des Active Probings als nicht erfüllt oder von einer Implementierung abhängend markiert waren, sollen durch die Entwicklung von DAGA in erfüllte Anforderungen umgewandelt werden und das in Abbildung 5.2 gezeigte Ergebnis ermöglichen.

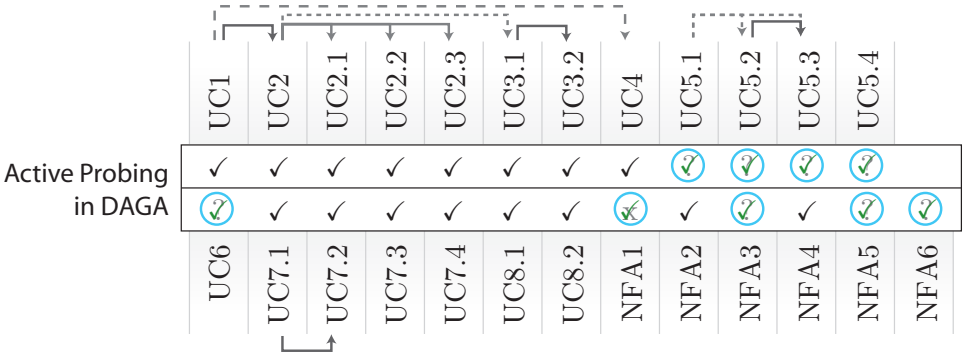


Abbildung 5.2: Angestrebtes Evaluationsergebnis von DAGA basierend auf den Ergebnissen des Active Probings

Aufgrund der generischen Definition des Active Probings kann es problemlos auch für die Gewinnung von Verfügbarkeits-Informationen in Grids eingesetzt werden, wodurch Herangehensweisen, die in anderen Anwendungsgebieten bereits seit längerer Zeit existieren, in Grid-Infrastrukturen übertragen werden können, in denen diese Herangehensweisen bisher neu sind. Im Folgenden wird die Verwendung des Active Probings begründet.

Obwohl in Grids Probleme an verschiedenen Stellen auftreten können [Tie+98], können mit Active Probing durch die interaktive Auswahl auszuführender Probes Problemursachen wesentlich schneller und effizienter gefunden werden als mit bestehenden Ansätzen. Zudem werden durch die Kernziele des Active Probings – Minimierung der Kosten bei gleichzeitiger Maximierung der diagnostischen Genauigkeit [Bro+03] – und die intelligente Auswahl der geeigneten Probes, durch die deren Anzahl minimiert und die Zeit bis zur Ermittlung des Problems

optimiert werden [Ris+04], viele Anforderungen wie eine geringe Intrusiveness (*NFA2*), eine autonome und schrittweise Annäherung an die Ursachen einer Nicht-Verfügbarkeit (*UC3.1*) inklusive der in den Probe-Typen implizit enthaltenen Lösungsvorschlägen sowie geringe Anforderungen an das Daten-Management [BRM01] erfüllt. Aus den genannten Vorteilen eignet sich Active Probing besonders gut für den Einsatz in Grid-Infrastrukturen und bildet wie eingangs erwähnt eines der Kernkonzepte von DAGA.

Durch die generische Formulierung des Active Probing–Algorithmus kann für die Auswahl der als nächstes auszuführenden Probe jeder beliebige Algorithmus verwendet werden. So ist es möglich, in verschiedenen Grids den jeweils besten Auswahl-Algorithmus einzusetzen und dennoch in allen Grids Active Probing zu verwenden. Diese Flexibilität wird im Entwurfsmodell und der prototypischen Implementierung weiter ausgeführt.

6 Entwicklung eines Entwurfsmodells

In diesem Kapitel wird ein *Entwurfsmodell* zur „Darstellung der Umsetzungslösung von gegebenen Anforderungen“ [ZGK04] für DAGA erstellt und die Frage beantwortet, „*wie* das Problem gelöst werden soll“ [Hen10a], das sich aus den in Kapitel 3 analysierten Anforderungen und den in Kapitel 5 definierten Zielen ergibt.

6.1 Vorgehen und Notation

Für die Entwicklung des Entwurfsmodells wird für ein methodisches Vorgehen gemäß Kapitel 5.1.1 der in Abbildung 6.1 dargestellte Prozess der objektorientierten Analyse in Anlehnung an [Hen10b] durchgeführt.

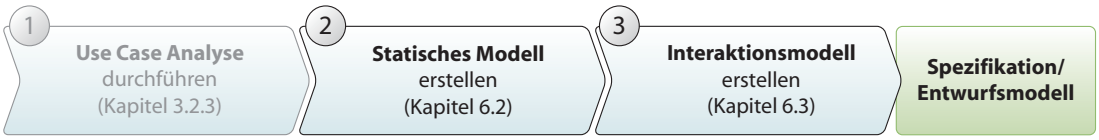


Abbildung 6.1: Schritte der objektorientierten Analyse

Nachdem die **Use Case Analyse** bereits in Kapitel 3.2.3 durchgeführt wurde, wird in Kapitel 6.2 das **statische Modell** für die Beschreibung der strukturellen und datenorientierten Aspekte von DAGA entwickelt, bestehend aus einem Datenmodell und einer Systemarchitektur mit den beteiligten Komponenten. Die möglichen *Interaktionen* der Komponenten, Subsysteme und Entitäten des statischen Modells werden anschließend in Kapitel 6.3 in einem **Interaktionsmodell** beschrieben, wobei eine Interaktion als „spezifisches Muster der Zusammenarbeit des Nachrichtenaustauschs zwischen Objekten zur Erledigung einer bestimmten Aufgabe“ [Hen10b] definiert ist. Ergebnis dieses Kapitels ist wie in Abbildung 6.1 dargestellt eine „(konstruktive) Entwurfsbeschreibung“ [Hen10a], auch *Spezifikation* oder *(Entwurfs-)Modell* genannt, die für die prototypische Implementierung in Kapitel 7 als Grundlage dient.

Für die Beschreibung des Entwurfsmodells werden die Diagrammtypen *Klassendiagramm*, *Komponentendiagramm* und *Sequenzdiagramm* der bereits in Kapitel 3.1.1 vorgestellten UML verwendet. Diese Diagrammtypen werden gewählt, da sie sich besonders gut für die Modellierung einer SOA, die auch das grundlegende Architektur-Paradigma von DAGA darstellt (Kapitel 6.2.2), eignen [Sta07]. So können einzelne Dienste mittels Klassen- und Komponentendiagrammen dargestellt werden, die Interaktionen der verschiedenen Komponenten mittels Interaktionsdiagrammen. Welche Diagrammtypen jeweils eingesetzt werden, wird an gegebener Stelle erläutert. Aufgrund der hohen Komplexität der verwendeten Diagrammtypen können diese jedoch nicht wie das Use Case–Diagramm in Kapitel 3 ausführlich vorgestellt werden. Stattdessen sei auf die offizielle Spezifikation unter [OMG07] und die umfassende Erläuterung in [RQZ07] verwiesen.

6.2 Statisches Modell

In diesem Kapitel werden die Bestandteile des statischen Modells entwickelt.

Dafür wird zunächst ein Datenmodell erstellt (Kapitel 6.2.1), das die beteiligten Elemente und deren Eigenschaften und Beziehungen zueinander darstellt. Daran anschließend wird eine Systemarchitektur entwickelt (Kapitel 6.2.2), mittels derer die Elemente des Datenmodells verarbeitet werden.

6.2.1 Datenmodell

Das in diesem Kapitel entwickelte Datenmodell beschreibt auf formalem Wege die in DAGA eingesetzten Informationsobjekte mittels ihrer Attribute und Beziehungen. Ziel ist dabei die „eindeutige Definition und Spezifikation der zu verwaltenden Objekte, ihrer für die Informationszwecke erforderlichen Attribute und der Zusammenhänge zwischen verschiedenen Informationsobjekten, um so einen Überblick über die Datensicht erhalten zu können“ [FS06].

Bei der Entwicklung des Datenmodells wurde neben der Berücksichtigung allgemeiner Anforderungen, wie die Freiheit von Redundanzen oder der hohen Effizienz, besonderer Wert auf die Erfüllung der in Kapitel 3 analysierten Anforderungen gelegt. Vier Anforderungen spielen hierbei eine besondere Rolle: aus *UC3.2* ergeben sich unmittelbare Auswirkungen auf das Datenmodell, da es möglich sein muss, Probe-Traces abzubilden. Eine hohe Flexibilität gegenüber Erweiterungen von Ressourcen- und Fehlertypen muss für *UC7.3* und *UC7.4* gewährleistet sein. Die meisten Auswirkungen ergeben sich aber aus *NFA1*, in der die Forderung nach der Verwendung eines in Grid-Infrastrukturen etablierten und verbreiteten Datenschemas für die Beschreibung der Grid-Entitäten gestellt wird. Um dieser Anforderung gerecht zu werden, wird das Datenmodell von DAGA nicht vollständig neu entwickelt, sondern es wird ein bestehendes Datenmodell um DAGA-spezifische Informationsobjekte erweitert.

Auswahl des zu erweiternden Datenmodells

Für die Beschreibung von Grid-Ressourcen kann aus einer Vielzahl von Möglichkeiten gewählt werden, wie beispielsweise dem *Grid Laboratory for a Uniform Environment*-Schema (GLUE), der *D-Grid Resource Definition Language* (D-GRDL) [Wei+08; Wol07], dem *Common Information Model* (CIM) [Urlb; Dia+08], der *Common Data Representation* (CORBA) [Sme+04] oder dem *NorduGrid Information Model* [Kón06]. Jedes der genannten Datenmodelle hat Vor- und Nachteile bezüglich seiner Flexibilität, der Unterstützung in Grid-Middlewares und seiner technischen Komplexität. Das OGSA WS-RF Basic Profile [FMS05] und das damit festgelegte *WS-Resource Framework* (WS-RF) als „grundlegenden Baustein für Zugriffe auf Bestandteile eines Grids“ [BFR07] eignen sich nicht als zugrunde liegendes Datenmodell, da sich diese Beschreibungen auf Web Services beschränken und nicht die notwendige Komplexität und Datenvielfalt aufweisen.

Als Grundlage für das DAGA-Datenmodell wird das von der GLUE Working Group [GW] innerhalb des Open Grid Forums [LRO] entwickelte GLUE-Datenmodell in der Version 2.0 [And+09] gewählt. Es beschreibt ein konzeptionelles Informationsmodell für Grid-Entitäten mittels natürlicher Sprache und UML–Klassendiagrammen [And+09] und ist durch sein hohes Abstraktionsniveau von konkreten Implementierungen unabhängig. GLUE 2.0 beinhaltet Beschreibungen für Standardtypen wie *Computing Element*, *Storage Element*, *Service* oder *Endpoint*, gleichzeitig gibt es aber auch Anknüpfungspunkte für individuelle Erweiterungen in Form des Feldes *OtherInfo*, das in allen GLUE-Klassen enthalten ist, und der

Extension-Klasse, mittels derer Schlüssel/Wert-Paare an jede beliebige GLUE-Entität angeknüpft werden können [And+09]. Zudem ist vorgesehen, bestehende Klassen des GLUE-Datenmodells durch eigene Subklassen zu erweitern und diese Subklassen beispielsweise für die Bildung von Assoziationen zu GLUE-fremden Klassen zu verwenden. Durch diese Erweiterungsmöglichkeiten ist es mit GLUE 2.0 möglich, den Standardisierungsentwicklungen des OGF zu folgen und gleichzeitig individuelle Anpassungen durchzuführen [BFR07]. Für eine einfache Verwendung des Datenmodells in verschiedenen Implementierungen werden in „GLUE 2.0 - Reference Realizations to Concrete Data Models“ [And+08] Anleitungen gegeben, wie das Mapping zwischen der abstrakten GLUE-Definition und einer Datenspeicherung in XML, MySQL und LDAP durchgeführt werden muss.

Es wird das GLUE-Schema in Version 2.0 verwendet, da es bei den Grid-Middlewares gLite und Globus Toolkit standardmäßig eingesetzt wird [Allb; GW; BFR07] und dadurch in bestehenden Grid-Lösungen sehr verbreitet ist [BFR07]. Es wird zudem in vielen wichtigen Grid-Projekten wie dem „Enabling Grids for e-Science“, dem „Worldwide Large Hadron Collider Computing Grid“ [Eura] oder dem „Open Science Grid“ [Urld] eingesetzt [Bau+09b]. Diese weite Verbreitung ermöglicht die Erfüllung der Anforderung *NFA1*. Daneben bietet GLUE 2.0 weitere Vorteile, die es für die Verwendung in DAGA interessant machen, wie beispielsweise die nachgewiesene Notwendigkeit der enthaltenen Datenfelder beim Einsatz in der Praxis [BFR07] oder die Erweiterbarkeit von GLUE 2.0 über die beschriebenen Erweiterungspunkte [And+09]. Abschließend ist GLUE 2.0 das einzige Datenschema für Grid-Ressourcen, das Virtuelle Organisationen unterstützt [Bau+09b], eine wichtige Voraussetzung für die Erfüllung von Anforderung *UC7.2*. Es wird Version 2.0 des GLUE-Datenmodells verwendet, da in dessen Entwicklung die Erfahrungen aus anderen Modellierungsansätzen in Produktiv-Infrastrukturen eingeflossen sind [And+09] und es somit wesentlich ausgereifter ist als die vorherigen Versionen.

Das DAGA-Datenmodell im Überblick

Nach der Auswahl des zu erweiternden Datenmodells kann nun das DAGA-Datenmodell entwickelt werden.

Das DAGA-Datenmodell (Abbildung 6.2, Seite 74), das ebenso wie das GLUE-Datenmodell als Klassendiagramm formalisiert ist, gliedert sich in drei Ebenen: die obere Ebene beschreibt die *GLUE-Klassen*, die für die Verwendung in DAGA relevant sind. Die Subklassen, die bestehende GLUE-Klassen erweitern und als *Schnittstellen-Klassen* zwischen den GLUE- und DAGA-Klassen dienen, sind in der mittleren Ebene dargestellt. In der unteren Ebene sind die *DAGA-Klassen* enthalten, die für die Speicherung der Verfügbarkeits-Informationen eingesetzt werden.

GLUE-Klassen Das Datenmodell enthält einige Klassen aus dem GLUE-Datenmodell, um die in DAGA relevanten Grid-Elemente beschreiben zu können. Die GLUE-Klassen sind nur ausschnittsweise erläutert, und es sind sowohl einige Assoziationen als auch einige Attribute nicht angegeben, da sie für DAGA unerheblich sind. Alle GLUE-Klassen inklusive der hier verwendeten sind in der GLUE 2.0 Spezifikation [And+09] umfassend beschrieben.

Die Klasse **Entity** stellt in GLUE die Superklasse aller Klassen dar und beschreibt die Attribute, die allen Klassen gemein sind, wie beispielsweise eine ID in Form eines *Uniform Resource Identifiers* (URI).

Die **Service**-Klasse des GLUE-Datenmodells wird verwendet, um die Dienst-Orientierung, eines der Kernkonzepte von DAGA (Kapitel 5.2.1), und die damit einhergehenden Dienste abbilden zu können.

Diese Dienste werden wiederum über **Endpoint**-Klassen nach außen verfügbar gemacht. Ein Endpoint hat dabei verschiedene Meta-Informationen wie beispielsweise die Adresse des beschreibenden WSDL-Dokuments (Erläuterung zu WSDL siehe Kapitel 7.2) oder Informationen zu geplanten Ausfallzeiten. Zudem sind über den Endpoint technische Informationen verfügbar, beispielsweise welche Fähigkeiten der Endpoint anbietet (**Capability_t**) oder über welche Technologie er implementiert wird (**EndpointTechnology_t**).

Für die geographische Lokalisierung von Objekten innerhalb einer Grid-Infrastruktur wird die **Location**-Klasse verwendet, die neben den üblichen Adressfeldern auch Längen- und Breitengrad enthält.

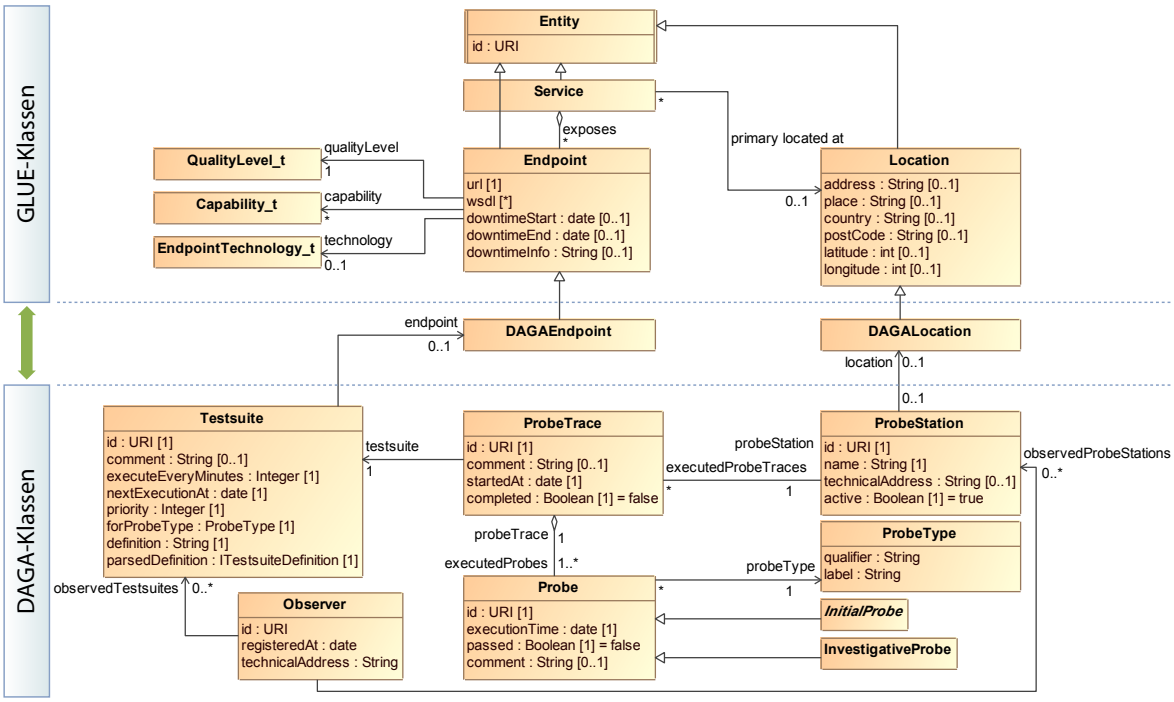


Abbildung 6.2: Datenmodell der in DAGA verwendeten Informationsobjekte

Schnittstellen-Klassen Für die Bildung von Assoziationen zwischen GLUE-Klassen und GLUE-fremden Klassen wird entsprechend der Empfehlung von [And+09] für jede zu verbindende GLUE-Klasse eine Subklasse angelegt, die für die Assoziationen verwendet wird. Diese Schnittstellen-Klassen haben keine zusätzlichen Attribute und dienen lediglich als Bindeglied zwischen den beiden Bereichen.

DAGA-Klassen Die DAGA-Klassen werden dazu verwendet, die gewonnenen Verfügbarkeits-Informationen in Form von Probes zu speichern und die Ergebnisse mittels Probe-Traces nachvollziehbar zu machen. Die einzelnen Klassen werden in Kapitel 6.2.1 umfassend erläutert.

Allgemeine Datenformate und Annahmen

Nachdem das DAGA-Datenmodell in einem Überblick dargestellt wurde, werden nun Datenformate und Annahmen, die für alle Klassen im DAGA-Datenmodell gelten, vorgestellt.

Objekt-Identifikation Alle DAGA-Klassen (ausgenommen die Klasse `ProbeType`) werden durch das obligatorische Attribut `id` identifiziert. Diese ID ist als URI modelliert, die die in Standard RFC 3986 [BLFM05] definierte Syntax verwendet und die in Code-Listing 6.1 gezeigte Form aufweist.

```
foo://example.com:8042/over/there?name=ferret#nose
\-/   \--------------/\---------/ \---------/ \--/
 |          |             |           |        |
scheme   authority      path        query   fragment
```

Listing 6.1: Generische Syntax einer URI gemäß RFC 3986

Die IDs der verschiedenen DAGA-Objekte verwenden als `scheme` doi, da es sich bei den IDs um *Digital Object Identifier* (DOI) handelt, als `authority` daga, um einen einheitlichen Namensraum zu bilden, und als `path` die Herkunft des jeweiligen Objekts. Code-Listing 6.2 zeigt den allgemeinen Aufbau einer ID eines DAGA-Objektes.

```
doi://daga/location/<id-aus-config>/probestation/<id-aus-config>/probetrace
    /<timestamp>/probe/<timestamp>
```

Listing 6.2: Allgemeine Syntax einer DAGA-ID

Diese Syntax wird aus zwei Gründen verwendet: zunächst ist es durch das `path`-Element möglich, den Kontext einzelner Objekte, wie beispielsweise einer Probe, eindeutig und schnell zu identifizieren, ohne dafür (komplexe) Datenbankanfragen ausführen oder Verknüpfungen erstellen zu müssen. Zudem können durch die Syntax eindeutige IDs von verschiedenen Komponenten innerhalb eines verteilten Systems generiert werden, ohne dabei inkonsistente Datenzustände durch die Mehrfachverwendung einer ID zu riskieren oder bei einer zentralen Stelle die nächste konsistente ID anfragen zu müssen.

Code-Listing 6.3 zeigt beispielhaft die ID einer Probe, anhand derer die genannten Gründe erläutert werden.

```
doi://daga/location/12/probestation/3/probetrace/4983457/probe/093458
```

Listing 6.3: Beispielhafte ID einer Probe

Der Kontext der identifizierten Probe kann direkt aus der ID abgelesen werden: die Probe wurde zum Zeitpunkt *093458* in einem Probe-Trace ausgeführt, der zum Zeitpunkt *493457* gestartet und von Probe-Station *3* auf der Location *12* initiiert wurde. Die Datenintegrität ist gewährleistet, da eine andere Probe, die ebenfalls zum Zeitpunkt *093458* in einem Probe-Trace *493457* ausgeführt wurde, nicht auf der gleichen Probe-Station existieren kann und daher eine andere ID zustande käme, wie beispielhaft in Code-Listing 6.4 gezeigt.

```
doi://daga/location/48/probestation/9/probetrace/4983457/probe/093458
```

Listing 6.4: Beispielhafte ID einer Probe auf einer anderen Probe-Station

Die gezeigte Unabhängigkeit von einer zentralen Instanz bei der Generierung der IDs ist von großer Wichtigkeit, da DAGA entsprechend einer SOA strukturiert ist (Kapitel 6.2.2), und daher alle Komponenten autonom und möglichst ohne Abhängigkeiten agieren können müssen. Zudem würde eine zentrale Vergabestelle für IDs einen systemkritischen Single-Point-of-Failure darstellen.

Zeitpunkte Zeitpunkte sind im DAGA-Datenmodell als `Date` und nicht als Unix-Zeitstempel modelliert, um die von vielen Datenbank-Systemen unterstützten Zeit- und Datumsfunktionen verwenden zu können. Ist ein Zeitpunkt als `Date` bzw. `Datetime` gespeichert, können arithmetische Funktionen auf den gespeicherten Werten ausgeführt werden, beispielsweise kann zu einem Datum ein Zeitraum addiert werden, wobei Tages-, Wochen- und Monatsgrenzen berücksichtigt werden, was bei Unix-Zeitstempel–Werten meist nicht ohne vorherige Umformungsschritte möglich ist.

Details der einzelnen Klassen

In den nun folgenden Abschnitten werden die einzelnen DAGA-Klassen des Datenmodells detailliert beschrieben. Die ID (`id`) für die eindeutige Identifikation der Objekte wird bei der Beschreibung nicht erwähnt, da jedes DAGA-Objekt (außer `ProbeType`) eine ID mit den bereits erläuterten Eigenschaften besitzt.

ProbeStation Die Klasse `ProbeStation` modelliert die Daten der in Kapitel 6.2.2 vorgestellte Probe-Station, also derjenigen Komponente, die Probe-Traces und die darin enthaltenen Probes ausführt.

Sie wird durch einen für Menschen lesbaren Namen (`name`) sowie eine technische Adresse (`technicalAddress`), beispielsweise einen Hostnamen, beschrieben. Die technische Adresse ist als `String` modelliert, um Einschränkungen auf bestimmte Formate zu vermeiden und so Schwierigkeiten bei der Verwendung und insbesondere bei der Erweiterung entgegenzuwirken. Wäre die Adresse beispielsweise auf das IPv4-Adressenformat beschränkt, könnten IPv6-Adressen nicht gespeichert werden, und für deren Verwendung müssten sowohl das Datenmodell als auch die Implementierung angepasst werden. Um bei der Verteilung von auszuführenden Testsuites auf verschiedene Probe-Stations nur diejenigen zu verwenden, die einsatzbereit sind, enthält die Klasse eine Markierung (`active`), die festlegt, ob eine Probe-Station verwendet werden kann oder nicht. Um ein Probe-Station-Objekt geographisch lokalisieren zu können, ist es über die Schnittstellen-Klasse `DAGALocation` mit einer GLUE-Location verbunden.

ProbeTrace Unter einem Trace versteht man alle Ergebnisse, die vom System generiert werden, von Beginn des Probe-Durchgangs bis zur Lösung des Problems [MsS93]. Der Probe-Trace fasst beliebig viele Probes zu einem Ausführungsstrang zusammen und unterstützt somit die Erfüllung der Anforderung *UC3.2*.

Neben einem optionalen Kommentar (`comment`), der beispielsweise für die spätere Auswertung verwendet werden kann, besitzt ein Probe-Trace-Objekt den Startzeitpunkt seiner Ausführung (`startedAt`) sowie eine Markierung, ob der Probe-Trace bereits abgeschlossen ist oder nicht (`completed`). Diese Markierung ist notwendig, um Probe-Traces bereits vor ihrer Fertigstellung abspeichern zu können, ohne dabei die Aussage der Daten zu verfälschen. Würde die Markierung fehlen, könnte es beispielsweise vorkommen, dass ein Probe-Trace, zu dem noch Investigative Probes ausgeführt werden, im Hintergrundspeicher abgelegt und für die Auswertung verwendet wird, obwohl sich noch Probes in der Ausführung befinden. Um nachvollziehen zu können, welche Probes innerhalb des Probe-Trace ausgeführt wurden, ist eine Assoziation angegeben.

Testsuite Ein Probe-Trace wird auf Veranlassung einer Testsuite ausgeführt, oder anders formuliert, ein Probe-Trace ist das Ergebnis einer Testsuite-Ausführung. Um die Details der in der Testsuite enthaltenen Vorgaben verwenden und so das Ergebnis eines Probe-Trace besser beurteilen zu können, besteht eine Assoziation zwischen einem Probe-Trace-Objekt und einem Testsuite-Objekt.

Der Test, den eine Testsuite definiert, kann sich auf einen Service beziehen und somit auf einen Endpoint, der diesen Dienst verfügbar macht. Für eine spätere Erweiterbarkeit ist die Multiplizität der Assoziation zwischen Testsuite und Endpoint aber als optional (`0..1`) angegeben, da auch Testsuites denkbar sind, die nicht einen Endpoint betreffen, da sie beispielsweise einen Grid-Job ausführen, für den kein Endpoint vorhanden ist. Durch die Verknüpfung einer Testsuite mit einem Endpoint, der wiederum mit weiteren Objekten innerhalb des GLUE-Schemas assoziiert ist, kann für jede Testsuite und damit auch jeden Probe-Trace und jede Probe die betroffene VO ermittelt werden, wodurch VO-spezifische Informationsausschnitte möglich sind [Bau+09b].

Die Attribute einer Testsuite können in *Ausführungsdaten* und *Testdaten* unterteilt werden.

Die **Ausführungsdaten** bestehen aus dem nächsten Ausführungszeitpunkt der Testsuite (`nextExecutionAt`), dem Zeitintervall, das zwischen zwei Ausführungen der Testsuite liegen soll (`executeEveryMinutes`), und der Priorität der Testsuite, angegeben in einem Bereich von 1 bis 5 (`priority`). Diese Felder spielen bei der Auswahl der als nächstes auszuführenden Testsuite eine Rolle: hier wird diejenige ausgewählt, deren nächster Ausführungszeitpunkt am weitesten in der Vergangenheit liegt und deren Priorität am höchsten ist. Gibt es in dieser Menge mehrere Objekte, wird zufällig eines gewählt. Bei der Ausführung wird als Wert für den nächsten Ausführungszeitpunkt die aktuelle Zeit, addiert um die Minuten im Attribut `executeEveryMinutes`, gespeichert. Der zeitliche Abstand ist in Minuten modelliert, da eine kleinere Zeiteinheit eine unnötige Systemlast erzeugen und eine größere Zeiteinheit zu grobe Ergebnisse liefern würde.

Die **Testdaten** sind für die konkrete Ausführung der Testsuite von Bedeutung: zunächst wird angegeben, welche Initial Probe für die Ausführung geeignet ist (`forProbeType`), die genauen Handlungsvorgaben für die Initial Probe werden in der Definition (`definition`) festgelegt. Um eine möglichst hohe Flexibilität und Erweiterbarkeit des Datenmodells an dieser Stelle zu erreichen, ist die Definition als String modelliert, wodurch sowohl strukturierte Daten als XML- oder JSON-String abgelegt werden können als auch unstrukturierte Daten in Form von Freitext. Damit die Initial Probe bei der Ausführung die Handlungsvorgaben der Definition interpretieren kann, wird zusätzlich eine maschinenlesbare Definition gespeichert (`parsedDefinition`), die aus der Definition (`definition`) bei der Ausführung der Testsuite erzeugt wird. Um auch hier größtmögliche Flexibilität zu erzielen, ist die maschinenlesbare Definition als Interface modelliert, das von beliebigen Klassen, die an den jeweiligen Inhalt der Definition angepasst sind, implementiert werden kann. Ein Beispiel für die Definition einer Testsuite, die einen Dienst mittels einer WSDL-Probe überprüft, ist in Code-Listing B.3 (Seite 131) angegeben, dessen Verwendung im Interaktionsmodell (Kapitel 6.3) beschrieben wird.

Aus der hohen Flexibilität bei der Definition von Testsuites und der gleichzeitig festen und transparenten Position der Testsuite innerhalb des DAGA-Datenmodells ergeben sich umfassende Einsatzgebiete für DAGA: so kann es in mehreren Grid-Middlewares zeitgleich verwendet werden, da es das Entwurfsmodell erlaubt, für jede Grid-Middleware passende Initial und Investigative Probes zu erstellen. Ebenso ist es möglich, neben Diensten mit einem Endpoint auch andere Funktionalitäten mit einer Initial Probe zu analysieren, da die Assoziation zwischen Testsuite und Endpoint optional modelliert ist und über den Inhalt einer

Testsuite-Definition keine Einschränkungen gegeben sind. So kann beispielsweise neben einem Web Service auch ein GridFTP-Server über dessen Dienst-Interface überprüft werden, wie es die prototypische Implementierung in Kapitel 7.4.4 zeigt.

Probe Eine Probe modelliert ein einzelnes Testpaket innerhalb eines Probe-Trace und unterteilt sich in *Initial Probes* und *Investigative Probes* (Kapitel 4.2.2).

Die Datenfelder, die allen Probes gemeinsam sind, sind in der Superklasse `Probe`, von der die verschiedenen Subklassen ableiten, zusammengefasst. Für eine chronologische Sortierung der Probes innerhalb des zugeordneten Probe-Trace wird ein Ausführungszeitpunkt für jede Probe gespeichert (`executionTime`), anhand dessen eine Ausführungsreihenfolge erstellt werden kann. Neben einem optionalen Kommentar (`comment`), der bei der Ergebnis-Auswertung verwendet werden kann, wird eine Markierung gespeichert, ob die Probe erfolgreich ausgeführt wurde oder nicht (`passed`). Die Bewertung des Ausführungsergebnisses, also ob die Ausführung erfolgreich war oder nicht, wird von der jeweiligen Probe selbst durchgeführt. Die Aussage der Markierung bezieht sich dabei nicht auf den Gesamtkontext, sondern lediglich darauf, ob die Ausführung entsprechend der Definition der Testsuite (bei Initial Probe) oder einer spezifischen Funktion (bei Investigative Probe) erfolgreich war. Die Subklassen **InitialProbe** und **InvestigativeProbe** werden modelliert, um bei der Entwicklung weiterer Probe-Typen eine Strukturierungsmöglichkeit zu bieten.

ProbeType Jede Probe hat einen eindeutigen Probe-Typ, der aus einem maschinenlesbaren Qualifier (`qualifier`) und einer Beschriftung (`label`) besteht. Der Qualifier wird für die interne Referenz verwendet, er besteht aus Großbuchstaben und enthält keine Zahlen oder Leer- und Sonderzeichen. Die Beschriftung kann alle Zeichen enthalten und beschreibt den Probe-Typ in natürlicher Sprache.

Observer Für die Erfüllung der Anforderung *UC5.3*, Push-Anfragen stellen zu können, ist die Verwaltung von Observern notwendig.

Dafür modelliert die Klasse `Observer` alle dafür benötigten Felder. Der Zeitpunkt der Registrierung (`registeredAt`) ist für die Auswertung der Observer vorgesehen. Um die jeweiligen Observer über Änderungen im Datenbestand informieren zu können, ist eine technische Adresse notwendig (`technicalAddress`), die ebenso wie die technische Adresse der Probe-Station als String modelliert ist. Es besteht eine Assoziation zu Testsuite-Objekten und Probe-Station-Objekten, da Observer diese beiden Elemente überwachen können.

6.2.2 Architekturmodell und Beschreibung der Komponenten

In diesem Kapitel werden das Architekturmodell von DAGA sowie die darin enthaltenen Komponenten entwickelt, die die im Datenmodell (Kapitel 6.2.1) beschriebenen Objekte einsetzen.

Wie auch bei der Entwicklung des Datenmodells wurde bei der Entwicklung des Architekturmodells neben der Einhaltung allgemeiner Richtlinien, wie beispielsweise Flexibilität und Stabilität, besonderer Wert auf die Erfüllung der analysierten Anforderungen (Kapitel 3) gelegt. So wurde beispielsweise darauf geachtet, die Erweiterbarkeit und Flexibilität des Datenmodells auch in der Systemarchitektur und den einzelnen Komponenten fortzuführen, um die Erfüllung der Anforderungen *UC7.3* und *UC7.4* zu ermöglichen.

Für einen einfachen Einstieg wird die Systemarchitektur zunächst informell in Abbildung 6.3 (Seite 79) dargestellt, um einen Überblick über die Komponenten und die allgemeine

Architektur zu geben. Die Komponenten sind grün, die Kommunikationswege blau und die von DAGA nur modellierten, aber nicht implementierten Komponenten grau gezeichnet. Sowohl die gesamte Systemarchitektur als auch die einzelnen Komponenten werden in den folgenden Unterkapiteln mittels UML-Diagrammen formal modelliert und ausführlich beschrieben.

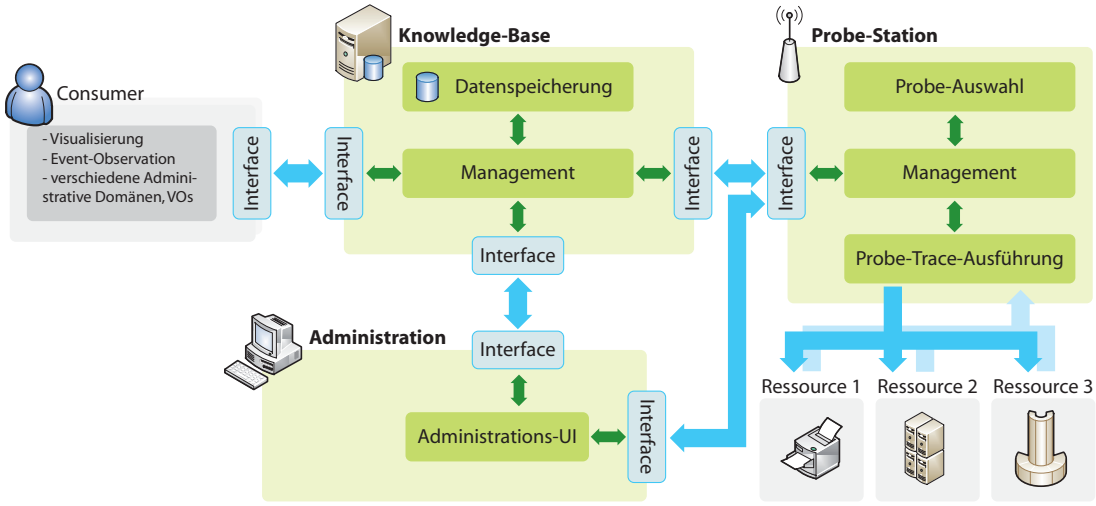


Abbildung 6.3: Informelles Architekturmodell von DAGA mit Komponenten und Kommunikationswegen

Zugrunde liegendes Architektur-Paradigma

Wie aus Abbildung 6.3 ersichtlich wird, wird DAGA entsprechend den Konzepten und Definitionen einer *Service-orientierten Architektur* (SOA) entwickelt.

Unter einer SOA versteht man „a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains“ [Mac+06]. Solch ein Dienst (Capability), auch *Service* oder *Komponente* [Str97] genannt, beschreibt eine eigenständige Funktionalität, mit der eine komplette Business-Funktion ausgeführt werden kann [HH10]. Für seine Verwendung stellt der Dienst eine öffentliche Schnittstelle (Interface) und die dazugehörige Schnittstellenbeschreibung zur Verfügung [HH10]. Kenntnisse über die Interna des Dienstes, wie beispielsweise die verwendeten Technologien, sind für dessen Verwendung nicht notwendig [HH10]. Da die interagierenden Dienste aus verschiedenen Domänen stammen (können), sollte ein Dienst bei der Ausführung seiner Funktionalitäten (theoretisch) unabhängig von anderen Diensten sein.

Als grundlegendes Architektur-Paradigma von DAGA wird eine SOA verwendet, da Grids ebenfalls als SOA strukturiert sind (Grid-Checkliste Kapitel 2.1.1) und durch die abstrakte Beschreibung der Dienste innerhalb einer SOA verschiedene technische Implementierungen möglich sind, was zu Plattformunabhängigkeit und Flexibilität führt.

Systemarchitektur und allgemeine Elemente

Systemarchitektur Das UML-Komponentendiagramm in Abbildung 6.4 (Seite 80) formalisiert den soeben gegebenen Überblick der Systemarchitektur und verdeutlicht, wie DAGA strukturiert ist, wie die Strukturen erzeugt werden und welche Wechselbeziehungen bestehen.

Es zeigt die in DAGA modellierten Komponenten *Knowledge-Base*, *Probe-Station*, *Ressource*, *Consumer* und *Administration* sowie deren Multiplizitäten, die in eckigen Klammern hinter dem jeweiligen Namen der Komponenten angegeben sind. Die Komponenten interagieren über *Ports*, die das Innenleben einer Komponente abstrahieren, wodurch entsprechend den Vorgaben einer SOA keine Kenntnisse über technische Interna der Komponenten für ihre Verwendung notwendig sind. Jeder Port ist mit einer Beschriftung versehen, die seine Verwendungsmöglichkeiten erläutert.

Neben der Strukturierung der Komponenten untereinander stellt das Komponentendiagramm auch die innere Aufteilung der Komponenten dar, die vorgenommen wird, um einzelne Subsysteme oder Artefakte austauschen zu können, ohne die gesamte Komponente verändern zu müssen, beispielsweise bei der Weiterentwicklung einzelner Subsysteme. Als spezifisches Artefakt wird zusätzlich der Stereotyp `Database` verwendet, der eine Datenbank-Software eines Drittanbieters modelliert, wie beispielsweise einen MSSQL-Server, eine Oracle-Datenbank oder eine MySQL-Datenbank. Dieser Stereotyp wird von DAGA nicht implementiert, sondern lediglich als abgeschlossenes System über dessen angebotene Schnittstellen verwendet. Der Stereotyp kann in jeder Komponente von einer anderen Datenbank-Software realisiert werden, um in jeder Komponente die technisch und konzeptionell am besten geeignete Lösung einsetzen zu können. So könnte die Speicherung der Daten in der Knowledge-Base durch ein verteiltes Datenbank-System implementiert werden, um einen Single-Point-of-Failure zu vermeiden, und die Queue innerhalb einer Probe-Station durch eine XML-Dokumentendatenbank, um keine dedizierten Datenbank-Server zu benötigen.

Die genauen Eigenschaften der Elemente des Komponentendiagramms werden in den folgenden Abschnitten erläutert.

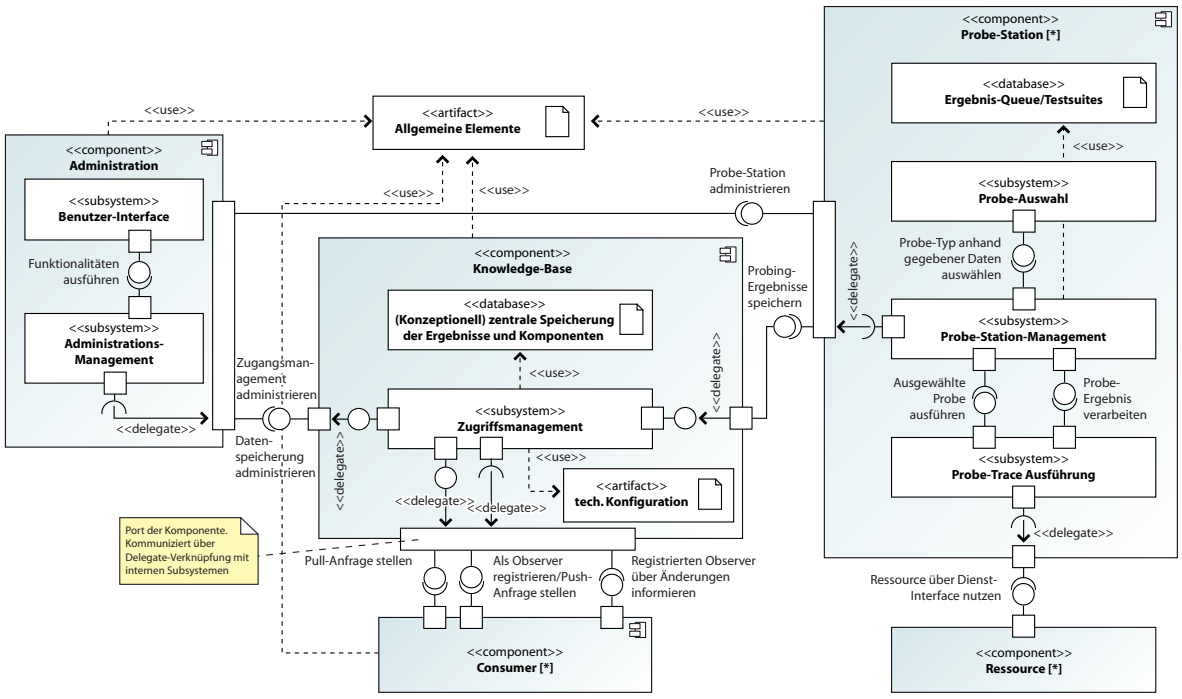


Abbildung 6.4: Das Komponentendiagramm stellt die verschiedenen Komponenten und deren Abhängigkeiten in DAGA dar

Artefakt „Allgemeine Elemente“ Wie bereits erläutert verwendet DAGA als grundlegendes Architektur-Paradigma eine SOA, d.h. die einzelnen Komponenten sind bei der Ausführung ihrer Funktionalitäten unabhängig voneinander. Dennoch verwenden alle Komponenten das gleiche, im vorherigen Kapitel vorgestellte Datenmodell und somit semantisch einheitliche Entitäten, wie beispielsweise Probes oder Probe-Traces.

Um diese globale Semantik und einen einheitlichen Zugriff auf die Daten zu vereinfachen und Datenkonsistenz zu unterstützen, enthält die Systemarchitektur im Artefakt „Allgemeine Elemente“ eine Sammlung von Klassen, die den Datenzugriff in DAGA betreffen und die von allen Komponenten verwendet werden (können). Diese Klassen beinhalten beispielsweise einen Generator für Objekt-IDs entsprechend der in Kapitel 6.2.1 gegebenen Syntax sowie die um verschiedene Methoden erweiterten Klassen aus dem Datenmodell.

Diese Klassen werden in das Artefakt ausgelagert, um auf Veränderungen am Datenmodell, beispielsweise durch neue Attribute oder Probe-Typen, schnell und einfach reagieren zu können. Die Änderungen müssen nur einmal an den im Artefakt enthaltenen Klassen durchgeführt werden und sind anschließend direkt für alle Komponenten verfügbar.

Zu beachten ist, dass die gemeinsame Verwendung der Klassen nicht bedeutet, dass daraus gegenseitige Abhängigkeiten entstehen, beispielsweise weil eine Komponente die Klassen einer anderen Komponente verwenden muss. Es ist vielmehr so, dass die Klassen unabhängig von den Komponenten entwickelt werden und dann in Form eines Bundles, z.B. eines jar-Archivs (Kapitel 7), in die einzelnen Komponenten importiert und dort verwendet werden können.

Die Klassen des Artefakts sind in die zwei Namensräume `daga.common.core` (Abbildung A.1, Seite 127) und `daga.common.elements` (Abbildung A.2, Seite 128) unterteilt. Die Klassen im erstgenannten Namensraum betreffen technische Gemeinsamkeiten, wie beispielsweise die Abbildung von Relationen einer Datenbank auf Objekte oder die Verwendung von Konfigurationsdateien. Die Klassen des zweiten Namensraums betreffen inhaltliche Gemeinsamkeiten, wie beispielsweise das DAGA-Datenmodell oder den ID-Generator. Wird das Datenmodell erweitert, z.B. um neue Initial Probes, müssen nur die Klassen des Namensraums `daga.common.elements` angepasst und an die einzelnen Komponenten verteilt werden, die das neue Datenmodell dann unmittelbar verwenden können.

Komponente „Ressource“

Die Komponente modelliert eine übliche Grid-Ressource mit den in Kapitel 2.1.2 beschriebenen Eigenschaften. Sie wird über ihr Dienst-Interface verwendet und von DAGA nicht implementiert, daher sind auch keine weiteren Details wie Klassendiagramme angegeben. Da es mit DAGA möglich sein soll, beliebig viele Ressourcen zu analysieren, ist die Multiplizität der Komponente als beliebig (*) angegeben.

Komponente „Consumer“

Diese Komponente modelliert einen Consumer, im Grid-Kontext eine Entität, die relevante und präzise Informationen in einem klaren Zeitfenster abrufen und dabei möglichst wenig Aufwand betreiben möchte [BGN02]. Entsprechend den analysierten Anforderungen (Kapitel 3) kann der Consumer Verfügbarkeits-Informationen mittels Pull- und Push-Anfragen stellen, die über die entsprechenden Ports modelliert werden. Die Verfügbarkeits-Informationen verwendet der Consumer für verschiedene Aktionen, beispielsweise textuelle Darstellung, Visualisierung von Zusammenhängen oder Ableitung weiterer Informationen [MsS93].

Der Consumer wird von DAGA nicht implementiert, daher sind keine weiteren Details wie Klassendiagramme zum Consumer angegeben. Da beliebig viele Consumer auf die von DAGA gewonnenen Verfügbarkeits-Informationen zugreifen können sollen, ist die Multiplizität der Komponente als beliebig (*) angegeben.

Komponente „Knowledge-Base“

In der Knowledge-Base werden alle Daten gespeichert, die von den Probe-Stations bei der Analyse der Ressourcen gesammelt werden und von Consumern anschließend abgefragt werden können.

Durch die (logisch) zentrale Sammlung aller Daten in dieser Komponente ergeben sich verschiedene Vorteile bei der *Abfrage der Daten* und der *Verwendung der Daten*, und es werden wesentlich *schlankere Probe-Stations* ermöglicht, wie nachfolgend erläutert wird.

Für die **Abfrage der Daten** muss für die Consumer nur eine Adresse bzw. ein Port publiziert werden, über den die Daten abgerufen werden können. Consumer müssen sich damit nicht bei mehreren Producern registrieren, wie es beispielsweise bei der GMA der Fall ist (Kapitel 4.1.1), um Daten abzufragen, sondern sie können bei der Knowledge-Base die Daten zentral abrufen. Durch die zentrale Position ist es zudem möglich zu protokollieren, welche Daten von wem wann angefragt wurden, und es kann einheitlich sichergestellt werden, dass nur autorisierte Anfragen entsprechend bestehender Nutzungs-Policies (Kapitel 8.2.2) möglich sind.

Bei der **Verwendung der Daten** ist es durch die zentrale Speicherung wesentlich einfacher, ein ganzheitliches Bild über den Zustand des Grids und dessen Verlauf zu erzeugen, als dies bei bestehenden Monitoring-Ansätzen möglich ist, da hier die Daten mitunter stark verteilt sind und nur schwierig aggregiert werden können (Kapitel 3.2.2). Ein ganzheitliches Bild ist aber beispielsweise eine Voraussetzung für die Ableitung von Zuverlässigkeits-Informationen (Kapitel 8.2.2).

Die zentrale Speicherung der Daten ermöglicht wesentlich **schlankere Probe-Stations**, da diese die Daten nur an die Knowledge-Base übermitteln müssen, die die Aufgabe der Datenverteilung übernimmt und Probe-Stations damit weder Consumer verwalten noch Anfragen bearbeiten müssen, wie es beispielsweise die GMA vorsieht.

Um eine hohe Flexibilität gegenüber Weiterentwicklungen zu erreichen, werden die Funktionalitäten der Knowledge-Base in das Subsystem *Zugriffsmanagement* und das Artefakt *Datenspeicherung* unterteilt.

Subsystem „Zugriffsmanagement“

Das Subsystem nimmt die an die Knowledge-Base gestellten Anfragen entgegen, bearbeitet diese und sendet eine Antwort an die jeweilige anfragende Komponente zurück. Zudem ist es für die Verwaltung der Observer verantwortlich. Die Datenspeicherung wird dabei vollständig vom Datenbank-Artefakt übernommen, das vom Subsystem für die Bearbeitung der Anfragen über die vom Datenbank-Artefakt bereitgestellten Schnittstellen verwendet wird. Abbildung 6.5 (Seite 83) zeigt das Klassendiagramm des Subsystems. Die bereits genannten Aufgabenbereiche *Anfragen-Handling* und *Observer-Management* sind in der Package-Struktur wiederzuerkennen.

Das **Anfragen-Handling** bearbeitet lesende Anfragen, die insbesondere für Consumer relevant sind, in der Klasse `ReadQueryHandler` und schreibende Anfragen, die von Probe-Stations und der Administration gestellt werden, in der Klasse `ModificationQueryHandler`.

Für eine verbesserte Performance verwenden beide Klassen die vorbereiteten Datenbank-Anfragen, die in der Klasse `PreparedStatementsKnowledgeBase` definiert sind. Alle Funktionalitäten sind im Package `daga.knowledgebase.accessmanagement.requests` zusammengefasst.

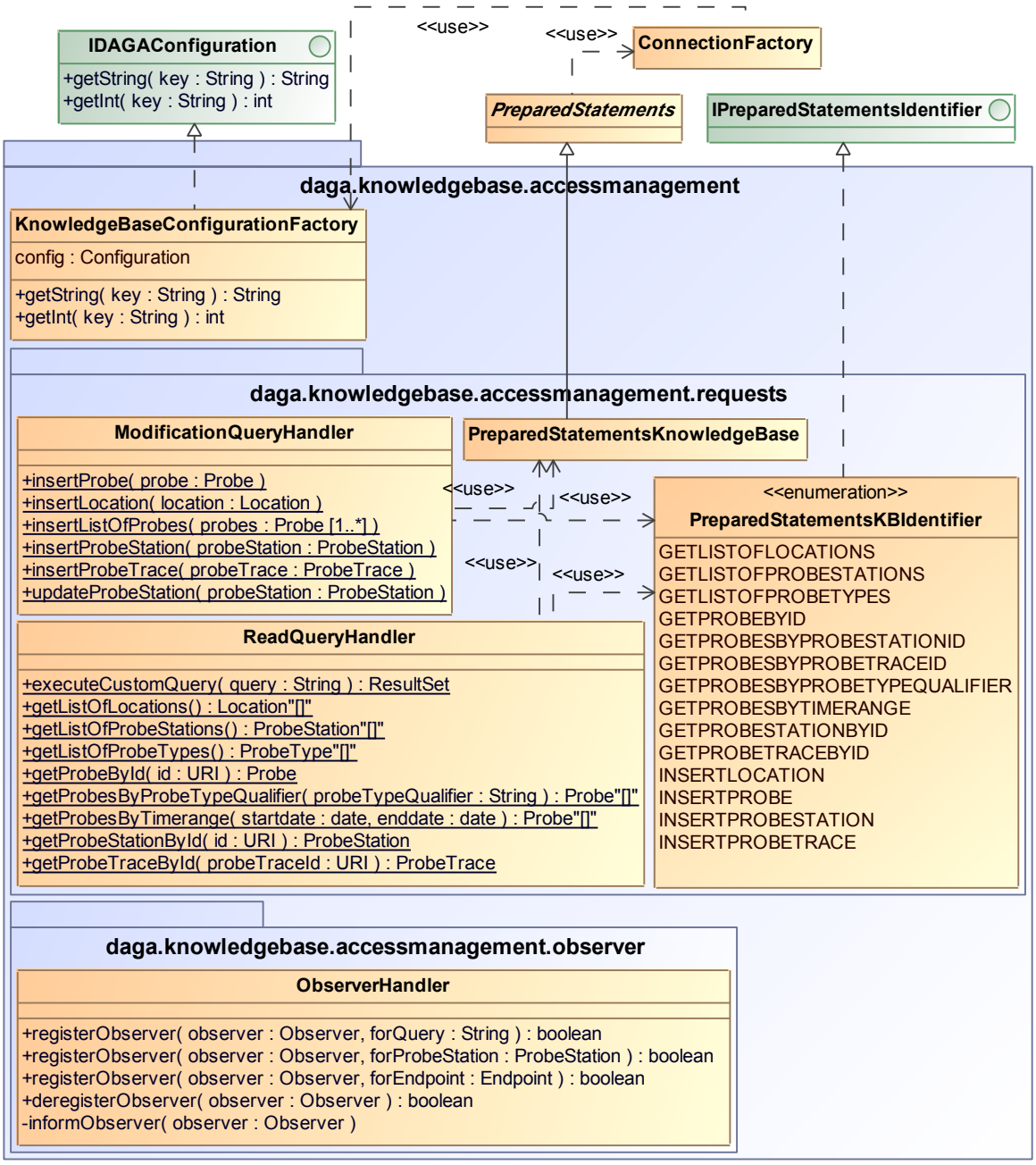


Abbildung 6.5: Klassendiagramm des Subsystems „Zugriffsmanagement“ der Komponente „Knowledge-Base“

Das Package `daga.knowledgebase.accessmanagement.observer` umfasst die Funktionalitäten für die Verwaltung von Consumern, die im Rahmen des **Observer-Managements**

[Fre+04] bei Veränderungen im Datenbestand benachrichtigt werden wollen. Es gibt verschiedene Methoden, über die sich ein Consumer als Observer registrieren kann, damit der Consumer die Benachrichtigung an verschiedene Bedingungen knüpfen kann, beispielsweise wenn ein bestimmtes Query eine Ergebnismenge hat oder wenn ein neuer Eintrag für eine Probe-Station oder einen Endpoint eingeht. Sobald eine Änderung auftritt, für die sich ein Observer registriert hat, wird er darüber benachrichtigt (`informObserver()`) und kann daraufhin die üblichen Abfrage-Mechanismen verwenden.

Im oberen Teil des Klassendiagramms, außerhalb der Package-Struktur, sind die Klassen und Interfaces zu sehen, die im Package `daga.common.*`, also im Artefakt „Allgemeine Elemente“, enthalten sind. Anhand der Use-Beziehung wird verdeutlicht, wofür diese Elemente verwendet werden.

Artefakt „Datenspeicherung“ Das Artefakt kapselt die physische Datenspeicherung der gewonnenen Verfügbarkeits-Informationen und der registrierten Observer in eine Drittanbieter-Datenbank. Die Daten werden in einer Drittanbieter-Datenbank gespeichert, um dem in Kapitel 5.1.3 formulierten Ziel zu folgen, bestehende und bewährte Lösungen einzusetzen. Durch die Kapselung in ein eigenes Artefakt kann die Implementierung der Datenspeicherung ausgetauscht werden, beispielsweise wegen ungenügender Performance, ohne dafür die Consumer anpassen zu müssen, da diese nur mit dem Subsystem „Zugriffsmanagement“ kommunizieren, nicht aber mit der Datenspeicherung. Um Performance-Engpässe und einen Single-Point-of-Failure zu vermeiden, muss die Datenspeicherung als verteilte und replizierte Datenbank implementiert werden. Zudem sollte eine horizontal skalierende Datenbank-Implementierung verwendet werden, da so theoretisch beliebig viele Datensätze gespeichert werden können, eine unbegrenzte Anzahl an Datenbank-Servern vorausgesetzt. Da die Daten in einer Drittanbieter-Datenbank gespeichert werden, werden keine weiteren Details wie UML-Klassendiagramme angegeben. Die Administration der Datenbank erfolgt über ein vom Drittanbieter-System bereitgestelltes, proprietäres Interface direkt auf dem jeweiligen Server. Daher sind im soeben vorgestellten Subsystem „Zugriffsmanagement“, das als Intermediär zwischen dem Artefakt der Datenspeicherung und den anderen Komponenten fungiert, nur *inhaltliche* Administrations-Funktionalitäten modelliert, nicht aber *technische* Funktionalitäten. Eine inhaltliche Funktionalität wäre die Aktualisierung einer Probe-Station, eine technische Funktionalität wäre das Anlegen einer neuen Tabelle.

Komponente „Probe-Station“

Die modellierte Probe-Station hat zwei Aufgabenbereiche: der erste Aufgabenbereich ist die Durchführung des Probing-Prozesses (Kapitel 4.2.2), wobei die Probe-Station als Parallele zu einem Sensor im Monitoring gesehen werden kann, der ebenso wie die Probe-Station eine Entität untersucht und daraus Ergebnisse bzw. Events generiert [ZS05]. Der zweite Aufgabenbereich ist die Übermittlung der gewonnenen Daten an die Knowledge-Base, wobei die Probe-Station als abgewandelter Producer betrachtet werden kann, da sie Informationen veröffentlichen möchte, ohne dabei Kenntnisse über die genaue Anzahl, Typen und Orte der potentiellen Consumer zu haben [BGN02]. Der Producer verlässt sich bei der Veröffentlichung der Daten auf den Information Service bzw. im Rahmen von DAGA auf die Knowledge-Base, dass die veröffentlichten Daten korrekt – im Sinne von Sicherheit, Robustheit und Auswahl der Daten – an die Consumer verteilt werden [BGN02].

Da die Aufgaben der Probe-Station unterschiedlich und vielfältig sind, ist in der Architektur für jeden Aufgabenbereich ein eigenes Subsystem innerhalb der Probe-Station modelliert, um einzelne Komponenten flexibel überarbeiten oder austauschen zu können, ohne die gesamte Probe-Station anpassen zu müssen. Ein mögliches Szenario wäre der Austausch des Selektions-Algorithmus der nächsten auszuführenden Probe: soll für die Auswahl der nächsten auszuführenden Probe eine statistische Methode anstatt eines Entscheidungsbaumes verwendet werden, kann das Subsystem durch eine andere Implementierung ausgetauscht werden, ohne die restlichen Subsysteme (negativ) zu beeinflussen. Außerdem verbessert die Trennung in Subsysteme die Übersichtlichkeit des Prozesses innerhalb einer Probe-Station, da es durch die Aufteilung zu einer Separation-of-Concerns, also einer Trennung der Belange, kommt.

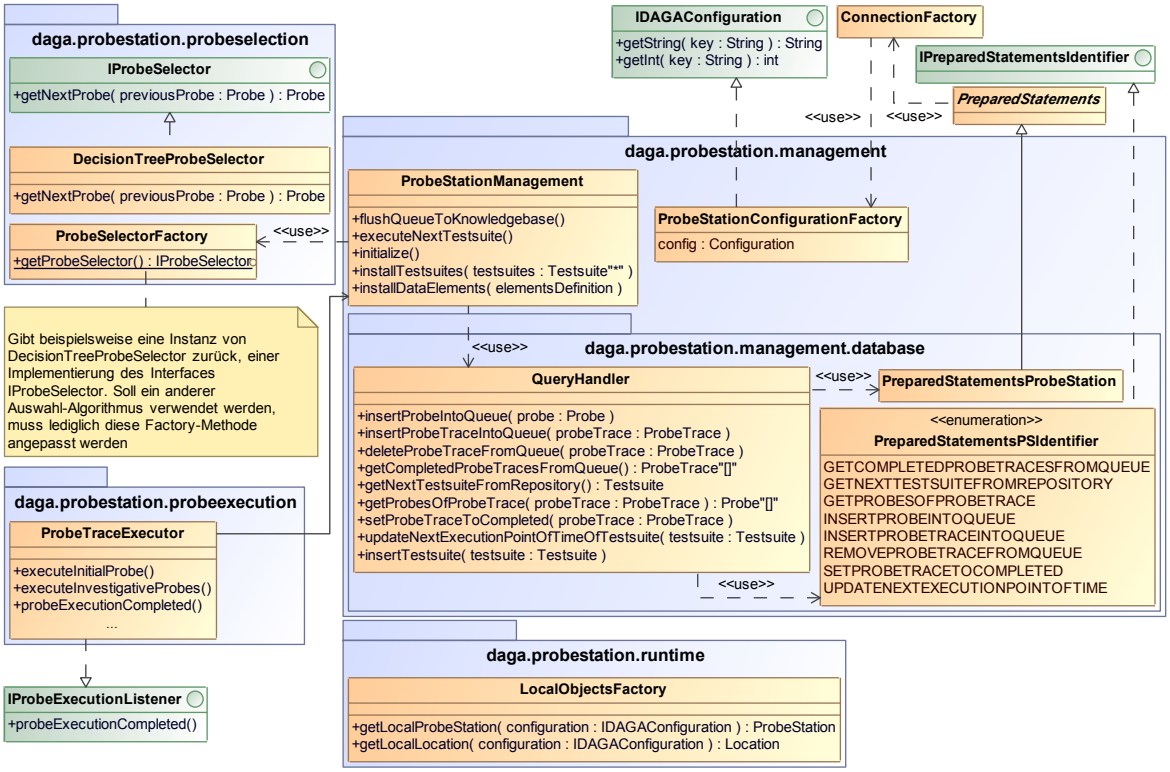


Abbildung 6.6: Klassendiagramm der Komponente „Probe-Station“

Abbildung 6.6 zeigt das Klassendiagramm der Probe-Station, wobei jedes Subsystem einen eigenen Namensraum besitzt. Die Subsysteme werden im Folgenden erläutert.

Subsystem „Probe-Auswahl“ Das Subsystem *Probe-Auswahl* fasst die Funktionalitäten zur Auswahl der nächsten auszuführenden Investigative Probe (Kapitel 4.2.2) anhand der Ergebnisse der vorherigen Probe zusammen und stellt das Auswahl-Ergebnis dem Subsystem „Probe-Station-Management“ zur Verfügung. Da es eine ganze Reihe von möglichen Auswahl-Algorithmen gibt, wie beispielsweise Greedy- oder Substractive-Suchen [Bro+03], regelbasierte Systeme [Wun06] oder Entscheidungsbäume, und die Anzahl sowie Beschaffenheit aller möglichen Algorithmen zum Zeitpunkt der Modellierung nicht bekannt sind, wird bei der Modellierung des Subsystems besonderer Wert auf eine hohe Flexibilität gegenüber den verwendbaren Auswahl-Algorithmen gelegt.

Die für die Probe-Auswahl im Package `daga.probestation.probeselection` (Abbildung 6.6, Seite 85) zusammengefassten Elemente gliedern sich in drei Bereiche: die allgemeine Repräsentation eines Auswahl-Algorithmus (`IProbeSelector`), die konkrete Implementierung eines Auswahl-Algorithmus (`DecisionTreeProbeSelector`) sowie ein Hilfsobjekt, das den zu verwendenden Auswahl-Algorithmus angibt (`ProbeSelectorFactory`). Die allgemeine Repräsentation ist durch ein Interface modelliert, um unabhängig von einer konkreten Implementierung den Objekttyp an verschiedenen Positionen angeben und die Methode `getNextProbe()` aufrufen zu können, beispielsweise in der `ProbeSelectorFactory`. Das Interface kann von beliebigen Klassen implementiert werden, wobei der jeweiligen Klasse überlassen ist, *wie* die Methode `getNextProbe()` umgesetzt ist. Durch die Verwendung der `ProbeSelectorFactory` kann der zu verwendende Auswahl-Algorithmus ausgetauscht werden, ohne den restlichen Programmcode anpassen zu müssen: überall, wo der Auswahl-Algorithmus verwendet wird, wird die statische Methode `getProbeSelector()` aufgerufen, die die aktuell zu verwendende Instanz eines Auswahl-Algorithmus zurückgibt. Code-Listing 6.5 zeigt einen beispielhaften Aufruf im Java-Code.

```
Probe nextProbe = ProbeSelectorFactory.getProbeSelector().
  getNextProbe(previousProbe);
```

Listing 6.5: Beispielhafte Verwendung der `ProbeSelectorFactory`-Klasse

Für die Verwendung eines anderen Auswahl-Algorithmus sind durch die verwendete Klassenstruktur nur zwei Schritte notwendig: zunächst muss die Klasse, die den alternativen Auswahl-Algorithmus implementiert, beispielsweise `RuleBasedProbeSelector`, dem Package `daga.probestation.probeselection` hinzugefügt werden. Anschließend wird die Methode `getProbeSelector()` dahingehend angepasst, dass nicht mehr eine Instanz der Klasse `DecisionTreeProbeSelector`, sondern der Klasse `RuleBasedProbeSelector` zurückgegeben wird. Durch dieses Vorgehen kann der Auswahl-Algorithmus ausgetauscht werden, ohne dafür andere Subsysteme der Probe-Station anpassen zu müssen.

Neben der eingangs geforderten Flexibilität gegenüber der Anzahl und Beschaffenheit der verwendbaren Auswahl-Algorithmen ermöglicht die beschriebene Modellierung zudem die in [BRM01] genannte Architektur für den gesamten Selektionsprozess der Probes.

Subsystem „Probe-Station–Management“ Das Subsystem „Probe-Station–Management“ bildet das Kernelement der Probe-Station und ist für die Kommunikation mit den anderen Komponenten des DAGA-Systems sowie die Orchestrierung der Subsysteme der Probe-Station verantwortlich. Die zentrale Rolle des Subsystems ist auch im UML-Klassendiagramm in Abbildung 6.6 erkennbar: Die Klasse `ProbeStationManagement` verwendet Klassen aus allen anderen Subsystemen bzw. Namensräumen. Die Prozesse, die innerhalb der Probe-Station ablaufen, werden im Interaktionsmodell in Kapitel 6.3 detailliert erläutert.

Artefakt „Ergebnis-Queue/Testsuites“ Das Artefakt wird vom Subsystem „Probe-Station–Management“ verwendet und modelliert eine Drittanbieter-Datenbank. Es speichert die Queue der an die Knowledge-Base zu übertragenden abgeschlossenen Probe-Traces und enthält die Testsuites, die die Probe-Station ausführen soll.

Es wird eine Queue für die Probe-Traces eingesetzt, um den Netzverkehr zwischen den DAGA-Komponenten zu reduzieren: die Ergebnisse einer Testsuite in Form eines Probe-Trace (Kapitel 6.2.1) werden nicht unmittelbar nach der Fertigstellung, sondern in festgelegten

Zeitabständen an die Knowledge-Base übertragen. So können mehrere abgeschlossene Probe-Traces in eine Nachricht gebündelt und übertragen werden, und es werden insgesamt weniger Nachrichten versandt. Die Payload der Nachrichten bleibt dabei zwar gleich, es fällt aber weniger Aufwand für die Generierung, Adressierung und Verteilung der Nachrichten an.

Subsystem „Probe-Trace–Ausführung“ Das Subsystem wird für die Ausführung eines einzelnen Probe-Trace verwendet. Die Klasse `ProbeTraceExecutor`, die die dafür notwendigen Funktionalitäten enthält, befindet sich im Package `daga.probestation.probeexecution` und wird vom Subsystem „Probe-Station–Management“ verwendet. Die Klasse interagiert bei der Ausführung eines Probe-Trace mit den zu überprüfenden Ressourcen und implementiert das im Artefakt „Allgemeine Elemente“ enthaltene Interface `IProbeExecutionListener`, um bei der Beendigung einer Probe-Ausführung benachrichtigt werden zu können. Diese Modellierung ermöglicht es, auch asynchrone Probe-Ausführungen verwenden zu können. Der genaue Ablauf und die Verwendung der Klassen und des Interface werden im Interaktionsmodell in Kapitel 6.3 beschrieben.

Komponente „Administration“

Die Komponente stellt die Funktionalitäten für die *inhaltliche* Administration des DAGA-Systems zur Verfügung. Dazu gehören die Erzeugung neuer Testsuites und deren Installation auf ausgewählten Probe-Stations, die Erzeugung neuer und die Bearbeitung bestehender Probe-Stations, die Erstellung von zusätzlichen aggregierten Untermengen des Datenbestandes der Knowledge-Base für eine effizientere Verwendung, die Auswertung der Nutzung einzelner DAGA-Komponenten und die Bearbeitung von Autorisierungseinstellungen.

Auch die inhaltliche Administration der Probe-Stations ist in diese Komponente ausgelagert, um beliebig viele Probe-Stations zeitgleich administrieren zu können und dadurch eine schnellere und effizientere Administration sowie eine höhere Einheitlichkeit zu erreichen. Beispielsweise kann bei allen Probe-Stations der Auswahl-Algorithmus für die nächste auszuführende Probe zeitgleich ausgetauscht werden, indem die Komponente ein Installationspaket mit den Änderungen zeitgleich an alle Probe-Stations sendet.

Die *technische* Administration, die beispielsweise das Anlegen neuer Datenbank-Benutzer oder die Anpassung einer Datenbank-Installation an Veränderungen im Datenmodell umfasst, wird aus zwei Gründen explizit aus dem Aufgabenbereich der Komponente ausgeschlossen: zum einen fordert eines der für die Entwicklung von DAGA formulierten Ziele die Verwendung von ausgereiften Software-Lösungen (Kapitel 5.1.3), weshalb davon ausgegangen werden kann, dass die eingesetzten Lösungen proprietäre Administrations-Schnittstellen anbieten, über die die jeweilige technische Administration durchgeführt werden kann. Zum anderen wäre für die Verwendung der Ports für die technische Administration umfangreiches Detailwissen über die verwendeten Technologien innerhalb einer Komponente bzw. eines Artefakts notwendig, was der Grundidee einer SOA widerspricht. Bietet der Administrations-Port der Komponente beispielsweise die Funktionalität, einen Fremdschlüssel in der Datenbank anzulegen, die verwendete Drittanbieter-Datenbank aber keine Fremdschlüssel unterstützt, kann der Administrations-Port ohne dieses Detailwissen nicht korrekt verwendet werden.

Die Komponente ist in die zwei Subsysteme „Administrations-Management“ und „Benutzer-Interface“, untergliedert, um Darstellung und Funktionalität zu trennen und die Darstellung flexibel austauschen oder mehrere Darstellungsformen parallel nutzen zu können, beispiels-

weise als GUI-Applikation, als Web-Interface oder als Integration in eine bestehende Anwendung.

6.3 Interaktionsmodell

In diesem Kapitel wird das Interaktionsmodell für DAGA entwickelt, das die „spezifischen Muster der Zusammenarbeit des Nachrichtenaustauschs zwischen Objekten zur Erledigung einer bestimmten Aufgabe“ [Hen10b] beschreibt. Die dabei verwendeten Objekte wurden im Kapitel 6.2 im Rahmen des statischen Modells eingeführt.

Das Interaktionsmodell in diesem Kapitel beschreibt nur diejenigen Abläufe und Interaktionen, die eine zentrale Rolle in DAGA spielen und für die Erfüllung der analysierten Anforderungen (Kapitel 3) von hoher Bedeutung sind, beispielsweise die Zusammenarbeit der DAGA-Komponenten beim Hinzufügen neuer Ressourcen oder das interne Verhalten einer Probe-Station während des Probing-Prozesses. Allgemeine Abläufe, wie die Registrierung und Benachrichtigung von Observern, werden nicht detailliert modelliert, da sie wegen ihrer sehr hohen Bekanntheit vorausgesetzt werden können.

Im ersten Teil des Kapitels werden Interaktionen auf Komponenten-Ebene, im zweiten Teil Komponenten-interne Interaktionen beschrieben. Die Modellierung erfolgt mittels UML-Sequenzdiagrammen, die die „Modellierung von festen Reihenfolgen und zeitlichen und logischen Ablaufbedingungen“ [RQZ07] ermöglichen.

6.3.1 Zusammenarbeit der Komponenten in DAGA

Administration des DAGA-Systems

Abbildung 6.7 (Seite 89) zeigt die bei der Administration des DAGA-Systems beteiligten Komponenten und auszuführenden Aktionen, die in zwei Bereiche untergliedert werden können: das *Hinzufügen neuer zu überwachender Ressourcen* (Aktion *1* bis *9*) und *allgemeine Administrationstätigkeiten* (Aktion *10* bis *17*).

Beim **Hinzufügen neuer zu überwachender Ressourcen** werden fünf bzw. neun Schritte ausgeführt, je nachdem, ob die neu hinzugefügten Ressourcen weitere Probe-Typen oder Anpassungen an den bestehenden Objekten erfordern. Wie in Anforderung *UC7.1* vorgegeben, wird die Liste der zu überwachenden Ressourcen aus einem im Grid eingesetzten Directory Service abgerufen (*1*) und kein proprietäres Ressourcen-Discovery durchgeführt. Nach dem Erhalt der aktuellen Ressourcenlandschaft (*2*) ist es möglich, dass die allgemeinen Elemente im Package `daga.common.elements.*` angepasst werden müssen, beispielsweise weil ein neues Attribut oder ein neuer Probe-Typ hinzugenommen werden muss, da die bestehenden Elemente nicht ausreichen, um die in der Liste enthaltenen Ressourcen zu überprüfen. Nachdem die entsprechenden Elemente angepasst wurden (*3*), werden sie für die Installation auf den DAGA-Komponenten vorbereitet (*4*), beispielsweise indem sie in ein Installations-Paket in Form eines jar-Archivs komprimiert werden (Kapitel 7.4.3). Dieses Installations-Paket kann dann die neuen bzw. angepassten Elemente auf den Probe-Stations (*5*) und der Knowledge-Base (*6*) einrichten. Nach dieser optionalen Anpassung des DAGA-Systems wird der eigentliche Prozess fortgesetzt. Für die Liste der neuen zu überwachenden Ressourcen werden die auszuführenden Testsuites generiert (*7*). In diesem Schritt wird jede neue Ressource untersucht und entsprechend ihrem Typ und angegebenen Endpoint eine passende Testsuite generiert. Diese Testsuites werden dann auf alle oder nur eine Auswahl von Probe-Stations

übertragen (*8*), die diese Testsuites anschließend ausführen (*9*). Die Ausführung einer Testsuite ist in Kapitel 6.3.2 detailliert erläutert. Das Sequenzdiagramm macht deutlich, dass entsprechend der Empfehlung aus [BSV03] eine Trennung zwischen Crawling und Probing stattfindet.

Die **allgemeinen Administrationstätigkeiten** umfassen das Anlegen (*10*) und Einrichten (*11*, *12*) neuer Probe-Stations, deren spätere Administration (*14*) sowie die Erstellung allgemeiner Auswertungen (*16*, *17*), beispielsweise welche Consumer welche Daten angefragt haben.

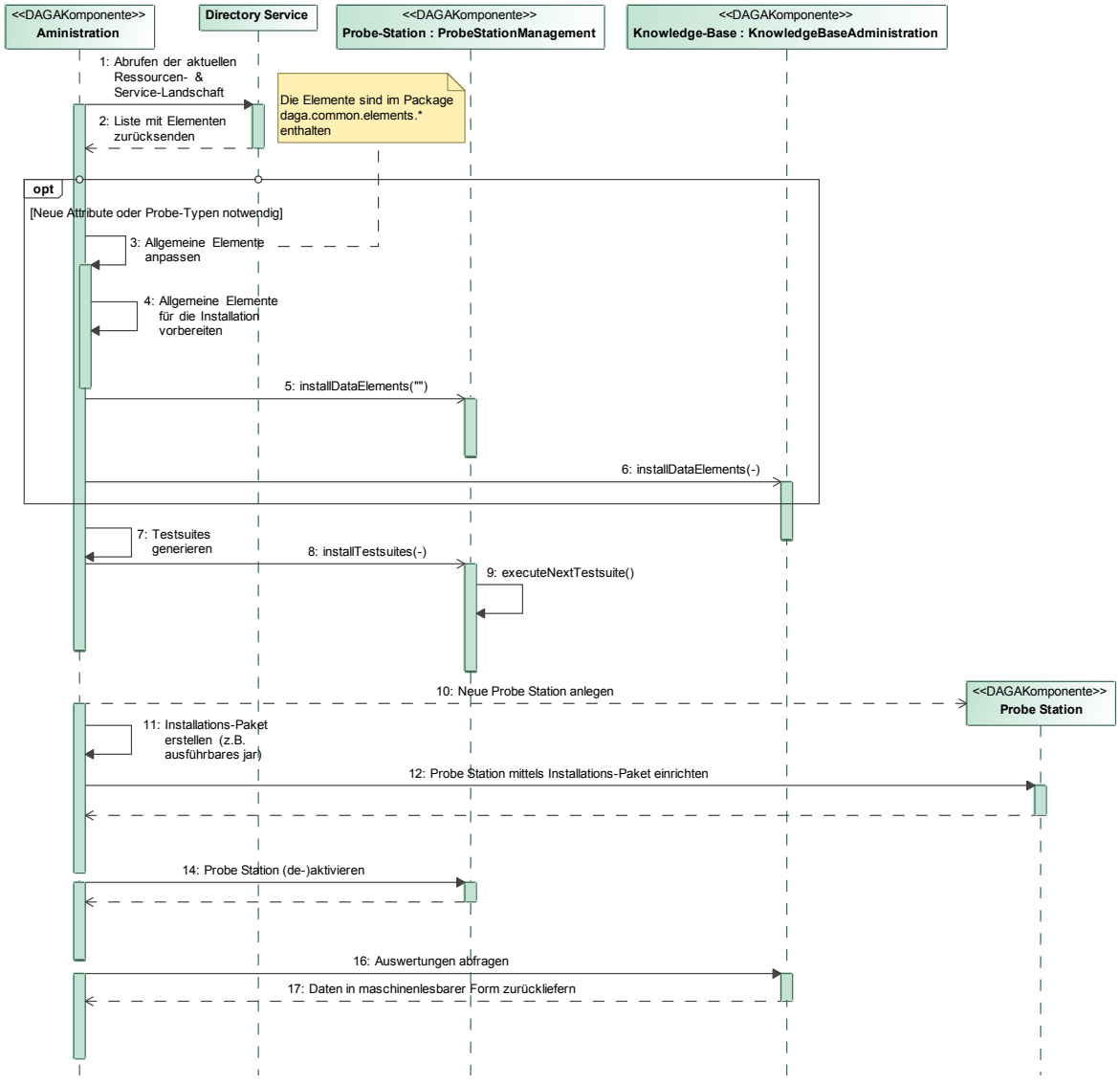


Abbildung 6.7: Modellierung der Administration des DAGA-Systems

Gewinnung und Verwendung von Verfügbarkeits-Informationen in DAGA

Abbildung 6.8 (Seite 90) zeigt die beteiligten Komponenten sowie die ausgetauschten Informationen bei der Gewinnung und Verwendung von Verfügbarkeits-Informationen im DAGA-System. Es ist ein sehr allgemeiner Überblick, da der Großteil der Aktivitäten während des

im nächsten Abschnitt modellierten Probing-Prozesses in der Probe-Station ausgeführt wird. Das Ergebnis der Ausführung einer Testsuite in Form eines Probe-Trace (Kapitel 6.2.1) wird vor der Übertragung an die Knowledge-Base in einer Queue gespeichert (*1*), um die in Kapitel 6.2.2 beschriebene Verringerung der Systemlast zu erreichen. Die in der Queue gespeicherten Probe-Traces samt der enthaltenen Probes werden in einem in einer Konfigurationsdatei hinterlegten Zeitintervall an die Knowledge-Base übertragen (*3*) und bei einer erfolgreichen Rückmeldung (*4*) aus der Queue gelöscht (*5*). Die Rückmeldung über die Speicherung einer Probe-Trace in der Knowledge-Base erhöht zwar die Netzlast, ist aber notwendig, da andernfalls Messergebnisse verloren gehen könnten, wenn der Probe-Trace nicht korrekt gespeichert wurde und dennoch aus der Queue entfernt wird.

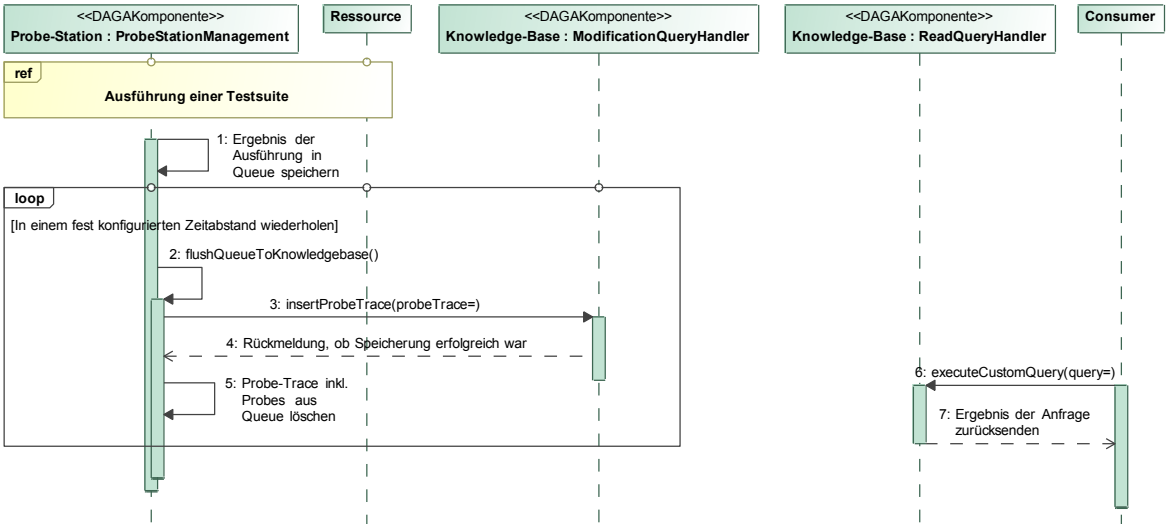


Abbildung 6.8: Sequenzdiagramm, das die bei der Gewinnung und Verwendung von Verfügbarkeits-Informationen in DAGA beteiligten Komponenten darstellt

Wie im statischen Modell in Kapitel 6.2.1 erläutert, verwendet das Subsystem „Zugriffsmanagement“ der Knowledge-Base zwei Klassen für schreibende und lesende Anfragen: das Einfügen von Probe-Traces aus der Queue der Probe-Station erfolgt über den für schreibende Zugriffe zuständigen `ModificationQueryHandler`, das Abfragen von Verfügbarkeits-Informationen (*6*, *7*) über den `ReadQueryHandler`. Die im Sequenzdiagramm 6.8 verwendete Methode `executeCustomQuery()` steht stellvertretend für alle Methoden, die von der Klasse `ReadQueryHandler` für die Abfrage von Verfügbarkeits-Informationen angeboten werden.

Interaktion von Observern und der Knowledge-Base

Wie in der Einführung des Interaktionsmodells begründet, wird für den Observing-Prozess in DAGA kein gesondertes Sequenzdiagramm erstellt, da hier keine DAGA-spezifischen Verhaltensweisen erläutert werden müssen. Stattdessen sei auf ein für diesen Anwendungsfall entwickeltes XML-basiertes Kommunikationsprotokoll verwiesen, das in [SGQ01] vorgestellt wird. Das Protokoll, das zu dem von der *Grid Forum Performance Working Group* definierten Event-Protokoll kompatibel ist [Smi01], beschreibt XML-Schemata, Nachrichten-Semantik und Abläufe, die den Einsatz von Observern in Grids stark vereinfachen.

6.3.2 Prozesse innerhalb einer Probe-Station

Ausführung einer Testsuite

Da der Betrieb der Probe-Station zu den komplexesten Abläufen in DAGA zählt, wird er in diesem Unterkapitel besonders herausgestellt und detailliert modelliert.

Abbildung A.3 (Seite 129) zeigt in einem Sequenzdiagramm die an der Ausführung eines Probe-Trace beteiligten Subsysteme der Probe-Station, welche Informationen sie austauschen und in welcher Reihenfolge die Ausführung stattfindet.

Zu Beginn wird die als nächstes auszuführende Testsuite entsprechend ihrer in Kapitel 6.2.1 angegebenen Eigenschaften aus der Datenbank ausgewählt (*1*). Die ausgewählte Testsuite wird dann an die neu erstellte Instanz des `ProbeTraceExecutor` übergeben (*2*), der die Testsuite in vier Phasen ausführt: die Vorbereitung der Probe-Trace Ausführung (*3*), die Ausführung der Initial Probe (*4*), die von dem Ergebnis der Initial Probe abhängige Ausführung weiterer Investigative Probes (*14*) und der Abschluss des Probe-Trace (*23*).

Die Vorbereitung (*3*) umfasst beispielsweise die Erzeugung eines `ProbeTrace`-Objekts und dessen Speicherung in der Datenbank. Daran anschließend wird die Initial Probe entsprechend der übergebenen Testsuite ausgewählt und in der Methode `executeInitialProbe()` ausgeführt (*4*). In dieser Methode wird eine neue Instanz der auszuführenden Initial Probe erzeugt (*5*), initialisiert (*6*) und ausgeführt (*10*), wobei sich die Initial Probe bei der Ressource authentifiziert und autorisiert. Vor der Ausführung der Initial Probe muss jedoch noch der `IProbeExecutionListener` gesetzt werden (*8*): dieses Interface wird vom `ProbeTraceExecutor` implementiert (Abbildung A.2, Seite 128), damit dieser von einer Initial Probe bei deren Beendigung benachrichtigt werden kann. Dieses Vorgehen ist notwendig, um die Ausführung asynchroner Aufrufe innerhalb einer Initial Probe zu unterstützen. Da hier für die Implementierung verschiedener Probe-Typen unterschiedliche Frameworks zum Einsatz kommen, wie beispielsweise das im Prototypen verwendete GoK Toolkit (Kapitel 7.4.4), das die Anfragen an die zu überprüfende Ressource in Form von asynchronen Aufrufen stellt, muss diese Asynchronität unterstützt werden. Wäre der `execute()`-Aufruf synchron und die Initial Probe Implementierung asynchron, würde das Ergebnis der Initial Probe aufgrund der synchronen Weiterausführung festgelegt, ohne dass das Ergebnis von der Ressource zurückgesandt wurde. Nach der Überprüfung der Ressource wird der `ProbeTraceExecutor` über die Beendigung der Initial Probe informiert (*13*) und die Ausführung der nachfolgenden Investigative Probes begonnen (*14*). In dieser Methode wird in einem ersten Schritt die zu verwendende Implementierung der Probe-Auswahl von der `ProbeSelectorFactory` abgefragt (*15*, *16*). Mittels dieser Auswahl-Implementierung kann nun überprüft werden, welche Investigative Probe als nächstes auszuführen ist (*17*). Wurde eine Investigative Probe ausgewählt, weil die Initial Probe fehlgeschlagen ist, wird mit der Ausführung der Schleife begonnen. Die Schleife erzeugt eine Instanz der in Schritt *17* bzw. *23* ausgewählten Investigative Probe und führt diese aus (*19*), wobei die aktuelle Investigative Probe die zu überprüfende Ressource analysiert. Nach der Ausführung der Investigative Probe wird erneut eine nachfolgende Probe gewählt (*23*) und die Schleife durchlaufen, falls dabei eine Probe gewählt wurde. Nach der (möglichen) Ausführung der Schleife wird der Probe-Trace abgeschlossen und für die Übertragung zur Knowledge-Base in die Queue gespeichert (*24*).

Die Auswahl der nächsten auszuführenden Testsuite erfolgt periodisch anhand eines fest konfigurierten Zeitintervalls. Die Verbesserungsmöglichkeiten dieses Vorgehens werden in Kapitel 8.2.1 erörtert.

Die Rolle des Grid-Schedulers

Wie in Kapitel 2.2 erläutert, spielen Scheduler bei der Ausführung von Grid-Jobs eine zentrale Rolle. Da die Testsuites entsprechend der Dienst-Orientierung von DAGA (Kapitel 5.2.1) aus Sicht eines Grid-Jobs ausgeführt werden, muss auch hier die Rolle der Scheduler bei der Ausführung einer Testsuite untersucht werden. Ob ein (Grid-)Scheduler bei der Ausführung einer Testsuite verwendet wird, hängt maßgeblich von der Testsuite-Definition und der daraus resultierenden Initial Probe ab.

Überprüft die Testsuite einen Endpoint mittels einer WSDL-Probe, werden keine Scheduler verwendet, da Anfragen an einen Endpoint, d.h. an den durch den Endpoint vertretenen Dienst, direkt gestellt und vom angefragten Dienst auch direkt bearbeitet werden, ohne die Anfrage zuvor an einen Scheduler zu senden. Wird also ein Endpoint mit der Testsuite verknüpft, ist kein Scheduler involviert.

Wird eine Testsuite nicht mit einem Endpoint verknüpft, ein durch die Optionalität der Assoziation zwischen beiden Klassen ebenfalls mögliches Szenario (Kapitel 6.2.1), hängt die Entscheidung vom jeweiligen Grid-Administrator ab, ob ein Scheduler verwendet werden soll oder nicht: soll ein (Grid-)Scheduler verwendet werden, muss dafür die Initial Probe lediglich dahingehend angepasst werden, dass die in der Testsuite-Definition festgelegte Analyse nicht direkt an die Ressource, sondern zunächst an einen zuständigen Scheduler gesendet wird. Soll kein Scheduler verwendet werden, wie es in der prototypischen Implementierung (Kapitel 7) realisiert ist, kann die Analyse direkt an die jeweilige Ressource gesandt werden.

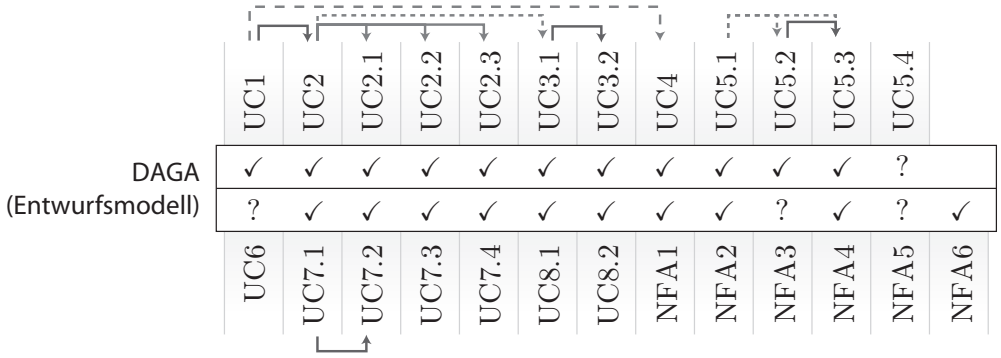
Da die Verfügbarkeits-Informationen aus Sicht eines Grid-Jobs gewonnen werden sollen (Kapitel 5.2.1) und die Verwendung eines (Grid-)Schedulers zu eben jener Grid-Job–Ausführung dazugehört, wäre die Verwendung eines Schedulers während des Probing-Prozesses auf den ersten Blick sinnvoll. Dennoch sollte im Allgemeinen davon abgesehen werden, da andernfalls die Aktualität der Ergebnisse nicht garantiert werden kann, wenn beispielsweise die Probe vom Scheduler erst längere Zeit später ausgeführt wird. Zudem sind die Last, die eine Initial Probe erzeugt, sowie die Anzahl der ausgeführten Probes so minimal, dass sie vernachlässigt werden können und die Probes nicht in einen Scheduling-Prozess eingegliedert werden müssen, sondern direkt zwischen zwei Grid-Jobs ausgeführt werden können. Eine Arbeit, die sich weiterführend mit der Rolle von Schedulern auseinandersetzt, ist [Sfi+08].

Für die Überprüfung des Scheduling-Prozesses können weitere Probe-Typen sowie eine Testsuite-Definition erstellt werden.

6.4 Evaluation des Entwurfsmodells

Entsprechend den in Kapitel 3.2.5 definierten Prozessschritten des Evaluations-Frameworks folgt nach Schritt 1, also „der Beschreibung der grundlegenden Konzepte, Methoden und Modelle“, der bei der Entwicklung des Entwurfsmodells in diesem Kapitel detailliert durchgeführt wurde, jetzt Schritt 2 mit der „Evaluation der beschriebenen, grundlegenden Konzepte“. Schritt 3 und 4 werden in Kapitel 7 bei der Entwicklung einer prototypischen Implementierung durchgeführt.

Bei der Evaluation der grundlegenden Konzepte von DAGA ergibt sich folgendes Ergebnis:



Evaluation Subsystem „A – Gewinnung von Verfügbarkeits–Informationen“ Das Entwurfsmodell von DAGA ermöglicht es bzw. sieht es durch die Initial Probes und insbesondere die WSDL-Probe explizit vor, das Dienst-Interface der zu überprüfenden Ressource zu verwenden (*UC1*/✓). Dabei kann eine Nicht-Verfügbarkeit erkannt (*UC2*/✓) und mittels der Flexibilität, die die Testsuite-Definitionen ermöglichen, als technische (*UC2.1*/✓) oder organisatorische (*UC2.2*/✓) Nicht-Verfügbarkeit erkannt werden. Eine technische Nicht-Verfügbarkeit wird erkannt, wenn beispielsweise das Dienst-Interface nicht antwortet. Eine organisatorische Nicht-Verfügbarkeit wird erkannt, da die flexible Modellierung der Testsuite-Definition und der Erweiterbarkeit der Probes im Package `daga.common.elements.probes.*` es erlauben, Credentials, beispielsweise in Form von Zertifikaten oder Benutzername- und Passwort-Kombinationen, bei der Überprüfung einer Ressource einzusetzen. Auch der Grad einer Nicht-Verfügbarkeit (*UC2.3*/✓) ist durch die Flexibilität der Testsuite-Definition und der Modellierung der WSDL-Probe erkennbar, da hier beispielsweise erkannt werden kann, ob der Rückgabewert einer Dienst-Anfrage korrekt ist oder nicht. Daraus kann ermittelt werden, ob die Ressource im Ganzen nicht verfügbar ist oder ob sie nur (teilweise) nicht richtig funktioniert. Zudem können beliebig viele Testsuites auf ein und demselben Endpoint bzw. ein und derselben Ressource ausgeführt und damit graduelle Unterschiede in der Ausführung erkannt werden. Aus den verschiedenen Probe-Typen und den darin implizit enthaltenen Lösungsvorschlägen [BRM01] können die Ursachen einer Nicht-Verfügbarkeit abgelesen werden (*UC3.1*/✓). Durch die Speicherung und Zusammenfassung der Ergebnisse einzelner Probes in Probe-Traces ist es ohne weiteres möglich, eine chronologische Abfolge von Probes und damit Teilursachen einer Nicht-Verfügbarkeit zu erstellen (*UC3.2*/✓). Wie bereits erwähnt können verschiedene Credentials in die Definition einer Testsuite eingebunden werden, wodurch es das Entwurfsmodell von DAGA ermöglicht, die Sicherheitsmechanismen einer Ressource zu überprüfen (*UC4*/✓). Wird eine (Initial) Probe mit zwei unterschiedlichen Credentials ausgeführt, wobei die Credentials jeweils aus der zugeordneten Testsuite ausgelesen werden können, und eine der beiden Ausführungen fehlschlägt, können daraus Rückschlüsse auf die korrekte Funktionsweise der Sicherheitsmechanismen gezogen werden.

Evaluation Subsystem „B – Verwendung von Verfügbarkeits-Informationen“ Durch die umfassende Vorbereitung verschiedener Anfrage-Methoden in der Knowledge-Base bzw. im `ReadQueryHandler` der Knowledge-Base können sowohl Anfragen in einer Anfragesprache formuliert (*UC5.1*/✓) als auch verschiedenste Pull-Anfragen gestellt werden (*UC5.2*/✓). Der modellierte `ObserverHandler` in der Knowledge-Base bietet alle notwendigen Funktionalitäten für die Verwendung von Push-Anfragen (*UC5.3*/✓), also die Verwendung von Observern, an. Ob eine Standard-UI angeboten wird, die einem Anwender die gewonne-

nen Verfügbarkeits-Informationen bereitstellt, hängt von einer möglichen Implementierung ab (*UC5.4*/?).

Evaluation Subsystem „C – System-Administration“ Gemäß dem technischen Ziel bei der Entwicklung von DAGA (Kapitel 5.1.3), möglichst bestehende und ausgereifte Software-Komponenten zu verwenden, die auch eine Administration bereitstellen, ist die Administration der Daten zwar theoretisch möglich, wird im Rahmen der Evaluation aber als von einer Implementierung abhängend bewertet (*UC6*/?), da die Administration über die proprietären Schnittstellen recht aufwendig und durch die unmittelbare Arbeit an den Rohdaten fehleranfällig sein kann. Neue zu überwachende Ressourcen werden wie im Interaktionsmodell beschrieben von einem im Grid eingesetzten Directory Service abgerufen (*UC7.1*/✓) und es wird kein proprietäres Discovery betrieben. Die Abfrage der Directory Services und die bei der Evaluation des Subsystems A erläuterte Verwendbarkeit unterschiedlicher Credentials ermöglicht das Hinzufügen neuer VOs (*UC7.2*/✓), DAGA kann also interorganisational eingesetzt werden. Durch die Sammlung aller relevanten Objekte im Artefakt „Allgemeine Elemente“, konkreter im Package `daga.common.*`, und die Erweiterbarkeit um neue Probe-Typen und Testsuites ist es möglich, neue Fehler- (*UC7.3*/✓) und Ressourcentypen (*UC7.4*/✓) zu DAGA hinzuzufügen, ohne umfangreiche Änderungen am System vornehmen zu müssen. Die hier genannte flexible Modellierung hat zusätzliche weitreichende Konsequenzen: so ist es beispielsweise möglich, DAGA auf mehreren Grid-Middlewares zeitgleich einzusetzen, da für jede Grid-Middleware eigene Initial bzw. Investigative Probes verwendet werden können. Ebenso ist es möglich, neben Diensten mit einem ausgewiesenen Endpoint andere Funktionalitäten, wie beispielsweise Dateiübertragungen mittels GridFTP (siehe Prototyp Kapitel 7), zu analysieren. Die flexible Modellierung der Probes und der Testsuites ermöglichen es außerdem, technische (*UC8.1*/✓) und inhaltliche (*UC8.2*/✓) Selbsttests in DAGA durchzuführen. Da eine Testsuite nur optional einen Endpoint für eine Überprüfung braucht, kann ebenso eine Testsuite erstellt werden, die eine andere Probe-Station analysiert. Eine passende Initial Probe bzw. weitere Investigative Probes können durch die bestehende Klassenhierarchie einfach hinzugefügt werden. Die gleiche Begründung gilt für die inhaltlichen Selbsttests.

Evaluation nicht-funktionaler Anforderungen Bei der Entwicklung des Datenmodells von DAGA wurde das GLUE-Schema in Version 2.0 (Kapitel 6.2.1) erweitert, wodurch ein einheitliches und etabliertes Datenschema für die Beschreibung der Ressourcen in DAGA verwendet wird (*NFA1*/✓). Da es nicht notwendig ist, bestehende Dateien wie Quellcode oder WSDL-Dokumente anzupassen oder zusätzliche Software zu installieren und außerdem nur eine geringe zusätzliche Last durch die Probes erzeugt wird, erzielt DAGA einen extrem niedrigen Intrusiveness-Wert (*NFA2*/✓). An einigen Stellen, beispielsweise der Ausführungshäufigkeit einer Testsuite, kann die Intrusiveness weiter gesenkt werden. Diese Punkte werden in Kapitel 8.2.1 behandelt. Das Entwurfsmodell von DAGA macht keine Aussagen über die Verwendung von Sicherheitsmechanismen bei der Interaktion der verschiedenen Komponenten. Die Einhaltung von Sicherheitsstandards hängt damit von einer möglichen Implementierung ab (*NFA3*/?). Die von DAGA gewonnenen Verfügbarkeits-Informationen haben aus zwei Gründen eine hohe Aktualität (*NFA4*/✓): 1) die Ausführungsintervalle der Testsuites können über das Feld `executeEveryMinutes` optimal an die Eigenschaften des Grids angepasst werden, d.h. der Zeitraum zwischen zwei Ausführungen kann an die Veränderungen in einem Grid individuell angepasst werden, wodurch ausreichend viele Messungen durchgeführt werden können; 2) die gewonnenen Ergebnisse werden direkt an die Knowledge-Base übertragen und sind somit unmittelbar für Consumer verfügbar. Es findet also weder bei der Erhebung noch bei der Verwendung der Daten eine Zeitverzögerung statt, die die Aktualität

der Daten negativ beeinflussen würde. Das Entwurfsmodell macht keine Aussagen zur Verwendung eines Zeitsynchronisationsmechanismus, der somit von einer möglichen Implementierung abhängt (*NFA5*/?). Da die Daten zentral in der Knowledge-Base gespeichert werden, ist weder für die generelle Nutzung der gewonnenen Verfügbarkeits-Informationen noch für ein gesamthaftes Abbild des Grids eine Daten-Migration oder ein Daten-Preprocessing notwendig (*NFA6*/✓).

7 Prototypische Implementierung

In diesem Kapitel wird eine prototypische Implementierung des in Kapitel 6 entwickelten Entwurfsmodells vorgestellt, und es wird beschrieben, welche Technologien eingesetzt werden, warum diese ausgewählt werden und wie der Prototyp konfiguriert und betrieben werden kann. Die Beschreibungen und Beispiele in diesem Kapitel richten sich nach der in [HH10] angegebenen Governance-Checkliste und der bei der Entwicklung des Prototyps verwendeten *integrierten Entwicklungs-Umgebung* (IDE) Eclipse [Foue].

Die Implementierung des Prototyps wird in Kapitel 7.2 mit der Auswahl einer Instanziierung des im Entwurfsmodell verwendeten Architektur-Paradigmas SOA (Kapitel 6.2.2) begonnen, da die Wahl der Instanziierung und deren Konzepte und Technologien maßgeblichen Einfluss auf die weitere Entwicklung des Prototyps hat und daher im ersten Schritt erfolgen muss. Anschließend wird in Kapitel 7.3 die Laufzeitumgebung, bestehend aus den eingesetzten Betriebssystemen, Frameworks und Programmiersprachen, erläutert, in der der Prototyp entwickelt und getestet wird. Nach diesen allgemeinen Vorbereitungen wird in Kapitel 7.4 die Implementierung der einzelnen Komponenten beschrieben und wie diese konfiguriert und verwendet werden können. Die Zusammenhänge zwischen den verschiedenen Konfigurationsdateien und einzelnen Schritten, die beim Deployment und bei möglichen Anpassungen des Prototyps durchzuführen sind, sind in Abbildung B.4 (Seite 132) zusammengefasst. Um DAGA mit den bestehenden Ansätzen aus Kapitel 4 vollständig vergleichen zu können, werden in Kapitel 7.5 die letzten beiden Prozessschritte des Evaluation-Frameworks (Kapitel 3.2.5) durchgeführt.

Alle Dateien der prototypischen Implementierung sind auf der in Anhang C beigefügten CD zu finden.

7.1 Abgrenzung als Prototyp

Bei der vorgestellten Implementierung handelt es sich um einen Prototypen, einer „teilweise, bewusst unvollständigen Implementierung einer Software. Er dient zur Demonstration ausgewählter Eigenschaften des Zielprodukts im praktischen Einsatz, für experimentelle Zwecke und zur Sammlung praktischer Erfahrungen“ [Fre07]. Bei der Entwicklung eines Prototyps findet also eine Konzentration auf besondere Eigenschaften statt, wohingegen andere Eigenschaften, die beispielsweise für die Entwicklung und den Betrieb eines Produktivsystems durchaus von Bedeutung sein können, vernachlässigt werden.

Der vorgestellte Prototyp soll aufzeigen, dass die Verfügbarkeit von Grid-Ressourcen mit Active Probing ermittelt werden kann, dass das Entwurfsmodell realisierbar ist und dass sich mit DAGA eine Lösung darstellt, die die analysierten Anforderungen (Kapitel 3) erfüllen kann. Eigenschaften, die bei der Erreichung der genannten Ziele des Prototyps eine sekundäre Rolle spielen und nur den Betrieb eines Produktivsystems betreffen, wie beispielsweise die Verwendung einer hoch skalierbaren Datenbank-Lösung, werden bei der Technologie-Auswahl und Entwicklung des Prototyps vernachlässigt, an gegebener Stelle aber explizit unter Angabe

von Alternativen erwähnt. Diese Vernachlässigung einzelner Aspekte hat für die Entwicklung des Prototyps keine Bedeutung, da sie keine konzeptionellen Schwachpunkte, sondern lediglich Implementierungsdetails darstellt. Zudem können wegen der Verwendung des SOA-Paradigmas (Kapitel 6.2.2) und der Gliederung der Komponenten in Subsysteme einzelne Teile des Prototyps leicht ausgetauscht werden, weshalb der Prototyp für den Produktiveinsatz nur weiterentwickelt, nicht aber vollständig neu entwickelt werden muss.

Für eine einfachere Referenzierung im Verlauf der Prototyp-Entwicklung werden die genannten Annahmen als *Prototyp-Prämissen* zusammengefasst.

7.2 SOA-Instanziierung

Das dem Entwurfsmodell von DAGA zugrunde liegende Architektur-Paradigma ist eine SOA (Kapitel 6.2.2). Da eine SOA aber nur ein Architekturmodell beschreibt und keine technischen Realisierungen oder Plattformen definiert [HH10], ist für die Entwicklung des Prototyps eine Realisierung bzw. Instanziierung des SOA-Architekturmodells notwendig.

Im Prototyp wird die SOA mittels der Technologien von Web Services instanziiert. Die Arbeitsgruppe des W3C definiert einen Web Service als „Software-Anwendung, welche eindeutig durch einen Uniform Ressource Identifier (URI) beschrieben wird. Die Schnittstellen und Protokolle einer solchen Anwendung können durch XML-Artefakte definiert, beschrieben und gefunden werden. Ein Web Service unterstützt die direkte Interaktion mit anderen Software-Anwendungen durch Verwendung XML-basierter Nachrichten und mittels internetbasierter Protokolle“ [HH10; Aus+02]. Ist eine Anwendung also über ein Computernetz erreichbar und wird mit ihr über eine Kombination von internetbasierten Protokollen wie HTTP oder XML kommuniziert, handelt es sich um einen Web Service [STK02].

Das Einsatzziel von Web Services, „to allow applications to work together over standard Internet protocols, without direct human intervention“ [Pap08], wird mit Hilfe des *Web Service Technology Stacks* erreicht, dessen Funktionsebenen sowie deren jeweilige Implementierungen durch die Protokolle HTTP, XML (SOAP, WSDL) und UDDI in Abbildung 7.1 [STK02; Pap08] dargestellt sind. Durch die Verwendung dieser Protokolle ist der entwickelte Prototyp konform zum *WS-I Basic Profile Version 1.1* [Bal+06].

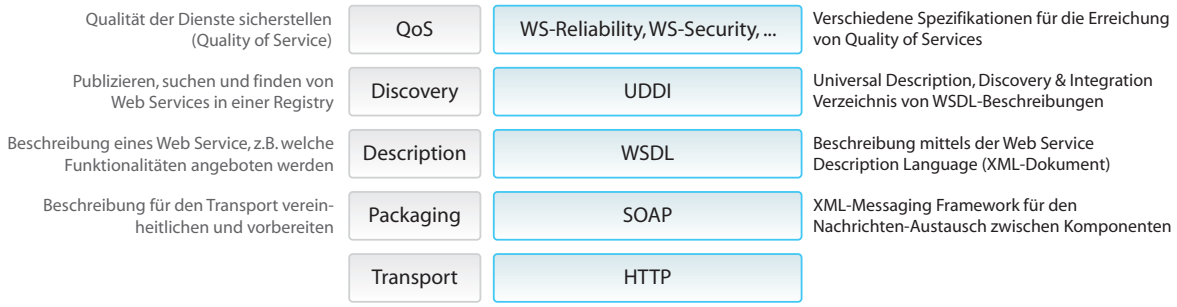


Abbildung 7.1: Elemente des Web Service Technology Stacks

Web Services werden aus zwei Gründen für die Instanziierung des SOA-Paradigmas gewählt.

Der erste Grund ist deren gute Eignung für den Einsatz in Grid-Umgebungen, da Web Services in Grids und insbesondere in der verwendeten Grid-Middleware Globus Toolkit 4 (Kapitel 7.3.1) weit verbreitet sind und dadurch umfangreiche Unterstützung in den wichtigsten Grid-Middlewares existiert. Außerdem ist die Ankopplung an die *Open Grid Service*

Architecture (OGSA) mittels WSDL und SOAP, die beide im Web Service Stack verwendet werden, leicht möglich [ZS05].

Der zweite Grund ist die Existenz vielfältiger, gut dokumentierter und häufig eingesetzter Bibliotheken im Java-Umfeld (Kapitel 7.3.3), die im Rahmen von Web Services eingesetzt werden können, was die Intention eines Prototyps unterstützt (Kapitel 7.1). Hier ist insbesondere die *Java API for XML - Web Services* (JAX-WS) zu erwähnen, die eine Schnittstelle für die Verwendung von XML in Web Services definiert. Die wichtigsten Implementierungen der JAX-WS sind die im GlassFish Metro Projekt enthaltene Referenz-Implementierung [Orac] sowie die Implementierung innerhalb des Apache CXF Projekts [Foua], die beide im Prototypen zum Einsatz kommen.

7.3 Laufzeitumgebung und eingesetzte Technologien

Nach der Festlegung von Web Services als SOA-Instanziierung können nun die für die Entwicklung und Ausführung des Prototyps einzusetzenden Betriebssysteme, Programmiersprachen und Frameworks ausgewählt werden. Die für die Auswahl relevanten Fragestellungen und die aus der Beantwortung dieser Fragestellungen resultierenden Auswahlergebnisse, die in Abbildung 7.2 in Form einer semi-formalen Übersicht unter Verwendung des in Kapitel 6.2.2 vorgestellten Komponentendiagramms dargestellt sind, werden nachfolgend erläutert. Abbildung 7.2 wird auch in Kapitel 7.4 für die Beschreibung der Implementierung der Komponenten verwendet.

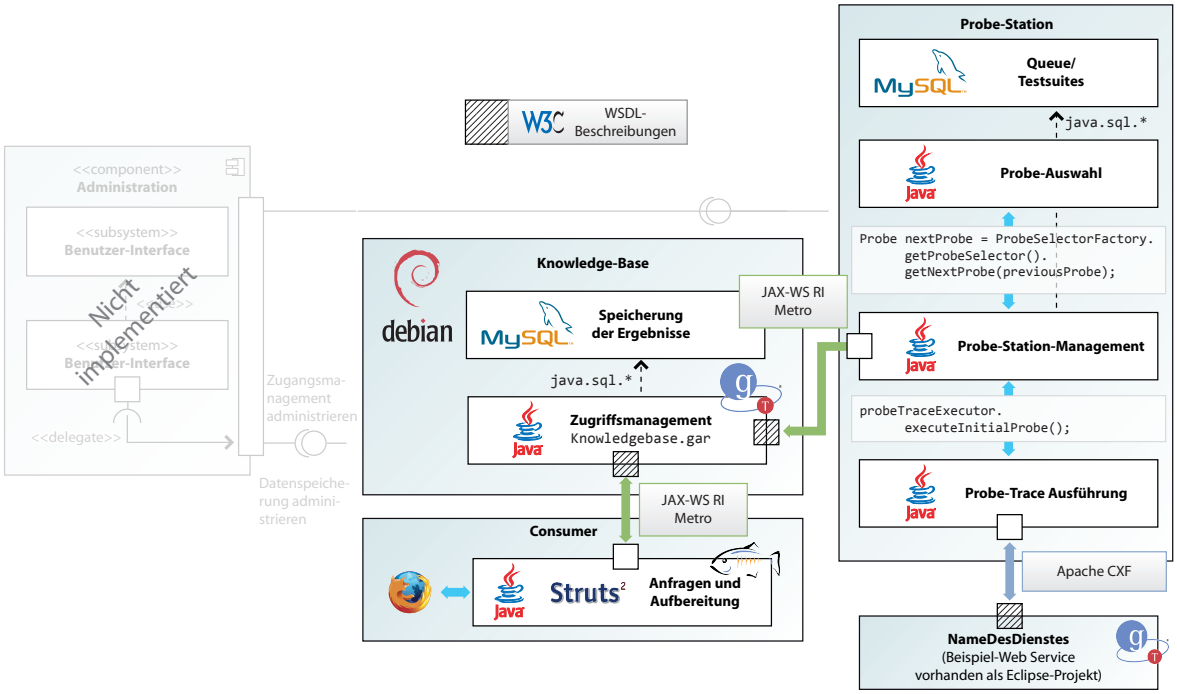


Abbildung 7.2: Übersichtsgrafik der im DAGA-Prototyp eingesetzten Betriebssysteme, Programmiersprachen und Frameworks

7.3.1 Globus Toolkit als Grid-Middleware

Auswahl und Eigenschaften

Um die hohe Heterogenität in Grids zu abstrahieren, werden sogenannte Grid-Middlewares eingesetzt, die auf den einzelnen Ressourcen installiert werden und anschließend eine einheitliche Schnittstelle nach außen anbieten sollen. Aufgrund dieser zentralen Rolle sind Grid-Middlewares, deren häufigste Vertreter das Globus Toolkit, UNICORE und gLite sind, in nahezu allen Grids anzutreffen.

Da DAGA für den Einsatz in Grid-Umgebungen entwickelt wird, muss es mit bestehenden Grid-Middlewares kompatibel sein. Es gilt also, eine Grid-Middleware auszuwählen, die für die Entwicklung und insbesondere den Testlauf des Prototyps verwendet werden kann.

Als Grid-Middleware kommen die genannten Globus Toolkit, UNICORE und gLite in Frage, da diese am häufigsten eingesetzt werden und DAGA dementsprechend auf diesen Grid-Middlewares eingesetzt werden können sollte. Da das Globus Toolkit den de-facto–Standard im Grid-Computing darstellt [Alla] und zu dieser Grid-Middleware die meiste Erfahrung an der *Lehr- und Forschungseinheit für Kommunikationssysteme und Systemprogrammierung an der LMU München* (LFKS), die diese Arbeit betreut, vorhanden ist, wird es als grundlegende Grid-Middleware für die Implementierung des Prototyps ausgewählt. Auch wenn die aktuelle Version 5 des Globus Toolkits bereits veröffentlicht wurde, wird Version 4.0.8 eingesetzt, da Version 5 nicht für den Einsatz im Prototypen geeignet ist: es sind viele konzeptionelle Fragen ungeklärt, beispielsweise wie die Elemente, die beim Versionssprung entfernt wurden (GT4 Java WS Core Container, WS-GRAM4, RFT) [Allc] in Version 5 ersetzt werden sollen. Eine besonders negative Auswirkung des Versionssprungs ist die allgemeine Unklarheit[1], wie Anwendungen im Globus Toolkit 5 deployt werden können, da dies in Form von Web Services durch den Wegfall des GT4 Java WS Core Container nicht mehr möglich ist. Das in den Release Notes zu Version 5.0.0 genannte Crux-Projekt, das diese Lücke schließen sollte, wurde noch vor seiner Veröffentlichung wieder beendet [Tue].

Deployment und Verwendung

Für die Installation des Globus Toolkit 4 sei auf die umfassende Installationsanleitung [Str11] verwiesen, die an der LFKS entwickelt wurde. Sie erläutert in einer detaillierten Schritt-für-Schritt–Anleitung, wie das Globus Toolkit 4 auf einem Debian 6 System zu installieren ist.

Das Globus Toolkit 4 bzw. der darin enthaltene „GT4 Java WS Core Container“ [Urlc] dient als Laufzeitumgebung für die Knowledge-Base, die als Web Service ihre Funktionalität den anderen Komponenten zur Verfügung stellt (Kapitel 7.4.1), und ebenfalls für den Web Service, der von der Probe-Station überprüft wird.

7.3.2 Betriebssysteme

Auswahl und Eigenschaften

Die Auswahl der Betriebssysteme richtet sich nach der verwendeten Grid-Middleware Globus Toolkit 4 und der vorhandenen Hardware.

Als Betriebssystem für die Ausführung des Globus Toolkit 4 wird die Linux-Distribution *Debian* in Version 6 gewählt, da dieser Distribution in den Platform Notes der Globus Toolkit

[1] Ergebnis einer intensiven Recherche in Online-Portalen, Usergroups und in der Literatur sowie von Gesprächen mit Mitarbeitern des Leibniz-Rechenzentrums

4 Dokumentation [Alld] neben *Ubuntu* eine besonders gute Eignung attestiert wird und da aufgrund anderer Projekte der LFKS, wie der erwähnten Installationsanleitung [Str11], eine klare Präferenz für Debian 6 besteht.

Als weiteres Betriebssystem kommt Mac OS 10.6.8 für die Consumer und die Probe-Station zum Einsatz, um mehrere Betriebssysteme einzusetzen und damit die Plattformunabhängigkeit des Entwurfsmodells und des Prototyps zu demonstrieren.

Deployment und Verwendung

Die für den Betrieb des Globus Toolkit 4 notwendigen Konfigurationsschritte in Debian 6 sind in der Installationsanleitung [Str11] angegeben. Mac OS 10.6.8 muss nicht spezifisch für DAGA konfiguriert werden.

7.3.3 Programmiersprache

Für die Entwicklung von Web Services können beliebige Hochsprachen wie C#, C++, Java oder .Net eingesetzt werden [HH10]. Aus der Vielzahl möglicher Sprachen wird *Java* [Orad] in der Version 6 mit der entsprechenden JVM 1.6.0 aus den folgenden Gründen ausgewählt.

Plattformunabhängigkeit und Grid-Middlewares – Da Java-Programme in virtuellen Maschinen, den sogenannten Java Virtual Machines (JVM), ausgeführt werden, sind sie plattformunabhängig und können in verschiedensten Umgebungen eingesetzt werden [SB08]. Die JVMs sind für die gewählten Betriebssysteme verfügbar und problemlos zu installieren, was bei .Net beispielsweise nicht der Fall ist. Neben einer allgemeinen Unterstützung der Programmiersprache Java ist ein weiterer Grund die explizite Unterstützung durch die verwendete Grid-Middleware Globus Toolkit 4 und hier insbesondere den Globus Toolkit 4 Java WS Core Container, ein Sammelbegriff für einen HTTP-Server, einen Application Server und eine SOAP-Engine [SC06].

Unterstützung bei der Entwicklung – Für Java gibt es umfassende und ausgereifte Bibliotheken, die die Entwicklung von Web Services stark vereinfachen und unterstützen, beispielsweise die bereits erwähnte JAX-WS (Kapitel 7.2). Des Weiteren gibt es eine sehr große Community, die bei möglichen Fragen helfen kann.

Weitere Gründe – Es fallen keine Lizenzgebühren an, weder für die Nutzung noch bei der Entwicklung, da es Entwicklungsumgebungen wie Eclipse [Foue] oder NetBeans [Oraf] im Open Source–Bereich gibt und nicht auf kommerzielle Produkte wie Microsoft Visual Studio bei der Entwicklung mit .Net zurückgegriffen werden muss. Abschließend ist die langjährige Java-Entwicklungserfahrung zu nennen.

Auch wenn andere Sprachen ebenfalls einige der genannten Gründe erfüllen, wie beispielsweise .Net, das eine umfassende Unterstützung für Web Services und insbesondere die Web Service Security Spezifikationen [WH03] bietet, überwogen die Gründe für die Programmiersprache Java.

7.3.4 Dienst-Beschreibungen mittels WSDL-Dokumenten

Die Dienstbeschreibungen der Knowledge-Base und des Test-Dienstes werden gemäß der Verwendung von Web Services (Kapitel 7.2) mit in der *Web Service Description Language* (WSDL) verfasster WSDL-Dokumente beschrieben. Hierbei wird die WSDL in Version 1.1. verwendet, da Version 2 weder von den Entwicklungswerkzeugen noch vom Java WS Core Container des Globus Toolkits 4 unterstützt wird.

7.4 Details zur Implementierung einzelner Komponenten

Nachdem in den vorigen Kapiteln die Grundlagen für die Entwicklung des Prototyps gelegt wurden, werden nun die prototypischen Implementierungen der Komponenten *Knowledge-Base*, *Consumer* und *Probe-Station* sowie der eingesetzten *Probes* erläutert. Für eine bessere Nachvollziehbarkeit wird für jede Komponente die Verzeichnis-Struktur des Eclipse-Projektes angegeben und in die Erklärung mit einbezogen. Die angegebenen Code-Listings sind in Anhang B (Seite 131) zu finden.

7.4.1 Knowledge-Base

Artefakt „Datenspeicherung“

Auswahl und Eigenschaften Bei der Wahl der Datenbank-Lösung zur Speicherung der Verfügbarkeits-Informationen gilt es, sowohl die Prämissen für die Entwicklung des Prototyps (Kapitel 7.1) einzuhalten als auch die analysierten Anforderungen (Kapitel 3) sowie die Vorgaben des Entwurfsmodells (Kapitel 6) zu berücksichtigen. Daraus ergeben sich vier Eigenschaften, die die Datenbank-Lösung vorweisen muss: eine einfache Installation und Konfiguration auf einem Debian 6 System (Kapitel 7.3.2), eine Schnittstelle für die Interaktion mit Java-Programmen (Kapitel 7.3.3), die Möglichkeit zur Abbildung des in Kapitel 6.2.1 vorgestellten Datenmodells und die Verfügbarkeit einer Anfragesprache (*UC5.1*).

Aus einer Vielzahl von bestehenden Datenbank-Lösungen, die relationale, schemalose und dateibasierte Ansätze ebenso umfassen wie lizenzpflichtige und Open Source-Projekte, wurde *MySQL* [Orae] für die Speicherung der Daten innerhalb der Knowledge-Base ausgewählt.

Die Wahl fiel auf MySQL, da dieses Datenbank-System alle vier genannten Eigenschaften aufweist: ein MySQL-Server ist mit den für den Debian 6 Package-Manager vorhandenen Paketen in kurzer Zeit installiert und konfiguriert, für die Interaktion mit Java-Programmen bietet Java native Programmbibliotheken an, das im Entwurfsmodell vorgestellte Datenmodell kann abgebildet werden, und es gibt mit SQL eine Anfragesprache, mit der komplexe Anfragen an die Datenbank formuliert werden können. Zudem besteht MySQL seit vielen Jahren und bildet den de-facto Standard unter den Open Source-Datenbanklösungen, woraus seine Stabilität und die große Community resultieren.

Der Nachteil der schwierigen Skalierung von „herkömmlichen“ relationalen Datenbanken [Edl+10], zu denen MySQL zu zählen ist, spielt im Rahmen der Prototyp-Entwicklung keine Rolle, da sich dieser Nachteil nur negativ auf den Betrieb eines Produktivsystems, nicht aber auf die Erfüllung der analysierten Anforderungen oder die im Entwurfsmodell entwickelten Konzepte auswirkt und daher gemäß den Prototyp-Prämissen vernachlässigt werden kann. Für den Produktiveinsatz kann durch die Kapselung der Datenspeicherung in ein eigenständiges Artefakt die Datenbank-Lösung ausgetauscht werden, ohne dafür umfangreiche Anpassungen am Subsystem „Zugriffsmanagement“ durchführen zu müssen, da hier nur die Implementierung der Klassen `ReadQueryHandler` sowie `ModificationQueryHandler` anzupassen wäre.

Eine Datenbank-Lösung, die für den Produktiveinsatz gut geeignet wäre, ist die NoSQL-Datenbank *Cassandra*, ein Projekt der Apache Foundation, „which develops a highly scalable second-generation distributed database, bringing together Dynamo's fully distributed design and Bigtable's ColumnFamily-based data model“[Lak+]. Von der Verwendung von Cassandra wird im Rahmen des Prototyps abgesehen, da diese Datenbank-Lösung die vier Eigenschaften wesentlich schlechter erfüllt als MySQL.

Eine weitere Alternative zu MySQL stellt die Verwendung von Directory Services dar, d.h. verteilten und durchsuchbaren Datenbanken [HSG99], auf die mittels des *Lightweight Directory Access Protocol*s (LDAP) zugegriffen wird. Sie würden sich für die Datenspeicherung in der Knowledge-Base eignen, da die IDs der in DAGA verwendeten Elemente in Form von Distinguished Names gespeichert werden (Kapitel 6.2.1) und somit gut von einen Verzeichnis-Dienst verwendet werden können. Außerdem haben LDAP-Server bei lesenden Anfragen eine sehr geringe Zugriffszeit, wodurch eine hohe Performance bei Zugriffen von Consumern erreicht werden kann. Mit ApacheDS™, „an extensible and embeddable directory server entirely written in Java, which has been certified LDAPv3 compatible by the Open Group“ [Foud], gibt es zudem eine frei verfügbare Server-Implementierung, die auf einem Debian 6 System installiert werden kann. LDAP wird in einigen GIS-Implementierungen und bestehenden Ansätzen, wie beispielsweise [Smi01], eingesetzt. Im Rahmen des Prototyps wird von der Verwendung von Directory Services abgesehen, da deren Verwendung nur ein Produktivsystem, nicht aber die Erreichung der Prototyp-Prämissen (Kapitel 7.1) unterstützen würde und zudem keine Erfahrungen mit LDAP-Systemen vorhanden sind.

Installation, Konfiguration und Verwendung Dank des bereits erwähnten Debian 6 Package-Managers gestaltet sich die Installation und Konfiguration des MySQL-Servers einfach. Die notwendigen Befehle zum Installieren der Pakete sind in Code-Listing B.1 (Seite 131) angegeben. Für eine vereinfachte Administration kann phpMyAdmin [Urle] eingesetzt werden, das über Code-Listing B.2 (Seite 131) installiert wird. Anschließend kann das Datenbank-Schema, das Teile der GLUE 2.0 Referenz-Implementierung aus [And+08] enthält, eingerichtet und der MySQL-Benutzer `knowledgebase` mit dem Passwort `knowledgebase` angelegt werden. Das Datenbank-Schema ist aufgrund seines Umfangs nicht in dieser Arbeit abgebildet, sondern im Eclipse-Projekt der Knowledge-Base in der Datei `/etc/KnowledgeBaseDBSchema.sql` hinterlegt. Weitere Konfigurationsschritte sind für die Verwendung von MySQL nicht notwendig.

Subsystem „Zugriffsmanagement“

Auswahl und Eigenschaften Das Subsystem handhabt die Anfragen der verschiedenen Komponenten an das Datenbank-System und muss somit auf der einen Seite mit der *MySQL-Datenbank kommunizieren* und auf der anderen Seite mit den *anderen DAGA-Komponenten interagieren*, indem die Anfragen der Consumer und der Probe-Station entgegengenommen und bearbeitet werden.

Durch die in den vorherigen Schritten gewählten Technologien der Web Services (Kapitel 7.2), der Programmiersprache Java (Kapitel 7.3.3) und des MySQL-Datenbank–Systems sind für das Subsystem neben der Auswahl einer passenden Implementierung der JAX-WS–Spezifikation aus den in Kapitel 7.2 genannten Kandidaten keine weiteren Elemente auszuwählen.

Für die **Kommunikation mit der MySQL-Datenbank** nutzt das Subsystem die Java-nativen Funktionalitäten des Packages `java.sql.*` [Orab]. Um die Ergebnisse der Datenbank-Anfragen in Java verwenden zu können, werden die Anfrage-Ergebnisse mittels der Klasse `ORMapper` im Package `daga.common.core.database` in *Plain Old Java Objects* (POJOs) umgewandelt, um die Daten aus der MySQL-Datenbank im weiteren Verlauf entsprechend der Objektorientierung von Java nutzen zu können. Eine beispielhafte Anfrage an die Datenbank gibt Code-Listing 7.1 (Seite 104).

```
// Ein Probe-Trace Objekt aus der Datenbank auslesen. Die Umwandlung des
// Datenbank-Ergebnisses in ein POJO, in diesem Fall vom Typ ProbeTrace,
// wird von der Methode der ReadQueryHandler-Klasse uebernommen
ProbeTrace probeTrace = ReadQueryHandler.getProbeTraceById(probeTraceId);
```

Listing 7.1: Beispiel einer Anfrage an die MySQL-Datenbank in der Knowledge-Base

Für die **Interaktion mit den anderen DAGA-Komponenten** muss eine Web Service Schnittstelle erstellt werden, die den Nachrichten-Austausch über SOAP-Nachrichten verwaltet und als Intermediär zwischen den Web Service Operationen und der bereits genannten Kommunikation mit der MySQL-Datenbank agiert. Für den SOAP-Nachrichten–Austausch sowie die damit zusammenhängenden Aufgaben kommt die in Kapitel 7.2 erwähnte JAX-WS–Spezifikation zum Einsatz. Da JAX-WS jedoch nur eine Spezifikation und keine Implementierung darstellt, gilt es, eine geeignete Implementierung auszuwählen. Als mögliche Kandidaten kommen die Referenz-Implementierung JAX-WS RI des Metro-Projekts, „a high-performance, extensible, easy-to-use web service stack“ [Orac], sowie die Implementierung des Apache CXF Projekts [Foua], einem sehr umfassenden Framework für die Entwicklung von Services mittels JAX-WS, in Frage.

Für die Implementierung des Subsystems wird der Metro Web Service Stack gewählt, da dieser auch im Globus Toolkit 4 Java WS Core Container eingesetzt wird und durch die Verwendung der gleichen Implementierung Kompatibilitätsprobleme sowohl im Java-Code als auch bei den Abhängigkeiten der Java-Bibliotheken vermieden werden können. Das folgende Code-Listing 7.2 zeigt, wie eine Anfrage an die Knowledge-Base entgegengenommen, bearbeitet und beantwortet wird. Die Parameter sowie die verschiedenen Klassentypen werden durch den Metro Web Service Stack vorgegeben.

```
// Das implementierte Interface wird bei der Java-Stubs-Erzeugung generiert
public class KnowledgebaseImpl implements KnowledgebasePortType {
   // Die Methode liefert ein ProbeTrace Objekt zur gegebenen ID zurueck
   // @param parameters Wrapper-Objekt, dass Anfrage-Parameter enthaelt
   // (Vom Metro WS Stack aus der Anfrage-SOAP-Nachricht generiert)
   // @return Das evtl. gefundene ProbeTrace Objekt, ebenfalls Wrapper-
       Objekt
   // (Umwandlung Wrapper-Objekt -> SOAP-Nachricht durch Metro WS Stack)
  public GetProbeTraceByIdResponse getProbeTraceById (
      GetProbeTraceByIdRequest parameters) {
    // Id aus dem Request-Objekt auslesen
    int pTID = parameters.getProbeTraceId();
    // Siehe Code-Listing 7.1
    ProbeTrace pT = ReadQueryHandler.getProbeTraceById(pTID);
    // Fuer Antwort ein weiteres Wrapper-Objekt erzeugen...
    GetProbeTraceByIdResponse response = new GetProbeTraceByIdResponse();
    // ... in das das gefundene ProbeTrace-Objekt gespeichert wird.
    // Da DAGA-Objekte tw. andere Eigenschaften haben als die
    // Metro WS Stack Objekte (POJOs), wird vor der Speicherung
    // umgewandelt
    response.setWsProbeTrace(POJOToWS.transformProbeTrace(pT));
    // Wrapper-Objekt fuer Versand als SOAP-Nachricht an Metro WS Stack
    return response;
  }
}
```

Listing 7.2: Auszug aus der den Web Service der Knowledge-Base implementierenden Klasse `KnowledgeBaseImpl`

Abbildung 7.3 zeigt die Verzeichnis-Struktur des Eclipse-Projekts der Knowledge-Base und erläutert die Aufgaben der enthaltenen Dateien. Die grün umrahmten Verzeichnisse enthalten Dateien, die für die Verwendung der Knowledge-Base als Web Service im Globus Toolkit 4 notwendig sind, die blau umrahmten Dateien betreffen die Implementierung der im Entwurfsmodell festgelegten Funktionalitäten der Knowledge-Base, und das grau umrahmte Verzeichnis enthält Bibliotheken, die für beide Bereiche eingesetzt werden.

Das die als Web Service veröffentlichten Funktionalitäten der Knowledge-Base beschreibende WSDL-Dokument (Kapitel 7.2) ist im Verzeichnis `/WebServiceAndDeployment/wsdl` des Eclipse-Projektes abgelegt. Bei der Erstellung der WSDL-Beschreibung werden alle Datentypen in *.xsd-Dateien im selben Verzeichnis ausgelagert. Zudem wird eine Namenskonvention eingeführt, die bei allen Elementen anzuwenden ist: sowohl bei Nachrichten als auch bei deren Inhalt ist die Anfragerichtung durch `Request` bzw. `Response` markiert. Nachrichtennamen enden zudem mit `Message`. Die Nachricht für die Abfrage der Details einer Probe anhand ihrer ID würde dementsprechend `getProbeByIdRequestMessage` lauten.

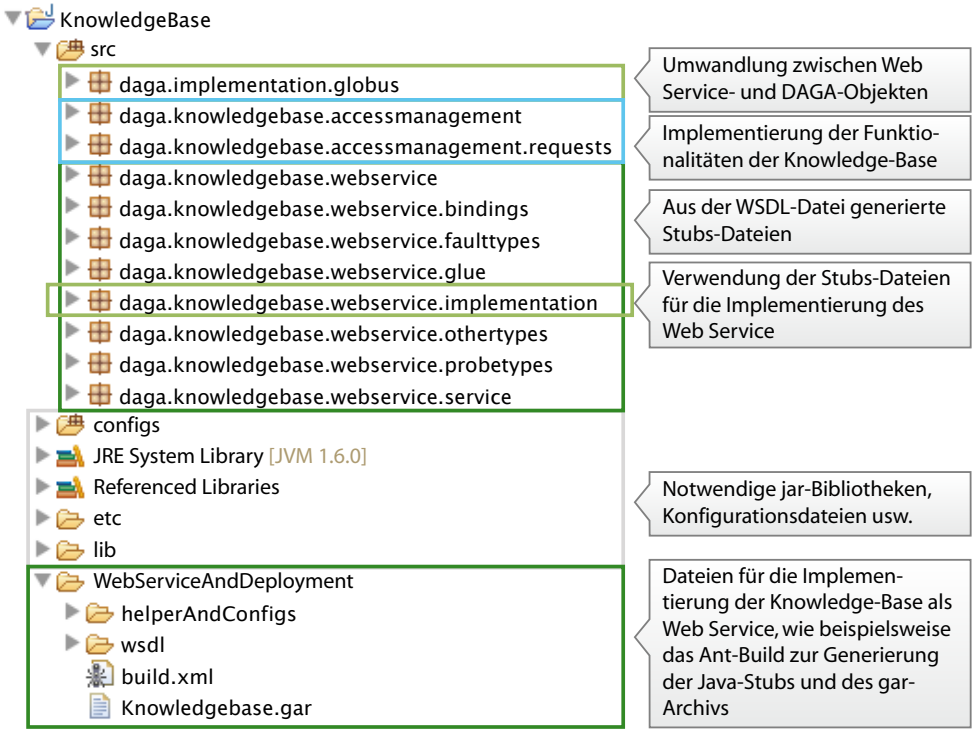


Abbildung 7.3: Verzeichnis-Struktur des Eclipse-Projekts der Knowledge-Base

Installation, Konfiguration und Verwendung Um einen Web Service im Globus Toolkit 4 Java WS Core Container bereitstellen zu können, wird ein sogenanntes *gar-Archiv* (Grid Archive) erstellt, das anschließend im Globus Toolkit 4 Java WS Core Container deployt wird. Das gar-Archiv muss eine vorgegebene, in Abbildung 7.4 (Seite 106) dargestellte Verzeichnis-Struktur aufweisen, um vom Globus Toolkit 4 während des Deployments erkannt zu werden. Die Verzeichnis-Struktur wird im Folgenden nur kurz erläutert, eine umfassende Erklärung ist in [Str11] zu finden. Für die Verwendung der Knowledge-Base im Globus Toolkit 4 müssen an den im Prototyp enthaltenen Dateien keine Anpassungen vorgenommen werden.

Mittels der jndi-Konfigurationsdatei wird festgelegt, welche Klasse den Web Service implementiert. Hier ist `daga.knowledgebase.webservice.implementation.KnowledgebaseImpl` anzugeben. Der hier angegebene Name muss mit dem in der Deployment-Descriptor–Datei `server-deploy.wsdd` angegebenen Namen übereinstimmen.

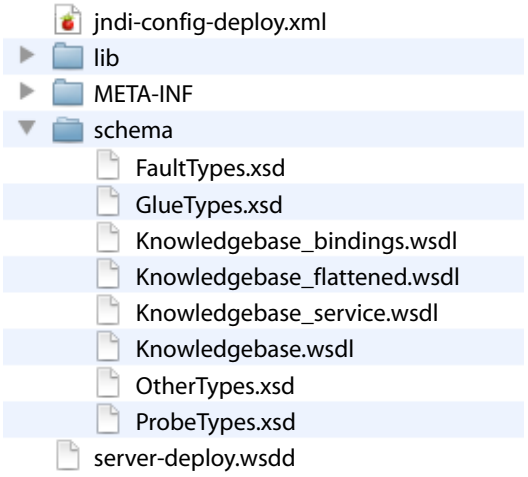


Abbildung 7.4: Die obligatorische Verzeichnis-Struktur eines gar-Archivs

Im `lib`-Verzeichnis sind alle jar-Archive enthalten, die vom Web Service verwendet werden, beispielsweise die MySQL-Treiber oder die log4J-Bibliotheken. Die Implementierung des Web Service selbst ist ebenfalls in Form von zwei jar-Archiven enthalten, dem `Knowledgebase.jar` und `Knowledgebase_Stubs.jar`, die beide während der Erstellung des gar-Archivs automatisch generiert werden. Die im `lib`-Verzeichnis enthaltenen jar-Archive werden beim Deployment im Globus Toolkit 4 in das allgemeine Verzeichnis `$GLOBUS_LOCATION/lib` kopiert und bei einem möglichen Undeployment des Web Service wieder gelöscht. Daher sollte darauf geachtet werden, dass hier keine bereits vom Globus Toolkit 4 installierten jar-Archive enthalten sind, da diese andernfalls gelöscht werden und das Globus Toolkit 4 anschließend nicht mehr funktionstüchtig ist.

Der Inhalt des `META-INF`-Verzeichnisses kann vernachlässigt werden.

Im `schema`-Verzeichnis sind die die WSDL-Dateien enthalten, die den Web Service beschreiben. Es sind alle vier Dateien notwendig, da sich die Beschreibung aus allen vier Dateien zusammensetzt und alle Dateien beim Deployment in GT4 an verschiedene Ziele kopiert werden.

Der Deployment-Descriptor `server-deploy.wsdd` [SC06] beschreibt, wie der Web Service beim Deployment im Globus Toolkit 4 Java WS Core Container eingerichtet werden soll.

Für die Erstellung des gar-Archivs ist im Eclipse-Projekt der Knowledge-Base ein Ant-Build enthalten, der alle notwendigen Schritte durchführt und ein unmittelbar verwendbares gar-Archiv erstellt. Der Ant-Build ist im Verzeichnis `WebServiceAndDeployment/build.xml` zu finden, das gar-Archiv wird im selben Ordner abgelegt. Vor der ersten Ausführung des Builds muss der absolute Pfad des Eclipse-Projekts in der zum Ant-Build gehörenden properties-Datei `WebServiceAndDeployment/helperAndConfigs/build.properties` angepasst werden.

Das gar-Archiv kann im Globus Toolkit 4 anschließend mit dem Befehl `globus-deploy-gar Knowledgebase.gar` deployt werden.

Erweiterungen Um die Knowledge-Base um neue Funktionalitäten zu erweitern, sind drei Schritte notwendig.

Im **ersten Schritt** wird die WSDL-Datei der Knowledge-Base um die gewünschten Funktionalitäten erweitert (`WebServiceAndDeployment/wsd/Knowledgebase.wsdl`). Wird der visuelle WSDL-Editor von Eclipse verwendet, muss nach dem Hinzufügen der Operationen das Binding neu generiert werden, da dies nicht automatisch durchgeführt wird und es andernfalls zu einem fehlerhaften WSDL-Dokument kommt.

Im **zweiten Schritt** werden die Java-Stubs–Dateien mit dem bereits erwähnten Ant-Build `WebServiceAndDeployment/build.xml` generiert, wobei das Target `compileStubs` verwendet wird. Für diesen Schritt muss zwingend das angegebene Ant-Build verwendet werden, da es bei der Verwendung anderer Tools wie beispielsweise `WSImport` zu Problemen mit dem Globus Toolkit 4 kommen kann.

Im **dritten Schritt** wird die Klasse `KnowledgebaseImpl` an die neu hinzugefügten Funktionalitäten angepasst (Package `daga.knowledgebase.webservice.implementation`). Da sich das von der Klasse implementierte Interface `KnowledgebasePortType` bei der Generierung der Java-Stubs–Dateien verändert, wird in einer IDE wie Eclipse angezeigt, welche Methoden zu implementieren sind. Die Implementierung einer Methode ähnelt dabei dem Muster des bereits gegebenen Beispiels in Code-Listing 7.2: zunächst werden die Parameter aus dem Anfrage-Objekt ausgelesen. Mittels dieser Parameter wird eine Anfrage an die Datenbank gestellt, deren Ergebnis anschließend in ein Antwort-Objekt gespeichert und dieses für die Übermittlung an die interagierende Komponente an den Metro Web Service Stack zurückgegeben wird.

Die hier beschriebenen Schritte sind auch in der Deployment-Übersicht in Abbildung B.4 (Seite 132) enthalten.

7.4.2 Consumer

Auswahl und Eigenschaften

Der Consumer muss Anfragen an den Web Service der Knowledge-Base stellen und die Anfrageergebnisse für den Endbenutzer grafisch aufbereiten können, er muss also einen *Web Service-Client* implementieren und zusätzlich eine *graphische Oberfläche* für den Endbenutzer anbieten.

Neben den in Kapitel 7.1 genannten Prämissen für die Entwicklung des Prototyps ist bei der Auswahl der Technologien des Consumers von besonderer Bedeutung, dass er möglichst einfach und möglichst schnell einzusetzen ist, d.h. ohne vorherige Installation von Programmen oder Bibliotheken, um den Aufwand für den Endbenutzer so gering wie möglich zu halten [BGN02].

Für die Funktionalitäten des **Web Service-Clients** wird wie schon bei der Knowledge-Base die JAX-WS Referenzimplementierung des Metro-Projekts eingesetzt, da damit bereits Erfahrungen während der Entwicklung der Knowledge-Base gesammelt werden konnten und zugleich das Tool `WSImport` [Oraa] enthalten ist, mit dem aus einer WSDL-Definition die für den Web Service-Client notwendigen Java-Stubs–Klassen erzeugt werden können. Anders als bei der Entwicklung der Knowledge-Base kann bei der Entwicklung des Consumers das Tool `WSImport` eingesetzt werden, da die hier erzeugten Java-Stubs-Klassen nicht im Globus Toolkit 4 zum Einsatz kommen, sondern in einem GlassFish Application Server, wo bei deren Verwendung keine Probleme auftreten. Eine typische Anfrage, die die mit `WSImport` generierten Java-Stubs-Klassen verwendet, an die Knowledge-Base ist im Code-Listing 7.3 (Seite

108) dargestellt. Dass die Anfrage an den Web Service geschickt und die Antwort wiederum als Java-Objekt verfügbar ist, wird durch den Metro Web Service Stack realisiert.

Die **graphische Oberfläche** für den Endbenutzer wird als Web-Anwendung realisiert, die über einen üblichen Browser verwendet werden kann. Dadurch muss weder spezielle Software installiert noch müssen Konfigurationseinstellungen angepasst werden wie beispielsweise die Freischaltung von Netz-Ports in der Firewall, da der Browser mit der Web-Oberfläche über den Internet-Port 80 interagieren kann. Um einen sauberen und wartbaren Quellcode zu entwickeln, kann aus einer Fülle von Frameworks wie Apache Wicket [Fouc], Spring [spr] oder Apache Struts^2 [Urla] gewählt werden, die allgemeine Funktionalitäten wie das Mapping von URLs auf Java-Methoden oder eine MVC-Architektur bei der Entwicklung verschatten und so einen Quellcode ermöglichen, der sich ausschließlich auf die Business-Logik der Web-Anwendung konzentrieren kann. Als Framework wird Struts^2 eingesetzt, „a free open-source solution for creating Java web applications“ [Urla]. Es wird dieses Framework gewählt, da es die Verwendung einer MVC-Architektur und die mehrsprachige Ausgabe von Nachrichten unterstützt, eine sehr kurze Einarbeitungszeit hat und mit *Java Server Pages* (JSP) eine mächtige Template-Sprache bereitstellt.

```
// Proxy-Objekt fuer Interaktion mit der Knowledge-Base erstellen
KnowledgebasePortType proxy = new KnowledgebaseService().
    getKnowledgebasePortTypePort();
// Verwendung der Metro WS Stack Wrapper-Objekte (Code-Listing 7.2)
GetProbeTraceByIdRequest requestTrace = new GetProbeTraceByIdRequest();
requestTrace.setProbeTraceId(probeTraceId);
// Anfrage-Objekt an Web Service senden...
GetProbeTraceByIdResponse response = proxy.getProbeTraceById(requestTrace);
// ... und dessen Antwort auswerten
probeTrace = WSToPOJO.transformProbeTrace(response.getWsProbeTrace());
```

Listing 7.3: Code-Beispiel für eine Anfrage an die Knowledge-Base mit anschließender Verarbeitung der Antwort

Die Ausführung eines Java-Programms und somit auch einer Java Web-Anwendung wie dem Consumer benötigt immer eine Laufzeitumgebung, in der die Java Virtual Machine und weitere Programmbibliotheken enthalten sind [LF10]. Für die Knowledge-Base übernimmt diese Aufgabe der im Globus Toolkit 4 enthaltene Java WS Core Container.

Die *Java Runtime Edition* (JRE), die in Mac OS X, dem Host-System des Consumers, enthalten ist, reicht für den Betrieb des Consumers allerdings nicht aus [HH10], da dafür neben der in der JRE bereitgestellten Java-Bytecode–Ausführung auch der Metro Web Service Stack verwendet und die grafische Benutzeroberfläche einem Browser zur Verfügung gestellt werden können muss. Für diesen komplexeren Anwendungsfall werden *Web Application Server* eingesetzt wie Apache Tomcat [Foub] oder Oracle GlassFish [jav11], die diese Funktionalitäten enthalten [HH10]. In diesen Web Application Servern kann die Java Web-Anwendung dann deployt und verwendet werden. Es wird der Oracle GlassFish Server gewählt, da es im Vergleich zum Apache Tomcat Server weder funktionale noch lizenzrechtlich relevante Unterschiede für den in dieser Arbeit erstellten Prototypen gibt und Oracle GlassFish in der bei der Entwicklung des Prototyps verwendeten Literatur als Beispiel eingesetzt wird.

Abbildung 7.5 (Seite 109) zeigt die Verzeichnis-Struktur des Eclipse-Projekts des Consumers. Die hellblau umrahmten Verzeichnisse und Dateien betreffen die Web-Anwendung und das dabei verwendete Struts^2-Framework, die grün umrandeten Verzeichnisse die Interaktion mit der Web Service Schnittstelle der Knowledge-Base und die grau umrandeten Verzeichnisse

die verwendeten Laufzeit-Dateien. Die gestrichelten Pfeile zeigen jeweils an, welche Dateien mit den angegebenen Ant-Builds generiert werden.

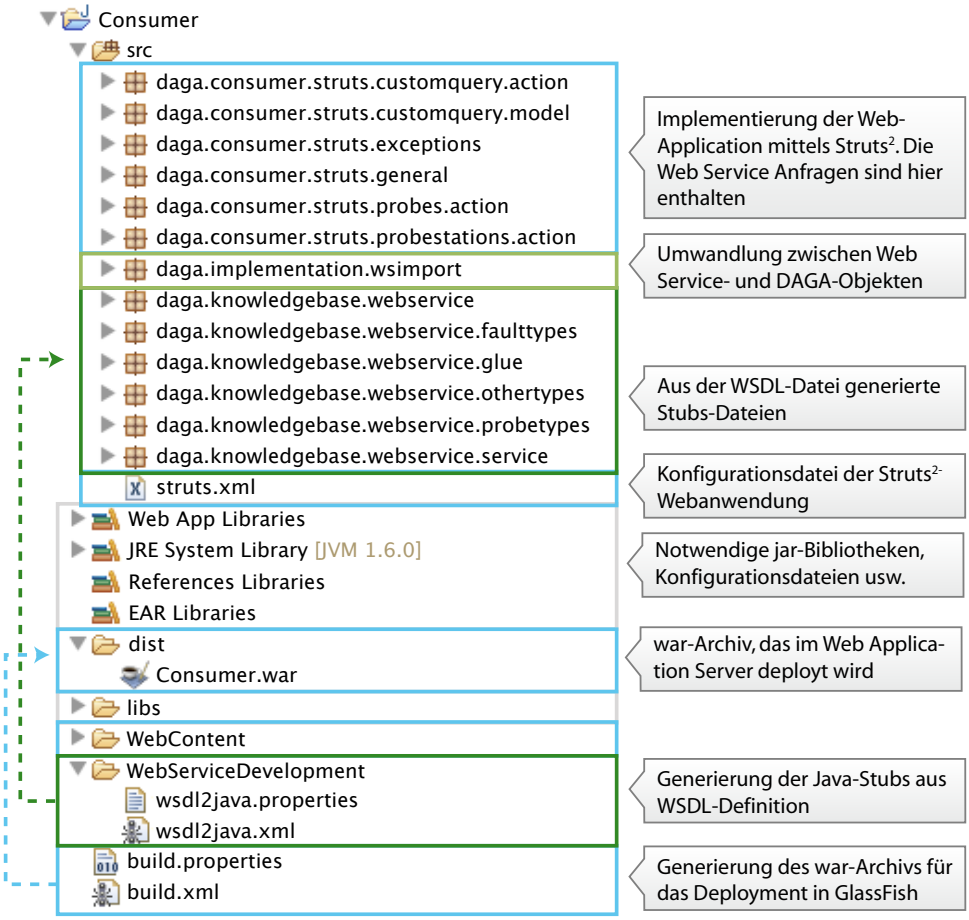


Abbildung 7.5: Verzeichnis-Struktur des Eclipse-Projekts des Consumers

Installation, Konfiguration und Verwendung In einem ersten Schritt muss der genannte Web Application Server GlassFish installiert werden. Dafür kann unter [jav11] ein Installationsassistent heruntergeladen werden, der den GlassFish Server installiert und konfiguriert.

Um den Consumer im GlassFish Server einzusetzen, muss ein sogenanntes *war-Archiv* (Web Archive) aus den Dateien des Consumers erstellt werden, das im GlassFish Server über dessen administrative Funktionalitäten deployt wird. Das war-Archiv `/dist/Consumer.war` kann mit dem vorhandenen Ant-Build `/build.xml` im Eclipse-Projekt des Consumers erzeugt werden, vor dessen erster Ausführung die Konfigurationsdatei `/build.properties` angepasst werden muss.

Wurden die Standard-Einstellungen beim GlassFish Server und dem Consumer beibehalten, ist der Consumer nach dem Deployment im GlassFish Application Server über die URL `http://localhost:8080/Consumer` zu erreichen. Abbildung 7.6 (Seite 110) zeigt die grafische Oberfläche des Consumers am Beispiel der Detailseite einer Probe: es werden die allgemeinen Daten einer Probe dargestellt sowie der die Probe beinhaltende Probe-Trace mit seiner Probe-Station und der Testsuite. Erweiterungen des Prototyps wären beispielsweise die Anzeige des

überprüften Endpoints (Kapitel 6.2.1) sowie eine Verlinkung zu diesem Element.

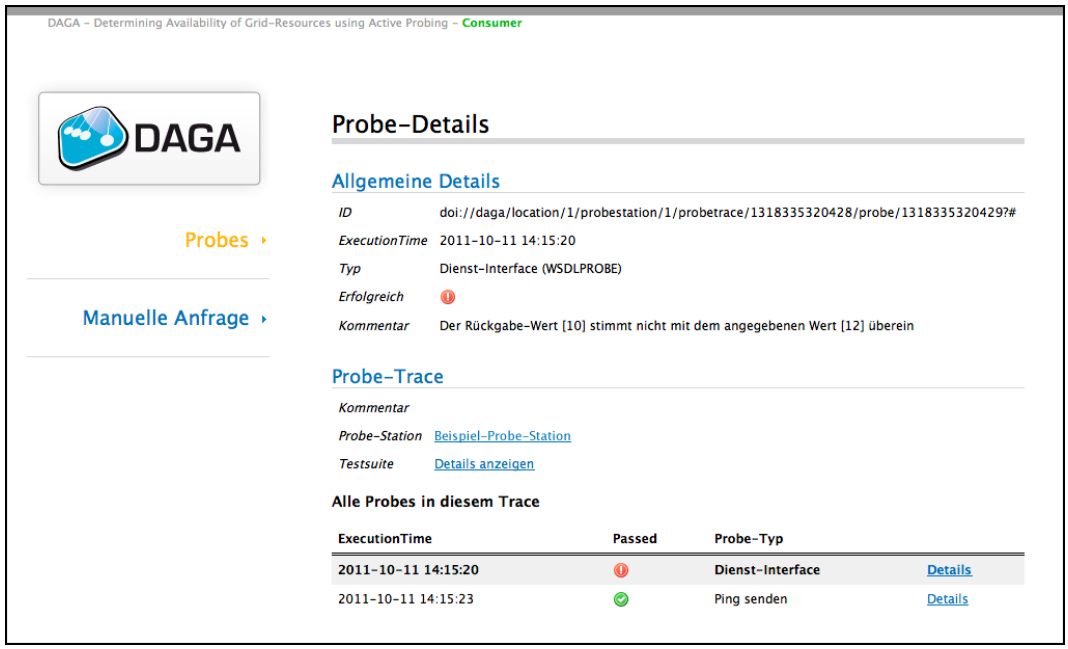


Abbildung 7.6: Screenshot der Darstellung von Probe-Details im Consumer

7.4.3 Probe-Station

Speicherung der Testsuites und der Queue

Auswahl und Eigenschaften Für die Speicherung der Testsuites und der Queue, die die an die Knowledge-Base zu übertragenden abgeschlossenen Probe-Traces enthält, wird wie schon in der Knowledge-Base eine MySQL-Datenbank eingesetzt. Neben den in Kapitel 7.4.1 genannten Gründen ergibt sich bei der Probe-Station der zusätzliche Vorteil, dass mit MySQL bei der Entwicklung der Knowledge-Base bereits Erfahrungen gesammelt wurden, die auf die Entwicklung der Probe-Station übertragen werden können. Da Probe-Stations nicht zwangsläufig auf leistungsstarker Hardware, sondern ebenso auf kleineren Endgeräten eingesetzt werden können, sollte für die Entwicklung eines Produktivsystems eine alternative Speicherlösung verwendet werden, bei der die Installation eines (MySQL-)Datenbank-Servers nicht notwendig ist. Möglich wäre hier beispielsweise eine dateibasierte Datenbank mit den dazugehörigen Java-Bibliotheken.

Installation, Konfiguration und Verwendung Die Installation der MySQL-Datenbank richtet sich nach dem Betriebssystem, auf dem die Probe-Station ausgeführt wird. Für den Betrieb auf einem Debian 6 System kann das bereits bei der Knowledge-Base angegebene Code-Listing B.1 eingesetzt werden. Im Rahmen des Prototyps wird die Probe-Station auf einem Mac OS X System ausgeführt, für das ein MySQL-Server zusammen mit dem Verwaltungsprogramm phpMyAdmin mittels MAMP („Macintosh, Apache, MySQL und PHP“ [Gmba]) installiert werden kann. Nach der Installation der Datenbank können das Datenbank-Schema eingerichtet und der Benutzer `probestation` mit dem Passwort `probestation` angelegt werden. Das Datenbank-Schema ist aufgrund seines Umfangs nicht in dieser Arbeit abgebildet, sondern im Eclipse-Projekt der Probe-Station in der Datei `/etc/ProbeStationDBSchema.sql` hinterlegt.

Probe-Auswahl

Auswahl und Eigenschaften Das Subsystem „Probe-Auswahl“ hat die Aufgabe, anhand einer gegebenen Initial oder Investigative Probe die nächste auszuführende Probe zu selektieren oder zu entscheiden, dass keine weitere Probe ausgeführt werden soll.

Die Auswahl der Probes ist in der in Kapitel 7.3.3 gewählten Programmiersprache Java implementiert. Die Wahl der Programmiersprache richtet sich hier wiederum nach den in Kapitel 7.1 genannten Prämissen des Prototyps. Da die Anzahl der auszuführenden Testsuites recht schnell ansteigen kann, wäre für den Einsatz in einem Produktivsystem die Evaluation einer alternativen Sprache wie C oder C++ ratsam, um hier eine möglichst ressourcenschonende Sprache zu wählen.

Code-Listing 7.4 zeigt, wie die im Entwurfsmodell (Kapitel 6) entwickelte Austauschbarkeit des Selektions-Algorithmus der nächsten auszuführenden Probe im Prototypen implementiert wird.

```
// Die naechste auszufuehrende Probe wird mit demjenigen Algorithmus
// ausgewaehlt, den die ProbeSelectorFactory zurueckgibt
Probe nextProbe = ProbeSelectorFactory.getProbeSelector().
  getNextProbe(previousProbe);
```

Listing 7.4: Verwendung der `ProbeSelectorFactory`

Die statische Methode `getProbeSelector()` gibt eine Instanz der Klasse zurück, die das Interface `IProbeSelector` implementiert und damit die Methode `getNextProbe()` anbietet (siehe Entwurfsmodell Kapitel 6).

Die folgenden Code-Listings 7.5 und 7.6 zeigen mögliche Implementierungen der Methode `getProbeSelector()` und dass ein Austausch dieser Methode ausreicht, um einen anderen Auswahl-Algorithmus zu verwenden.

```
public class ProbeSelectorFactory {
  // Die statische Methode gibt einen Auswahl-Algorithmus zurueck,
  // der einen Entscheidungsbaum verwendet
  public static IProbeSelector getProbeSelector() {
    return new DecisionTreeProbeSelector();
  }
}
```

Listing 7.5: Einsatz des DecisionTreeProbeSelectors

```
public class ProbeSelectorFactory {
  // Hier gibt die gleiche Methode einen anderen Auswahl-Algorithmus
  // zurueck, der statt eines Entscheidungsbaums eine regelbasierte
  // Entscheidungsfindung durchfuehrt
  public static IProbeSelector getProbeSelector() {
    return new RuleBasedProbeSelector();
  }
}
```

Listing 7.6: Einsatz des RuleBasedProbeSelectors

Das Beispiel zeigt, dass die Auswahl der nächsten auszuführenden Probe `nextProbe` immer von dem Objekt ausgeführt wird, das die `ProbeSelectorFactory` zurückgibt (Code-Listing 7.4). Gibt die Factory statt der DecisionTree-Implementierung (Code-Listing 7.5) eine alternative Implementierung zurück (Code-Listing 7.6), wird diese für die Auswahl der nächsten auszuführenden Probe eingesetzt. Dadurch kann die Probe-Auswahl schnell und zur Laufzeit ausgetauscht werden, indem die Implementierung der `ProbeSelectorFactory` ersetzt wird.

Installation, Konfiguration und Verwendung Die Probe-Auswahl ist in der Probe-Station enthalten und wird daher bei deren Installation, die in einem späteren Abschnitt erläutert wird, ebenfalls installiert und konfiguriert.

Probe-Station-Management

Das Subsystem „Probe-Station-Management“ muss die Ausführung der vorhandenen Testsuites initiieren und die dabei generierten Daten aus der Queue auslesen und in vorgegebenen Zeitintervallen an die Knowledge-Base übertragen.

Diese Tätigkeiten werden mit den bereits beschriebenen MySQL-Datenbank-Anfragen und der Verwendung der JAX-WS Referenzimplementierung des Metro Web Service Stack durchgeführt. Da es hierbei keine Besonderheiten im Rahmen des Subsystems gibt, wird der im Eclipse-Projekt der Probe-Station enthaltene Quellcode des Probe-Station-Managements nicht detailliert besprochen.

Probe-Trace-Ausführung

Die Ausführung eines Probe-Traces ist in der Klasse `ProbeTraceExecutor` implementiert. Dabei wird zunächst ein neues Probe-Trace Objekt erzeugt, anschließend eine Initial Probe ausgeführt, wobei sich der `ProbeTraceExecutor` als Listener für die Fertigstellung dieser Initial Probe registriert und bei deren Beendigung die Ausführung der Investigative Probes anstößt. Die Notwendigkeit der Registrierung als Listener wurde bereits im Entwurfsmodell in Kapitel 6 erläutert. Code-Listing 7.7 zeigt ausschnittsweise die Ausführung eines Probe-Traces.

```
// Die Klasse implementiert das Interface IProbeExecutionListener,
// um sich bei der In. Probe als Listener registrieren zu koennen
public class ProbeTraceExecutor implements IProbeExecutionListener {
  // ...
  public ProbeTraceExecutor(...) {
    // Neue Probe-Trace anlegen und Eigenschaften setzen
    probeTrace = new ProbeTrace();
    // wobei hier der IDGenerator aus dem Package daga.common.elements
    // sowie die lokalen Objekte verwendet werden
    probeTrace.setId(IDGenerator.generateProbeTraceId(LocalObjectsFactory.
      getLocalProbeStation(manager.getConfiguration())));
    // ...
  }

  public void executeInitialProbe() {
    if (probeTrace.getTestsuite().getForProbeType().
      getQualifier().equals(ProbeTypeQualifier.WSDLProbeQUALIFIER)) {
      initialProbe = new WSDLProbe();
      // ...
    }
    initialProbe.initialize(probeTrace);
    // Als Listener registrieren...
    initialProbe.setListener(this);
    // ... und anschliessend ausfuehren
    initialProbe.execute();
  }
```

```
  // Methode des IProbeExecutionListener-Interfaces, die von
  // der Initial Probe bei deren Beendigung aufgerufen wird
  public void probeExecutionCompleted() {
    // ...
    // Nach der Ausfuehrung der Initial Probe
    // koennen die Investigative Probes ausgefuehrt werden
    executeInvestigativeProbes();
  }
  // Diese Methode wird von der Listener-Methode aufgerufen
  public void executeInvestigativeProbes() {
    // Naechste auszufuehrende Probe mit der von der ProbeSelectorFactory
    // zurueckgegebenen Implementierung auswaehlen
    Probe nextProbe = ProbeSelectorFactory.getProbeSelector().
      getNextProbe(initialProbe);
    // Solange eine weitere Probe ausgefuehrt werden soll...
    while (nextProbe != null) {
      // ...
      nextProbe.execute();
      // ...
      // Naechste auszufuehrende Probe waehlen
      nextProbe = ProbeSelectorFactory.getProbeSelector().
        getNextProbe(nextProbe);
    }
    // Probe-Trace abschliessen und in die Datenbank speichern
  }
}
```

Listing 7.7: Ausführung eines Probe-Trace im `ProbeTraceExecutor`

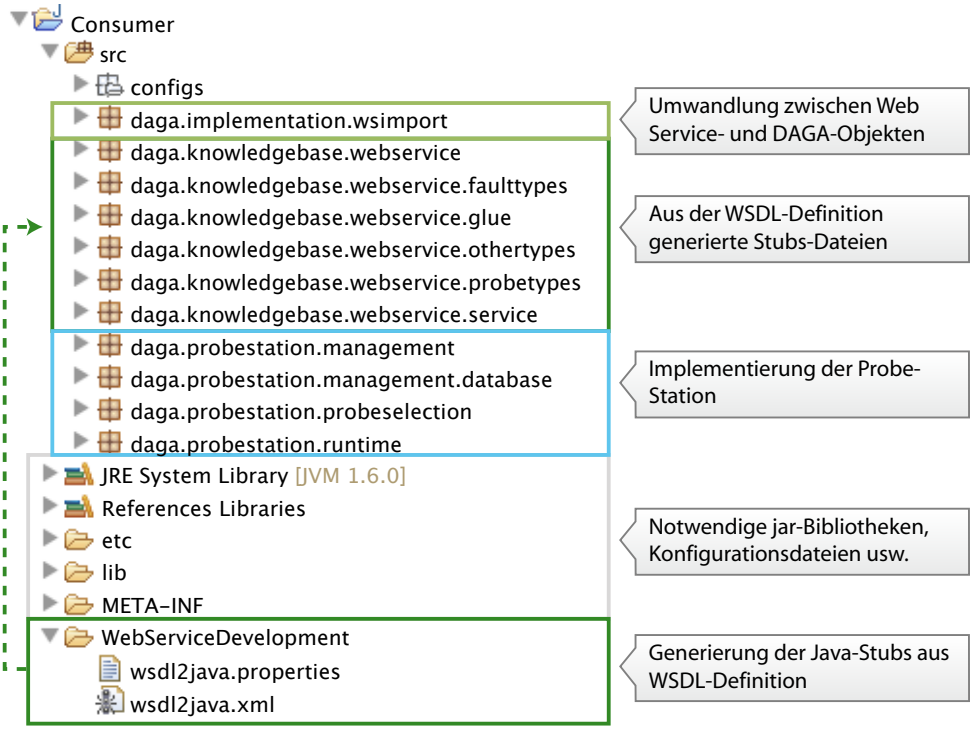


Abbildung 7.7: Verzeichnis-Struktur des Eclipse-Projekts der Probe-Station

Installation, Konfiguration und Verwendung der Probe-Station

Für die Probe-Station kann ein ausführbares jar-Archiv erstellt werden, womit die Probe-Station in jeder beliebigen Java-Umgebung ausgeführt werden kann. Dies ist möglich, da in der Klasse `ProbeStationManagement` eine `main()`-Methode angegeben ist, die bei der Ausführung des jar-Archivs gestartet wird. Das jar-Archiv kann über den Eclipse-Export–Assistenten erstellt werden.

Abbildung 7.7 (Seite 113) zeigt die Verzeichnis-Struktur des Eclipse-Projekts der Probe-Station. Wie bei den anderen beiden Abbildungen 7.3 (Seite 105) und 7.5 (Seite 109) sind die Verzeichnisse und Dateien für die Implementierung der im Entwurfsmodell vorgestellten Funktionalitäten hellblau umrandet, die Verzeichnisse für die Kommunikation mit dem Web Service Interface der Knowledge-Base grün und die verschiedenen Laufzeit-Bibliotheken grau.

7.4.4 Probe

Auswahl und Eigenschaften

Rahmenwerk und Test-Ausführung Die in Kapitel 5.2.2 getroffene Aussage, dass Probes auf der zu überprüfenden Ressource ausgeführt werden, wird für die Entwicklung des Prototyps differenziert: die Probe setzt sich aus einem Ausführungsrahmen, der auf einer Probe-Station ausgeführt wird, und der eigentlichen Probing Aktivität, die in der in jeder Probe implementierten Methode `execute()` definiert ist, zusammen. Diese Aufteilung ist notwendig, um die Probes auf einer Probe-Station in Form von Java-Objekten verwenden zu können und gleichzeitig die Funktionalitäten zur Überprüfung einer Ressource zu bündeln. Code-Listing 7.8 zeigt beispielhaft die beiden genannten Elemente.

```
public class BeispielProbe extends InitialProbe {
  protected IProbeExecutionListener listener;
   // Die in der Superklasse aufgerufene Template-Methode definieren, um
   // fuer die Probe spezifische Einstellungen vorzunehmen (z.B. Qualifier)
   public void specificInitialize(ProbeTrace assignedProbeTrace) {
      setProbeType(new ProbeType(ProbeTypeQualifierEXAMPLEPROBEQUALIFIER));
   }
  // Die Methode enthaelt die Mechanismen zur Ueberpruefung einer Ressource
  public void execute() {
    // Super-Methode aufrufen, um allgemein notwendige Vorbereitungen fuer
    // die Ausfuehrung zu treffen, z.B. Ausfuehrungszeitpunkt setzen
    super.execute();
    // Die Probe auf der zu ueberwachenden Ressource "ausfuehren",
    // z.B. durch eine Web Service Anfrage oder einen Ping
    boolean myResult = executeSomeAction();
    // Das Ergebnis in den Probe-Attributen speichern
    if(myResult) {
      setPassed(true);
      setComment("Probe erfolgreich ausgefuehrt");
    }
    // Den ProbeTraceExecutor ueber die Beendigung informieren
    listener.probeExecutionCompleted();
  }
}
```

Listing 7.8: Darstellung des Ausführungsrahmens einer Probe und der darin enthaltenen Probing Aktivität

Die `execute()`-Methode enthält neben der Ausführung, stellvertretend dargestellt durch die Methode `executeSomeAction()`, auch das Setzen der vom Ausführungsergebnis abhängigen Probe-Attribute `comment` und `passed`, die beide in der Superklasse `Probe` definiert sind (siehe UML-Diagramm in Abbildung A.2). Nachdem die Ausführung abgeschlossen ist, wird bei der Initial Probe noch der auf der Probe-Station zuständige `ProbeTraceExecutor` benachrichtigt, da Initial Probes auch asynchron ausgeführt werden können (siehe GridFTP-Probe in Code-Listing 7.11, Seite 117), d.h. die Ausführung der Probe läuft noch, während der aufrufende Thread weiterläuft.

Das Rahmenwerk wird ebenfalls in Java (Kapitel 7.3.3) entwickelt, da auch die Probe-Station, auf der das Rahmenwerk ausgeführt wird, in Java erstellt wurde. Würde für das Rahmenwerk eine andere Programmiersprache verwendet, wäre hier eine Schnittstelle notwendig, die Aufrufe und Ergebnisse zwischen den Programmiersprachen transformiert. Da dies verschiedene Nachteile nach sich zieht, müssen Probe und Probe-Station in der gleichen Programmiersprache entwickelt werden.

Diese Einschränkung gilt für den Methodenrumpf der `execute()`-Methode nur teilweise, da hier Java-Methoden eingesetzt werden, um (möglicherweise) Funktionalitäten *außerhalb* Javas aufzurufen, beispielsweise einen Shell-Command mittels `Process pr = run.exec(„my command„)`. Diese Schnittstelle ist nicht zu umgehen, da viele Funktionalitäten, wie beispielsweise `ping`, nicht in Java vorliegen und daher über diesen Weg aufgerufen werden müssen. Für die Verwendung des Globus Toolkit 4 gibt es mit dem *Java CoG Kit* ein Framework, das es ermöglicht, Grid-Funktionalitäten aus einer Hochsprache wie Java aufzurufen. Das Java CoG Kit wird in der Initial Probe GridFTP-Probe (Code-Listing 7.11, Seite 117) eingesetzt.

Testsuite Definition Wie im Entwurfsmodell bereits erläutert wurde, steht eine Initial Probe immer am Anfang eines Probe-Trace, der auf Veranlassung einer Testsuite ausgeführt wird. Aus diesem Grund führt eine Initial Probe in der `execute()`-Methode einen weiteren Schritt aus, nämlich die Aufbereitung der Testsuite-Definition, d.h. das Parsen des Strings `definition` und die Umwandlung in ein Objekt vom Typ `ITestsuiteDefinition`. Code-Listing 7.9 zeigt diesen Schritt am Beispiel der WSDL-Probe.

```
// Objekt zur Speicherung der geparsten Testsuite-Definition vorbereiten
WSDLTestsuiteDefinition wsdlTestsuiteDef = new WSDLTestsuiteDefinition();
// String mit der die Testsuite-Definition aus der Testsuite auslesen ...
String definition = getProbeTrace().getTestsuite().getDefinition();
// ... und anschliessend parsen. Die parse()-Methode ist in jeder
// TestsuiteDefinition-Klasse entsprechend dem Format der Definition
// implementiert. Wird z.B. ein XML-String angegeben, ist in der
// parse()-Methode ein XML-Parser enthalten
wsdlTestsuiteDef.parse(definition);
// Nach dem Parsen kann auf die Attribute der Testsuite zugegriffen werden
wsdlAddress = wsdlTestsuiteDef.getWsdlAddress();
```

Listing 7.9: Parsen der Testsuite-Definitionen und Umwandlung in ein ITestsuiteDefinition-Objekt

Wie das Beispiel zeigt, wird zunächst eine zum Typ der Initial Probe passende Instanz einer Testsuite Definition-Klasse erzeugt. Da eine WSDL-Probe verwendet wird, handelt es sich um die Klasse `WSDLTestsuiteDefinition`. Daran anschließend wird die Methode `parse()` aufgerufen, die alle `TestsuiteDefinition`-Klassen implementieren (siehe UML-Diagramm in Abbildung A.2, Seite 128). Nachdem die `parse()`-Methode aufgerufen wurde, können die einzelnen Felder der Testsuite-Definition von der WSDL-Probe verwendet werden.

Implementierte Probes Im Rahmen des Prototyps werden drei Probe-Typen implementiert, zwei Initial Probes (`WSDLProbe`, `GridFTPProbe`) und eine Investigative Probe (`PingProbe`), deren Besonderheiten und Eigenschaften im Folgenden erläutert werden. Das Rahmenwerk ist bei allen gleich, es wird in den Code-Listings nur der Inhalt der jeweiligen `execute()`-Methode angegeben.

Code-Listing 7.10 zeigt ausschnittsweise die **WSDL-Probe**. Der vollständige Quellcode kann in der Klasse `daga.common.elements.probes.initialprobes.WSDLProbe` nachgesehen werden. Um möglichst flexibel die Testsuite-Definitionen verwenden zu können, wird für die Implementierung der WSDL-Probe das Apache CXF-Framework [Foua] verwendet, mit dem es möglich ist, dynamische Web Service Clients zu erstellen, die keine vorherige Generierung statischer Java-Stubs Dateien erfordern. Die WSDL-Probe verwendet eine Testsuite-Definition, wie sie in Code-Listing B.3 (Seite 131) angegeben ist.

```
// Den Dynamic Client des Apache CXF Frameworks erstellen
JaxWsDynamicClientFactory factory = JaxWsDynamicClientFactory.newInstance();
// Mit den Definitionen aus der WSDL Testsuite den Client einrichten
// Die Definition wird aus Code-Listing 7.9 verwendet
Client client = factory.createClient(wsdlTestsuiteDef.getWsdlAddress());
ClientImpl clientImpl = (ClientImpl) client;
Endpoint endpoint = clientImpl.getEndpoint();
// ...
// Alle Operationen durchlaufen, die entsprechend der WSDL Testsuite
// ueberprueft werden sollen
for (Operation operation : wsdlTestsuiteDefinition.getOperations()) {
  // Den Namen der Operation aus der Testsuite-Definition auslesen
  QName opName = new QName(operation.getNamespace(), operation.getName());
  BindingOperationInfo boi = binding.getOperation(opName);
  // ...
  // Alle festgelegten Parameter einsetzen
  for (RequestParam requestParam : operation.getRequestParams()) {
    // ...
  }
  // Anfrage ausfuehren und Antwort analysieren
  Object[] result = client.invoke(opName, inputObject);
  // ...
  Class<?> wsResponseClass = wsResponse.getClass();
  // Den Rueckgabe-Typ mit der Vorgabe aus der Testsuite vergleichen.
  // Bei einem Fehler die Probe als fehlgeschlagen markieren
  if (!wsResponseClass.getCanonicalName().equals(operation.getResponse().
      getType())) {
    // ...
    setPassed(false);
    listener.probeExecutionCompleted();
  }
  // Den Rueckgabewert vergleichen
  if (!"".equals(operation.getResponse().getValue())) {
    // ...
  }
  // Bei erfolgreicher Ausfuehrung aller Tests den Probe-Status setzen
  setPassed(true);
  listener.probeExecutionCompleted();
}
```

Listing 7.10: Methodenrumpf der execute()-Methode der WSDL-Probe

Eine weitere Initial Probe ist die **GridFTP-Probe**, mittels der gezeigt wird, dass auch die

Sicherheitsmechanismen des Grids überprüft werden können. Dafür wird eine Anfrage an den GridFTP-Dienst des Globus Toolkits 4 gestellt. Die Probe verwendet das *Java Commodity Grid Kit* (CoG) [Las+01], mit dem es möglich ist, auf Grid-Dienste wie beispielsweise GridFTP oder GRAM-Job–Ausführung mittels Java zuzugreifen. Code-Listing 7.11 zeigt einen Ausschnitt der `execute()`-Methode der GridFTP-Probe.

```
public void execute() {
  // Zum Probe-Typ passende Testsuite erstellen
  GridFTPProbeTestsuiteDefinition testsuiteDef = new
      GridFTPProbeTestsuiteDefinition();
  testsuiteDef.parse(getProbeTrace().getTestsuite().getDefinition());
  // Felder aus der Testsuite auslesen
  URI sourceURI = new URI(testsuiteDef.getSourceURI());
  URI destinationURI = new URI(testsuiteDef.getDestURI());
  Task task = new FileTransferTask("MyTestTransferTask");
  // Spezifikationen der Test-Uebertragung definieren
  FileTransferSpecification spec = new FileTransferSpecificationImpl();
  // ...
  // Sicherheits-Kontext und Credentials einrichten
  // ...
  GSSCredential cred = manager.createCredential(GSSCredential.ACCEPT_ONLY);
  sourceSecurityContext.setCredentials(cred);
  // ...
  // Den Task ausfuehren. Da der TaskHandler die Tasks asynchron ausfuehrt,
  // muss der bereits erwaehnte Listener verwendet werden,
  // um reihenfolgeabhaengige Fehler zu vermeiden
  TaskHandler handler = new GenericTaskHandler();
  handler.submit(task);
}

public void statusChanged(StatusEvent event) {
  // Status der Uebertragung mittels des CoG Frameworks ueberwachen
  if (event.getStatus().getStatusCode() == Status.FAILED) {
    if (event.getStatus().getException() != null) {
      setComment("Probe execution failed");
      setPassed(false);
      // ...
      // Listener benachrichtigen, dass Probe beendet wurde
      listener.probeExecutionCompleted();
    }
  }
}
```

Listing 7.11: Auszug aus der GridFTP-Probe

Die in Code-Listing 7.12 gezeigte Investigative Probe ist deutlich übersichtlicher als die beiden Initial Probes. Da Investigative Probes wesentlich weniger Parameter erwarten als die Initial Probes und nie als erste Probe eines Probe-Trace ausgeführt werden, wird in diesen Probes auch keine Testsuite verwendet.

```
ip = new URL(ip).getHost();
// Wenn der Host erreichbar ist, Attribute der Probe setzen
if (InetAddress.getByName(ip).isReachable(timeout)) {
  setComment("Adresse [" + ip + "] ist mit Timeout [" + timeout + "]
      erreichbar");
  setPassed(true);
  return;
}
```

```
    setPassed(false);
    setComment("Adresse [" + ip + "] ist mit Timeout [" + timeout + "] nicht
        erreichbar");
  // Sofern ein Fehler auftritt, ist die Probe fehlgeschlagen!
  } catch (Exception e) {
    setPassed(false);
    setComment("Ein Fehler ist aufgetreten: " +
      DAGAHelper.stacktraceToString(e));
  }
```

Listing 7.12: Auszug aus der Investigative Probe Ping-Probe

7.4.5 Administration

Wie bereits bei der Abgrenzung des Prototyps (Kapitel 7.1) angegeben, wird die Administration nicht implementiert, da sie für die Erreichung der Ziele des Prototyps (Kapitel 7.1) keinen nennenswerten Beitrag leisten kann. Die nachträgliche Implementierung würde wie auch die der anderen Komponenten entsprechend dem Entwurfsmodell (Kapitel 6) durchgeführt.

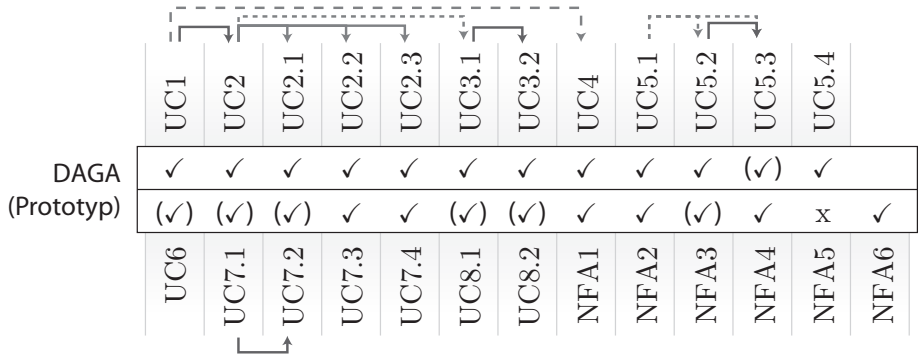
7.5 Evaluation der prototypischen Implementierung

Nachdem die ersten beiden Prozessschritte des Evaluation-Frameworks (Kapitel 3.2.5) in Kapitel 6.2 und 6.3 mit der Beschreibung der grundlegenden Konzepte und in Kapitel 6.4 mit deren Evaluation durchgeführt wurden, folgen nun die Evaluation der prototypischen Implementierung (Prozessschritt 3) und das Evaluations-Fazit (Prozessschritt 4).

Für die Evaluation des Prototyps wird zu den in Kapitel 3.2.5 genannten Symbolen ein weiteres eingeführt, die *mögliche* Erfüllung einer Anforderung, dargestellt durch ein eingeklammertes Häkchen ((✓)). Sie wird eingeführt, da es sich bei der evaluierten Implementierung um einen Prototypen handelt, der gemäß den in Kapitel 7.1 genannten Prämissen entwickelt wurde und daher keinen Anspruch auf Vollständigkeit erhebt. Ist eine Anforderung als mögliche Erfüllung, d.h. mit dem eingeklammerten Häkchen, markiert, bedeutet dies, dass die Anforderung im Prototypen nicht erfüllt ist, durch eine technisch und/oder zeitlich wenig aufwendige Erweiterung aber problemlos erfüllbar ist. So ist in der prototypischen Implementierung beispielsweise kein Administrationsbereich vorhanden. Dieser kann aber ohne Anpassungen des bestehenden Prototyps, also mit geringem zeitlichen und technischen Aufwand, hinzugefügt werden, da die dafür notwendigen Methoden bereits von der Knowledge-Base implementiert werden bzw. vorgesehen sind. Das Fehlen eines Administrationsbereichs wurde bei bestehenden Ansätzen (Kapitel 4) mit einer Nicht-Erfüllung (x) evaluiert, da es sich bei den betrachteten Implementierungen um Produktivsysteme handelt, bei denen die Prototyp-Prämissen (Kapitel 7.1) nicht anzuwenden sind und die Implementierungen diese Funktionalität daher enthalten *müssen*. Als Nicht-Erfüllung werden bei der Evaluation des Prototyps nur solche Anforderungen bewertet, die entweder gar nicht möglich sind oder deren Erfüllung eine nicht unerhebliche Anpassung und Bearbeitung des Prototyps nach sich ziehen würden.

Schritt 3 – Evaluation konkreter Implementierungen der Konzepte

Die Evaluation des Prototyps ergibt die in der folgenden Ergebnistabelle dargestellten Ergebnisse.



Evaluation Subsystem „A – Gewinnung von Verfügbarkeits–Informationen“

Der Prototyp und insbesondere die WSDL-Probe (Kapitel 7.4.4) haben gezeigt, dass es möglich ist, das Dienst-Interface einer zu überprüfenden Ressource programmatisch zu verwenden (*UC1*/✓) und dabei dessen Verfügbarkeit zu ermitteln (*UC2*/✓). Neben der Ermittlung einer technischen Nicht-Verfügbarkeit anhand einer Ping-Probe (*UC2.1*/✓) ist auch die Ermittlung einer organisatorischen Nicht-Verfügbarkeit anhand einer GridFTP-Probe (*UC2.2*/✓) einer Ressource möglich. Außerdem können für diese Use Cases weitere Investigative Probes hinzugefügt werden. Durch die detaillierte Auflistung der Ergebnisse innerhalb eines Probe-Trace, die beispielhaft in Abbildung 7.6 (Seite 110) gezeigt wurde, und die hohe Flexibilität der Testsuite-Definition kann auch der Grad einer Nicht-Verfügbarkeit ermittelt werden (*UC2.3*/✓), wie das Beispiel des erwarteten Ergebnisses einer Web Service Anfrage gezeigt hat. Die Ursache (*UC3.1*/✓) sowie der Trace aus verschiedenen (Teil-)Ursachen (*UC3.2*/✓) können bereits mit der Standard-UI in Form des Consumers abgerufen werden, eine Verwendung der Daten in einem individuellen Consumer ist aber ebenfalls möglich. Der Probe-Typ GridFTP-Probe zeigt beispielhaft, dass auch eine Überprüfung der Sicherheitsmechanismen möglich ist (*UC4*/✓), da eine der Fehlerursachen der GridFTP-Probe die Verwendung ungültiger Credentials sein kann.

Evaluation Subsystem „B – Verwendung von Verfügbarkeits-Informationen“

Der Prototyp ermöglicht die Verwendung der gewonnenen Verfügbarkeits-Informationen in allen für die Erfüllung der Anforderungen notwendigen Formen: die Knowledge-Base bietet mit der Operation `executeCustomQuery()` sowie der Verwendung von SQL die Möglichkeit, komplexe Anfragen in Form einer Anfragesprache auszuführen (*UC5.1*/✓). Zudem sind vielfältige Operationen vorhanden, mit denen Einzelabfragen durchgeführt werden können (*UC5.2*/✓), wie beispielsweise `getProbeById()`. Das Senden von Push-Anfragen, also die Registrierung als Observer, ist nach wenigen Anpassungen des Prototyps ebenfalls möglich (*UC5.3*/(✓)), da das im Entwurfsmodell vorgestellte Datenmodell vollständig implementiert ist und nur die Funktionalitäten für das An- und Abmelden sowie die Benachrichtigung der Observer in der Knowledge-Base implementiert werden müssen. Der Screenshot in Abbildung 7.6 zeigt die Standard-UI von DAGA, über die Daten ausgewählt werden können, die anschließend von der Standard-UI graphisch aufbereitet dargestellt werden (*UC5.4*/✓).

Evaluation Subsystem „C – System-Administration“

Wie zu Beginn der Evaluation erläutert, ist die Administration der Daten durch die Erweiterung des Prototyps entsprechend den Anforderungen erfüllbar (*UC6*/(✓)), ebenso das Hinzufügen neuer zu überwachender Ressourcen (*UC7.1*/(✓)), wobei für die Erfüllung lediglich das in Kapitel 6.3 modellierte Vorgehen umgesetzt werden muss: *UC7.1* ist erfüllbar, wenn im Administrationsbereich eine Liste der vorhandenen Ressourcen angezeigt wird und anschließend für einen Eintrag dieser Liste eine neue Testsuite angelegt werden kann. Die Liste der vorhandenen Ressourcen kann mittels einer Web Service Anfrage an den Globus Index Service, die anschließend mit *XPath* [CD99] ausgewertet wird, erstellt werden. Das Hinzufügen einer neuen VO wird durch eine Anpassung bzw. Erweiterung der WSDL-Probe erfüllbar (*UC7.2*/(✓)), da dafür lediglich die Definition der Testsuite erweitert werden muss. *UC7.2* kann gemeinsam mit der bereits untersuchten Anforderung *UC4* erfüllt werden. Neue Fehlertypen können durch die Implementierung der im Entwurfsmodell gegebenen Flexibilität ohne Weiteres hinzugefügt werden (*UC7.3*/✓), indem die Probes erweitert und angepasst werden. So kann beispielsweise die Ping-Probe dahingehend um einen Fehlertyp erweitert werden, dass der vorhandene Fehler differenziert wird in „Ressource ist nicht erreichbar“ und „Ressource ist innerhalb eines vorgegebenen Zeitfensters nicht erreichbar“. Auch das Hinzufügen neuer Ressourcentypen ist möglich (*UC7.4*/✓), da die hierfür notwendigen Klassen im Paket `daga.common.elements` bereits so implementiert und eingesetzt werden, dass sie bei einer Erweiterung schnell in allen Komponenten verfügbar sind. Technische (*UC8.1*/(✓)) und inhaltliche (*UC8.2*/(✓)) Selbsttests sind durch die Erweiterung um neue Probe-Typen möglich.

Evaluation nicht-funktionaler Anforderungen

Das vom Entwurfsmodell vorgegebene Datenschema wird vom Prototypen ohne Anpassungen umgesetzt, wodurch ein einheitliches und etabliertes Datenschema verwendet wird (*NFA1*/✓). Da die Testparameter in einer Testsuite gekapselt sind und keine Anpassungen an den überwachten Ressourcen, beispielsweise durch Installation zusätzlicher Software oder die Erweiterung der WSDL-Dokumente, notwendig sind, hat der Prototyp einen extrem geringen Intrusiveness-Wert (*NFA2*/✓). Die Einhaltung der Sicherheitsstandards ist im Prototyp nicht implementiert, kann aber durch den Einsatz von WS-Security leicht hinzugefügt werden (*NFA3*(✓)). Die hohe Aktualität der Daten ist gewährleistet (*NFA4*/✓), da sowohl die Ressourcen häufig überprüft als auch die gewonnenen Daten regelmäßig an die Knowledge-Base übertragen werden und dort den Consumern zur Verfügung stehen. Zudem können beide Zeitpunkte an die Eigenschaften des Grids angepasst werden: die Häufigkeit der Überprüfungen einer Ressource können über das Feld `executeEveryMinutes` der Testsuite gesteuert werden (Kapitel 6.2.1), die Intervalle der Übertragungen an die Knowledge-Base werden von jeder Probe-Station aus einer Konfigurationsdatei gelesen. Eine Zeitsynchronisation ist im Prototypen nicht enthalten und auch nicht ohne größeren Aufwand hinzuzufügen (*NFA5*/x). Dieser Punkt wird daher in den Erweiterungsmöglichkeiten in Kapitel 8.2.1 aufgegriffen. Wie vom Entwurfsmodell (Kapitel 6) vorgegeben, erfolgt aufgrund der dort modellierten Architektur auch im Prototypen keine Daten-Migration oder Daten-Transformation (*NFA6*/✓).

Schritt 4 – Fazit „DAGA“

Wie aus der Ergebnistabelle erkennbar ist, erfüllt der Prototyp von DAGA bis auf *NFA5* alle analysierten Anforderungen. Müssen Erweiterungen des Prototyps durchgeführt werden, beschränken sich diese meist auf die Erstellung neuer Probe-Typen oder die Anpassung der Objekte im Package `daga.common.elements`, wobei diese Erweiterungen explizit in der Modellierung (Kapitel 6) und im Prototypen vorgesehen sind. Es sind also nur punktuelle und vorgesehene Erweiterungen, nicht aber grundsätzliche Veränderungen am Prototypen notwendig, um alle analysierten Anforderungen zu erfüllen. Damit kann abschließend festgestellt werden, dass DAGA einen Ansatz darstellt, der alle Anforderungen, die an ein System zur Gewinnung von Verfügbarkeits-Informationen in Grid-Infrastrukturen gestellt werden, erfüllt.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

In der vorliegenden Arbeit wurde gezeigt, dass Verfügbarkeits-Informationen für Grids von grundlegender Bedeutung sind, da sie in allen Betriebsbereichen und bei der Grid-Job–Ausführung eine zentrale Rolle für eine korrekte und optimale Funktionsweise einnehmen (Kapitel 2.2). Ihrer Gewinnung muss daher besondere Aufmerksamkeit geschenkt werden. Dabei ergibt sich aufgrund der Eigenschaften eines Grids (Kapitel 2.1) die besondere Herausforderung, dass es nicht nur technische Gründe für die Nicht-Verfügbarkeit einer Ressource gibt, wie beispielsweise eine defekte Netzkarte, sondern auch die für Grids besonderen organisatorischen Gründe, wie beispielsweise eine Überschreitung des Nutzungskontingents für die zugehörige VO.

Vor diesem Hintergrund wurde eine umfassende Anforderungsanalyse an ein System zur Gewinnung von Verfügbarkeits-Informationen in Grid-Infrastrukturen durchgeführt (Kapitel 3), um bestehende und in Grids bereits eingesetzte Ansätze evaluieren zu können. Bei der durchgeführten Evaluation hat sich gezeigt, dass die analysierten Anforderungen von bestehenden Ansätzen zur Gewinnung von Verfügbarkeits-Informationen in Grid-Infrastrukturen nicht oder nur teilweise erfüllt werden (Kapitel 4), was auf die fehlende Ausrichtung auf die spezifischen Eigenschaften eines Grids zurückzuführen ist.

Aufgrund der gefundenen Fehlstellen in bestehenden Ansätzen wurde in dieser Arbeit mit DAGA ein neuer Ansatz entwickelt, der die systemimmanenten Schwachstellen der bestehenden Ansätze nicht mehr aufweist. Der neue Ansatz, bei dessen Entwicklung alle Schritte des Software-Entwicklungs–Prozesses durchgeführt wurden (Kapitel 5.1.1), verfolgt dabei einen völlig neuen Betrachtungswinkel, bei dem nicht einzelne technische Endgeräte wie Massenspeicher oder CPUs überwacht werden, sondern die im Grid angebotenen Dienste über ihre nach außen sichtbaren Dienst-Schnittstellen, da diese auch von den in Grids ausgeführten Grid-Jobs (Kapitel 2.1.4) verwendet werden. Der neue Ansatz folgt somit der in Grids vorherrschenden Dienst-Orientierung (Kapitel 2.1, 5.2.1). Ebenfalls neu bei der Gewinnung von Verfügbarkeits-Informationen ist die Verfolgung eines aktiven anstelle eines bisherigen passiven Ansatzes. Dies ist durch den Einsatz von Active Probing möglich (Kapitel 4.2.2), einem Vorgehen, bei dem kleine Testpakete, sogenannte Probes, an die zu überwachenden Ressourcen geschickt werden und die erhaltene Antwort unmittelbar ausgewertet wird. Wird dabei ein Fehler erkannt, werden weitere Probes mit möglichst hohem Informationsgewinn autonom ausgewählt und an die fehlerhafte Ressource geschickt, bis die Ursache für den Fehler eingegrenzt oder gefunden wurde.

Aufbauend auf diesen neuen Kernkonzepten und den analysierten Anforderungen wurden das technische Konzept und die Architektur des neuen Ansatzes entwickelt (Kapitel 6) und prototypisch implementiert (Kapitel 7).

8.2 Weiterführende Themen

Auch wenn DAGA bis auf *NFA5* alle analysierten Anforderungen erfüllt (Kapitel 7.5), bestehen dennoch verschiedene Aspekte, die weiterentwickelt werden können. Sie sind nicht in der vorliegenden Arbeit bearbeitet worden, da es entweder den Rahmen dieser Arbeit überstiegen hätte (Kapitel 8.2.1) oder weil sie keine grundlegende Anforderung für die Gewinnung von Verfügbarkeits-Informationen darstellen und mehr als Erweiterung zu verstehen sind (Kapitel 8.2.2).

8.2.1 Verbesserungspotentiale der prototypischen Implementierung

Da es sich bei der in Kapitel 7 vorgestellten Implementierung um einen Prototypen und nicht um eine für den Produktiveinsatz entwickelte und getestete Software handelt, wurden bei der Implementierung des Prototyps Aspekte vernachlässigt, die keine unmittelbare Bedeutung für die Erfüllung der analysierten Anforderungen (Kapitel 3) oder die Überprüfung der Machbarkeit der entwickelten technischen Konzepte (Kapitel 6) haben. Diese Aspekte werden im Folgenden kurz erläutert, und es wird jeweils ein möglicher Lösungsansatz vorgestellt, wobei der Lösungsansatz als Hinweis, nicht aber als bereits evaluierter Weg zu verstehen ist.

Minimierung der Intrusiveness

Sowohl das Entwurfsmodell als auch die prototypische Implementierung von DAGA wurden so entwickelt, dass sie die Anforderung nach einer geringen Intrusiveness erfüllen. Dennoch könnte die Intrusiveness weiter gesenkt werden: wie in Kapitel 6.3 beschrieben, werden die Initial Probes in einem festgelegten Zeitintervall an die verschiedenen Ressourcen geschickt. Dieses Zeitintervall könnte dynamisch ermittelt werden, beispielsweise indem das Intervall nach einer festgelegten Menge von erfolgreich ausgeführten Initial Probes schrittweise gesenkt wird oder die Ressourcen einen Expires-Wert senden, wie oft die Ressource selber meint, überprüft werden zu müssen. Bei der Wahl des Intervallwerts sollte berücksichtigt werden, dass ein relativ kleines Intervall, also eine häufige Ausführung, zu mehr Netz- und Systemlast führt, aber nicht zwangsläufig zu Performance-Verbesserungen des gesamten Grids durch mehr Verfügbarkeits-Informationen [Ios+07]. Andererseits ist zu beachten, dass die durchschnittliche „mean time between failures“ bei 12 Minuten auf Grid-Ebene, fünf Stunden auf Cluster-Ebene und zwei Tagen auf Computer-Node-Ebene liegt [Ios+07].

Alternativ könnte auch eine Priorisierung von Ressourcen aufgestellt werden und Ressourcen mit geringer Priorität seltener überprüft werden als hochpriorisierte Ressourcen. Dadurch würden insgesamt weniger Probes versandt und ausgeführt, was einen positiven Effekt auf die Lasterzeugung und damit die Intrusiveness im gesamten Grid hätte. Weitere Ansätze, die Probes hinsichtlich einer geringeren Intrusiveness zu verbessern, gibt beispielsweise [SRL11].

Maßnahmen für Sicherheit und Zuverlässigkeit

Die GSI, die bei der Entwicklung der technischen Konzepte berücksichtigt wurde (Kapitel 6) und damit die entsprechende Anforderung nach Verwendung von Grid-Sicherheitsstandards erfüllt wird, wurde bei der prototypischen Implementierung nicht umgesetzt, da sie nicht unmittelbar mit der Überprüfung der Machbarkeit des technischen Konzepts zusammenhängt. Der Prototyp kann somit um GSI-Mechanismen erweitert werden.

Eine zusätzliche Verbesserung des Prototyps betrifft die Integritätssicherung der einzelnen Probes und der ermittelten Ergebnisse. Da mit diesen Daten zu einem späteren Zeitpunkt Service-Level-Agreements (SLA) untermauert werden sollen, ist hier eine Sicherung der Integrität der Daten notwendig, um beispielsweise sicherzustellen, dass einzelne Probes nicht abgefangen und verändert werden.

Beide Bereiche können beispielsweise mittels der *Web Service Reliable Messaging Technology* (WS-RM) [Dav+08], *WS-Security* und *WS-PolicyAssertion* erreicht werden, da mit deren Kombination eine sichere und garantierte Nachrichtenübertragung gewährleistet ist [HH10] und auf die Belange in einem Grid, beispielsweise bei der Autorisierung in VOs [WH03], eingegangen werden kann. Ebenfalls denkbar wäre die Nutzung von Policy Decisions Points (PDP), beispielsweise bei der Anfrage von Verfügbarkeits-Informationen durch verschiedene Consumer.

Verwendung einer Zeitsynchronisation

Die einzige Anforderung, die die prototypische Implementierung von DAGA nicht erfüllt, ist die Verwendung eines Zeitsynchronisationsmechanismus (Kapitel 7.5). Um auch diese Anforderung und somit alle analysierten Anforderungen zu erfüllen, könnte beispielsweise das Network Time Protocol [Mil92] verwendet werden. Der Mechanismus müsste zwischen den Consumern und der Knowledge-Base eingerichtet werden, also zwischen der Gewinnung und der Verteilung der gewonnenen Verfügbarkeits-Informationen.

Verbesserung der Performance

Die Performance kann maßgeblich mittels Caching-Funktionalitäten verbessert werden, wie die Studie [ZFS03] herausgefunden hat. Durch die Architektur von DAGA werden einige Anknüpfungspunkte für Caching, wie sie in [ZS05] genannt werden, bereits vermieden. Dennoch gibt es einige Stellen, an denen Caching eingeführt werden könnte, beispielsweise auf Consumer-Seite. Eine weitere Möglichkeit wäre, die Knowledge-Base in zwei Datenbank-Systeme aufzuteilen, ein vorgelagertes, schnelles System, das häufige Anfragen vorhält, und eine große, verteilte Datenbank im Hintergrund.

8.2.2 Hinzufügen neuer Funktionalitäten

Wie eingangs erläutert gibt es verschiedene Funktionalitäten, durch die DAGA sinnvoll erweitert werden könnte, die in der vorliegenden Arbeit aber nicht näher untersucht wurden, da sie nicht unmittelbar für die Gewinnung von Verfügbarkeits-Informationen notwendig sind. Diese Funktionalitäten werden im Folgendem kurz vorgestellt.

Ableitung von Zuverlässigkeits-Informationen

Aus den gewonnenen Verfügbarkeits-Informationen könnten in einem nachfolgenden Schritt Zuverlässigkeits-Informationen abgeleitet werden, also Aussagen über „the ability of a system to avoid service failures that are more frequent and more severe than is acceptable“ [Avi+04] getroffen werden. Da Probe-Traces und Zeitserien gebildet werden können (Kapitel 6), ist die notwendige Datengrundlage für die Ableitung solcher Informationen bereits vorhanden. Zuverlässigkeit fasst als Oberbegriff Sicherheit und Fehlertoleranz zusammen [BGN02].

Last-Simulation

Mit DAGA ist es möglich, einzelne Ressourcen gezielt unter Stress zu setzen und deren Verhalten zu überprüfen. Dafür könnte ein neuer Probe-Typ eingeführt werden, der in einem besonders kurzen Intervall an die zu testende Ressource geschickt und dort ausgeführt wird.

Aussagen über die Erreichung von Service-Level–Agreements

Da bei der Entwicklung von DAGA eine globale Sicht auf die Ressourcenlandschaft eingeführt wurde (Kapitel 5.1.1), ist es mittels der abgeleiteten Zuverlässigkeits-Informationen und der Ergebnisse der Stresstests in einem weiteren Schritt möglich, Aussagen über die Erreichung von Service-Level-Agreements zu treffen.

2-Phasen–Scheduling und Job-Monitoring

Durch die Flexibilität von DAGA gegenüber den eingesetzten Probe-Typen und durch die Dienst-Orientierung wäre ein 2-Phasen–Scheduling denkbar. Dafür könnte ein Probe-Typ eingeführt werden, der die lokalen Scheduler einer Ressource fragt, ob sie zu einem bestimmten Zeitpunkt einen Job mit spezifischen Eigenschaften und Anforderungen ausführen könnten, beispielsweise drei CPUs für vier Tage. Diese Probes würden in der ersten Phase aufgrund statistischer Auswertungen an relevante Ressourcen geschickt und die Ergebnisse in der Knowledge-Base gespeichert. In einer zweiten Phase könnte der Grid-Scheduler die Ergebnisse aus der zentralen Knowledge-Base abfragen statt einzelne Ressourcen überprüfen und validieren zu müssen.

Ebenso denkbar wäre ein weiterer Probe-Typ, mit dem der Status eines bestimmten Jobs beim lokalen Scheduler abgefragt werden könnte. Welche Jobs wo mit welchem Zustand laufen könnte ebenfalls bei der zentralen Knowledge-Base abgefragt werden. Diese Funktionalität wäre besonders im Globus Toolkit sehr hilfreich, da Job-Monitoring und -Discovery hier nur sehr eingeschränkt nutzbar sind [BFR07].

Formulierung von Policies für die Nutzung der Verfügbarkeits-Informationen

Eine Policy ist ein Regelsatz, der die beteiligten Subjekte (z.B. Consumer) und Objekte (z.B. Knowledge-Base) definiert und deren Verhältnisse beschreibt [Fos+98]. Eine Policy stellt somit allgemeine Anwendungs- und Verhaltensregeln auf, beispielsweise, wer zu welchem Zeitpunkt auf welche Weise Verfügbarkeits-Informationen von der Knowledge-Base abrufen darf oder welche Angaben ein Subjekt benötigt, um sich bei einem Objekt zu authentifizieren [WH03]. Auch aus rechtlicher Sicht haben Policies eine wichtige Funktion: wegen der Integration verschiedenster realer Organisationen aus theoretisch unterschiedlichen Ländern in einer VO (Kapitel 2.1.3) müssen viele unterschiedliche und meist nur national geltende Bestimmungen eingehalten werden. Durch die Transparenz, die durch die Einigung auf gemeinsame Policies entsteht, kann die Einhaltung dieser Bestimmungen besser kontrolliert und nachverfolgt werden [Fos+98].

Um DAGA für weitere Funktionalitäten vorzubereiten und den Zugriff auf die gewonnenen Verfügbarkeits-Informationen feingranular festlegen zu können, könnten verschiedene Policies definiert werden. Das Policy-Enforcement würde auf der jeweils betroffenen Komponente durchgeführt werden, beispielsweise unter Zuhilfenahme des in [WH03] vorgestellten Vorgehens.

Anhang

A UML-Diagramme

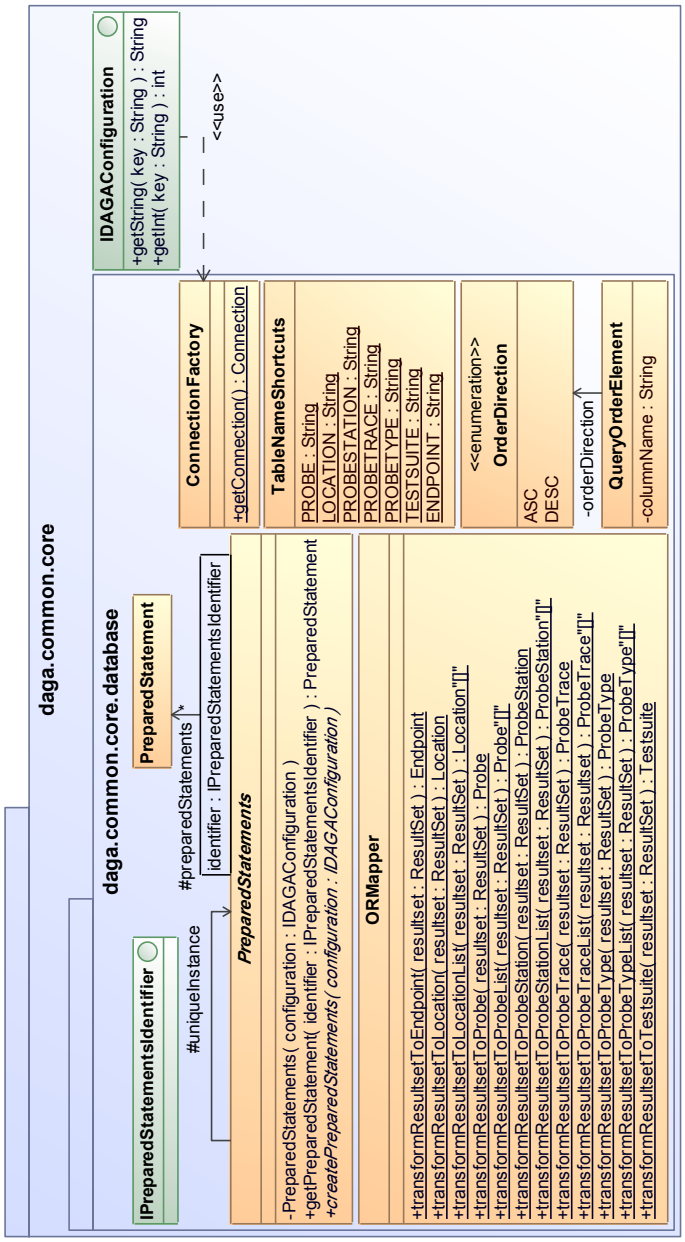


Abbildung A.1: Klassen, die von allen DAGA-Komponenten in technischer Hinsicht verwendet werden

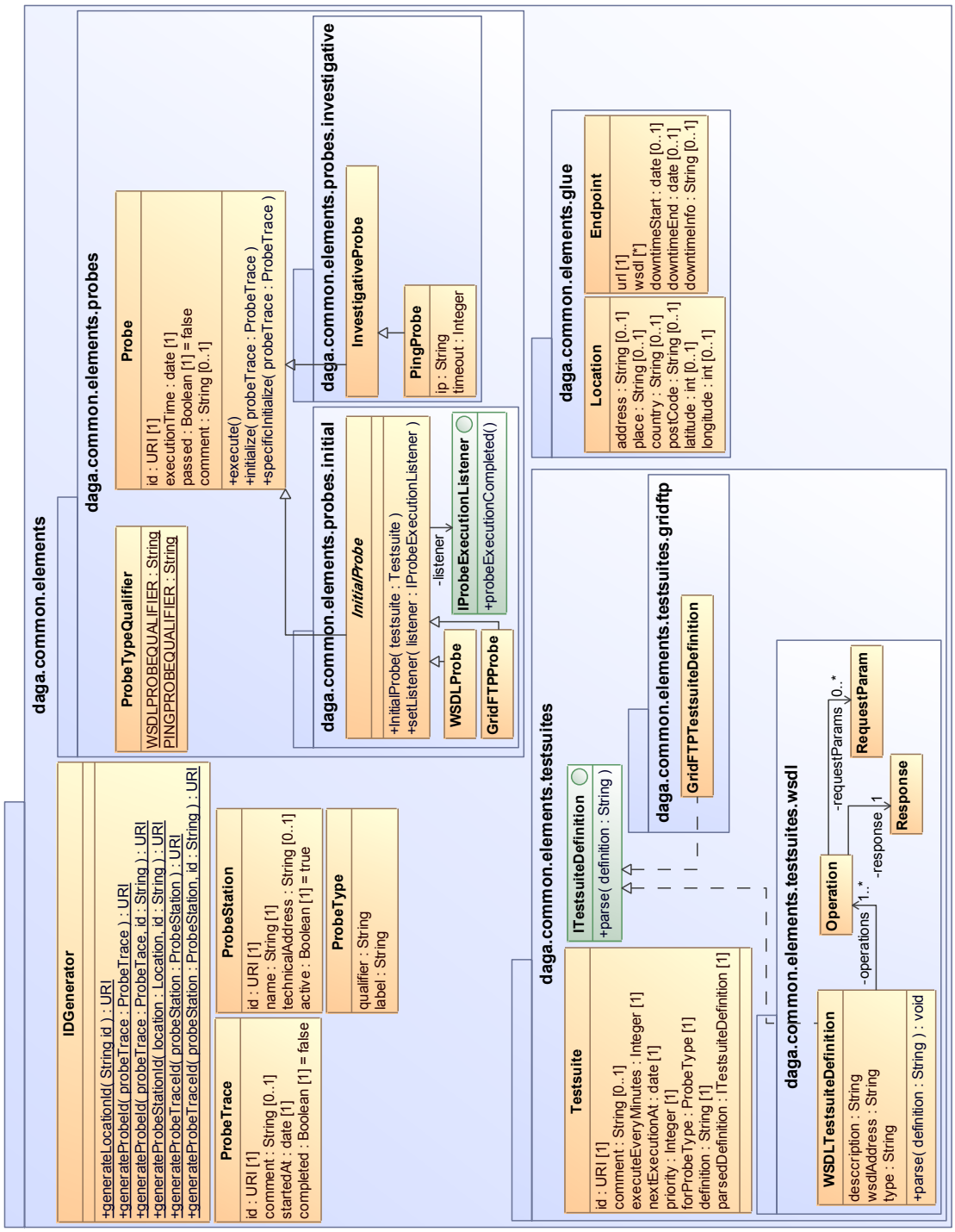


Abbildung A.2: Klassen, die von allen DAGA-Komponenten für eine einheitliche Semantik verwendet werden

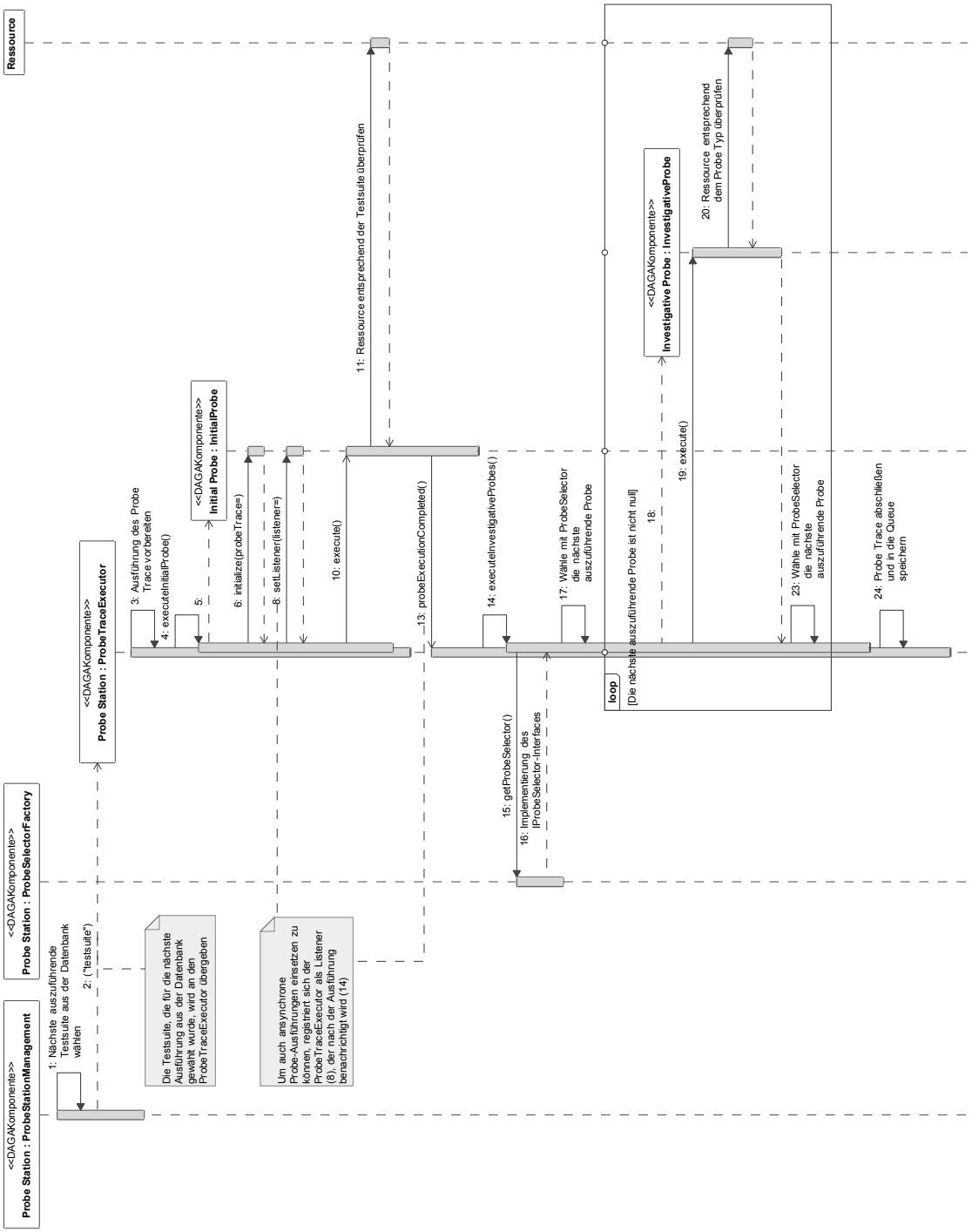


Abbildung A.3: Detailliertes Sequenzdiagramm der Ausführung einer Testsuite auf einer Probe-Station

B Code-Listings und Deployment-Übersicht

Code-Listings

```
apt-get install mysql-server mysql-client
```

Listing B.1: Einrichtung eines MySQL-Servers auf Debian 6

```
apt-get install apache2 apache2-doc apache2-mpm-prefork
apt-get install apache2-utils apache2-suexec libexpat1 ssl-cert
apt-get install php5 php5-common php5-curl php5-dev php5-gd php5-idn
apt-get install php5-imagick php5-mysql php5-xcache libapache2-mod-php5
apt-get install phpmyadmin
```

Listing B.2: Einrichtung von phpMyAdmin auf Debian 6 für die Verwaltung der Datenbank

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="testsuiteSchema.xsd">
  <description>
    Die Testsuite ueberprueft die Methode AddNumbers,
    indem fuer zwei Eingabeparameter die Summe berechnet
    und das Ergebnis mit der Vorgabe verglichen wird
  </description>
  <wsdladdress>http://192.168.56.101:8080/wsrf/services/NameDesDienstes?wsdl
    </wsdladdress>
  <type>WSDL-Probe</type>
  <binding namespace="http://mmm/namedesdienstes/webservice/bindings"
    name="NameDesDienstesPortTypeSOAPBinding" />
  <testoperations>
    <!-- Es wird die Operation AddNumbers ueberprueft ... -->
    <operation namespace="http://mmm/namedesdienstes/webservice"
      name="AddNumbers">
      <!-- ... wobei der Request zwei Parameter enthaelt... -->
      <request>
        <param>
          <propertyName>firstNumber</propertyName>
          <paramType>Integer</paramType>
          <value>5</value>
        </param>
        <param>
          <propertyName>secondNumber</propertyName>
          <paramType>Integer</paramType>
          <value>5</value>
        </param>
      </request>
      <!-- ... und die Antwort mit diesem Wert verglichen wird -->
      <response>
        <name>additionResult</name>
        <type>java.lang.Integer</type>
        <value>10</value>
      </response>
    </operation>
  </testoperations>
</testsuite>
```

Listing B.3: Beispielhafte Testsuite-Definition für eine WSDL-Probe

Deployment-Übersicht

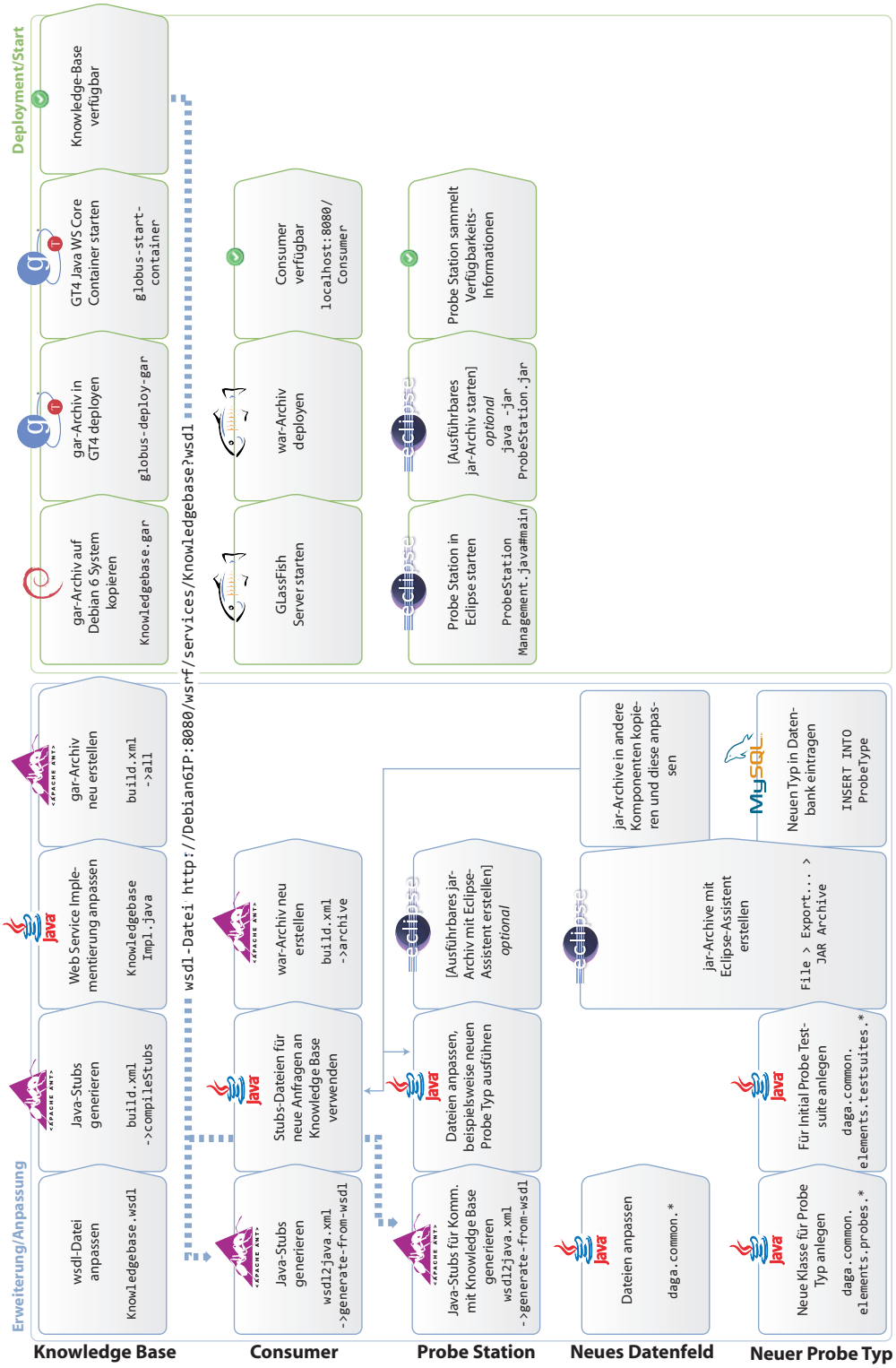


Abbildung B.4: Notwendige Schritte bei der Anpassung und dem Deployment der einzelnen DAGA-Komponenten

C Datenträger mit der prototypischen Implementierung

Die CD enthält die prototypische Implementierung von DAGA sowie die Quelldateien dieser Arbeit.

Der Prototyp besteht aus den Eclipse-Projekten der Komponenten *Consumer*, *Knowledge-Base* und *Probe-Station*, dem Eclipse-Projekt der *allgemeinen Elemente* und einem Web Service, der für Testzwecke im Globus Toolkit 4 deployt und anschließend mit DAGA analysiert werden kann. Die in Code-Listing B.3 (Seite 131) angegebene Testsuite ist für die Funktionalitäten dieses Web Service angelegt. Die Verzeichnisstrukturen der Eclipse-Projekte wurden bereits im Rahmen der prototypischen Implementierung (Kapitel 7.4) erläutert.

×

Abbildungsverzeichnis

Literatur

[Ale+08] M. Alef u. a. „Integration of Multiple Middlewares on a Single Computing Resource“. In: *Future Generation Computer Systems* 25 (2008), S. 268–274.

[Alla] Globus Alliance. *About the Globus Toolkit.* http://www.globus.org/toolkit/about.html.

[Allb] Globus Alliance. *Globus Toolkit 4 – GT4 Admin Guide – Chapter 11. WS GRAM Admin Guide.* http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch11.html#id2565123.

[Allc] Globus Alliance. *Globus Toolkit 5 – Globus Toolkit Official Documentation – GT 5.0.0 Release Notes.* http://www.globus.org/toolkit/docs/5.0/5.0.0/rn/#rn.

[Alld] Globus Alliance. *GT4 Admin Guide – Chapter 3. Software Prerequisites – 3. Platform Notes.* http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch03.html#s-platform.

[And+03] S. Andreozzi u. a. „GridICE: a Monitoring Service for the Grid“. In: *Proceedings of the Third Cracow Grid Workshop (CGW2003).* 2003, S. 220–226.

[And+08] Sergio Andreozzi u. a. *GLUE v. 2.0 – Reference Realizations to Concrete Data Models.* Techn. Ber. Open Grid Forum, 2008.

[And+09] Sergio Andreozzi u. a. *GLUE Specification v. 2.0.* Techn. Ber. GFD-R-P.147. Open Grid Forum, 2009.

[ANF11] Noriyani Mohd Zin an Ahmad Noraziah und Ainul Azila Che Fauzi. „Neighbour Replication on Grid Deadlock Detection Framework“. In: *Digital Enterprise and Information Systems.* Hrsg. von Ezendu Ariwa und Eyas El-Qawasmeh. Springer, 2011, S. 350–358.

[Arp95] R. Arpaci. „The Interaction of Parallel and Sequentiell Workloads on a Network of Workstations“. In: *International Conference on Measurement and Modeling of Computer Systems.* 1995, S. 267–278.

[Atk+04] Malcolm Atkinson u. a. „Data Access, Integration, and Management“. In: Hrsg. von Ian Foster und Carl Kesselman. 2. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. Kap. 22.

[Aus+02] Daniel Austin u. a. *Web Services Architecture Requirements – W3C Working Draft 11 October 2002.* http://www.w3.org/TR/2002/WD-wsa-reqs-20021011. 2002.

[Avi+04] Algirdas Avižienis u. a. „Basic Concepts and Taxonomy of Dependable and Secure Computing“. In: *IEEE Transactions on Dependable and Secure Computing* 1 (2004), S. 11–33.

[Bal+01] Zoltán Balaton u. a. *Use Cases and the Proposed Grid Monitoring Architecture*. Techn. Ber. LDS-1/2001. Computer und Automation Research Institute of the Hungarian Academy of Sciences, 2001.

[Bal+06] Keith Ballinger u. a. *Basic Profile Version 1.1*. Techn. Ber. Web Services Interoperability Organization (WS-I), 2006.

[Bau+06] Timo Baur u. a. „Monitoring-Anwendungsfälle und Anforderungen“. In: *Ergebnisse der Studie und Anforderungsanalyse in den Fachgebieten Monitoring, Accounting, Billing bei den Communities und Ressourcenanbietern im D-Grid*. Hrsg. von C.-P. Rückemann. 2006, S. 45–63.

[Bau+08] Timo Baur u. a. *Middleware-übergreifendes Monitoring: Evaluierung und Auswahl von Komponenten*. Techn. Ber. D-Grid – D-MON Forschungsprojekt, 2008.

[Bau+09a] Timo Baur u. a. „Adopting GLUE 2.0 for an Interoperable Grid Monitoring System“. In: *Proceedings of the Cracow Grid Workshop (CGW08)*. Cracow, Poland, 2009.

[Bau+09b] Timo Baur u. a. „An Interoperable Grid Information System for Integrated Resource Monitoring Based on Virtual Organizations“. In: *Journal of Grid Computing* 7.3 (2009), S. 319–333.

[BFR07] Timo Baur, Nils gentschen Felde und Helmut Reiser. *Konzepte zum Monitoring im Kern D-Grid*. Techn. Ber. Leibniz Rechenzentrum – Munich Network Management Team, 2007.

[BGN02] Zoltán Balaton, Gábor Gombás und Zsolt Németh. „Information System Architecture for Brokering in Large Scale Grids“. In: *Proceedings of the DAPSYS 2002 on Parallel and Distributed Systems: Cluster and Grid Computing*. Kluwer, Linz, 2002, S. 57–65.

[BHP02] Franck Bonnassieux, Robert Harakaly und Pascale Primet. „MapCenter: An Open Grid Status Visualization Tool“. In: *Proceedings of the ISCA 15th International Conference on Parallel and Distributed Computing Systems*. 2002, S. 2–3.

[BKS05] Rüdiger Berlich, Marcel Kunze und Kilian Schwarz. „Grid Computing in Europe: from Research to Deployment“. In: *Proceedings of the 2005 Australasian Workshop on Grid Computing and e-Research*. Bd. 44. ACSW Frontiers '05. Newcastle, Australia: Australian Computer Society, Inc., 2005, S. 21–27.

[BLFM05] T. Berners-Lee, R. Fielding und L. Masinter. *RFC 3986 – Uniform Resource Identifier (URI): Generic Syntax*. Techn. Ber. Network Working Group, 2005.

[BMA09] Asgarali Bouyer, Ehsan Mohebi und Abdul Hanan Abdullah. „Using Self-Announcer Approach for Resource Availability Detection in Grid Environment“. In: *Proceedings of the 2009 Fourth International Multi-Conference on Computing in the Global Information Technology*. Washington, DC, USA: IEEE Computer Society, 2009, S. 151–156.

[Boo+04] David Booth u. a. *Web Services Architecture*. W3C Working Group Note 11. World Wide Web Consortium, 2004. URL: `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`.

[Bra97] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels.* http://www.ietf.org/rfc/rfc2119.txt. 1997.

[BRM01] Mark Brodie, Irina Rish und Sheng Ma. „Optimizing Probe Selection for Fault Localization". In: *12th International Workshop on Distributed Systems: Operations and Management DSOM.* Hrsg. von O. Festor und A. Pras. 2001.

[Bro+02] Mark Brodie u. a. *Active Probing.* Techn. Ber. IBM T.J. Watson Research, 2002.

[Bro+03] Mark Brodie u. a. „Active Probing Strategies for Problem Diagnosis in Distributed Systems". In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, S. 1337–1338.

[BS02] Mark Baker und Garry Smith. „GridRM: A Resource Monitoring Architecture for the Grid". In: *Proceedings of the Third International Workshop on Grid Computing.* GRID '02. Springer, 2002, S. 268–273.

[BS03] Mark Baker und Garry Smith. „GridRM: An Extensible Resource Monitoring System". In: *Proceedings of the IEEE International Cluster Computing Conference.* IEEE Computer Society, 2003.

[BSV03] Ranjita Bhagwan, Stefan Savage und Geoffrey M. Voelker. „Understanding Availability". In: *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03).* Bd. 2735. 2003, S. 256–267.

[Bur07] Herbert Burbiel. *SOA & Webservices in der Praxis – Service Oriented Architecture mit XML, SOAP, .NET, Java & Co.* FRANZIS Professional Series, 2007.

[Bus+96] Frank Buschmann u. a. *Pattern-Oriented Software Architecture.* Bd. 1. Chichester, UK: Wiley & Sons, 1996.

[But+00] Randy Butler u. a. „A National-Scale Authentication Infrastructure". In: *IEEE Computer* 33.12 (2000), S. 60–66.

[Car72] Gary Carlson. „How to Save Money with Computer Monitoring". In: *Proceedings of the ACM Annual Conference.* Bd. 2. ACM '72. New York, NY, USA: ACM, 1972, S. 1018–1023.

[CD97] Surajit Chaudhuri und Umeshwar Dayal. „Data Warehousing and OLAP for Decision Support". In: *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data.* SIGMOD '97. ACM, 1997, S. 507–508.

[CD99] James Clark und Steve DeRose. *XML Path Language (XPath) – Version 1.0.* Techn. Ber. World Wide Web Consortium, 1999.

[Cer02] Ethan Cerami. *Web Services Essentials – Distributed Applications with XML-RPC, SOAP, UDDI & WSDL.* O'Reilly Media, Inc., 2002.

[CFK04] Karl Czajkowski, Ian Foster und Carl Kesselman. „Resource and Service Management". In: Hrsg. von Ian Foster und Carl Kesselman. 2. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. Kap. 18.

[Che+05] Ling-Jyh Chen u. a. „Ad Hoc Probe: Path Capacity Probing in Wireless Ad Hoc Networks". In: *Proceedings of the First International Conference on Wireless Internet.* IEEE Computer Society, 2005, S. 156–163.

[Chr+01] Erik Christensen u. a. *Web Services Description Language (WSDL) 1.1.* W3C Note. World Wide Web Consortium, 2001. URL: `http://www.w3.org/TR/wsdl`.

[Chu+04] Greg Chun u. a. „Benchmark Probes for Grid Assessment“. In: *18th International Parallel and Distributed Processing Symposium (IPDPS 2004).* IEEE Computer Society, 2004, S. 26–30.

[CLL02] Jacky Chu, Kevin Labonte und Brian Neil Levine. „Availability and Locality Measurements of Peer-To-Peer File Systems“. In: *Proceedings of ITCom: Scalability and Traffic Control in IP Networks.* 2002.

[Cona] Condor Team. *Condor – High Troughput Computing.* `http://www.cs.wisc.edu/condor/`.

[Conb] Condor Team. *Condor Version 7.5.6 Manual.* `http://www.cs.wisc.edu/condor/manual/v7.5/`. 09.04.2011.

[Coo+03] Andy Cooke u. a. „R-GMA: An Information Integration System for Grid Monitoring“. In: *Proceedings of the 10th International Conference on Cooperative Information Systems.* 2003, S. 462–481.

[Cui+06] Jianqun Cui u. a. „A Resource Discovery Algorithm with Probe Feedback Mechanism Based on Advance Reservation“. In: *Proceedings of the Fifth International Conference on Grid and Cooperative Computing, 2006. GCC 2006.* 2006, S. 281–286.

[Cza+01] Karl Czajkowski u. a. „Grid Information Services for Distributed Resource Sharing“. In: *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing.* Washington, DC, USA: IEEE Computer Society, 2001, S. 181–194.

[Dav+08] Doug Davis u. a. *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1.* Techn. Ber. OASIS, 2008. URL: `http://docs.oasis-open.org/ws-rx/wsrm/v1.1/wsrm.html`.

[Dew+97] Tony Dewitt u. a. *ReMoS: A Resource Monitoring System for Network-Aware Applications.* Techn. Ber. 1997.

[Dia+08] I. Diaz u. a. „Integrating the Common Information Model with MDS4“. In: *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing.* GRID '08. Washington, DC, USA: IEEE Computer Society, 2008, S. 298–303.

[Din+01] Peter A. Dinda u. a. „The Architecture of the Remos System“. In: *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing.* IEEE Computer Society, 2001, S. 252–265.

[Du+04] Wenliang Du u. a. „Uncheatable Grid Computing“. In: *In 24th IEEE International Conference on Distributed Computing Systems.* 2004, S. 4–11.

[Edl+10] Stefan Edlich u. a. *noSQL – Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken.* Hanser, 2010.

[Eura] European Organization of Nuclear Research. *LCG – Worldwide LHC Computing Grid.* http://lcg.web.cern.ch/LCG/.

[Eurb] European Organization of Nuclear Research. *The Large Hadron Collider*. `http://public.web.cern.ch/public/en/lhc/LHC-en.html`.

[Eurc] European Organization of Nuclear Research. *The name CERN*. `http://public.web.cern.ch/public/en/About/Name-en.html`.

[Fie] Laurence Field. *Getting Grids to work together: interoperation is key to sharing*. http://cerncourier.com/cws/article/cnl/26301. (Besucht am. 01. 11. 2006).

[Fit+97] Steven Fitzgerald u. a. „A Directory Service for Configuring High-Performance Distributed Computations“. In: *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing*. HPDC '97. Washington, DC, USA: IEEE Computer Society, 1997, S. 365–375.

[FK04a] Ian Foster und Carl Kesselman, Hrsg. *The Grid 2: Blueprint for a New Computing Infrastructure*. 2. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. Kap. 20.

[FK04b] Ian Foster und Carl Kesselman. „The Grid in a Nutshell“. In: Hrsg. von Jarek Nabrzyski, Jennifer M. Schopf und Jan Weglarz. Norwell, MA, USA: Kluwer Academic Publishers, 2004. Kap. 1.

[FKT01] Ian Foster, Carl Kesselman und Steven Tuecke. „The Anatomy of the Grid - Enabling Scalable Virtual Organizations“. In: *International Journal of Supercomputer Applications* 15 (2001), S. 200–222.

[FL99] A. Frenkiel und H. Lee. „EPP: A Framework for Measuring the End-to-End Performance of Distributed Applications“. In: *Proceedings of Performance Engineering "Best Practices" Conference, IBM Academy of Technology*. 1999.

[FMS05] Ian Foster, T. Maguire und D. Snelling. *OGSA WSRF Basic Profile 1.0*. Techn. Ber. Open Grid Forum, 2005.

[Fos02] Ian Foster. „What is the Grid? A Three Point Checklist“. In: *GRIDtoday* 1 (2002).

[Fos+06] I. Foster u. a. *The Open Grid Services Architecture, Version 1.5*. Techn. Ber. Open Grid Forum, 2006.

[Fos+98] Ian Foster u. a. „A Security Architecture for Computational Grids“. In: *Proceedings of the 5th ACM conference on Computer and Communications Security*. CCS '98. New York, NY, USA: ACM, 1998, S. 83–92.

[Foua] Apache Software Foundation. *Apache CXF*. http://cxf.apache.org/.

[Foub] Apache Software Foundation. *Apache Tomcat*. http://tomcat.apache.org/.

[Fouc] Apache Software Foundation. *Apache Wicket*. http://http://wicket.apache.org/.

[Foud] Apache Software Foundation. *ApacheDS^{TM}*. http://directory.apache.org/apacheds/.

[Foue] Eclipse Foundation. *Eclipse IDE*. http://www.eclipse.org/.

[Fre+04] Eric Freeman u. a. *Head First Design Patterns*. Hrsg. von Mike Hendrickson und Mike Loukides. Bd. 1. O'Reilly Media, Inc., 2004.

[Fre07] Tessen Freund. *Software Engineering durch Modellierung wissensintensiver Entwicklungsprozesse*. GITO mbH – Verlag für Industrielle Informationstechnik und Organisation, 2007.

[FS06] Otto K. Ferstl und Elmar J. Sinz. *Grundlagen der Wirtschaftsinformatik*. 5., überarbeitete und erweiterte Auflage. Oldenbourg, 2006.

[Ger+04] Michael Gerndt u. a. *Performance Tools for the Grid: State of the Art and Future*. Bd. 30. LRR-TUM Research Report Series. Aachen: Shaker Verlag, 2004.

[GJbXy06] Chen Gang, XIa Jing-bo und Cai Xiao-yong. *Probing Techniques for Remote Operating Systems Based on TCP Fingerprint*. Techn. Ber. Telecommunication Engineering Institute of the Air Force Engineering University,Xi'an Shanxi 710077,China, 2006.

[Gma] *Global Grid Forum, Grid Monitoring Architecture Working Group*. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/.

[Gmba] appsolute GmbH. *MAMP – Macintosh, Apache, Mysql und PHP*. http://www.mamp.info/de/index.html.

[Gmbb] D-Grid GmbH. *D-Grid gGmbH: D-MON - Horizontal Integration of Resource and Service Monitoring in the D-Grid*. `http://www.d-grid-gmbh.de/index.php?id=52&L=1`.

[Gou+06] Anastasios Gounaris u. a. „Practical Adaptation to Changing Resources in Grid Query Processing". In: *Proceedings of the 22nd International Conference on Data Engineering*. ICDE '06. Washington, DC, USA: IEEE Computer Society, 2006, S. 165–.

[Gri] GridCafé. *The Five Big Ideas Behind Grid Computing*. `http://www.gridcafe.org/version1/challenges/challanges.html`.

[Gud+07] Martin Gudgin u. a. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. W3C Recommendation. World Wide Web Consortium, 2007. URL: `http://www.w3.org/TR/soap12-part1/`.

[GW] Grid Schema Working Group (GLUE-WG). *GLUE Schema*. http://forge.gridforum.org/sf/projects/glue-wg. (Besucht am. 30. 05. 2011).

[HAN99] Heinz-Gerd Hegering, Sebastian Abeck und Bernhard Neumair. *Integriertes Management vernetzter Systeme – Konzepte, Architekturen und deren betrieblicher Einsatz*. dpunkt, 1999.

[Hen10a] Rolf Hennicker. *Softwaretechnik – Kapitel 1 – Softwaretechnik: Uberblick*. Vorlesungsskript. 2010.

[Hen10b] Rolf Hennicker. *Softwaretechnik – Kapitel 3 – Objektorientierte Analyse*. Vorlesungsskript. 2010.

[HH10] Oliver Heuser und Andreas Holubek. *Java Web Services in der Praxis – Realisierung einer SOA mit WSIT, Metro und Policies*. dpunkt.verlag, 2010.

[Hit+05] Martin Hitz u. a. *UML@Work – Objektorienter Modellierung mit UML 2*. Bd. 3. dpunkt.verlag, 2005.

[Hos+00] Wolfgang Hoschek u. a. „Data Management in an International Data Grid Project". In: *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*. GRID '00. Springer, 2000, S. 77–90.

[HSG99] T. Howes, M. Smith und G. Good. *Understanding and Deploying LDAP Directory Services*. Macmillan Technical Publications, 1999.

[HT04] Jeffrey Hollingsworth und Brian Tierney. „Instrumentation and Monitoring“. In: Hrsg. von Ian Foster und Carl Kesselman. 2. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. Kap. 20.

[Ios+07] Alexandru Iosup u. a. „On the Dynamic Resource Availability in Grids“. In: *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*. GRID '07. Washington, DC, USA: IEEE Computer Society, 2007, S. 26–33.

[IRD05] E. Imamagic, B. Radic und D. Dobrenic. „CRO-GRID Grid Monitoring Architecture“. In: *27th International Conference on Information Technology Interfaces*. Cavtat, Croatia, 2005, S. 65–72.

[Jac92] Ivar Jacobson. *Object Oriented Software Engineering: A Use Case Driven Approach*. Wokingham, UK: Addison-Wesley, 1992.

[jav11] java.net. *GlassFish - Open Source Application Server*. http://glassfish.java.net/. 2011.

[Joe+01] Bill Allcock Joe u. a. „Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing“. In: *Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*. MSS '01. Washington, DC, USA: IEEE Computer Society, 2001.

[KB06] Kyong Hoon Kim und Rajkumar Buyya. „Policy-Based Resource Allocation in Hierarchical Virtual Organizations for Global Grids“. In: *Proceedings of the 18th International Symposium on Computer Architecture and High Performance Computing*. Washington, DC, USA: IEEE Computer Society, 2006, S. 36–46.

[KH04] Ezzat Kirmani und Cynthia S. Hood. „Diagnosing Network States Through Intelligent Probing“. In: *Proceedings of IEEE/IFIP Network Operations and Management Symposium*. Bd. 1. IEEE Computer Society, 2004, S. 147–160.

[Kle08] Stephan Kleuker. *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten*. Vieweg+Teubner Verlag, 2008.

[Kón06] Balázs Kónya. *The NorduGrid/ARC Information System*. Techn. Ber. NORDUGRID-TECH-4. NorduGrid, 2006.

[KS10] Dieter Kranzlmüller und Michael Schiffers. *Grid Computing – Grundlagen und Einführung – Verteilte Systeme*. Vorlesungsskript. 2010.

[Lak+] Avinash Lakshman u. a. *Apache Cassandra – Official Website*. http://cassandra.apache.org/.

[Las+01] Gregor von Laszewski u. a. „A Java Commodity Grid Kit“. In: *Concurrency and Computation: Practice and Experience* 13 (2001), S. 643–662.

[LB05] Maozhen Li und Mark A. Baker. *The Grid – Core Technologies*. Wiley, 2005.

[LF10] Robert J. Liguori und Edward G. Finegan. *SCJA – Sun Certified Java Associate Study Guide*. McGrawHill, 2010.

[Liv11] Miron Livny. *Condor Hawkeye – A Monitoring and Management Tool for Distributed Systems*. http://www.cs.wisc.edu/condor/hawkeye/. 2011.

[LLC08] PatentStorm LLC. *US Patent Application 20080209269 - Active Probing for Real-Time Diagnosis*. http://www.patentstorm.us/applications/20080209269/description.html. 2008.

[LMG95] Darrell Long, Andrew Muir und Richard Golding. *A Longitudinal Survey of Internet Host Reliability.* Techn. Ber. Palo Alto, DC, USA: Hewlett-Packard Laboratories, 1995.

[LRO] Craig Lee, Joel Replogle und Melissa Osner. *Open Grid Forum.* http://www.ogf.org/.

[Mac+06] C. Matthew MacKenzie u. a. *Reference Model for Service Oriented Architecture 1.0.* Techn. Ber. OASIS, 2006.

[Mar+07] Moreno Marzolla u. a. „Open Standards-Based Interoperability of Job Submission and Management Interfaces across the Grid Middleware Platforms gLite and UNICORE". In: *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing.* IEEE Computer Society, 2007, S. 592–601.

[MCC04] Matthew L. Massie, Brent N. Chun und David E. Culler. „The Ganglia Distributed Monitoring System: Design, Implementation, and Experience". In: *Parallel Computing* 30.7 (2004), S. 817–840.

[Mil92] David L. Mills. *Network Time Protocol Version 3 – Specification, Implementation and Analysis.* Specification RFC 1305. Universität von Delaware, 1992.

[MN06] James W. Mickens und Brian D. Noble. „Exploiting Availability Prediction in Distributed Systems". In: *Proceedings of NSDI'06 Symposium on Networked Systems Design & Implementation.* Bd. 3. NSDI'06. Berkeley, CA, USA: USENIX Association, 2006, S. 73–86.

[MSS93] M. Mansouri-Samani und M. Sloman. „Monitoring Distributed Systems". In: *IEEE Network* 7.6 (1993), S. 20–30. URL: `http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=244791`.

[MsS93] Masoud Mansouri-samani und Morris Sloman. *Monitoring Distributed Systems (A Survey).* Techn. Ber. DOC92/23. London, UK: Imperial College, 1993.

[MSZ06] Jens-Michael Milke, Michael Schiffers und Wolfgang Ziegler. „Virtuelle Organisationen in Grids: Charakterisierung und Management". In: *Praxis der Informationsverarbeitung und Kommunikation (PIK).* Bd. 29. 3. 2006, S. 165–170.

[NPF08] Farrukh Nadeem, Radu Prodan und Thomas Fahringer. „Characterizing, Modeling and Predicting Dynamic Resource Availability in a Large Scale Multipurpose Grid". In: *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid.* Washington, DC, USA: IEEE Computer Society, 2008, S. 348–357.

[NSW04] Jarek Nabrzyski, Jennifer M. Schopf und Jan Weglarz, Hrsg. *Grid Resource Management – State of the Art and Future Trends.* Norwell, MA, USA: Kluwer Academic Publishers, 2004.

[OMG07] Inc. Object Management Group. *OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2.* Specification. Object Management Group, Inc., 2007. URL: `http://www.omg.org/spec/UML/2.1.2/`.

[Oraa] Oracle. *Java SE Documentation – wsimport - Java^{TM} API for XML Web Services (JAX-WS) 2.0.* http://download.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html.

[Orab] Oracle. *Java^{TM} Platform Standard Edition 6 – API – Package java.sql*. http://download.oracle.com/javase/6/docs/api/java/sql/package-summary.html.

[Orac] Oracle. *JAX-WS Reference Implementation Project*. http://jax-ws.java.net/.

[Orad] Oracle. *Learn About Java Technology*. http://java.com/en/about/.

[Orae] Oracle. *MySQL – Die populärste Open-Source-Datenbank der Welt*. http://www.mysql.de.

[Oraf] Oracle. *NetBeans IDE*. http://netbeans.org/.

[Pap08] Michael P. Papazoglou. *Web Services: Principles and Technology*. Pearson, Prentice Hall, 2008.

[PB08] Behnaz Pourebrahimi und Koen Bertels. „Adaptation to Dynamic Resource Availability in Ad Hoc Grids through a Learning Mechanism“. In: *Proceedings of the 2008 11th IEEE International Conference on Computational Science and Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, S. 171–178.

[PDL02] Beth Plale, Peter Dinda und Gregor von Laszewski. „Key Concepts and Services of a Grid Information Service“. In: *Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems (PDCS 2002)*. 2002, S. 437–442.

[Pie04] V. Pietrobon. *Performance Fault Prediction Models*. Techn. Ber. CS-2004-3. University of Venice, 2004.

[Pre98] IEEE Press. *IEEE Std. 830-1998. IEEE Recommended Practice for Software Requirements Specifications*. Techn. Ber. IEEE Standards Board, 1998.

[Pro] GridSphere Project. *GridSphere Portal Framework*. http://www.gridsphere.org.

[Rib+98] Randy L. Ribler u. a. „Autopilot: Adaptive Control of Distributed Applications“. In: *Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing*. HPDC '98. IEEE Computer Society, 1998, S. 172–179.

[Ris+04] Irina Rish u. a. „Real-time Problem Determination in Distributed Systems using Active Probing“. In: *Network Operations and Management Symposium*. Bd. 1. Seoul, South Korea: IEEE Computer Society, 2004, S. 133–146.

[RL07] Brent Rood und Michael J. Lewis. „Multi-State Grid Resource Availability Characterization“. In: *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*. GRID '07. Washington, DC, USA: IEEE Computer Society, 2007, S. 42–49.

[RL08] Brent Rood und Michael J. Lewis. „Resource Availability Prediction for Improved Grid Scheduling“. In: *Proceedings of the 2008 Fourth IEEE International Conference on eScience*. Washington, DC, USA: IEEE Computer Society, 2008, S. 711–718.

[RP09] Chris Rupp und K. Pohl. *Basiswissen Requirements Engineering: Aus- und Weiterbildung zum Certified Professional for Requirements Engineering – Foundation Level nach IREB-Standard*. dpunkt.verlag, 2009.

[RQZ07] Chris Rupp, Stefan Queins und Barbara Zengler. *UML 2 glasklar – Praxiswissen für die UML-Modellierung*. 3. Aufl. Hanser, 2007.

[Rup09] Chris Rupp. *Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis*. 5. Aufl. München: Hanser, 2009.

[SB08] Kathy Sierra und Bert Bates. *SCJP – Sun Certified Programmer for Java 6 Study Guide*. McGrawHill, 2008.

[SC06] Borja Sotomayor und Lisa Childers. *Globus Toolkit 4 – Programming Java Services*. The Elsevier Series in Grid Computing. Morgan Kaufmann Publishers, 2006.

[Sch04] Jennifer M. Schopf. „Ten Actions when Grid Scheduling“. In: Hrsg. von Jarek Nabrzyski, Jennifer M. Schopf und Jan Weglarz. Norwell, MA, USA: Kluwer Academic Publishers, 2004. Kap. 2.

[Sch07] Michael Schiffers. „Management dynamischer virtueller Organisationen in Grids“. Diss. Ludwig-Maximilians-Universität München: Fakultät für Mathematik, Informatik und Statistik, 2007.

[SEH] Shava Smallen, Kate Ericson und Paul Hoover. *INCA Version 2.5*. http://inca.sdsc.edu.

[Sfi+08] I. Sfiligoi u. a. „Pilot Job Accounting and Auditing in Open Science Grid“. In: *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*. GRID '08. IEEE Computer Society, 2008, S. 112–117.

[SG06] Bianca Schroeder und Garth A. Gibson. „A Large-Scale Study of Failures in High-Performance Computing Systems“. In: *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2006)*. Washington, DC, USA: IEEE Computer Society, 2006, S. 249–258.

[SGQ01] Warren Smith, Dan Gunter und Darcy Quesnel. *A Simple XML Producer-Consumer Protocol*. Techn. Ber. GWD-Perf-8-2. Grid Forum Performance Working Group, 2001.

[SJS01] Commitee for Software und systems engineering JTC 1/SC 7. *ISO/IEC 9126-1:2001 – Software engineering – Product quality – Part 1: Quality model*. Techn. Ber. International Organization for Standardization, 2001.

[Sma+07] Shava Smallen u. a. „User-Level Grid Monitoring with Inca 2“. In: *Proceedings of the 2007 Workshop on Grid Monitoring*. GMW '07. ACM, 2007, S. 29–38.

[Sme+04] S. De Smet u. a. „A Performance Oriented Grid Monitoring Architecture“. In: *Proceedings of the 2nd IEEE Workshop on End-to-End Monitoring Technics and Sercives (E2EMON)*. Hrsg. von E. Al-Shaer. 2004.

[Smi01] Warren Smith. *A Framework for Control and Observation in Distributed Environments*. Techn. Ber. NAS-01-006. NASA Advanced Supercomputing Division, NASA Ames Research, 2001.

[spr] springsource. *Core Spring Framework*. http://www.springsource.org.

[SRL11] Alexandre Strube, Dolores Rexachs und Emilio Luque. „Improving Probe Usability“. In: *Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*. WAINA '11. IEEE Computer Society, 2011, S. 381–387.

[Sta07] Michael Stal. *Service-orientierte Architekturen und UML 2.0*. 2007.

[Ste+98] Paul Stelling u. a. „A Fault Detection Service for Wide Area Distributed Computations“. In: *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*. HPDC '98. IEEE Computer Society, 1998, S. 268–278.

[STK02] James Snell, Doug Tidwell und Pavel Kulchenko. *Programming Web Services With SOAP*. Hrsg. von Nathan Torkington. O'Reilly Media, Inc., 2002.

[Str11] Christian Straube. *Installationsanleitung Globus Toolkit 4.0.8*. Lehr- und Forschungseinheit für Kommunikationssysteme und Systemprogrammierung. 2011.

[Str97] Alois Stritzinger. *Komponentenbasierte Softwareentwicklung – Konzepte und Techniken für das Programmieren und Modellieren in Smalltalk*. Addison-Wesley, 1997.

[Tan07] Andrew S. Tanenbaum. *Computernetzwerke*. Bd. 4. München, Bayern, Germany: Pearson Studium, 2007.

[TG03] Brian Tierney und Dan Gunter. „NetLogger – A Toolkit for Distributed System Performance Tuning and Debugging“. In: *Proceedings of the IFIP/IEEE 8th International Symposium on Integrated Network Management*. 2003, S. 97–100.

[Tha+03] Douglas Thain u. a. „Condor and the Grid“. In: Hrsg. von F. Berman, A. Hey und G. Fox. John Wiley & Sons, Ltd, 2003. Kap. 11.

[Tie+00] Brian Tierney u. a. „A Monitoring Sensor Management System for Grid Environments“. In: *Proceedings of the IEEE High Performance Distributed Computing Conference (HPDC-9)*. HPDC '00. Washington, DC, USA: IEEE Computer Society, 2000, S. 97–104.

[Tie+02] Brian Tierney u. a. *A Grid Monitoring Architecture*. 2002.

[Tie+98] Brian Tierney u. a. „The NetLogger Methodology for High Performance Distributed Systems Performance Analysis“. In: *Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing*. HPDC '98. Chicago, Illinois: IEEE Computer Society, 1998, S. 260–267.

[TS08] Andrew S. Tanenbaum und Maarten van Steen. *Verteilte System – Prinzipien und Paradigmen*. 2. Aufl. München, Bayern, Germany: Pearson Studium, 2008.

[Tue] Steve Tuecke. *Globus Toolkit User List – Crux Project*. http://lists.globus.org/pipermail/gt-user/2011-February/009687.html.

[Urla] *Apache Struts – Official Website*. http://struts.apache.org/.

[Urlb] *Distributed Management Task Force, Inc. – Common Information Model (CIM)*. http://www.dmtf.org/standards/cim.

[Urlc] *Globus Toolkit 4 – Globus Toolkit 4.0 Release Manuals – Java WS Core*. http://www.globus.org/toolkit/docs/4.0/common/javawscore/.

[Urld] *OSG – Open Science Grid – A National, Distributed Computing Grid for Data-Intensive Research*. http://www.opensciencegrid.org/.

[Urle] *phpMyAdmin*. http://www.phpmyadmin.net/home_page/index.php.

[Wei+08] Anette Weisbecker u. a. *Fraunhofer Enterprise Grid*. Techn. Ber. Fraunhofer Gesellschaft, 2008.

[WH03] Glenn Wasson und Marty Humphrey. „Policy and Enforcement in Virtual Organizations“. In: *Proceedings of the 4th International Workshop on Grid Computing*. GRID '03. Washington, DC, USA: IEEE Computer Society, 2003, S. 125–132.

[Wil09] Barry Wilkinson. *Grid Computing – Techniques and Applications*. Hrsg. von Horst D. Simon. Chapman & Hall/CRC Computational Science. Chapman & Hall, 2009.

[Wol+04] Rich Wolski u. a. „Grid Resource Management“. In: Norwell, MA, USA: Kluwer Academic Publishers, 2004. Kap. Performance Information Services for Computational Grids, S. 193–213.

[Wol07] A. Wolf. *Spezifikation der D-Grid Ressourcenbeschreibungssprache DGRDL*. Techn. Ber. Fraunhofer Gesellschaft, 2007.

[WSH99] Rich Wolski, Neil T. Spring und Jim Hayes. „The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing“. In: *Future Generation Computing Systems* 15 (1999), S. 757–768.

[Wu+08] Libing Wu u. a. „A Resource Discovery Algorithm with Probe Feedback Mechanism in Multi-Domain Grid Environment“. In: *Proceedings of the 3rd International Conference on Advances in Grid and Pervasive Computing*. GPC'08. Springer, 2008, S. 178–186.

[Wun06] Lars Wunderlich. *Java Rules Engines – Entwicklung von regelbasierten Systemen*. entwickler.press, 2006.

[WWW10] Xiaoguang Wang, Hui Wang und Yongbin Wang. „A Unified Monitoring Framework for Distributed Environment“. In: *Intelligent Information Management*. Bd. 2. 7. Scientific Research. 2010, S. 398–405.

[ZFS03] Xuehai Zhang, Jeffrey L. Freschl und Jennifer M. Schopf. „A Performance Study of Monitoring and Information Services for Distributed Systems“. In: *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*. HPDC '03. Washington, DC, USA: IEEE Computer Society, 2003, S. 270–282.

[ZGK04] Wolfgang Zuser, Thomas Grechenig und Monika Köhle. *Software Engineering mit UML und dem Unified Process*. Bd. 2. Pearson Studium, 2004.

[Zho+06] XiaoLi Zhou u. a. „ReCon: A Fast and Reliable Replica Retrieval Service for the Data Grid“. In: *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*. CCGRID '06. IEEE Computer Society, 2006, S. 446–453.

[ZL03] Marcia Zangrilli und Bruce B. Lowekamp. „Comparing Passive Network Monitoring of Grid Application Traffic with Active Probes“. In: *Proceedings of the 4th International Workshop on Grid Computing*. GRID '03. Washington, DC, USA: IEEE Computer Society, 2003.

[ZP10] Yue Zhang und Yunxia Pei. „A Resource Discovery Algorithm in Mobile Grid Computing Based on IP-Paging Scheme“. In: *Proceedings of the 2010 IFIP international conference on Network and parallel computing*. NPC'10. 2010, S. 402–411.

[ZS04] Serafeim Zanikolas und Rizos Sakellariou. „Towards a Monitoring Framework for Worldwide Grid Information Services“. In: *10th International Euro-Par Conference*. Hrsg. von Lecture Notes in Computer Science. Bd. 3149. Pisa, Italy: Springer, 2004, S. 417–422.

[ZS05] Serafeim Zanikolas und Rizos Sakellariou. „A Taxonomy of Grid Monitoring Systems“. In: *Future Generation Computer Systems (FGCS) Journal* 21 (2005), S. 163–188.