INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwicklung eines dynamischen Lizenzschemas für Grids

Michael Höber

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwicklung eines dynamischen Lizenzschemas für Grids

Michael Höber

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Michael Schiffers
Abgabetermin:	25. März 2010

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

München, den 25. März 2010

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Grid-Computing ist eines der großen Forschungsgebiete der heutigen Informatik. Das Konzept ist jedoch bereits schon so weit fortgeschritten, dass Grids im wissenschaftlichen Bereich bereits zum Alltag gehören und sehr häufig für umfangreiche Berechnungen und Arbeiten eingesetzt werden. Aber nicht nur im wissenschaftlichen Umfeld werden Grids eingesetzt, sondern auch im kommerziellen Bereich und der Industrie trifft man immer häufiger Grid-Umgebungen an, da sie viele komplexe Probleme in relativ kurzer Zeit lösen können.
Der Einsatz von kommerzieller Software unterliegt meistens einem Lizenzabkommen und folglich kann die Software nur mit einer gültigen Lizenz genutzt werden. Herkömmliche Lizenzierungsmodelle genügen jedoch nicht den Ansprüchen zum Einsatz in einem Grid auf Grund verschiedenster Problematiken, so dass neue, erweiterte Modelle entwickelt werden müssen.
Diese Diplomarbeit befasst sich mit der Entwicklung eines Anforderungskatalogs für ein allgemein in Grid-Umgebungen nutzbares Lizenzkonzept. Dieser Anforderungskatalog dient danach dazu, andere Ansätze zu evaluieren, die sich mit Lizenzierung in Grid-Umgebungen beschäftigen. Außerdem befasst sich die Diplomarbeit mit der Entwicklung eines neuen Lizenzkonzepts, das sämtlichen Anforderungen des Katalogs genügt und somit für den allgemeinen Einsatz in kommerziellen Grids geeignet ist. Auf Basis dieses Lizenschemas wird ein Prototyp implementiert, der das Lizenzschema auf einem Globus-Toolkit-4-Testbed umsetzt. Die Ergebnisse der Arbeit finden sich im letzten Kapitel dieser Diplomarbeit.

Inhaltsverzeichnis

1 Einleitung

Grid-Computing ist eines der großen Forschungsgebiete der heutigen Informatik. Das Konzept ist jedoch bereits schon so weit fortgeschritten, dass Grids im wissenschaftlichen Bereich bereits zum Alltag gehören und sehr häufig für umfangreiche Berechnungen und Arbeiten eingesetzt werden. Aber nicht nur im wissenschaftlichen Umfeld werden Grids eingesetzt, sondern auch im kommerziellen Bereich und der Industrie trifft man immer häufiger Grid-Umgebungen an, da sie viele komplexe Probleme in relativ kurzer Zeit lösen können. Nach [Wik10c] hat ein Grid die Zielsetzung der gemeinsamen, koordinierten Nutzung von Ressourcen. Mittels dieser Ressourcen sollen gemeinschaftlich Probleme innerhalb dynamischer, institutionsübergreifender Virtueller Organisationen gelöst werden. So wird nach Festlegung von Abrechnungsdaten und Rechten ein direkter Zugang zu beispielsweise Rechenleistungen, Anwendungen, Daten oder Instrumenten gemeinschaftlich ermöglicht. Im Zusammenhang mit Grids ist eine Virtuelle Organisation (VO) ein dynamischer Zusammenschluss von Individuen und/oder Institutionen, um gemeinschaftliche Ziele mittels Nutzung des Grids zu erreichen. Oft liegt der Fokus im Bereich des verteilten Rechnens. Oberstes Ziel ist dennoch, analog zu der Entstehung des Internets, die Entwicklung eines einheitlichen, globalen Grids. Da der Einsatz von Grids immer verbreiteter wird, steigt auch das Verlangen nach der Nutzung von kommerzieller Software in der Grid-Umgebung. Kommerzielle Software unterliegt einem Lizenzabkommen und kann folglich nur mit einer Lizenz und Zustimmung des zugrundeliegenden Lizenzvertrags genutzt werden. Herkömmliche Lizenzierungsmodelle genügen jedoch nicht den Ansprüchen zum Einsatz in einem Grid, so dass neue, erweiterte Modelle entwickelt werden müssen. Diese neuen Modelle sollten jedoch weitestgehend unabhängig von der verwendeten Grid-Middleware sein.
Als Grid-Middleware bezeichnet man die Software, die das Grid und dessen Ressourcen dem Benutzer zur Verfügung stellt. Der Aufgabenbereich einer Grid-Middleware ist jedoch sehr vielfältig. So fallen darunter Aufgaben wie Benutzer-Authentisierung und -Autorisierung, Ressourcen-Verwaltung, Job-Zuteilungen und -Verwaltung und so weiter. Momentan sind mehrere Grid-Middlewares verfügbar, die keinem einheitlichen Konzept folgen. Daher kann es zu einer gewissen Inkompatibilität führen, da die verschiedenen Middlewares aus unterschiedlichen Bereichen der Wissenschaft und Industrie stammen und daher selbstverständlich mit unterschiedlichen Anforderungen entwickelt wurden. Diese Inkompatibilitäten und unterschiedlichen Konzepte erschweren allgemein die Entwicklung von verallgemeinerten Konzepten und Strukturen, so zum Beispiel auch das in dieser Diplomarbeit entwickelte Lizenzmodell auf Basis virtueller Lizenzen.

1.1 Motivation

Die klassische Nutzung von kommerzieller Software in nicht-Grid-Umgebungen erfordert in der Regel immer die Bereitstellung einer Lizenz. Diese Lizenz ermöglicht dem Independent-Software-Vendor (ISV), also dem Hersteller der Software, seine Kunden an den Software-Lizenzvertrag zu binden und dessen Einhaltung besser zu kontrollieren. Eine lizenzierte Software kann nur mit einer gültigen Lizenz gestartet werden.
Da bei Verwendung von Grids Software-Anwendungen dynamisch auf die zugehörigen Computer, sprich Ressourcen, bei Starten eines Jobs zugeteilt werden, müssen auch die notwendigen Lizenzen bei jedem Job-Start dynamisch als Teil des Jobs zugeteilt werden. Da ein Grid sich in der Regel über mehrere administrative Domänen erstreckt, ist die Authentisierung der Benutzer deutlich erschwert. Die meisten Lizenzschemata, die auf einer Client-Server-Struktur basieren, authentifizieren Computer anhand ihrer IP. Lizenzen die auf einen bestimmten IP-Bereich beschränkt sind, sind daher nutzlos, da in der hochdynamischen Umgebung einer Virtuellen Organisation ständig neue Ressourcen hinzukommen oder nicht mehr benötigte entfernt werden.
Aus einem anderen Winkel betrachtet möchte eine Organisation zum Beispiel externe Computing Resources eines Resource-Providers nutzen und dort lizenzierte Anwendungen innerhalb eines Jobs laufen lassen. Um die Lizenzen auf Legalität zu überprüfen müsste diese Organisation ihre Firewall soweit öffnen, dass von jeder theoretisch möglichen Resource eine Anfrage zur Gültigkeitsprüfung und Authentisierung der Lizenz eingehen könnte. Dies wäre ein erheblicher Sicherheitsmangel und stellt eine potentielle Angriffsstelle dar.
Ein anderer Gesichtspunkt ist die Einschränkung der Benutzung einer Anwendung mittels dem zugehörigen Lizenzvertrag auf eigene Ressourcen. Diese Einschränkung ist nahezu immer gegeben. Daher wäre die Nutzung auf externe Ressourcen strikt untersagt oder mit erheblichen Zusatzkosten verbunden, da man das Lizenzabkommen erweitern müsste. Obwohl aus Sicht der Organisation diese externen Ressourcen eigentlich *eigene* Ressourcen zum Zeitpunkt der Existenz der Virtuellen Organisation wären, wären aus Sicht des ISVs oder des Lizenzvalidierers diese Ressourcen *extern* angesiedelt.
Eine weitere Problematik ist die Tatsache, dass Grids ursprünglich vor allem als Universitäts- oder Forschungsnetze benutzt wurden. Lizenzabkommen erlauben oft den Einsatz der lizenzierten Software in universitärem beziehungsweise nicht-kommerziellem Umfeld. Da aber mittlerweile Grids auch häufig in kommerziellen Projekten eingesetzt werden, ist eine Unterscheidung zwischen Universitäts-Lizenzen und kommerziellen Lizenzen sowie den dazugehörigen Jobs notwendig, so dass nicht illegalerweise Lizenzen dynamisch für Forschungsprojekte zugeteilt werden und fälschlicherweise für kommerzielle Projekte genutzt werden.
Die allgemeine Problematik der Lizenznutzung ist die Tatsache, dass Grids in der Regel auf mehrere administrative Domänen verteilt sind. Es gelten überall andere Sicherheitspolitiken und andere vertragliche Verpflichtungen. Eine Kommunikationsmöglichkeit zwischen diesen Domänen ist in der Regel nicht direkt vorhanden, sodass eine Lizenzanfrage an einen Lizenzserver mit hoher Wahrscheinlichkeit nicht ankommt.
Diese extreme und vor allem hochdynamische Heterogenität erschwert die sinnvolle Nutzung von Softwarelizenzen innerhalb Grid-Umgebungen.
Diese und andere Problematiken treten beim Einsatz von kommerzieller Software in Grid-Umgebungen auf. Ziel dieser Diplomarbeit ist somit die Entwicklung eines Lizenzschemas, das, dem im Laufe der Arbeit entstandenen, Anforderungskatalog genügt und somit die Nutzung von kommerzieller lizenzierter Software in Grid-Umgebungen ermöglicht.

1.2 Aufgabenstellung

Die Aufgabenstellung dieser Diplomarbeit ist [Leh09] zu entnehmen und wird nachfolgend zitiert.

1.2.1 Hintergrund der Arbeit

Grid Computing bezeichnet die koordinierte Nutzung von Rechnern, Software und Daten in Virtuellen Organisationen (VO) zur Lösung großer wissenschaftlicher Probleme (die so genannten Grand Challenges). Ein Charakteristikum Virtueller Organisationen ist deren Dynamik im Äußeren (ausgedrückt durch den Lebenszyklus) und im Inneren (Ressourcen, Dienste und Mitglieder können „kommen und gehen“). Im Rahmen mehrerer nationaler und internationaler Grid-Projekte untersucht der Lehrstuhl Fragen des Managements von und in Grids. Insbesondere interessieren uns Fragestellungen zum Management Virtueller Organisationen in Grids.

1.2.2 Ziel der Arbeit

Die hier ausgeschriebene Arbeit ist im erweiterten Kontext des Managements Virtueller Organisationen (VO) in Grids anzusiedeln. Das Ziel Virtueller Organisationen (genauer: deren Mitglieder) ist das organisationsübergreifende koordinierte Problemlösen auf der Basis gemeinsam genutzter (und benutzbarer) Ressourcen. Dazu sind neben einer Menge technischer Herausforderungen zahlreiche nicht-triviale organisatorische Anforderungen zu erfüllen. Eine solche Fragestellung betrifft die Lizenzierung von Software (und anderer Ressourcen) in hoch-dynamischen, multi-organisationalen Kontexten, wie VOs sie darstellen. In der Arbeit soll zunächst ein umfangreicher Anforderungskatalog an ein adäquates VO-zentrisches (d.h. unter besonderer Berücksichtigung der VO-Perspektive) Lizenzschema erstellt werden. Anschließend soll der aktuelle Stand der Technik gegen diese Anforderungen evaluiert werden. Da heutige Schemata jeweils nur Teilaspekte erfüllen, soll schließlich ein neues tragfähiges Konzept auf der Basis „virtueller Lizenzen“ erstellt werden, das sämtlichen Anforderungen genügt. Das erarbeitete Konzept soll schließlich prototypisch implementiert werden und für ein Deployment in Produktions-Grids (z.B. D-Grid) vorbereitet werden.
Die Ergebnisse der Arbeit werden zur weiteren Diskussion den internationalen Standardisierungsgremien und den nationalen Grid-Betriebszentren zur Verfügung gestellt. Die Arbeit kann auf einige Vorarbeiten des Lehrstuhls aufsetzen. Ein Grid-Testbed ist am Lehrstuhl vorhanden.

1.3 Aufteilung der Arbeit

Nachdem nun die grundsätzliche Aufgabe der Diplomarbeit definiert wurde, wird im folgenden Abschnitt die Entstehung der Arbeit kurz erklärt, gefolgt von der näheren Beschreibung der einzelnen Kapitel.

1.3.1 Vier-Phasen-Entstehung

Die Entstehung dieser Diplomarbeit kann in vier Phasen aufteilt werden.

- **Brainstorming**
  In der ersten Phase wurde umfangreiches Brainstorming betrieben. So wurden in dieser Zeit grundsätzliche Informationen bezüglich Grid-Umgebungen und deren Einsatzzwecke und Möglichkeiten gesammelt. Außerdem wurden die häufigsten Lizenzschemata kommerzieller Software betrachtet und analysiert. Anschließend wurden diese Modelle auf ihre Problematiken für den Einsatz in Grid-Umgebungen untersucht.

- **Entwurf und State-of-the-Art-Betrachtung**
  In der zweiten Phase wurden die aus der ersten Phase gesammelten Informationen abstrahiert, um für eine Generierung eines Anforderungskatalogs zu dienen. Dieser Anforderungskatalog soll alle notwendigen Anforderungen und Risiken einer Software-Lizenzierung innerhalb Grid-Umgebungen abdecken. Außerdem wurden derzeitige State-of-the-Art-Lizenzkonzepte verglichen und Gleichheiten sowie Unterschiede herausgearbeitet und gegen den Anforderungskatalog evaluiert.

- **Konzeption**
  In der dritten Phase wurde ein neues Lizenzmodell entwickelt, das den Anforderungen des Katalogs genügt und somit für den Einsatz in Grid-Szenarien geeignet ist. Diese Phase betrifft sowohl das Umsetzen des Anforderungskatalogs in Funktionalitäten, als auch die gesamte konzeptionelle Entwicklung eines Lizenzmodells.

- **Entwicklung**
  In der vierten Phase wurde ein Prototyp zu dem konzeptionierten Lizenzmodell entwickelt. Dieser Prototyp wurde für eine Globus Toolkit Middleware programmiert.

1.3.2 Die Kapitel

Die Arbeit gliedert sich in sieben Kapitel.

1. Kapitel

Das 1. Kapitel befasst sich mit der Motivation der Thematik, der Erläuterung der grundsätzlichen Struktur der Diplomarbeit sowie allgemeinen Beschreibungen.

2. Kapitel

Im 2. Kapitel werden konzeptionelle Grundlagen behandelt. So werden zuerst Definitionen bezüglich Grid-Umgebungen, Virtuellen Organisationen und zusammenhängende Konzepte gegeben. Der zweite Abschnitt dieses Kapitels befasst sich mit Grundlagen und Definitionen

in Zusammenhang mit Software-Lizenzen. Außerdem erläutert dieser Abschnitt Lizenzschemata, die bei klassischer Softwarelizenzierung eingesetzt werden sowie deren Vor- und Nachteile sowohl aus Lizenznehmer- als auch Lizenzgebersicht. Abschließend gibt dieses Kapitel einen kurzen Einblick in die Entwicklungsumgebung g-Eclipse

3. Kapitel

Das 3. Kapitel befasst sich mit der Entwicklung eines Anforderungskatalogs an ein Lizenzmodell für Grid-Umgebungen. Dieser Anforderungskatalog wurde im Laufe der Entstehung der Diplomarbeit erarbeitet und ständig ergänzt, umgeschrieben, revidiert und verändert, da im Laufe der Zeit sich ständig neue Probleme, Entdeckungen und Ideen auftaten, die natürlich in diesem Anforderungskatalog nicht fehlen sollten.

4. Kapitel

Das 4. Kapitel befasst sich mit Lizenzierungen in Grid-Umgebungen. In diesem Kapitel werden einige der fortgeschrittensten State-of-the-Art-Ansätze der Software-Lizenzierung in Grids betrachtet und deren Vor- und Nachteile herausgearbeitet. Außerdem werden die Ansätze gegen den Anforderungskatalog evaluiert. Es wird aufgezeigt, dass keines dieser Konzepte allen Anforderungen genügt und daher nicht als allgemeines Konzept gelten kann. Außerdem wird in diesem Kapitel gezeigt, dass jedes der vorgestellten Konzepte andere Ziele vorweist.

5. Kapitel

Das 5. Kapitel betrachtet die Entwicklung eines tragfähigen Konzeptes auf Basis des zuvor erarbeiteten Anforderungskatalogs. So soll hier dargestellt werden, über welche Wege die jeweiligen Anforderungen gelöst werden, beziehungsweise in welcher Hinsicht die Anforderungen im endgültigen Konzept Anwendung finden. Außerdem wird das Konzept genau erläutert und die Aufgaben der einzelnen Komponenten definiert.

6. Kapitel

Das 6. Kapitel erklärt die Entwicklung eines Prototypen, der das im 5. Kapitel entwickelte Konzept implementiert. Es wird hierbei auf den Entwicklungsverlauf und die dabei aufgetretenen Probleme eingegangen. Außerdem wird aufgezeigt, inwieweit es sich um eine prototypische Implementierung handelt, und worin folglich für eine Produktivumgebung noch Arbeitsbedarf besteht.

7. Kapitel

Das 7. Kapitel fasst die vorangegangenen Kapitel nochmals zusammen und gibt eine Interpretation der Ergebnisse. Außerdem wird auf Schwierigkeiten bei der allgemeinen Erstellung der Arbeit eingegangen. Abschließend gibt das letzte Kapitel noch einen kritischen Ausblick in die Zukunft, abgerundet durch ein kleines Fazit.

2 Konzeptionelle Grundlagen

Dieses Kapitel befasst sich mit Definitionen und Begrifflichkeiten, die für das Verständnis der Arbeit relevant sind. Zuerst wird der Begriff Dienst erklärt und warum Dienste in Grids eine große Rollen spielen. Anschließend wird definiert, was ein Grid ist, beziehungsweise was man unter Grid Computing versteht. Grids basieren immer auf sogenannten Middlewares. Als Beispiel für eine Middleware wird in diesem Kapitel das Globus Toolkit erklärt. Außerdem wird auf ein typisches Konzept, das in Grids Verwendung findet, den Virtuellen Organisationen (VO), ausführlich eingegangen.
Der zweite Abschnitt dieses Kapitels befasst sich mit Definitionen im Zusammenhang mit Software-Lizenzen. Hierbei wird Wert gelegt, dass allgemeine Begrifflichkeiten erläutert werden, die für den späteren Verlauf der Diplomarbeit von grundlegender Bedeutung sind.
Der letzte Abschnitt dieses Kapitels befasst sich mit einer kurzen Erklärung von g-Eclipse.

2.1 Begriffliche Zusammenhänge

Diese Grafik gibt eine Übersicht über verschiedene Begriffe, die insbesondere in Grid-Umgebungen eine Rollen spielen. Im Folgenden werden diese Begriffe erklärt.

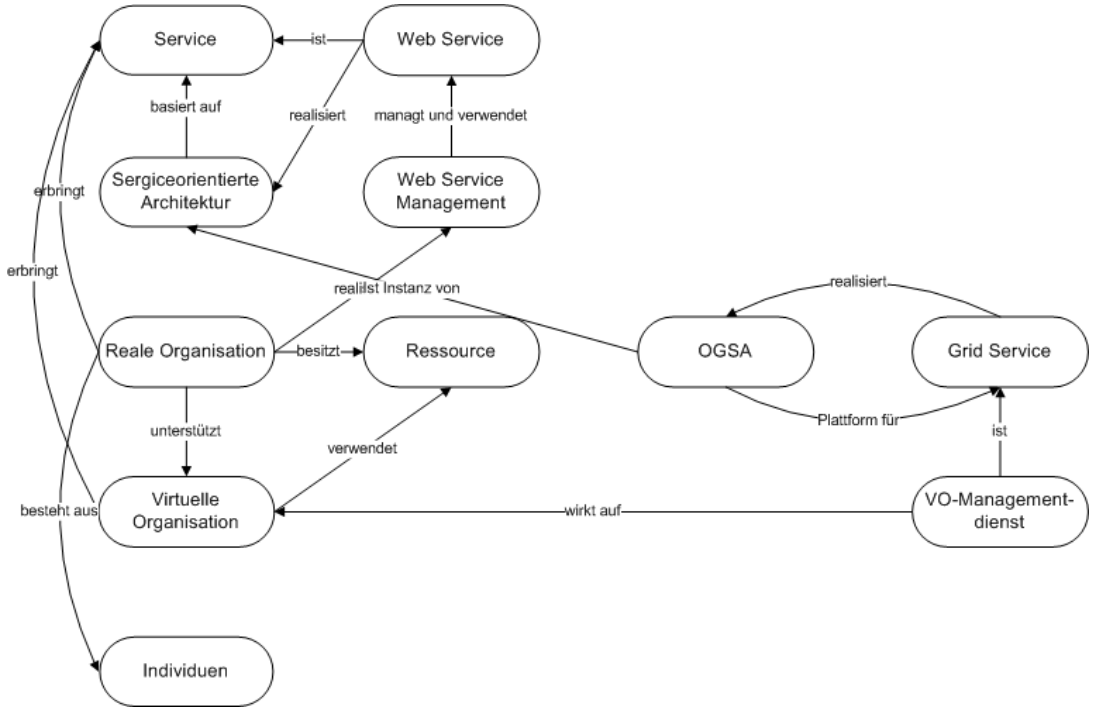

Abbildung 2.1: Struktur der Begriffe nach [Sch07]

2.2 Definitionen im Zusammenhang mit Diensten

Der folgende Abschnitt befasst sich mit Definitionen und Erläuterungen im Zusammenhang mit Diensten. So wird darauf eingegangen, was genau ein Dienst ist, wofür man ihn benötigt und was man von ihm erwarten kann. Außerdem wird aufgezeigt, in wie weit Dienste in Grid-Umgebungen eine Rolle spielen.

2.2.1 Allgemeine Charakterisierung des Dienstbegriffes

Allgemein betrachtet ist ein Dienst eine Leistung einer Person, einer Entität oder Maschine gegenüber einer anderen Person oder Entität oder Maschine.

2.2.2 Web Service

Ein Spezialfall eines Dienstes ist ein Web Service. Laut [Wik10h] hat das W3C ([Wor10]) einen Web Service folgendermaßen definiert: Ein Web Service dient als Unterstützung zur Zusammenarbeit zwischen verschiedenen Anwendungen, die auf unterschiedlichen Plattformen basieren und betrieben werden. Web Services werden mittels eindeutigen Uniform Resource Identifier (URI) identifiziert und gefunden. Die Schnittstellen von Web Services werden anhand von XML Ausdrücken definiert, beschrieben und im Netzwerk zur Verfügung gestellt. Zu Kommunikation werden XML-basierte Nachrichten mittels einer HTTP-Verbindung genutzt. Typischerweise wird das SOAP-Protokoll über HTTP benutzt, um Requests an einen Service an einer bestimmten URI zu senden, auf welchem der Web Service angesprochen wird, und die Nachricht nach einer Bearbeitung als Response zurücksendet.
Web Services stellen Services zur Verfügung, welche verschiedene Arten von Netzkomponenten wie zum Beispiel Clients, Servers, Anwendungen, Benutzer, Programme und so weiter integrieren. Daher ist es nötig, dass Web Services gut skalierbar und einfach zu pflegen sind. Web Services ähneln einer Serviceorientierten Architektur (SOA) und vereinen somit laut [Wik10h] verteilte und objektorientierte Programmierstandards.
Die Grundlage von Web Services sind folgende auf XML basierende Standards:

- **UDDI**
  UDDI ist ein Verzeichnisdienst und dient im Zusammenhang mit Web Services als ebensolcher zur Registrierung von Web Services. Damit wird ein dynamisches Finden von Web Services unterstützt.

- **WSDL**
  Die Web Services Description Language (WSDL) ist eine Beschreibungssprache die speziell für Web Services entwickelt wurde. Ein WSDL-Dokument enthält alle relevanten Informationen eines Web Services, wie zum Beispiel seinen Namen, wie er erreichbar ist, welche Funktionen (Capabilities) er bietet, welche Eigenschaten (Properties) er innehat, wie diese angefragt werden können und welche Antworten jeweils zu erwarten sind.

- **SOAP**
  Das Simple Object Access Protocol dient zur Kommunikation zwischen den einzelnen Services beziehungsweise der aufrufenden Instanz und den Services.

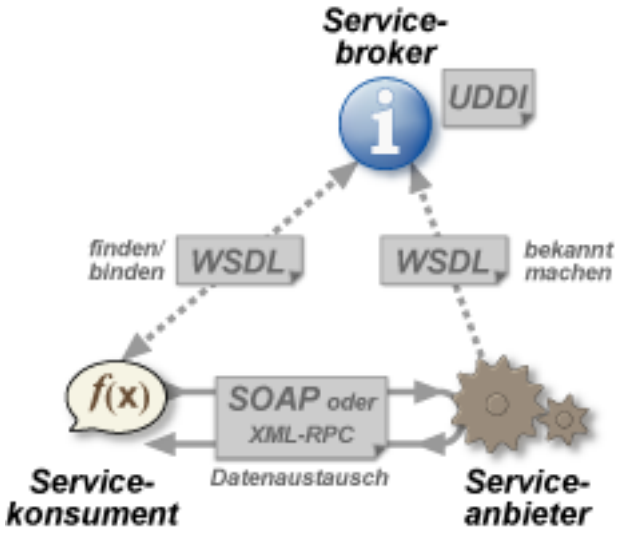

Abbildung 2.2: Web Service nach [Wik10h]

2.2.3 Web Services Resource Framework (WSRF)

Das Web Services Ressource Framework (WSRF) ([OAS10c]) ist eine Sammlung von Spezifikationen von OASIS ([OAS10a]). Das WSRF ist eine gemeinsame Entwicklung der Grid- und Web Services Communities.
Mittels den Spezifikationen des WSRF ist es laut [SC06] möglich, Web Services stateful zu machen. Dies geschieht durch Kopplung eines Web Services mit einer oder mehreren sogenannten Resources. Diese Resources sind als Bibliotheken implementiert, die die Service-Aufrufe des Web Services umsetzen. Die Verbindung eines Web Services mit einer Resource wird als Web Service Resource beziehungsweise WS-Resource bezeichnet. Diese WS-Resource ist zustandsbehaftet.
Das WSRF beinhaltet zur Benutzung von WS-Resources laut [SC06] vier Teilspezifikationen:

- **WS-ResourceProperties**
  Diese Spezifikation definiert die Interfaces, um die Eigenschaften und Variablen, die sogenannten ResourceProperties, zu durchsuchen, zu modifizieren und auf sie zuzugreifen.

- **WS-ResourceLifetime**
  Da Resources einen dynamischen Lebenszyklus unterliegen, muss dieser exakt definiert werden. Die WS-ResourceLifetime-Spezifikation definiert die dazu notwendigen Methoden und Funktionen.

- **WS-ServiceGroup**
  Diese Spezifikation erlaubt das Gruppieren von Web Services zu logischen Gruppen. Dies erleichtert die Benutzung von Web Services, denen die gleiche Aufgabe übergeben werden soll.

- **WS-BaseFaults**
  Da Web Services in der Regel entfernte Dienste darstellen, kann es bei der Kommunikation mit ihnen zu Fehlern kommen. Die WS-BaseFaults-Spezifikation definiert standardisierte Möglichkeiten zum Fehlermanagement.

Das WSRF beinhaltet jedoch keine Spezifikation, wie mit WS-Resources interagiert werden kann, sondern referenziert dabei auf die WS-Addressing-Spezifikation. Diese definiert, wie WS-Resources adressiert werden können und folglich, wie sie erreicht werden können.

2.2.4 Dienste in Grids

Grid Umgebungen werden mittels der zugrunde liegenden Middleware auf der Basis von Web Services und WS-Resources aufgezogen, daher spielen Web Services und WS-Resources eine bedeutende Rolle in diesem Zusammenhang. Alle Funktionalitäten eines Grids basieren somit auf Diensten.

2.2.5 Service Level Agreements

Ein Service Level Agreement (SLA) ist eine Dienstgütevereinbarung und bezeichnet nach [Wik10e] einen Vertrag zwischen Auftraggeber und Dienstleister. Eines der wichtigsten Ziele ist es, die Kontrollmöglichkeiten für den Auftraggeber transparent zu machen. Dies geschieht durch genaue Beschreibung der zugesicherten Leistungseigenschaften wie zum Beispiel Leistungsumfang, Reaktionszeit, Verfügbarkeitsgarantien, Schnelligkeit der Bearbeitung und so weiter. Ein Service Level ist hierbei eine festgelegte Dienstgüte, die die vereinbarte Leistungsqualität beschreibt. Charakteristisch für ein SLA ist die Tatsache, dass ein Dienstleister die relevanten Dienstleistungsparameter in verschiedenen Dienstgütestufen anbietet und sich der Auftraggeber aus diesem Katalog sein spezielles SLA zusammenbauen kann.
SLAs entstanden zuerst für IT-Dienstleistungen, da in diesem Gebiet eine genaue Definition der verschiedenen Levels möglich ist. Mittlerweile werden SLAs jedoch für alle Arten von Dienstleistungen verwendet. Bekannt geworden ist der Begriff durch die IT Infrastructure Library (ITIL).
Das Abschließen eines SLAs wird als Negotiation bezeichnet. SLAs sind jedoch keineswegs statisch, sondern in gewissem Umfang jederzeit änderbar und anpassbar. Dieser Vorgang wird als Re-Negotiation bezeichnet.

2.3 Definitionen im Zusammenhang mit Grid-Umgebungen

Definitionen rund um Grids und Grid-Umgebungen werden im Folgenden näher erläutert.

2.3.1 Allgemein

Nach [Wik10c] bezeichnet ein Grid eine Clusterung von lose gekoppelten Computern. Dieser Cluster diente ursprünglich zum verteilten Rechnen im Sinne eines Supercomputers um insbesondere wissenschaftliche sowie mathematische Probleme zu lösen. Heute werden Grids jedoch vermehrt auch im kommerziellen Umfeld eingesetzt, vor allem in forschungsintensiven Industriezweigen wie zum Beispiel Pharmaunternehmen oder aber auch im elektronischen Handel sowie im Risikomanagement, in der Baudynamik und im Bereich Finanzmanagement.
Ein Grid ist zwar eine Clusterung von Computern, jedoch gibt es gewisse Unterschiede zu „typischen“ Computerclustern. So sind Grids in der Regel sehr lose gekoppelt und unterliegen häufig einer starken Heterogenität. Die beteiligten Computer sind meist weit voneinander entfernt und daher in keiner örtlichen Nähe. Ein Grid basiert normalerweise auf standardisierten Programmbibliotheken und einer Middleware, dazu jedoch später mehr.

2.3.2 Definition

In der ersten Auflage des Buches „The Grid: Blueprint for a New Computing Infrastructure“ von Ian Foster und Carl Kesselman ([FK98] wird einer der ersten Versuche unternommen, ein Grid zu definieren:
„A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.“
„Ein Computational Grid ist eine Hardware- und Software-Infrastruktur, die einen zuverlässigen, konsistenten, von überall erreichbaren und preiswerten Zugriff auf die Kapazitäten von Hochleistungsrechnern ermöglicht.“
Das Buch ist jedoch in seiner Erstauflage bereits vor den ersten Grids erschienen und definierte somit Grids nur im theoretischen Sinne. In einer späteren Auflage ([FK03]), wurde die Definition überarbeitet:
„The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization.“
„Die gemeinsame Nutzung von Ressourcen, mit der wir uns hier beschäftigen, ist nicht primär der Austausch von Dateien, sondern vielmehr der direkte Zugriff auf Computer, Software, Daten und andere Ressourcen, wie sie bei einer Reihe von kollaborativen, problemlösenden und Ressourcen vermittelnden Strategien benötigt werden, die zurzeit in Industrie, Wissenschaft und im Ingenieurwesen auftauchen. Diese gemeinsame Nutzung von Ressourcen ist, notwendigerweise, in einem Höchstmaß kontrolliert, wobei die Anbieter und Konsumenten der Ressourcen klar und eindeutig festlegen, welche Ressourcen geteilt werden, wem die gemeinsame Nutzung erlaubt ist, und unter welchen Bedingungen die gemeinsame Nutzung

erfolgt. Eine Menge von Individuen und/oder Institutionen, die sich durch solche Richtlinien zur gemeinsamen Nutzung von Ressourcen ergeben, formen das, was wir eine Virtuelle Organisation nennen.“
In dieser überarbeiteten Definition tritt vor allem die gemeinsame Nutzung von Ressourcen durch Virtuelle Organisationen (VO) in den Vordergrund. Diese gemeinsamen Ressourcen und Virtuelle Organisationen spielen in den heutigen Grid-Implementierungen eine zentrale Rolle.
Ian Foster hat in seinem Buch „What is the Grid? A Three Point Checklist“ ([Fos02]) außerdem eine 3-Punkte-Checkliste eingeführt, um Grids genau zu spezifizieren. Nach dieser Liste ist ein Grid ein System, das:

- Ressourcen koordiniert, die nicht einer zentralen Instanz untergeordnet sind.
  Dieser Punkt besagt, dass ein Grid Ressourcen unterschiedlicher Organisationen benutzt und den jeweiligen Mitgliedern Zugriff darauf erteilt. Die Definition kann jedoch auch auf einer niedrigeren Ebene betrachtet werden, so dass ein Grid Ressourcen und Benutzer mehrerer administrativer Domänen innerhalb eines Unternehmens koordiniert und integriert. Als Ressourcen werden alle physischen als auch logischen Objekte innerhalb des Grids bezeichnet. Darunter fallen zum Beispiel Cluster, Massenspeicher, Datenbanken, Anwendungen, Strukturen, CPUs und so weiter.

- offene, standardisierte Protokolle und Schnittstellen verwendet.
  Diese Definition ist nötig, da ein Grid über mehrere Unternehmen oder Domänen verteilt sein kann und daher auf eine mögliche Heterogenität trifft. Da diese heterogenen Ressourcen gemeinsam für Rechenaufgaben oder als Datenspeicher nutzbar gemacht werden, ist es notwendig, dass ein Grid allgemein gehaltene Protokolle und Schnittstellen verwendet, die offen und standardisiert sind, sowie auf eine gemeinsame Sprache setzen. Grundsätzliche Funktionen für die Authentisierung, die Autorisierung, die Ressourcen-Ermittlung und den Ressourcen-Zugriff sollten ebenfalls standardisiert gewährleistet werden.

- nicht-triviale Dienstgüten bereitstellt.
  Ein Grid soll die bestehenden Ressourcen in einer koordinierten Art und Weise verwenden, um verschiedene Dienstgüten bereitzustellen, abhängig von beispielsweise der Antwortzeit, dem Durchsatz, der Erreichbarkeit oder Sicherheit. Um komplexen Benutzererwartungen zu entsprechen, erfolgt eine Neueinteilung mehrerer Ressourcentypen, damit der Nutzen des kombinierten Systems signifikant größer ist als die Summe seiner Teile.

Diese Checkliste trennt somit Grids von anderen Systemen, die aus gekoppelten Computern bestehen. Systeme aus den Bereichen Cluster-Computing, Peer-to-Peer-Computing, Meta-Computing und vor allem das bekannte Distributed Computing (wie zum Beispiel Seti@home ([SET10] oder BOINC([BOI10])) sind somit definitiv vom Grid Computing zu unterscheiden. Diese Systeme weisen zwar Teilaspekte des Grid Computings auf, jedoch fehlen wesentliche Punkte.
Foster, Kesselman und Tuecke definieren eines der wichtigsten Ziele eines Grids in „The Anatomy of the Grid“ ([FKT01])wie folgt:
„The real and specific problem that underlies the Grid concept is coordinated resource sharing und problem solving in dynamic, multi-institutional virtual organizations“

Aus dieser Perspektive ist also ein Zusammenschluss von Ressourcen erst dann als Grid anzusehen, wenn über dem Grid eine dynamische institutionsübergreifende Virtuelle Organisation steht und das Grid mittels gemeinsamer Ressourcennutzung zur Lösung einer gemeinsamen Aufgabe verwendet wird.

2.3.3 Herkunft

Bereits in den 60er Jahren gab es Konzepte für die Verteilung von rechenintensiven Aufgaben. Das Konzept eines Grids kommt jedoch eher aus frühen Experimenten mit Hochgeschwindigkeitsnetzen.
Der Begriff „Grid“ entstammt wahrscheinlich einem kleinen Wortspiel:
Im späten Mittelalter war man froh, als die Häuser mehr und mehr fließendes Wasser direkt im Haus bekamen, und man nicht mehr selbst für die Beschaffung von Wasser verantwortlich war.
Als der elektrische Strom zur Massenware wurde, wurde er über Kabel den Haushalten zur Verfügung gestellt, so dass man nicht mehr selbst für die Herstellung der elektrischen Energie verantwortlich war. (Im Zuge von Solar- und Photovoltaikanlagen ändert sich diese Einstellung jedoch derzeit wieder...)
In Naher Zukunft soll mittels Grids die Möglichkeit geschaffen werden, Computerrechenleistung „aus der Steckdose“ zu erhalten, so dass man als Privathaushalt oder Unternehmen nicht mehr länger selbst für die Bereitstellung von Rechenleistung mittels Computern verantwortlich ist.
Aus der englischen Bezeichnung „Power Grid“ für Stromnetz leitet sich der Begriff „Grid“ ab. Mit Grids soll es dem Benutzer ermöglicht werden, seinen Auftrag über genormte Schnittstellen an ein Grid zu übergeben. Das Grid kümmert sich dann um Ressourcenallokation und Ausführung der Berechnungen und liefert das Ergebnis an den Benutzer zurück.

2.3.4 Zielsetzung

Wie kam man nun eigentlich auf die Idee, sich das Konzept eines Grids zu überlegen?
Eine Zielsetzung war nach [FKT01] die gemeinsame, koordinierte Nutzung von Ressourcen und gemeinsame Lösung von Problemen innerhalb dynamischer Virtueller Organisationen. Das bedeutet, nachdem Abrechnungsdaten sowie Zugriffsrechte festgelegt worden sind, soll ein direkter Zugang zu Rechenleistungen, Anwendungen, Daten gemeinschaftlich ermöglicht werden.
Aktuell liegt zwar der Fokus von Grids im Bereich verteilten Rechnens, aber dennoch ist als oberste Zielsetzung die Entwicklung eines einheitlichen globalen Grids zu sehen, ganz analog zur Entstehung und Verbreitung des Internets sowie dessen Zugänglichmachen für die Allgemeinheit.

2.3.5 Klassifizierung von Grids

Grids lassen sich laut [Wik10c] in mehrere verschiedene Klassen einteilen, je nachdem wozu sie dienen.

- **Rechengrids (Computing Grids)**
  Computing Grids stellen die ursprüngliche Auffassung von Grids dar, nämlich das Bereitstellen von Rechenleistung „aus der Steckdose“, so dass sich der Benutzer keine

Gedanken mehr machen muss, woher die Rechenleistung stammt und wie seine Aufgaben abgearbeitet werden.

- **Datengrids (Data Grids)**
  Data Grids bieten Zugriff auf verteilte Datensätze beziehungsweise Datenbanken. Hier sind also nicht nur Computer zur gemeinschaftlichen Rechenleistung lose gekoppelt, sondern es werden auch Datenbestände zur gemeinsamen Nutzung zur Verfügung gestellt.

- **Service Grids**
  Service Grids sind Grids, die zur Erbringung gewisser Dienste entworfen wurden. Ein Service Grid unterscheidet sich von einem Computing Grid darin, dass ein Computing Grid Rechenoperationen von quasi beliebigen benutzerbestimmten Anwendungen ausführt, wohingegen Benutzer eines Service Grids auf die Dienste des Grids angewiesen sind.

Gemeinsam für alle Grids ist die Bereitstellung von Netzwerkressourcen. Diese bereitgestellten Ressourcen werden auch als „gridifiziert“ oder im Englischen als „gridified“ bezeichnet. Diese Ressourcen sollen innerhalb des Grids aus einem Pool von Ressourcen aufgrund bestimmter QoS-Parameter automatisch ausgewählt werden. Idealerweise sollte die Wahl der Ressourcen applikationsgetrieben, also abhängig von der Anwendung sein.

2.3.6 Architektur und Implementierungen

Es gibt mehrere Konzepte zur Architektur eines Grids laut [Wik10c]. Diese Konzepte haben jedoch alle gemeinsam, dass es sowohl nachfragende als auch koordinierende Instanzen geben muss. Die nachfragende Instanz fordert eine gewisse Leistung, die die koordinierende Instanz durch Zuteilung von Ressourcen, Rechenleistung und Zusammenführung von Teilleistungen erfüllen muss. Diese Koordination muss einer strengen Hierarchie unterliegen, die die Zuteilung der Leistungen und Ressourcen nach objektiven Kriterien verwaltet und durchführt beziehungsweise ausschließt. Im Grid ist jeder Computer zuallererst eine, den anderen Computer gleichgestellte, Einheit (Peer-to-Peer).
Aus diesem Konzept heraus folgt, dass mit zunehmender Leistung des Grids durch Ressourcenansammlung und Vergrößerung des Rechnerpools auch der Aufwand der Verwaltung und Koordination drastisch ansteigt. Daher folgt aus einer Verdoppelung der theoretischen Rechenleistung nie eine Verdoppelung der praktischen Rechenleistung. Dieser Aspekt der Overhead-Betrachtung ist in sehr engen SLA-Abkommen nicht zu vernachlässigen, tritt aber bei genügend Rechenreserven in den Hintergrund.
Die heute wohl am verbreitetste Softwarearchitektur für Grids ist die von Ian Foster mitentwickelte Open Grid Service Architecture (OGSA (siehe 2.3.9). Diese wird in Ansätzen bereits in Ihrem Vorläufer, der Open Grid Services Infrastructure (OGSI ([TCF+03])), beschrieben. Die Grundlage dieser Architektur ist die Darstellung aller Komponenten (wie zum Beispiel Rechner, Speicherplatz, sonstige Hardware), Objekte und Ressourcen (beispielsweise CPU-Leistung, Rendering-Leistung) als Grid-Services in einer offenen Komponentenstruktur. Mit der Konvergenz der Web Services des W3C ([Wor10]) und des OGSA Standards wurden Grid-Services zu Web Services, die Grid-Funktionalitäten sowie die technischen Funktionalitäten der Grid-Middleware ermöglichen. OGSA schlägt in diesem Zusammenhang den Einsatz von WSRF (dem Web Services Resource Framework (siehe 2.2.3)) als grundlegendes

Framework der Web Services vor. So bekommen die Web Services, deren Einsatz einheitliche Zugriffsverfahren auf die einzelnen Dienste eines Grids ermöglicht, zusätzlich noch einen Zustand. Erst mit diesen zustandsbehafteten (stateful) Diensten können Funktionalitäten eingeführt werden, die sich über mehrere Tranksaktionen erstrecken. Die Verknüpfung eines Web Services mit einer zustandsbehafteten Ressource und der damit verbundenen Übergabe des Zustands an den Service bildet eine sogenannten WS-Resource nach dem Web Services Resource Framework.

Foster, Kesselman, Tuecke in „Anatomy of the Grid“ ([FKT01]) beschreiben die Grid-Architektur aus einem etwas anderen Gesichtspunkt, nämlich abstrahiert in fünf Schichten.

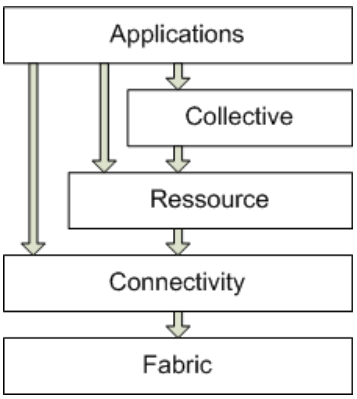

Abbildung 2.3: Grid Architektur nach [FKT01]

- **Fabric Layer**
  Die Fabric Layer ist die unterste Schicht einer Grid-Architektur. Hier befinden sich alle Ressourcen wie zum Beispiel Computer, Cluster, Supercomputer, Speicher, Datenbanken und ähnliches, die über das Grid zur Verfügung gestellt werden. Außerdem werden hier die lokalen, ressourcenspezifischen Operationen implementiert, auf die höhere Schichten zugreifen können.

- **Connectivity Layer**
  Diese Schicht stellt die grundsätzlichen Kommunikationstechnologien sowie sicherheitsrelevanten Aspekte zur Verfügung. In dieser Schicht sind die grundlegenden Kommunikationsprotokolle wie zum Beispiel TCP/IP, HTTP, DNS und ähnliche, sowie Protokolle die für die Sicherheit, sprich Autorisierung und Authentisierung, zuständig sind, um eine sichere Kommunikation zu ermöglichen, angesiedelt. Außerdem beinhaltet diese Schicht auch Standards wie X.509-Zertifikate, die Grid Security Infrastructure (GSI) und ähnliche. Darüber hinaus werden hier viele weitere Protokolle mit einbezogen, die insbesondere innerhalb eines Grids von Bedeutung in Bezug auf Kommunikation und Sicherheit sind und somit von der Grid Community als notwendig angesehen werden.

- **Resource Layer**
  In dieser Schicht werden die Protokolle zur Informationsbeschaffung und zum Management individueller Ressourcen definiert. Damit ist es möglich, Zustände von Ressourcen zu initialisieren, zu analysieren, zu beobachten und Accounting-relevante Daten

zu erhalten. Die Managementprotokolle dienen in erster Linie dazu, den Zugriff auf die Ressourcen zu ermöglichen und darauf Programme innerhalb des Grids mittels Jobs auf diesen Ressourcen zu starten und zu verwalten. Diese Schicht bezieht sich jedoch noch nicht auf die globale Interaktion mehrerer Ressourcen innerhalb des Grids, sondern nur auf individuelle Ressourcen.

- **Collective Layer**
  Diese Schicht bietet die Möglichkeit der Gruppierung von Ressourcen mittels Diensten und Protokollen. Hier werden also mehrere Ressourcen zusammengefasst und verwaltet. Diese Schicht bietet daher auch Dienste nach oben hin an. Die wichtigsten sind im Folgenden genannt:
  - **Resource registries**
    Dieser Dienst ist notwendig, um Ressourcen innerhalb einer Virtuellen Organisation zu discovern (also aufzufinden) und ihre Properties (also Fähigkeiten) zu erfragen.
  - **Allocation und Scheduling Services**
    Bei Übermittlung eines Jobs an das Grid entscheidet der Benutzer normalerweise nicht, auf welchen Ressourcen innerhalb des Grids sein Job, also seine Anwendung, zur Ausführung übermittelt wird. Der Allocation Service übernimmt genau diese Aufgabe. Er erfragt in einem Resource Directory alle von der Collective Layer discoverten Ressourcen und entscheidet, welche Ressourcen für den zu startenden Job relevant und benutzbar sind. Der Scheduling Service ist dann verantwortlich dafür, wann welcher Job gestartet wird, um das Grid nach den jeweiligen Policies und Priorities möglichst gut auszulasten.
  - **Monitoring Services**
    Es gibt verschiedene Dienste, die für das globale Monitoring zuständig sind. Diese ermöglichen die Überwachung der Ressourcen.
  - **Data Management Services**
    In der Regel brauchen die jeweiligen Jobs bestimmte Datasets, sprich Inputdaten oder Parameter. Die Data Management Dienste sind nun dafür verantwortlich, dass die den jeweiligen Jobs zugeteilten Ressourcen die notwendigen Datasets rechtzeitig bekommen. Hier kann auch die Zuteilung einer Software-Lizenz zu einer Anwendung die auf einem Computer (also Ressource) des Grids gestartet werden soll, eine Rolle spielen.
- **Application Layer**
  Hier befinden sich alle Benutzer-Anwendungen, also die Anwendungen, die der User tatsächlich auf dem Grid innerhalb einer Virtuellen Organisation ausgeführt haben will. Die Application Layer kann direkt auf alle darunter liegenden Schichten außer der Fabric Layer zugreifen.

Eines der globalen Probleme von Grids ist momentan die Tatsache, dass es keine standardisierten Protokolle zwischen den verschiedenen Grid-Middlewares gibt und die jeweiligen Middlewares die Schichten unterschiedlich interpretieren und verschwimmen lassen.

2.3.7 User Roles in Grids

Benutzer (Users) in Grids können laut [SG08] verschiedene Rollen einnehmen. So kann ein Benutzer eine oder auch mehrere Rollen innehaben. Die wichtigsten Rollen werden im Folgenden dargestellt.

- **Grid application user**
  Grid application users sind die Benutzer, die ihre Anwendungen (Applications) auf dem Grid laufen lassen wollen. Dazu müssen sie die Möglichkeit haben, Aufgaben (Jobs) an das Grid zu übermitteln. Außerdem müssen sie natürlich ihre Jobs überwachen können sowie den Fortschritt ihrer Jobs erkennen können und auf Wunsch bestimmte Jobs vor dem Ende abbrechen können. Da die meisten Jobs Eingabedaten sowie Ausgabedaten haben, müssen Grid application users auch Zugriff auf Dateien im Grid haben, wie zum Beispiel den Standardoperationen wie kopieren, verschieben, umbenennen, neue Dateien anlegen, Daten löschen und so weiter. Die Ergebnisse sollten den Grid application users sowohl lokal als auch remote zugänglich gemacht werden, damit sie die Daten auswerten können.

- **Grid operators**
  Grid operators sind eher als die „Manager“ eines Grids zu sehen. Sie müssen Zugriff auf die managed Resources des Grids haben. Außerdem sind sie verantwortlich für die Konfiguration der Grid-Infrastruktur. Die Überwachung der managed Resources des Grids fällt auch in den Zuständigkeitsbereich des Grid operators. Damit das Grid jederzeit korrekt arbeitet, sollten die Grid operators auch die Möglichkeit haben, das Grid zu testen und zu benchmarken. VO Manager gehören auch zur Kategorie der Grid operators.

- **Grid application developers**
  Grid application developers sind Entwickler, die Anwendungen für Grids entwickeln. Es gibt verschiedene Entwicklungsumgebungen für diesen Zweck. Eine davon ist das weiter unten angesprochene g-Eclipse. Die Grid application developers haben die Möglichkeit, ihre Anwendungen direkt in das Grid zu deployen. Die dort zum Testen gestartete Anwendung kann dann auf dem lokalen Computer, der zur Entwicklung dient, als auch direkt im Grid gedebuggt werden. g-Eclipse dient hierbei dem Grid application developer als Interface zwischen dem lokalem IDE (Eclipse) und der remote Grid Infrastruktur.

2.3.8 Begrifflichkeiten in Zusammenhang mit Grids

In diesem Abschnitt werden einige Begriffe, die in Zusammenhang mit Grids stehen, näher betrachtet.

Computing Resource

Als Computing Resource bezeichnet man einen Cluster, Supercomputer oder Standard-Computer, auf den durch das lokale Resource Management System (RMS) oder über eine Grid Middleware zugegriffen werden kann.

Distributed Resource Manager (DRM)

Ein Distributed Resource Manager (DRM) wird oft in Cluster-Umgebungen oder Grid-Umgebungen benutzt. Der DRM ist verantwortlich dafür, die User-Jobs auf die jeweiligen passenden High Performance Computing Resources (HPC) zu verteilen.

Grid Orchestrator

Der Grid Orchestrator ist im Prinzip der Verwalter der Jobs der Benutzer eines Grids. Der Orchestrator ist dafür zuständig, wo und wann der übermittelte Job ausgeführt wird und allokiert die zugehörigen Ressourcen.

HPC Resource

Als High Performance Computing Resource (HPC) bezeichnet man Computing Resources, die für High Performance Computing Aufgaben benutzt werden.

User-Group

Als User-Group bezeichnet man eine Gruppe von Benutzern, die in Hinsicht auf bestimmte Attribute ähnlich oder gleich sind. Im Zusammenhang mit Grids ist eine typische User-Group eine Virtuelle Organisation, die per Definition ein Zusammenschluss von Organisationen ist, die ein gemeinsames Ziel verfolgen.

Resource Broker

Ein Resource Broker kümmert sich um die Verteilung der Grid Ressourcen. So fällt unter seinen Aufgabenbereich, dass er je nach Job passende Ressourcen zuteilt und diese Ressourcen verwaltet.

Resource Management System (RMS)

Das Resource Management System ist dafür zuständig, dass die Jobs auf die lokalen Computing Resources dispatched werden.

2.3.9 Open Grid Services Architecture (OGSA)

Die Open Grid Services Architecture (OGSA) ([FKS+05]), die vom Global Grid Forum (Heute: Open Grid Forum als Zusammenschluss aus Global Grid Forum und Enterprise Grid Alliance ([Ope10a])) entwickelt wurde, zielt daraufhin ab, eine gemeinsame, standardisierte und offene Architektur für alle Grid-basierten Anwendungen zu bieten. OGSA hat sich selbst als Ziel gesetzt, dass alle Dienste eines Grid Systems (wie zum Beispiel Job Management Dienste, Resource Management Dienste, Sicherheits Dienste und so weiter) standardisiert werden. Dazu bietet OGSA Spezifikationen für Standard-Interfaces für all diese Dienste an. OGSA ist noch nicht abgeschlossen, sondern immer noch in der Entwicklung, bietet aber bereits heute einen großen Anforderungskatalog an die jeweiligen Schnittstellen und viele Standardisierungsvorschläge.
OGSA selbst basiert auf der Web Services Technologie als darunter liegende Schicht. Ein Problem ergab sich jedoch bei der Entscheidung, diese Technologie zu nutzen: Die Web

Services-Architektur erlaubt zwar prinzipiell sowohl zustandslose, als auch zustandsbehaftete Web Services, in der Regel sind sie jedoch nicht stateful, sondern stateless und es gibt keine standardisierte Möglichkeit, Web Services stateful zu machen. Diese Zustandsbehaftigkeit ist jedoch eine Grundvoraussetzung für die Open Grid Services Architecture. Daher wurde die Web Services Definition durch OGSA um eine weitere Standardisierung erweitert, nämlich die Definition, wie ein Web Service stateful wird (siehe 2.2.3).

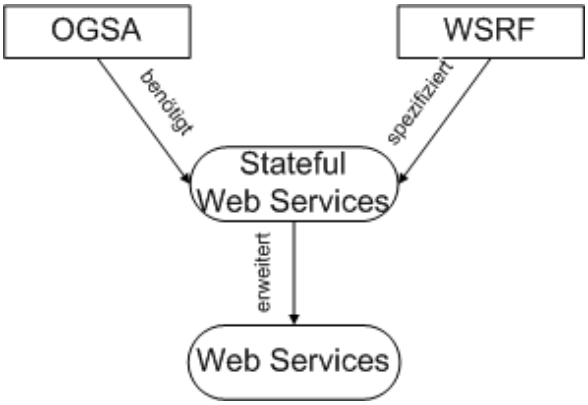

Abbildung 2.4: WSRF und OGSA nach [SC06]

2.4 Globus Toolkit 4

Abbildung 2.5: Globus Toolkit 4

Das Globus Toolkit 4 ([The09]) ist die derzeitig wohl weitverbreitetste Middleware, die es für Grids gibt. Globus Toolkit 4 bezieht sich auf die Version 4.0.8, die auch im Zuge dieser Diplomarbeit eingesetzt wurde. Eine aktuellere Version im 4.x-Branch ist die Version 4.2.1. Diese Version ist die zum Zeitpunkt der Diplomarbeit aktuellste 4.x Version. Jedoch ist seit 20.1.2010 bereits Globus Toolkit 5 verfügbar. Alle in dieser Diplomarbeit dargelegten Definitionen und Beschreibungen beziehen sich auf Version 4.0.8.
Das Toolkit bietet eine große Sammlung an Werkzeugen, die notwendig sind, um Grid-Anwendungen zu entwickeln, und die daraus resultierenden Jobs an Grids zu übergeben sowie die Ergebnisse zu erhalten. Das Globus Toolkit 4 bietet somit die Dienste an, die dafür notwendig sind, wie zum Beispiel Resource-Monitoring, Discovery, Job Submission, Security, Data Management und viele weitere. Da die Arbeitsgruppen von GGF noch immer damit beschäftigt sind, mittels OGSA weitere Dienste zu standardisieren und zu spezifizieren, kann man bisher nur sagen, dass das Globus Toolkit 4 sich zwar an die Standards und

Spezifikationen hält, aber keine „echte“ Implementierung von OGSA darstellt. Durch diese Nähe zu OGSA und dadurch, dass die OGSA-Anforderungen eingehalten werden, wird laut [SC06] das GT4 als de facto Standard-Middleware und Toolkit gesehen.
Ein Großteil der Dienste des Globus Toolkit 4 wird auf Basis des WSRF entwickelt, einzelne Dienste jedoch auch nicht. Diese Dienste werden als non-WS Komponenten bezeichnet. Das Globus Toolkit 4 bietet aber eine komplette Implementierung des WSRF und deckt daher einen sehr wichtigen Teil ab.
Im folgenden Bild werden die Zusammenhänge nach [SC06] von Globus Toolkit 4, OGSA, WSRF und Web Services dargestellt:

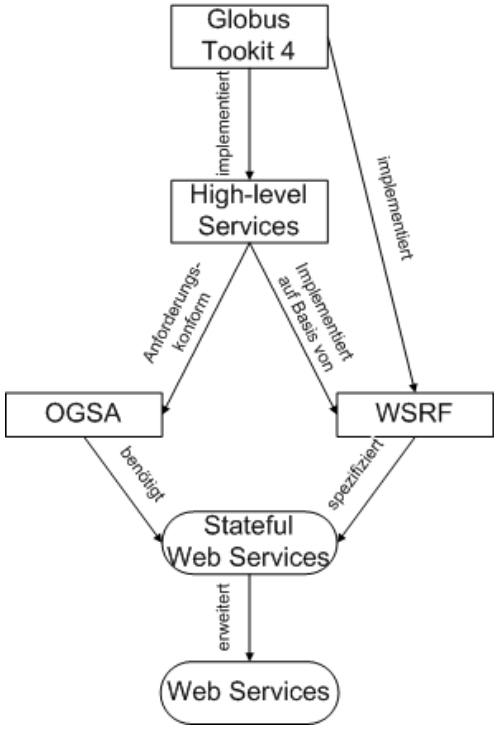

Abbildung 2.6: Globus Toolkit 4 mit OGSA und WSRF nach [SC06]

Das folgende Bild trennt die Zusammenhänge als kleines Schichtenmodell auf:

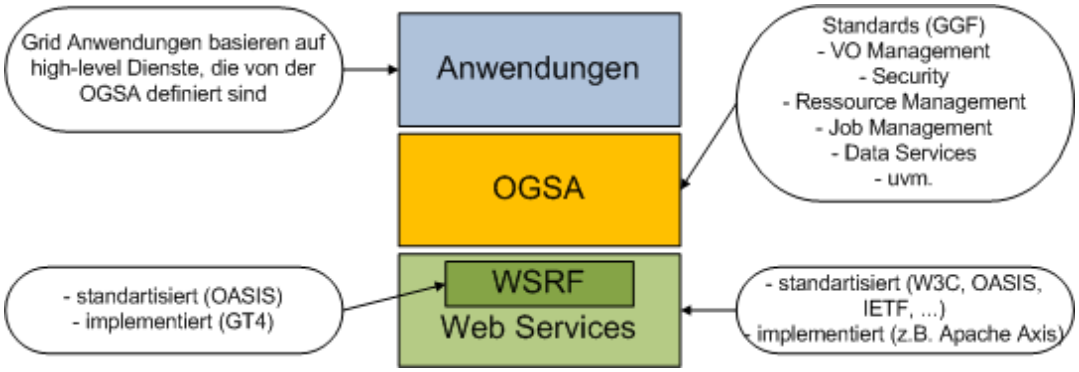

Abbildung 2.7: Schichten nach [SC06]

Das Globus Toolkit wird laut [SC06] in mehrere Komponenten unterteilt, die jeweils für sich einen gewissen Funktionsumfang abdecken. Diese Komponenten sind zueinander nur lose gekoppelt. Jede Komponente für sich besteht aus Diensten, Bibliotheken und Entwicklungswerkzeuge. Die Komponenten können nach [SC06] in fünf Bereiche gegliedert werden.

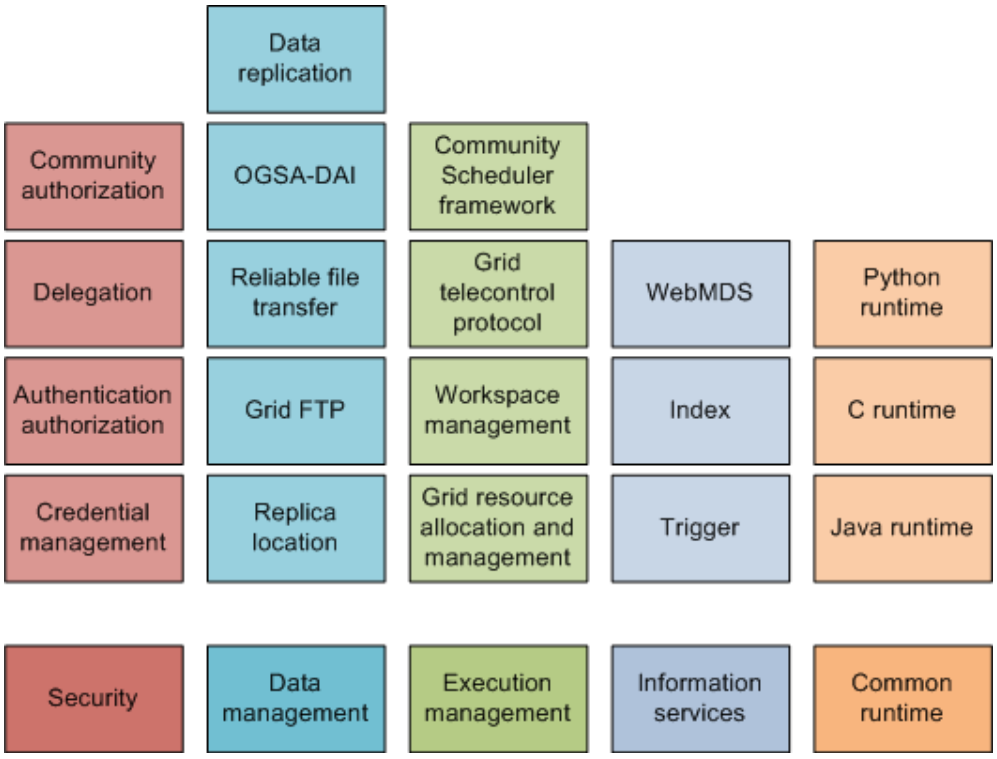

Abbildung 2.8: Globus Toolkit Komponenten nach [SC06]

Im folgenden werden die einzelnen Komponenten näher erläutert:

2.4.1 Security

Die Security Komponenten, die auch als Grid Security Infrastructure (GSI) bezeichnet werden, beschäftigen sich mit sicheren (secure) Kommunikationen und der Durchsetzung der Sicherheitspolitiken der Systeme.

- **Community Authorization**
  Virtuelle Organisationen können den Community Authorization Service zur Verwaltung der Authorization Policies ihrer Ressourcen nutzen.

- **Delegation**
  Hier sind Dienste enthalten, die notwendig sind, um Berechtigungen zu delegieren.

- **Authentication authorization**
  Diese Komponente beinhaltet Bibliotheken und Werkzeuge zur Verwaltung der Zugriffsberechtigungen auf Services und Resources. Außerdem bietet diese Komponente ein Framework, das die Möglichkeit bietet, verschiedene Authentisierungs- und Autorisierungs-Mechanismen zu verwenden und sogar selbst implementierte Mechanismen für diese Zwecke zu verwenden.

- **Credential Management**
  Diese Komponente beinhaltet die SimpleCA, eine Certificate Authority, die zur Zertifizierung des Globus Toolkit Hosts sowie den zugehörigen Benutzern dient, wenn man keine „echte“ Zertifizierungsstelle mit einbeziehen möchte. Außerdem ist in dieser Komponente der MyProxy enthalten, der ein Credential Repository darstellt.

2.4.2 Data Management

Diese Komponenten dienen zum Transferieren und Auffinden von Daten sowie deren Verwaltung und Zugriff darauf.

- **Data Replication**
  Der Data Replication Service (DRS) garantiert unter Verwendung des Replica Location Service (RLS) mittels dem Reliable File Transfer (RFT), dass lokale Kopien der Replicas auf den Hosts verfügbar sind, die diese gerade benötigen.

- **OGSA-DAI**
  Die Komponente OGSA Data Access and Integration (OGSA-DAI) bietet ein Framework, um Zugriff auf verschiedene Datasets in einem Grid zu erhalten und diese Datasets integrieren zu können.

- **Reliable File Transfer**
  Der Reliable File Transfer (RFT) ist ein Dienst, der auf einem WSRF-basierenden Web Service aufbaut und unter Verwendung von GridFTP zur Übermittlung von großen Datenmengen eingesetzt wird. Der RFT-Dienst bietet dabei weitaus mehr Möglichkeiten und Features, als GridFTP, wie zum Beispiel das Fortsetzung (Resuming) von unterbrochenen Datentransfers.

- **GridFTP**
  Diese Komponente stellt einen GridFTP-Server sowie einige Client-seitige Utilities zur Verfügung. Das GridFTP-Protokoll ist eine Abart des normalen FTP-Protokolls in Hinblick auf die Optimierung zum Transferieren von sehr großen Datenmengen.

- **Replica Location**
  Der Replica Location Service (RLS) bietet die Möglichkeit, die Replicas eines Datasets innerhalb einer Virtuellen Organisation aufzufinden, um eine Übersicht zu erhalten.

2.4.3 Execution Management

Die Execution Management Komponenten sind, wie der Name schon sagt, für die Ausführung, Verwaltung, Deployment, Scheduling und Monitoring der Anwendungen, die mittels Jobs auf dem Grid laufen, zuständig.

- **Community Scheduler Framework**
  Diese Komponente bietet ein Interface zu den verschiedenen Resource Schedulers wie zum Beispiel PBS, Condor, LSF, SGE und so weiter.

- **Grid Telecontrol Protocol**
  Diese Komponente bietet einen WSRF-basierenden Web Service zur Kontrolle von Remote Instrumenten.

- **Workspace Management**
  Mittels dieser Komponente können User dynamisch Workspaces auf den Remote Hosts anlegen und verwalten.

- **Grid Resource Allocation and Management (GRAM)**
  Diese Komponente ist das Herz des Globus Toolkit Execution Managements, denn diese Komponente bietet alle Dienste an, die zum Deployen, Ausführen und Überwachen von Jobs auf dem Grid notwendig sind.

2.4.4 Information Services

Die Information Services sind allgemein bekannt als Monitoring and Discovery System (MDS). Wie dieser Name schon sagt, befassen sich die Komponenten dieser Kategorie mit dem Monitoring und der Discovery von Ressourcen innerhalb einer Virtuellen Organisation.

- **WebMDS**
  Diese Komponente bietet Informationen zu den, durch die Globus Toolkit Aggregator Services (Index Service und Trigger Service) gesammelten Daten mittels einer Web-Oberfläche.

- **Index Service**
  Der Index Service sammelt Informationen über die Ressourcen einer Virtuellen Organisation.

- **Trigger Service**
  Der Trigger Service sammelt ebenfalls Informationen über die Ressourcen einer Virtuellen Organisation. Darüber hinaus führt er aber auch noch bestimmte Funktionen basierend auf diesen Daten beziehungsweise mit diesen Daten aus.

2.4.5 Common Runtime

Die Komponenten dieser Gruppe bieten grundlegende Bibliotheken und Werkzeuge an, um bereits bestehende Services zu hosten. Außerdem werden in dieser Gruppe Werkzeuge, Tools und Bibliotheken geboten, die zur Entwicklung neuer Services in den verschiedenen Sprachen dienen. Aktuell werden diese Objekte für Pyhton, C und Java geboten.

2.4.6 Java GT4 Container

Der Globus Toolkit 4 Container ist ein Spezialfall eines Web Services Container. Ein Web Services Container ist nach [SC06] eine Kombination aus einer SOAP-Engine zusammen mit einem Application Server und eventuell einem HTTP Server. Somit bietet ein Web Services Container eine Runtime Umgebung für Web Services. Da viele der GT4 Komponenten als Web Services implementiert sind, müssen diese Komponenten innerhalb einem Web Services Container laufen. Daher bietet Globus Toolkit 4 der Einfachheit halber selbst einen Java Web Services Container an, der den Ansprüchen genügt. Der Container basiert auf Apache Axis ([Apa10b]) und kann sowohl die bei GT4 mitgelieferten Web Services als auch benutzerspezifische Web Services mittels der GT4-WS-Implementierung ausführen. Da dieser Container nicht sehr viele Features bietet, ermöglicht Globus Toolkit, dass die Web Services in einen anderen Application Server, wie zum Beispiel Apache Tomcat ([Apa10d]) deployt werden können.

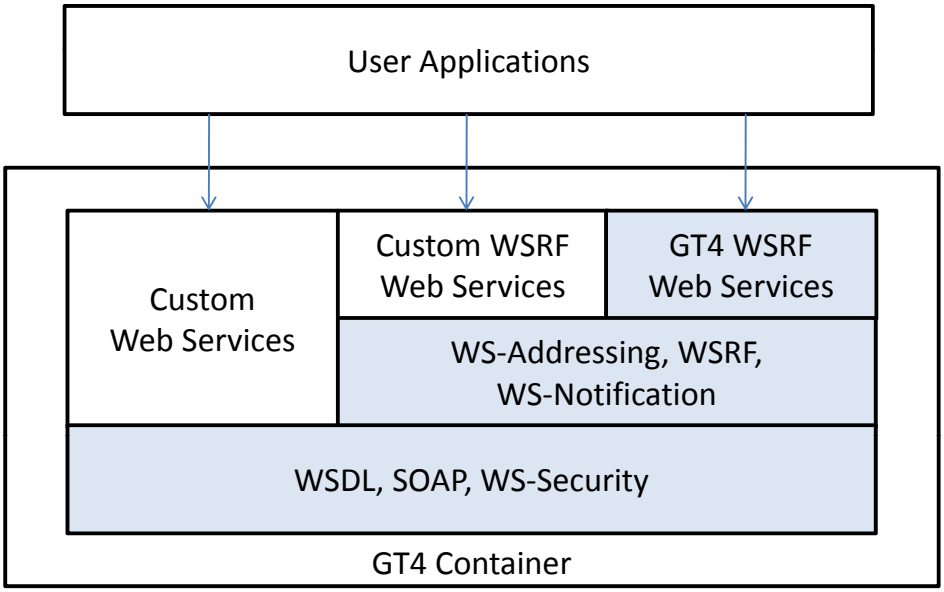

Abbildung 2.9: GT4 Container nach [SC06]

2.5 Definitionen im Zusammenhang mit Virtuellen Organisation

Der folgende Abschnitt befasst sich mit Virtuellen Organisationen, deren genaue Definition sowie den Zusammenhang mit Grids. Außerdem werden Konzepte dargelegt und die Struktur einer Virtuellen Organisation genauer erläutert.

2.5.1 Allgemein

Eine Virtuelle Organisation (VO) entsteht laut [Wik10g] durch den Zusammenschluss von rechtlich unabhängigen Unternehmen und/oder Einzelpersonen. Dieser Zusammenschluss zu einem gemeinsamen Geschäftsverbund ist in der Regel nur durch eine lose Kopplung, zum Beispiel per Internet, und normalerweise für einen begrenzten Zeitraum hinweg. Die Virtuelle Organisation ist in der Regel über mehrere physische Standorte verteilt. Gegenüber Dritten oder den Auftraggebern tritt die VO jedoch als eigenständiges und einheitliches Unternehmen auf. Ein Unternehmen kann gleichzeitig mehreren Virtuellen Organisationen angehören. Bei der Gründung einer Virtuellen Organisation wird insbesondere auf kooperative Zusammenarbeit der Partner wert gelegt. Optimalerweise ergänzen sich die Mitglieder in ihren Kernkompetenzen, so dass die Virtuelle Organisation an sich einen größeren Wissensbereich abdeckt sowie umfangreichere Produktionsmöglichkeiten besitzt.
In der folgenden Abbildung nach [Sch07] wird eine Virtuelle Organisation X dargestellt, die aus verschiedenen Ressourcen und Mitarbeitern aus zwei Organisationen besteht.

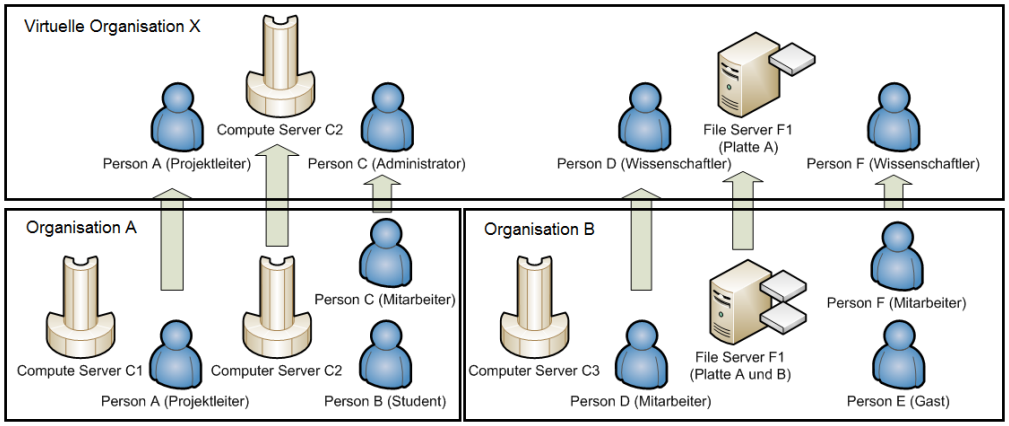

Abbildung 2.10: Virtuelle Organisation nach [Sch07]

In dieser Abbildung ist ersichtlich, dass Organisation A einen ihrer Compute Server der Virtuellen Organisation bereitstellt, sowie zwei Personen der Virtuellen Organisation beitreten. Organisation B stellt eine Festplatte des File Servers zur Verfügung und lässt 2 Personen beitreten. Man sieht hierbei, dass die Rollen, die die Personen in ihrer „echten“ Organisation einnehmen, sich von den Rollen in der Virtuellen Organisation unterscheiden (können). Diese, im wissenschaftlichen Umfeld gegründete Beispiels-VO, dient den beiden Organisationen A und B einem gemeinsamen Zweck und zum Erreichen eines bestimmten gemeinsamen Ziels. Hier ist auch ersichtlich, dass eine Virtuelle Organisation nur für eine bestimmte Zeitspanne gegründet wird und auch dynamisch Zuwachs und Abgang von Ressourcen beziehungsweise Mitarbeiter haben kann.

2.5.2 Merkmale einer Virtuellen Organisation

Virtuelle Organisationen werden laut [Wik10g] gebildet, ohne dafür ein Unternehmen zu gründen, eine Unternehmensform auszuwählen, Personal einzustellen, eine Organisation aufzubauen oder sonstige notwendige Tätigkeiten zur Gründung eines „normalen“ Unternehmens einzuleiten und durchzuführen. Wesentliche Merkmale einer Virtuellen Organisation sind somit Flexibilität sowie keine direkte Kommunikation zwischen den Mitgliedern, da die lose Organisation der einzelnen Unternehmen zur zeitlich begrenzten Ausnutzung von Marktpotentialen dient. Im Gegensatz zu anderen Kooperationen wie zum Beispiel Joint Ventures wurde auf die Institutionalisierung von zentralen Managementfunktionen verzichtet. Einzelne Teilprozesse werden auf die jeweiligen Mitglieder der Virtuellen Organisation aufgeteilt und können somit dezentralisiert bearbeitet und durchgeführt werden. Eine Virtuelle Organisation bietet somit einem Unternehmen die Möglichkeit, andere Unternehmen in einer losen Kooperation für eine bestimmte Aufgabe zu sich zu nehmen.
Die Teilnehmer einer Virtuellen Organisation sind rechtlich unabhängig, somit entsteht auch beim Zusammenschluss keine Gesellschaftsform. Die Kooperation entsteht meist nur nach Leistungsaustauschverträgen. In Deutschland fällt die Virtuelle Organisation jedoch unter die BGB-Gesellschaft, außer es wird explizit eine andere gesellschaftliche Form festgelegt.

Das Konzept der Virtuellen Organisation wird nicht nur in der Informatik verwendet, sondern auch in anderen Wissenschaften. Allen Virtuellen Organisationen gemein ist aber der darüberliegende Begriff der Organisation und der Einteilung in gewisse Kategorien beziehungsweise Gruppen und Rollen sowie Aufgabenbereiche. Der Begriff der Virtuellen Organisation taucht laut [Sch07] zum ersten Mal 1986 im angloamerikanischen Sprachraum auf. So wurden zu diesem Zeitpunkt laut [Sch07] Virtuelle Organisationen im Umfeld instrumentell orientierter Organisationen gegründet, um dem schnellen Wechsel und Änderungen im Wirtschaftskreislauf, insbesondere der Wandlung von Anbieter- und Käufermärkten sowie der Individualisierung von Produkten und Dienstleistungen nachzukommen. Die Leistungssteigerungen der Informations- und Kommunikationssysteme führten dabei zu einer Auflösung traditioneller Organisationsstrukturen zu Gunsten von Zusammenschlüssen und Kollaborationen mehrerer Organisationen.
[Sch07] beschreibt bei der Definition einer Virtuellen Organisation die grundsätzliche Trennung einer intra-organisatorischen Gestaltung im Sinne einer Zusammenarbeit innerhalb einer Organisation zur Überwindung räumlicher und zeitlicher Grenzen um die Vorteile des verteilten Operierens und des dezentral verteilten Wissens nutzen zu können. Erst die inter-organisatorische Gestaltung entspricht dem herkömmlich benutzen Begriff der Virtuellen Organisation. Ein Zusammenschluss mehrerer rechtlich selbstständiger Organisationen um mittels einem kooperativen, flexiblen Netzwerk Teile ihrer jeweiligen Ressourcen und Möglichkeiten gemeinsam nutzen zu können und die jeweiligen Kernkompetenzen sinnvoll verwenden zu können. Nach außen hin erscheint das Ergebnis einer VO als Einheit, obwohl es faktisch das Ergebnis mehrerer Teilorganisationen und Teilerbringungen von Leistungen mehrerer Organisationen sind, die mittels einem verteilten Leistungserstellungsprozess geplant, durchgeführt und vereint wurden.
Virtuelle Organisationen werden in der Literatur sehr häufig beschrieben. Diese Definitionen unterscheiden sich jedoch sehr oft oder spezialisieren den allgemeinen Begriff der Virtuellen Organisation zu genau, so dass es zu Widersprüchen von Teildefinitionen zwischen den verschiedenen Literaturwerken kommt. [Sch07] definiert eine Virtuelle Organisation sehr all-

gemein und trifft damit den Kern der jeweiligen Definitionen, ohne auf wichtige Elemente zu verzichten:
„Eine virtuelle Organisation ist eine zeitlich begrenzte koordinierte Kooperation von Elementen in Form von Individuen, Gruppen von Individuen, Organisationseinheiten oder ganzer Organisationen, die Teile ihrer physischen oder logischen Ressourcen oder Dienste auf diesen, ihre Kenntnisse und Fähigkeiten sowie Teile ihrer Informationsbasis in Form virtueller Ressourcen und Dienste über eine Grid-Infrastruktur derart zur Verfügung stellen, dass die gemeinsam vereinbarten Ziele unter Berücksichtigung lokaler und globaler Policies erreicht werden können."

2.5.3 Vor- und Nachteile einer Virtuellen Organisation

Der Zusammenschluss von Unternehmen zu Virtuellen Organisationen bietet laut [Wik10g] natürlich Vorteile, aber auch Nachteile.

Vorteile

- Durch die Gründung einer Virtuellen Organisation entsteht selbstverständlich ein Kosteneinsparungspotenzial. Wohl eines der wichtigsten Ziele von Wirtschaftsunternehmen. Kosten können gespart werden durch den Wegfall von Raum-, Organisations- und Reisekosten sowie selbstverständlich durch den Wegfall durch Einkäufe von Fremdwissen oder -produktionsmöglichkeiten.
- Mittels Virtuellen Organisationen ist es möglich, sich schnell und flexibel auf wechselnde Marktanforderungen anzupassen beziehungsweise wissenschaftliche Informationen auszutauschen und zu erhalten.
- Virtuelle Organisationen bieten im Prinzip eine Symbiose zwischen den einzelnen Mitgliedern. Jedem einzelnen Unternehmen werden neue Möglichkeiten geboten, dafür, dass dieses Unternehmen den anderen Mitgliedern ebenfalls Möglichkeiten bietet.
- Da jedes Mitglied seine jeweiligen Kernkompetenzen in die Virtuelle Organisation einbringt, kommt es zu einem effizienteren Arbeitsergebnis, als wenn jedes Unternehmen für sich allein handeln würde. Auch ist es damit möglich, neue Märkte gemeinsam zu betreten, da die Kosten nicht mehr auf das jeweilige Unternehmen einzeln umgewälzt werden.

Nachteile

- Da Virtuelle Organisationen grundsätzlich nur lose gekoppelt sind, sind diese schwierig zu kontrollieren. Da die einzelnen Mitglieder der VO stark voneinander abhängen beziehungsweise sich ergänzen, kann es passieren, dass einzelne Mitglieder diese Tatsache zu ihrem eigenen Vorteil ausnutzen (zum Beispiel Weitergabe von vertraulichen Daten an Dritte). Virtuelle Organisationen sollten daher nur zwischen Unternehmen gegründet werden, die sich vertrauen oder bereits Geschäftspartner sind.
- Die Mitarbeiter der einzelnen Unternehmen einer Virtuellen Organisation können auch Probleme mit der Identifizierung der Virtuellen Organisation haben. Insbesondere Personen, die „klassische" Unternehmensstrukturen gewöhnt sind.

- Virtuelle Organisationen setzen eine voll funktionsfähige Informations- und Kommunikationstechnologie voraus. Bei einem Ausfall oder Problemen kann es zu komplexen Folgefehlern kommen.

2.5.4 Grundbegriffe einer Virtuellen Organisation

Nach [GHP+08] spielen folgende Oberbegriffe in Virtuellen Organisationen eine wichtige Rolle.

Virtuelle Ressource und virtueller Dienst

Eine Virtuelle Organisation stellt virtuelle Ressourcen und virtuelle Dienste bereit. Diese Bereitstellung bedeutet, dass aus der jeweiligen VO heraus auf die Ressourcen zugegriffen werden kann und Dienste benutzt werden können.

Mitglied

Wenn jemand berechtigt (autorisiert) ist, Ressourcen oder Dienste, die von einer Virtuellen Organisation bereitgestellt werden, zu benutzen, so ist diese Person ein Mitglied dieser VO. Auch wenn eine Person einer Rolle in der VO zugeordnet werden kann, ist diese Person Mitglied der VO.
Personen und Individuen können Gruppen bilden. Eine solche Gruppe ist Mitglied einer Virtuellen Organisation, wenn *jedes* Gruppenmitglied ein Mitglied der VO ist.
Eine Organisation hingegen ist bereits Mitglied einer Virtuellen Organisation, wenn *mindestens ein* Organisationsmitglied ein Mitglied der VO ist. Andererseits ist eine Organisation auch bereits dann Mitglied einer VO, wenn die Organisation eine Ressource oder einen Dienst der VO zur Verfügung stellt.

2.5.5 Virtual Organization Membership Service

Der Virtual Organization Membership Service (VOMS) ist laut [Wik10f] ein Dienst innerhalb Grid-Umgebungen zur Authentisierung und Autorisierung der Grid-Benutzer. Der Dienst basiert auf einer einfachen Account Datenbank mit festgelegten Formaten zum Informationsaustausch. Er unterstützt Single-Sign-On, Expiration Time und bietet die Möglichkeit, User mit Zugehörigkeit zu mehreren Virtuellen Organisationen zu verwalten. Mittels administrativer Tools kann ein VO-Administrator die Rollen und Gruppen der jeweiligen Nutzer in der Datenbank anpassen. Die Benutzer, die in der VOMS-Datenbank gelistet sind, können mittels einem Tool einen lokalen Proxy auf ihren Daten basierend generieren. Dieser Proxy dient als Berechtigungsnachweis und somit zur grundsätzlichen Authentifizierung. Dieser VOMS-Proxy beinhaltet die gleichen Informationen wie ein Standard Grid Proxy und zusätzlich die jeweiligen Rollen- und Gruppeninformationen aus der VOMS-Datenbank. Anwendungen, die mit der VOMS-Struktur harmonieren, können die VOMS-Daten zur Authentifizierung innerhalb des Grids beziehungsweise der Virtuellen Organisation heranziehen.

Komponentenstruktur

Das VOMS System besteht laut [ACC+] aus folgenden Komponenten:

- **User Server**
  Der User Server empfängt Anfragen von einem Client und antwortet mit den Informationen über den angefragten User.
- **User Client**
  Der User Client setzt eine Anfrage mittels einem User Zertifikat an den User Server ab. Er erwartet als Antwort Informationen über die Gruppen, Rollen und Capabilities dieses Users.
- **Administration Client**
  Der Administration Client wird indirekt von den VO-Administratoren zum Hinzufügen von Benutzern, Anlegen neuer Gruppen, Definieren von Rollen und so weiter benutzt.
- **Administration Server**
  Der Administration Server nimmt die Anfragen des Administration Clients entgegen und führt die geforderten Updates in der Datenbank aus.

Kommunikationsablauf

Das folgende Bild zeigt die grundsätzliche Kommunikation:

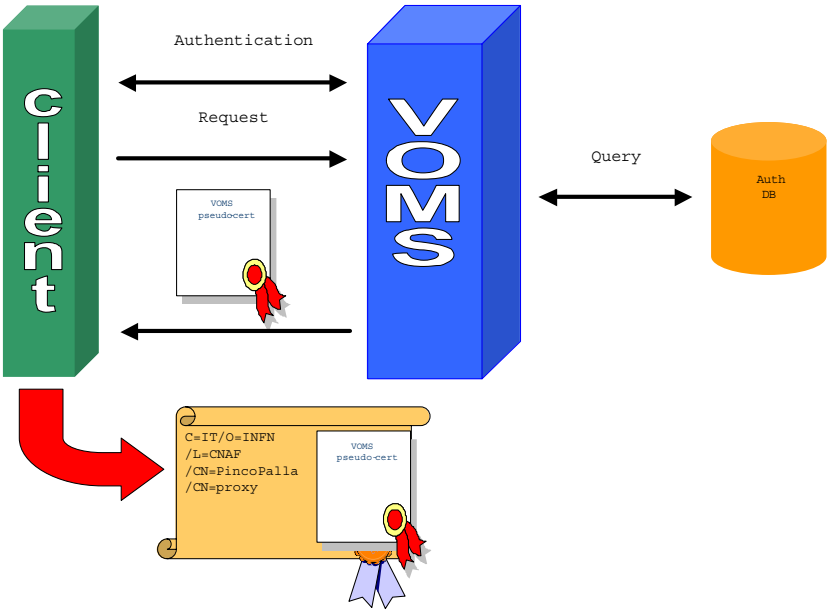

Abbildung 2.11: VOMS Architektur nach [ACC+]

Client und Server bauen nach einer gegenseitigen Authentisierung einen sicheren Kommunikationskanal auf. Über diese Verbindung stellt der Client als nächstes seine Anfrage an den Server. Der Server bearbeitet die Anfrage nach einer Überprüfung auf Korrektheit dieser Anfrage und sendet die angeforderten Informationen signiert mit dem eigenen Zertifikat zurück. Als letztes validiert der Client die Antwort. Der Client kann nun ein Proxy-Zertifikat generieren, das die vom VOMS-Server signierten Informationen enthält und mit diesem weiterarbeiten.

Technische Betrachtung aus User-Sicht

In Grid-Umgebungen ohne VOMS wird das Proxy-Zertifikat mittels *grid-proxy-init* generiert. Dieses Proxy-Zertifikat ermöglicht dem User die Benutzung des Grids. Beim Einsatz von VOMS muss natürlich ein VOMS-Proxy-Zertifikat generiert werden. Dies geschieht laut [ACC+] mittels *voms-proxy-init*. Dieses Zertifikat beinhaltet wie oben beschrieben die Standardinformationen des normalen Proxy-Zertifikats und zusätzlich die VOMS-spezifischen User-Informationen. Das VOMS-Proxy-Zertifikat beinhaltet somit sowohl die Benutzer- als auch die Server-Berechtigungen sowie eine Gültigkeit. Der VOMS-Server hat dieses Informationspaket signiert. Der Middleware-Gatekeeper ist nun nicht nur zuständig für die notwendige Authentisierung des Benutzers, sondern muss auch die erweiterten VOMS-Informationen zur Autorisierung heranziehen.

Technische Betrachtung aus Administrator-Sicht

Der Administrator-Server kann mittels dem SOAP-Protokoll erreicht werden. Daher kann er laut [ACC+] leicht in einen OGSA Service konvertiert werden. Der Server an sich besteht aus drei Komponenten.

- **Core**
  Der Core ist die Kernkomponente des Servers, die für grundlegende Aufgaben verantwortlich ist.
- **Admin**
  Diese Komponente verfügt über die notwendigen Mittel zur Administration der VOMS-Datenbank.
- **History**
  Die History-Komponente bietet Funktionen für das Logging und Accounting.

Die Einträge der VOMS-Datenbank besitzen alle laut [ACC+] die Spalten *createdBy* und *createdSerial*. Die *createdBy*-Spalte beinhaltet die ID des Anfragestellers sowie die Operation, die dazu führte, dass die Zeile angelegt wurde. Die *createdSerial*-Spalte beinhaltet eine Datenbankweite eindeutige sortierte Seriennummer, die bei jeder Modifikation in der Datenbank hochgezählt wird. Die Operation ist somit eindeutig anhand dieser Transaktions-ID identifizierbar.
Wenn eine Zeile aus der VOMS-Datenbank komplett gelöscht werden soll (zum Beispiel, weil es einen Benutzer nicht mehr gibt), so wird diese Zeile in eine Archiv-Tabelle verschoben. Diese Tabelle besitzt dasselbe Schema wie die eigentliche Datenbank und hat zusätzlich noch zwei weitere Spalten: *deletedBy* und *deletedSerial*. Erstere beinhaltet den Administrator, der den Eintrag aus der eigentlichen VOMS-Datenbank entfernt hat und Zweitere referenziert

die Transaktions-Nummer der Operation, die zur Löschung führte. Diese komplette Daten-Archivierung dient insbesondere zur Rückverfolgung aller Operationen und Aktionen und zur späteren Prüfung von Benutzerrechten.

Sicherheitsbetrachtungen

Laut [ACC+] fügt der VOMS-Server keine weiteren Sicherheits-Features auf User-Ebene hinzu. Er prüft klassischerweise das User-Zertifikat bevor er Rechte zuteilt. Dieses Zertifikat muss selbstverständlich von einer Certificate Authority (CA) signiert sein, der der Server vertraut. Außerdem darf das Zertifikat natürlich nicht revoked sein und muss gültig sein.
Falls der VOMS-Server kompromittiert werden würde, könnte dennoch kein illegaler Zugriff auf die Ressourcen erfolgen, da die Autorisierungsdaten in einem User-Proxy-Zertifikat vorliegen müssen, das vom Benutzer selbst signiert ist. Jedoch bietet der VOMS-Server ein Angriffsziel für Denial-of-Service-Attacken. Bei einem erfolgreichen DoS-Angriff könnten die Benutzer ihre Autorisierungs-Daten nicht mehr vom VOMS-Server abholen und somit auch nicht mehr die Ressourcen nutzen.
Problematisch ist darüber hinaus, dass die Proxy-Zertifikate keinen Revocation-Mechanismus besitzen und daher ihre Laufzeit nicht vorzeitig beendet werden kann. Dies ist jedoch auch nur ein relatives Sicherheitsrisiko, da diese Zertifikate in der Regel nur eine kurze Gültigkeit von einigen Stunden besitzen.

2.6 Definitionen im Zusammenhang mit Lizenzen

Dieser Abschnitt beschäftigt sich mit Lizenzen. Zuerst wird allgemein betrachtet, was genau eine Lizenz ist. Danach wird darauf eingegangen, welche verschiedenen klassischen Lizenzschemata es gibt und wofür diese eingesetzt werden können und vor allem wann diese Schemata sinnvoll sind.

2.6.1 Was sind Lizenzen?

Der Begriff „Lizenz“ stammt laut [Wik10d] vom lateinischen *licere* ab, was soviel wie „erlauben“ bedeutet. Eine Lizenz „erlaubt“ somit gewisse Dinge, die ohne Lizenz nicht erlaubt wären. Jemandem eine Lizenz für etwas geben wird als „lizenzieren“ bezeichnet. Damit ist gemeint, jemandem eine Erlaubnis für etwas zu erteilen. Umgangssprachlich wird jedoch unter „lizenzieren“ oft fälschlicherweise angenommen, dass man eine Lizenz erwerbe (also aus Sicht des Lizenznehmers).
Eine Lizenz wird immer vom Lizenzgeber zum Lizenznehmer übergeben.

2.6.2 Lizenzen in der Softwareentwicklung

Lizenzen werden im Softwarebereich laut [Wik10a] vergeben, um Personen oder Institutionen, Rechte zur Nutzung einer bestimmten Software zu erteilen. Diese sogenannten Endbenutzer-Lizenzvereinbarungen beziehungsweise Endbenutzerlizenzverträge werden auch als EULA (aus dem Englischen: End User License Agreement) bezeichnet. Dieser Vertrag beinhaltet zusätzlich zur Lizenz auch noch gewisse Regelungen, die den Benutzer einschränken sowie Haftungen des Lizenzgebers beschreiben.

Anders als in den meisten Ländern gilt in Deutschland hier ein Sonderfall laut [AB03]: Lizenzvereinbarungen bei sogenannter Standardsoftware sind nur dann gültig, wenn sie bereits vor dem Kauf vereinbart wurden. Sollten die Lizenzvereinbarungen erst nach dem Kauf einsehbar sein oder darauf hingewiesen werden, so sind diese wirkungslos. Auch in dem Fall, dass der Benutzer der Software während der Installation den Vereinbarungen zustimmt. Sollten die Lizenzbedingungen direkt beim Kauf sichtbar sein, zum Beispiel durch Bereitstellen der Bedingungen als pdf unmittelbar vor dem Absenden der Order beziehungsweise im Ladengeschäft auf der Rückseite der Verpackung der Software, so gelten diese Lizenzvereinbarungen laut [AB03] als Allgemeine Geschäftsbedingungen, die der Inhaltskontrolle durch die AGB-Regelungen des BGB unterliegen. Daraus folgt in Deutschland, dass viele Lizenzbedingungen oder Teile davon ungültig sind, da sie den Nutzer einseitig oder ungewöhnlich einschränken (§ 407 BGB) oder gegen konkrete Vorschriften nach § 308 BGB oder § 309 BGB verstoßen.

2.6.3 Begriffe im Zusammenhang mit Lizenzen

Da nun geklärt ist, was genau Lizenzen sind und wofür sie wichtig sind, wird im Folgenden erläutert, welche Begrifflichkeiten dabei außerdem eine Rolle spielen. Diese Definitionen wurden [MBC+08] entnommen.

Application Service Provider (ASP)

Ein Application Service Provider (ASP) bietet Anwendungen innerhalb eines Netzwerks. Diese Anwendungssoftware wird auf einem zentralen Server gehostet, der auch einen Lizenzserver beinhalten kann, so dass der Benutzer sowohl auf die Anwendungen zugreifen kann, als auch direkt die notwendigen Lizenzen erhält.

Feature

Als Feature bezeichnet man einen Teil einer Software, der einzeln lizenziert werden kann. Einige Anwendungen bestehen aus mehreren Features, so dass der Anwender nur die Teile erwerben muss, die er auch tatsächlich nutzen will.

Independent Software Vendor (ISV)

Independent Software Vendor (ISV) ist die grundsätzliche Bezeichnung für Software-Entwickler. Die ISVs haben daher die Möglichkeit, ihre entwickelten Anwendungen unter Zuhilfenahme von Lizenzen zu verkaufen.

License Contract Framework

Als License Contract Framework bezeichnet man den Vertrag, der bei Lizenzierung einer Anwendung oder einem Dienst zwischen Lizenzgeber und Lizenznehmer geschlossen wird. In diesem Vertrag werden alle Bestimmungen und Einschränkungen, die bei Erwerb der Lizenz vereinbart wurden, dargestellt. Der Vertrag beinhaltet daher grundsätzliche Regelungen zur Nutzung (und eventuell Weitergabe) der lizenzierten Software, sowie die Kosten (und eventuell Folgekosten) der Lizenzierung. Außerdem sind in diesem Vertrag Haftungsausschlüsse geregelt.

License protected Software

Eine License protected Software ist eine Anwendung, die mittels einer Lizenz geschützt ist. Diese Lizenz dient vorwiegend der Durchsetzung von Copyrights sowie Nutzungs- und Weitergaberechten.

License Server

Ein License Server ist eine Anwendung, die Lizenzen verwaltet. Der Lizenzserver verwaltet somit die Lizenzpools aller seiner betreuten Kunden. Außerdem besitzt er Informationen, um alle Anfragen nach Lizenzen bearbeiten zu können.

License System Administrator

Der License System Administrator ist verantwortlich für den License Server sowie den dazugehörigen notwendigen Kommunikationen. Teil seiner Aufgabe ist auch das Einspielen von Lizenzen auf den License Server, so dass diese von Benutzern abgeholt werden können. Auch muss der Administrator die Lizenzen up-to-date halten und deren Nutzung nach dem Lizenzvertrag ermöglichen.

License Virtualization

Als License Virtualization bezeichnet man das Anlegen und Reservieren von Lizenzen von einem License Server.

Negotiation

Als Negotiation bezeichnet man die Zustimmung des License Contract Frameworks einer Lizenzgeschützen Software. Aus dieser Zustimmung resultiert ein Service Level Agreement oder ein rechtsgültiger Lizenzvertrag.

User

Ein User ist eine Einzelperson oder Firma, die eine Lizenz benutzt.

License Scheduler

Ein License Scheduler wird in der Regel dann eingesetzt, wenn die Anzahl der gesamten Lizenzen nicht ausreicht, um einen reibungslosen Betrieb zu gewährleisten, so dass es immer wieder zu Pausieren oder Blockieren von einzelnen Anwendungen oder Jobs kommt, da keine Lizenz verfügbar ist. Ein License Scheduler kümmert sich nun lokal um die Verteilung der Lizenzen. In der Regel wird dabei ein First-Come-First-Serve-Prinzip eingesetzt. Selten gibt es auch andere Verfahren wie zum Beispiel Round Robin, so dass zwar jeder mal eine Lizenz zur Verfügung gestellt bekommt, aber nie lange. Eine deutlich sinnvollere Lösung wäre es, mehr Lizenzen zu erwerben, was jedoch wirtschaftliche Konsequenzen hat.

2.6.4 Lizenzschemata

In diesem Abschnitt werden herkömmliche Lizenzschemata betrachtet. Je nach Vermarktung oder Einsatzzweck wird Software unterschiedlich lizenziert. Diese Modelle beziehungsweise Schemata haben dabei unterschiedliche Vorteile und Nachteile.
Viele Applikationen werden unter mehreren Lizenzmodellen lizenziert, so dass es eine Absprache mit dem Kunden geben sollte, welche Lizenzierung Sinn hat und welche eher nicht. Dabei sollten sowohl die wirtschaftlichen Interessen des Lizenznehmers, als auch die wirtschaftlichen Interessen des Lizenzgebers berücksichtigt werden. Auch muss bei der Wahl des Lizenzmodells an die Sicherheits-Politiken und damit zusammenhängende Einschränkungen der Kommunikation gedacht werden.

Einzelplatzlizenz / Node-Locked-License

Die Einzelplatzlizenz beziehungsweise Node-Locked-License ist eine Lizenz, die an einen bestimmten Computer gebunden ist. Das bedeutet, die Anwendung, die mittels einer solchen Lizenz lizenziert ist, kann nur auf diesem einen Computer verwendet werden. Als klassisches Beispiel ist hier das Windows-Betriebssystem von Microsoft zu nennen. Die Lizenz von Windows erlaubt die Nutzung an genau einem Rechner. Microsoft bestimmt in den Lizenzbedingungen dazu sogar, dass der beim Kauf beiliegende Lizenzaufkleber auf den Computer angebracht wird. Dies bindet die Lizenz an genau diesen einen Computer. Damit wird der Weiterverkauf unterbunden, was in manchen Ländern rechtlich verboten ist. Daher ist dieses Beispiel in manchen Ländern nicht ganz der Einzelplatzlizenz entsprechend.

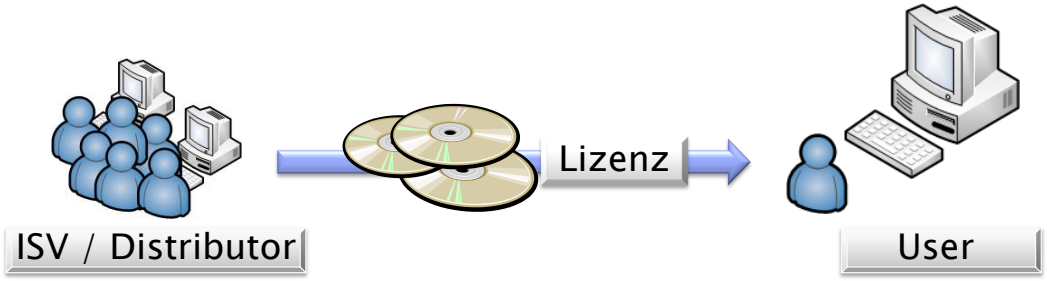

Abbildung 2.12: Node-Locked-Lizenz

Floating-Lizenz

Floating-Lizenzen werden heutzutage von vielen Software-Vendors eingesetzt. Ein Lizenznehmer erwirbt eine bestimmte Anzahl von Lizenzen, zum Beispiel zehn Stück, und ist damit berechtigt, die lizenzierte Anwendung auf beliebigen Computern von beliebigen Benutzern insgesamt zehn mal gleichzeitig zu verwenden. Zur Überprüfung der Einhaltung der maximalen Anzahl gleichzeitiger Instanzen ist in der Regel ein Lizenzserver zuständig, der die Lizenzen je nach Anfrage dynamisch verteilt und zuweist. Eine weitere Möglichkeit der Überprüfung der Einhaltung ist die Anwendung selbst, die nach bereits laufenden Instanzen im Netzwerk oder auf dem lokalen Rechner sucht. Diese Methode setzt jedoch ein hohes Maß an Vertrauen an den Lizenznehmer voraus, da diese Überprüfung sehr leicht mittels Firewalls oder Zugriffsberechtigungen zu umgehen ist.

Beim Einsatz eines Lizenzservers ist die Sicherheitspolitik des Lizenznehmers nicht zu vernachlässigen, da eine Kommunikation zwischen Server und Client notwendig ist. Der Lizenzserver kann einerseits beim Lizenznehmer untergebracht sein, was zu einem erhöhten Administrationsaufwand des Lizenznehmers führt und wieder gewisse Risiken in Hinblick auf Manipulation des Servers birgt. Andererseits kann der Lizenzserver zentral beim Independent Software Vendor zur Verwaltung aller Lizenznehmer auf einer Serverinstanz untergebracht werden. Dies führt jedoch für den Lizenzgeber zu einem erhöhten administrativen Aufwand. Außerdem muss der Lizenznehmer dem Lizenzgeber vertrauen, dass keine Daten, die für eine Lizenzanfrage relevant sind, an den zentralen Lizenzserver gesendet werden. Desweiteren muss eine Kommunikation von Lizenznehmer (eventuell über einen Lizenz-Proxy) zum Lizenzserver des Lizenzgebers ermöglicht werden und somit eventuell die Sicherheits-Politiken angepasst werden.
Ein allgemeiner Problempunkt bei Einsetzen eines Lizenzservers ist die Verfügbarkeit des Servers beziehungsweise eine Festlegung, was bei einem Ausfall des Servers passieren soll.

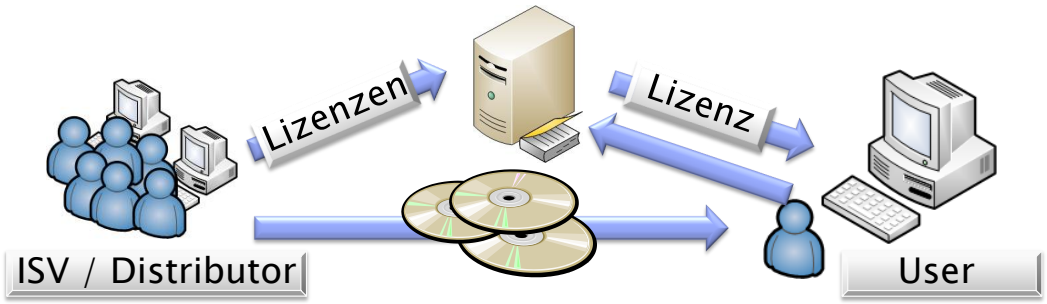

Abbildung 2.13: Floating-Lizenz

Usage-Based Lizenz beziehungsweise Pay-per-Use-Lizenz

Usage-Based-Lizenzen, die auch als Pay-per-Use-Lizenzen bezeichnet werden, setzen einen umfangreichen Monitoring- und Accounting- Einsatz voraus, um den Gebrauch der lizenzierten Anwendung mit dem Lizenzgeber korrekt abzurechnen.
Eine Usage-Based-Lizenz kann als Spezialfall einer Floating-Lizenz gesehen werden. Der Lizenzgeber stellt jedoch in diesem Fall keine feste Anzahl an gleichzeitig nutzbaren Lizenzen zur Verfügung, sondern genehmigt die Nutzung einer bestimmten Anzahl von Lizenzen innerhalb einer bestimmten Zeitspanne. So könnte ein Lizenzvertrag zum Beispiel die Nutzung von zehn Lizenzen für jeweils 24 Stunden an 30 Tagen innerhalb 30 Tage erlauben. Dieses Beispiel bietet dem Lizenznehmer nun die Möglichkeit alle 10 Lizenzen rund um die Uhr einzusetzen. Als Alternative kann er jedoch auch 40 Instanzen je 6 Stunden an 30 Tagen innerhalb 30 Tagen nutzen, um zum Beispiel an Load-Peaks nicht ohne Lizenz dazustehen. Alternativ wäre es auch möglich nur 15 Tage der 30 Tage rund um die Uhr 20 Lizenzen zu nutzen. Dieses Lizenzmodell bietet also eine sehr hohe Flexibilität der Nutzung und ist sehr Lizenznehmer-freundlich, da der Lizenznehmer für selten auftretende Peaks eine deutlich größere Anzahl an Lizenzen erwerben muss.
Das soeben dargelegte Modell stellt im Prinzip ein Prepaid-Modell dar. Alternativ lässt sich das Usage-Based-Modell auch als Postpaid-Modell darstellen. Hierbei wird im License-Contract der Preis einer Lizenz pro Zeiteinheit vereinbart mit eventuellen Maximal-Be-

schränkungen oder sonstigen Einschränkungen. Die Kosten werden erst im Nachhinein abgerechnet. Dies hat für den Lizenznehmer den Vorteil, dass er keine ungenutzten Lizenzen bezahlt, sondern nur für tatsächlich genutzte Lizenzzeiten bezahlen muss.
Sicherheitspolitisch sind die selben Fragen wie bei einer Floating-Lizenz zu klären sowie überdies hinaus inwieweit anonymisiert die tatsächlichen Usage-Statistiken abgerechnet werden. Dies ist jedoch eine Frage des Datenschutzes und nicht des dahinter liegenden Konzeptes.

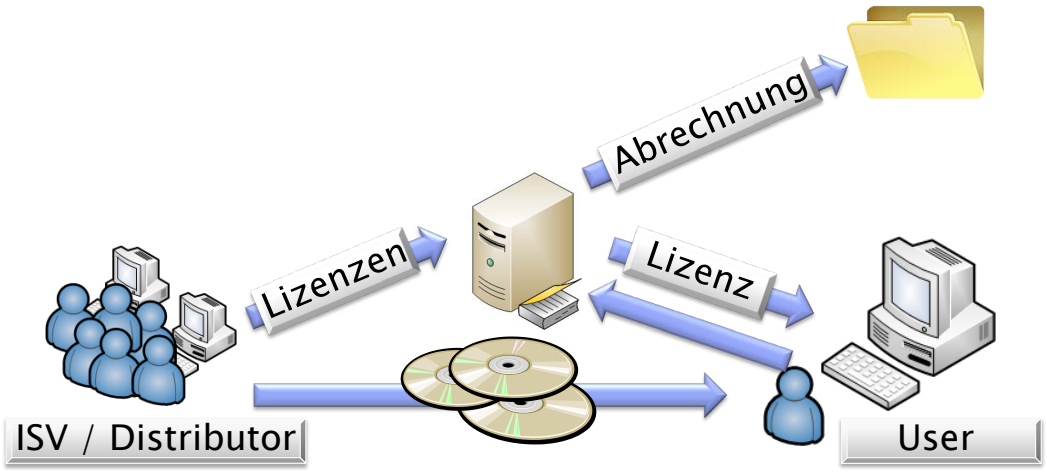

Abbildung 2.14: Usage-based Lizenzen

On-Demand-Lizenz

Ein Spezialfall einer Pay-per-Use-Lizenz ist die On-Demand-Lizenz. Wie der Name schon suggeriert, ist eine On-Demand-Lizenz nur temporär verfügbar. On-Demand-Lizenzen werden nach Anfrage zeitweilig zur Verfügung gestellt. Bezahlt wird dann genau diese Zeit. Nach Nutzung der On-Demand-Lizenz ist der Vorgang abgeschlossen. Somit bietet sich der Einsatz von On-Demand-Lizenzen dann an, wenn eine Anwendung nur einmalig oder sehr selten gebraucht wird oder noch nicht vorhersehbar ist, ob diese Anwendung jemals wieder benötigt wird. On-Demand-Lizenzen werden häufig von Application Service Providern eingesetzt, um Kunden die Möglichkeit zu geben, eine Anwendung, die der Kunde nicht selbst besitzt, zu nutzen.

Evaluation/Demo Lizenz

Als Evaluationslizenz oder Demo-Lizenz bezeichnet man die Lizenzierung einer Software zum „Ausprobieren“. Das bedeutet für den Lizenznehmer, dass er eine Lizenz bekommt, die den vollen Funktionsumfang der Anwendung zusichert aber in der Regel kurzfristig zeitlich beschränkt ist, zum Beispiel auf zwei Wochen. Meist wird hierbei ein einfaches Lizenzmodell gewählt, so dass der Lizenznehmer zum Ausprobieren der Software keinen großen Aufwand hat. Eine Evaluationslizenz wird interessierten Kunden zugeteilt, um diese als feste Kunden zu bekommen.

2.6.5 Möglichkeiten zur Sicherstellung der korrekten Lizenznutzung

Dieser Abschnitt befasst sich mit Möglichkeiten, die zur Durchsetzung des Lizenzvertrags dienen. Diese Durchsetzung ist wichtig, damit der ISV sicher sein kann, dass die Lizenzen nur legal und mit dem Contract übereinstimmend genutzt werden, da ein ISV sich nicht auf Ehrlichkeit des Vertragspartners verlassen kann.

Lizenzdatei

Eine Software ist in der Regel immer gebundlet mit einer Lizenzdatei. Diese Lizenzdatei enthält Informationen über die Software, den Benutzer beziehungsweise die Hardware auf der die Software ausgeführt werden darf. Außerdem beinhaltet die Datei einen verschlüsselten Hash über diese Informationen. Beim Anwendungsstart überprüft die Anwendung das Vorhandensein der Lizenzdatei, die darin enthaltenen Informationen sowie die Validität. Außerdem weiß die Anwendung, wie der Hash zu bilden ist und hasht die Informationen gefolgt von einer Validitätsprüfung des Hashes aus der Lizenzdatei. Eine absolute Absicherung gegen den Missbrauch beziehungsweise multiplen Einsatz solcher Lizenzen ist der Einsatz einer Lizenzdatei erst dann, wenn genügend überprüfbare Informationen enthalten sind.

Zählung der laufenden Instanzen

Zusätzlich zum Vorhandensein von Lizenzdateien kann die Anwendung beim Start oder in regelmäßigen Zeitabständen mittels einem Broadcast die gleichzeitig laufenden Anwendungen auf anderen Computern in derselben Domäne abfragen, um das Kontingent von Floating-Lizenzen zu überprüfen. Dieser Ansatz ist jedoch sehr eingeschränkt nutzbar, da häufig Broadcasts komplett verworfen werden oder eine restriktive Firewall einfach die Anfragen nach weiteren Instanzen unterbindet. Daher ist diese Zählung ein sehr rudimentärer und vielleicht sogar naiver Ansatz, um Zählungen von Floating-Lizenz-Instanzen durchzuführen.

Lizenzserver

Ein weitaus besserer aber auch deutlich komplexerer Ansatz als die triviale Instanz-Zählung, ist der Einsatz eines Lizenzservers. Ein Lizenzserver beziehungsweise ein Cluster von Lizenzservern ist dafür verantwortlich, dass er Benutzer beziehungsweise Computer authentisiert und ihnen, falls sie autorisiert sind, temporär Lizenzen zuteilt, die er einem Lizenzpool entnimmt. Der anfragende Benutzer kann diese Lizenz(en) dann solange nutzen, wie er sie braucht beziehungsweise er sie abgeben muss, da ein priorisierter Nutzer die Lizenzen benötigt. Der Lizenzserver kann einerseits bei der Organisation beziehungsweise Firma stehen, die die Anwendung benutzen möchte. Dann muss jedoch die Firewall soweit geöffnet sein, dass der Lizenzserver selbstverständlich interne Anfragen nach Lizenzen entgegen nehmen kann, aber auch von extern vom betreuenden ISV oder Distributor administriert werden kann. Andererseits kann der Lizenzserver direkt beim Independent Software Vendor oder Distributor untergebracht sein. Dann muss die Firewall bei den Anwendungsnutzern soweit geöffnet werden, dass Verbindungen zu diesem externen Lizenzserver möglich sind. Außerdem muss der ISV oder Distributor seine eigene Firewall soweit öffnen, dass er Verbindungen von allen Kunden entgegen nehmen kann. Der Einsatz von Lizenzservern muss also mit den jeweiligen Firmen-Politiken vereinbar sein.

Lizenzserver ermöglichen den Einsatz von Pay-per-Use-Lizenzen, sowohl Prepaid als auch Postpaid, da der Lizenzserver das Accounting und Billing übernehmen kann.

Dongle

Ein eher klassischer Ansatz ist der Einsatz von Hardwaredongles. Diese Geräte werden an den Computer über eine Schnittstelle (früher oft seriell am COM-Port, heute auch USB oder intern PCI / PCI-e) angeschlossen. Der Dongle ist im Lieferumfang der einzusetzenden Software enthalten und wird somit als Teil des Produkts verkauft. Im Dongle sind im Prinzip ähnliche Informationen enthalten wie in einer Lizenzdatei. Das Physisch-Vorhandensein des Dongles unterbindet somit die multiple Nutzung einer Lizenz.
Heute wird ein Dongle jedoch selten zum Schutze von Einzelplätzen eingesetzt, sondern meist nur zum Schutze eines Lizenzservers. Dabei wird sichergestellt, dass ein Lizenzserver nur funktionsbereit ist, wenn er Zugriff auf „seinen“ Dongle hat und dieser als gültig validiert wurde.

Hashbildung

Allen Sicherheitsmaßnahmen liegt die Bildung von Hashes zugrunde. Ein verschlüsselter Hash ist eine einfache Möglichkeit um Daten auf Veränderung zu überprüfen, daher finden Hashes allgemein bei der Überprüfung von Lizenzen eine wichtige Rolle.

2.7 g-Eclipse

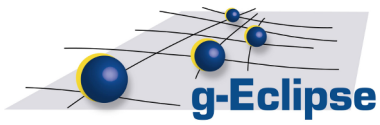

Abbildung 2.15: g-Eclipse

g-Eclipse ist eine Entwicklungsumgebung insbesondere für den Bereich von Grid-Umgebungen. Zum Zeitpunkt der Diplomarbeit ist die aktuellste Version 1.0 RC3 vom 30.11.2009. Das Entwicklungspaket g-Eclipse ist zu beziehen von der offiziellen Homepage ([g-E09]) und lauffähig nach Extraktion des Archivs.

2.7.1 Allgemein

Die Entwicklungsumgebung g-Eclipse ist basierend auf der weit verbreiteten Entwicklungsumgebung Eclipse ([Ecl10a]), die ursprünglich eine reine Java-Entwicklungsumgebung darstellte. Seit einiger Zeit gibt es jedoch schon viele Pakete auch für andere Entwicklungsumgebungen wie php, JaveEE, C++ und weitere. Durch sogenannte Plugins lässt sich die Entwicklungsumgebung jedoch beliebig erweitern. Das Open-Source-Projekt g-Eclipse ist laut [Wik10b] ein Teilprojekt des Technologieprojekts der Eclipse Foundation und wird seit 2006 aktiv von einem Konsortium bestehend aus dem Forschungszentrum Karlsruhe, dem Poznan Supercomputing and Networking Center, der Johannes Kepler Universität Linz, der Universität Zypern, der Innoopract GmbH (Deutschland), der Universität Reading und IT Innovation Centre weiterentwickelt. Außerdem wird das g-Eclipse Konsortium durch das 6. Forschungsrahmenprogramm gefördert.
g-Eclipse ist im Prinzip eine Sammlung und Entwicklung von Plugins für Eclipse, die bereits eingebunden und konfiguriert sind, so dass auf der offiziellen Homepage von g-Eclipse ([g-E09]) nur noch ein fertiges Archiv gedownloadet werden muss, um das g-Eclipse-Framework zu bekommen. Einige Plugins sind in diesem Archiv jedoch bereits veraltet, so dass g-Eclipse nach dem ersten Start manuell mit dem internen Update-Verfahren geupdatet werden sollte.

2.7.2 Warum g-Eclipse verwenden?

Die Entwicklungsumgebung g-Eclipse ist ein Open-Source-Framework zur Unterstützung von Anwendern, Administratoren und Entwicklern bei der Arbeit mit Grids.
g-Eclipse erweitert laut [Wik10b] Eclipse mit Hilfe der Plugins um die dazu nötigen Funktionalitäten und grafischen Benutzeroberflächen sowie spezielle Funktionen, mit deren Hilfe auf existierende Grid-Infrastukturen zugegriffen werden kann und diese verwaltet werden können. Außerdem ist es mit g-Eclipse möglich, Anwendungen zu erstellen und in Grids zu deployen. g-Eclipse bietet darüber hinaus die Möglichkeit, Grids direkt zu administrieren und die Ressourcen einzurichten beziehungsweise zu überwachen.
Da g-Eclipse Open-Source ist und außerdem auf der Plugin-Struktur von Eclipse aufsetzt, ist es selbstverständlich möglich das g-Eclipse-Framework noch eigenen Wünschen anzupassen und sogar durch weitere Plugins zu erweitern.

2.7.3 Perspektiven

- **Anwenderperspektive**
  Diese Perspektive richtet sich an den Anwender von Grid-Ressourcen. Hier ist es möglich, Anwendungen im Grid zu starten und zu beenden. Die Perspektive hilft außerdem bei der Überwachung der Anwendungsausführung und erlaubt das Verwalten der eigenen Daten.

- **Operatorperspektive**
  Die Operatorperspektive ist, wie der Name schon andeutet, auf den Bedarf eines Grid-Administrators zugeschnitten. Mittels verschiedener Views und Editoren lassen sich lokale Ressourcen und insbesondere Virtuelle Organisationen verwalten.

- **Developerperspektive**
  Diese Perspektive dient dem Entwickler von Anwendungsprogrammen zum Erstellen, Debugging und Installieren von Grid-Anwendungen. Eine Integration mit anderen Eclipse-Entwicklungswerkzeugen wie C/C++ Development Tooling (CDT) und Java Development Tools (JDT) ist ebenfalls gegeben.

2.7.4 Das g-Eclipse Framework

Das eigentliche Eclipse-Framework basiert laut [SG08] auf Java und läuft somit auf allen gängigen Betriebssystemen wie Linux, Windows, Sun Solaris und Max OS X. Da g-Eclipse darauf basiert, läuft es höchstwahrscheinlich ebenso auf allen gängigen Betriebssystemen, wurde aber offiziell nur für Windows, Linux und Mac OS X freigegeben und getestet.
Sehr viele Software-Entwickler sind bereits mit Eclipse in Berührung gekommen, so dass sie mit den Grundfunktionalitäten von g-Eclipse bereits vertraut sind.
Eclipse besteht aus vielen einzelnen Modulen, die als Plugins realisiert sind. Auf Grund dieser Struktur bietet Eclipse jederzeit die Möglichkeit einzelne Module der Architektur auszutauschen oder abzuändern. Eclipse geht aber sogar einen Schritt weiter als herkömmliche Software, die nur eine Erweiterung mittels Plugins erlauben. In Eclipse ist im Prinzip alles ein Plugin und kann während der Laufzeit je nach Bedarf dynamisch geladen oder entfernt werden. Ein Plugin von Eclipse kann selbstverständlich auch auf andere Plugins zugreifen. Dazu ist es nötig, dass jedes Plugin angibt, welches Plugin zum korrekten Ausführen benötigt wird. Das dem Eclipse-Framework zugrunde liegende OSGi-Framework bietet dazu die notwendigen Möglichkeiten, um Dependencies zwischen den Plugins zu definieren sowie festzulegen, zu welchem Zeitpunkt welche Plugins aktiv sein müssen. Eclipse bietet aber nicht nur die Plugin-Funktionalität, sondern auch die Möglichkeit von Extension-Points. Diese Extension-Points definieren, wie eine bestehende Funktionalität erweitert werden kann. Dazu wird von einem Plugin ein Interface bereitgestellt, das jemand anderes implementieren kann und somit die Funktionalität von diesem Plugin erweitern kann. Ein weiteres Plugin implementiert dann diese Schnittstelle und gegebenenfalls noch weitere Funktionalitäten. Die Plugins können folglich sehr stark ineinander verwoben sein und voneinander abhängig sein.
Diese technischen Erweiterungsmöglichkeiten führen letztendlich dazu, dass g-Eclipse an sich von der darunter liegenden Grid-Middleware unabhängig ist. Wenn eine Middleware bestimmte Funktionalitäten und Mechanismen benötigt, können diese mittels Plugins einfach dem bestehenden g-Eclipse-Framework während der Laufzeit hinzugefügt werden.

2.7.5 Nativ unterstütze Middleware

Das interne Modell von g-Eclipse ist im Prinzip unabhängig von der verwendeten Grid-Middleware. Zum Zeitpunkt dieser Diplomarbeit existieren für g-Eclipse spezifische Grid-Middleware-Implementierungen für gLite sowie für GRIA.

2.8 Zusammenfassung des 2. Kapitels

Im 2. Kapitel wurden alle grundlegenden Definitionen und Beschreibungen erläutert, die für das Verständnis dieser Diplomarbeit wichtig sind. Zuallererst wurde definiert, was genau ein Dienst ist und wo dieser Begriff im Zusammenhang mit Grids auftritt. Anschließend wurde ausführlich erklärt, um was es sich konkret bei einem Grid handelt und was man unter Grid-Computing versteht. Dabei wurden die Begrifflichkeiten aus der Welt der Grid-Umgebungen erklärt. Grids haben bereits in ihrer Entstehungsgeschichte mehrmals einen Wandel der Definition erlebt. Es wurde erläutert, wie ein Grid eindeutig als Grid identifiziert werden kann und wie es technisch aufgebaut ist.
Die technische Grundlage von Grids bilden sogenannte Middlewares, die auf verschiedensten Spezifikationen beruhen und die grundlegende Implementierung von Grid-Umgebungen anbieten. Als eine der am häufigsten eingesetzten Middlewares wurde das Globus Toolkit vorgestellt. Dabei wurde sowohl die interne Funktionsweise, als auch die theoretischen Hintergründe erklärt.
Da Grids in der Regel innerhalb Virtuellen Organisationen eingesetzt werden, wurde dieser Begriff in den Kontext von Grid-Umgebungen gerückt und aufgezeigt, um was es sich bei Virtuellen Organisation handelt, wie diese entstehen, wozu sie dienen und welche begrifflichen Zusammenhänge bestehen.
Zusätzlich zu diesen Definitionen wurden nach einer Spezifikation des Lizenz-Begriffs klassische Möglichkeiten der Lizenzierung dargestellt und Begriffe bezüglich der Softwarelizenzierung erläutert.
Als Abschluss dieses Kapitels wurde die g-Eclipse-Entwicklungsumgebung kurz betrachtet und ein grober Überblick darüber gegeben.

3 Entwicklung eines Anforderungskatalogs

In den vorangegangenen Kapiteln wurde eine Übersicht über Lizenzmodelle und Möglichkeiten zur Lizenzvergabe in normalen (eventuell vernetzten) Umgebungen gezeigt. Dieses Kapitel befasst sich nun mit Überlegungen, die sowohl die Lizenzvergabe unter „normalen“ Bedingungen sowie die speziellen Gegebenheiten in Grids berücksichtigen. Ganz konkret wird dieses Kapitel die Entwicklung eines Anforderungskatalogs darstellen, der alle Anforderungen an ein Lizenzschema innerhalb Grid-Umgebungen reflektiert. Dabei soll berücksichtigt werden, dass die Anforderungen die Grid-spezifischen Problematiken abdeckt sowie die besonderen Gegebenheiten innerhalb einer Virtuellen Organisation berücksichtigt. Dieser Anforderungskatalog wird mittels Fallbeispielen erarbeitet.

3.1 Beispielszenarios

Dieser Abschnitt stellt einige Use-Cases beispielhaft dar, um einen möglichen Einsatz von kommerzieller, lizenzierter Software in Grid-Umgebungen darzulegen und einige dort auftretende Problematiken aufzuzeigen. Die Beispiele dienen dabei als Einleitung in die Problematik und vor allem zur Herleitung der Anforderungen.

3.1.1 Fallbeispiel 0: Das unmögliche Szenario

Dieses Szenario ist ein *unmögliches Szenario*. Das Fallbeispiel beschreibt ein Unternehmen, bezeichnen wir es als UC0, das eine Anwendung einsetzen möchte, deren Nutzungsrechte das Unternehmen zusammen mit einer Lizenz erworben hat. Die Anwendung prüft bei ihrer Ausführung auf das Vorhandensein eines Hardware-Dongles, der im Lieferumfang der Anwendung enthalten war. Nur bei Vorhandensein und erfolgreicher Prüfung auf Gültigkeit dieses Dongles, kann die Anwendung benutzt werden. UC0 hat 5 Lizenzen für den gleichzeitigen Gebrauch erworben und somit auch 5 Hardware-Dongles erhalten. Diese Dongles sind somit nur lokal einsetzbar und nur auf Computing Resources, die man physisch erreichen kann. Somit ist es ein Ding der Unmöglichkeit, externe Computing Resources oder Resources eines anderen Unternehmens im Sinne einer Virtuellen Organisation einzusetzen.
Nehmen wir nun an, anstatt eines Hardware-Dongles liege eine Node-locked-License vor: Auch in diesem Fall ist es für UC0 unmöglich, die Anwendung auf externe Computing Resources einzusetzen beziehungsweise auf zufälligen Ressourcen einer Virtuellen Organisation, da die Anwendung explizit an bestimmte Computer gebunden ist.
Es zeigt sich in diesem Szenario, dass der Einsatz eines Hardware-Dongles beziehungsweise der Einsatz von Node-locked-Licenses ein erhebliches Problem darstellt und in der Regel nicht lösbar ist.

3.1.2 Fallbeispiel 1: Privates Grid

Dieses erste Fallbeispiel ist aus eigener Erfahrung sehr verbreitet.
In diesem Use-Case existiert ein Unternehmen, nennen wir es UC1, das eine rechenintensive Anwendung ausführen möchte zu kommerziellen Zwecken zum eigenen Nutzen. UC1 hat die notwendigen Nutzungsrechte vom Entwickler der Software erworben. Außerdem hat UC1 eine gewisse Anzahl an Lizenzen erworben, die für die Nutzung der Anwendung notwendig sind. Diese Lizenzen basieren auf einem Floating-Schema, das den Einsatz der Lizenzen innerhalb der Domäne von UC1 gestattet. Damit UC1 die gewünschten Rechenergebnisse schnell bekommt, besitzt UC1 mehrere Computing Resources auf denen die Anwendung parallel ausgeführt werden kann. Diese Computing Resources sind innerhalb eines Grids angesiedelt, das von UC1 aufgesetzt wurde und auf das nur UC1 Zugriff hat. Es handelt sich also um ein privates Grid.
Die Anwendung benötigt auf jeder Computing Resource eine Lizenz, damit sie ausgeführt werden kann. Diese Lizenz bekommt sie von einem Lizenzserver, der lokal bei UC1 gehostet wird. Der Lizenzserver verwaltet ausschließlich den Lizenzpool, in dem die Lizenzen der Anwendung für UC1 gelistet sind. Verwaltet wird der Lizenzpool direkt vom ISV, der Remote-Zugriff auf den Lizenzserver besitzt.
Dieses Szenario ist durchaus realistisch, da viele Unternehmen derzeit mit Grids experimentieren und selbst ein kleines Grid aufsetzen.
Die größte Problematik in diesem Szenario ist, dass von den Grid-Ressourcen kein Zugriff auf einen klassischen License-Server möglich ist. Außerdem sind es vor allem wirtschaftliche Gründe, die eine wichtige Rolle in diesem Szenario spielen. Klassischerweise basiert eine Lizenzierung auf Floating-Lizenzen. Dies ist sehr teuer, da der Peak die Gesamtanzahl der Lizenzen bestimmt. Eine Lösung mittels Pay-per-Use-Lizenzen wäre hingegen relativ günstig, da nur zu Peak-Zeiten eine große Anzahl von Lizenzen für eine in der Regel kurze Zeit benötigt werden.

3.1.3 Fallbeispiel 2: Computing Resources von einem Resource Provider

Dieses Szenario handelt von einem Unternehmen, bezeichnen wir es als UC2, das selbst keine Hochleistungsrechner besitzt, aber diese von Zeit zu Zeit benötigt. UC2 hat jedoch wie auch UC1 eine Anwendung samt Lizenzen erworben. Da UC2 aber selbst keine Hochleistungsrechner besitzt und meist nur wenige der Lizenzen auf den eigenen Rechnern parallel nutzt und daher auch kein Grid oder Cluster privat für den eigenen Zweck hosten möchte, bietet sich jedoch für UC2 an, von einem externen Resource Provider Ressourcen temporär zu mieten, falls für kurze Zeit dennoch eine hohe Rechenleistung benötigt wird. Dies hat den Vorteil, dass UC2 keine eigenen Computer anschaffen muss, aber dennoch den Vorteil von Hochleistungsrechnern auf Basis eines Mietvertrags nutzen kann, wenn Bedarf besteht.
Der Lizenzserver, den UC2 wie auch UC1 selbst hostet, verwaltet analog zu UC1 die Lizenzen.
Solange UC2 die Anwendung nur auf den lokalen eigenen Computern in der eigenen Domäne einsetzt, besteht kein Problem mit der Lizenzzuteilung. Erst wenn hohe Rechenleistung gefordert wird und UC2 vorübergehend Ressourcen von einem Resource-Provider mietet, kommt es zu Problemen:

- Wie kommuniziert die Anwendung, die auf den externen Computing Resources läuft mit dem lokalen Lizenzserver?
- Woher weiß die Anwendung, wen sie kontaktieren muss?
- Muss die lokale Firewall von UC2 für alle in Frage kommenden externen Resource-Provider geöffnet werden?
- Was passiert, wenn der Lizenzserver nicht antwortet oder die Antwort beziehungsweise die Anfrage auf dem Weg durch das Internet verloren geht?
- Der Lizenzserver lehnt die Lizenzanfrage ab, da die ausführenden Computing Resources nicht in der Domäne von UC2 liegen. Was nun?
- Die Lizenzen sind auf die Nutzung innerhalb der Domäne von UC2 beschränkt, aber die Computing Ressourcen liegen theoretisch gesehen in der Domäne des Resource Providers, sind jedoch praktisch zu diesem Zeitpunkt für UC2 reserviert und somit als temporärer Teil von UC2 zu sehen: Darf UC2 überhaupt die Anwendung auf den Computing Resources eines Resource-Providers nutzen oder ist das eine illegale Handlung gegenüber dem Lizenzvertrag zwischen UC2 und dem Distributor beziehungsweise ISV?

Erweitern wir dieses Szenario noch um eine Kleinigkeit: Die Anwendung wird von mehreren Jobs eventuell gleichzeitig benötigt, da mehrere Abteilungen unabhängig die Anwendung für ihre jeweiligen Projekte einsetzen möchten.

- Die Jobs können sich gegenseitig blockieren, da sie nicht genügend Lizenzen zugeteilt bekommen, aber die bereits erhaltenen Lizenzen nicht freigeben. Wie ist mit diesem Deadlock umzugehen?

Dieses Fallbeispiel zeigt sehr klar auf, dass der Einsatz einer Anwendung auf fremden Computing Resources nicht auf einfache Art lösbar ist. Zum einen ist die Kommunikation zwischen Anwendung und Lizenzserver an sich bereits problematisch. Sollte dieses Problem gelöst sein, so findet sich das nächste Problem bei der erfolgreichen Authentifizierung und somit Lizenzzuteilung. Und auch wenn die Authentifizierung erfolgreich durchgeführt werden kann, besteht noch immer eine rechtliche Divergenz zwischen der Lizenznutzung und dem Lizenzvertrag.
Außerdem kann es zu einem Deadlock kommen, wenn mehrere Jobs gleichzeitig Lizenzen benötigen und gleichzeitig die bereits erhaltenen nicht freigeben.
Dieses Fallbeispiel zeigt bereits die grundlegenden Problematiken einer Softwarelizenzierung in Grid-Umgebungen. Die oben genannten Fragestellungen sind Voraussetzung für einen erfolgreichen Einsatz der beschrieben Anwendung im dargelegten Einsatzgebiet.

3.1.4 Fallbeispiel 3: Shared Resources einer Virtuellen Organisation

Dieses Szenario beschreibt eine Virtuelle Organisation, die aus Personen und Ressourcen verschiedener realer Organisationen besteht. Aus den in Kapitel 2 beschriebenen Gründen wurde diese Virtuelle Organisation gegründet. Ein Unternehmen dieser Virtuellen Organisation bezeichnen wir als UC3a, ein anderes Unternehmen als UC3b.
Diese Virtuelle Organisation bietet nun jeder beteiligten Person der beteiligten Unternehmen die Möglichkeit alle beteiligten Computing Resources zu nutzen.
Nehmen wir nun an, Unternehmen UC3a besitzt Lizenzen einer Anwendung, die sie auf dem Grid der Virtuellen Organisation ausführen wollen. Der Lizenzserver wird von UC3a gehostet, der ISV beziehungsweise Distributor hat auf den dort vorhandenen Lizenzpool Zugriff.
Unternehmen UC3b besitzt ebenfalls eine gewisse Anzahl von Lizenzen derselben Anwendung, hat aber einen eigenen Lizenzpool. Außerdem wird der notwendige Lizenzserver nicht bei UC3b gehostet, sondern direkt im Lizenzserverpool des ISVs. Unternehmen UC3b möchte ebenfalls die Anwendung auf dem Grid der Virtuellen Organisation ausführen.
Aus Sicht der beiden Unternehmen treten ähnliche Probleme auf wie im Fallbeispiel 2, da aus Sicht der beiden Lizenzserver einige der Computing Resources aus fremden Domänen stammen.

- Wie kommuniziert die Anwendung, die auf den Ressourcen von UC3b läuft mit dem lokalen Lizenzserver von UC3a?
- Woher weiß die Anwendung, wen sie kontaktieren muss?
- Für welche Computing Resources muss UC3a die Firewall zum Lizenzserver öffnen?
- Was passiert, wenn der Lizenzserver von UC3a nicht antwortet oder die Antwort beziehungsweise die Anfrage auf dem Weg durch das Internet zwischen UC3a und UC3b verloren geht?
- Der Lizenzserver lehnt die Lizenzanfrage ab, da die ausführenden Computing Resources von UC3b nicht in der Domäne von UC3a liegen. Was nun?
- Der Lizenzserver, den UC3b kontaktieren muss, nimmt nur Anfragen von autorisierten Domänen entgegen, also in diesem Fall nicht von Computing Resources, die aus der Domäne von UC3a stammen. Wie kann UC3b also die gemeinsamen Computing Resources überhaupt nutzen?
- Die Lizenzen von UC3a sind auf die Nutzung innerhalb der Domäne von UC3a beschränkt, aber einige der Computing Resources liegen theoretisch gesehen in der Domäne von UC3b, sind jedoch praktisch zu diesem Zeitpunkt für UC3a reserviert und somit als temporärer Teil von UC3a zu sehen: Darf UC3a überhaupt die Anwendung auf den Computing Resources von UC3b nutzen oder ist das eine illegale Handlung gegenüber dem Lizenzvertrag zwischen UC3a und dem Distributor beziehungsweise ISV?
  Selbige Fragestellung gilt analog für UC3b.

Außerdem treten noch einige sehr spezifische Problematiken auf, die jedoch eher rechtlicher oder organisatorischer Natur sind:

- Darf und kann UC3a Lizenzen von UC3b nutzen, wenn UC3b diese im Moment nicht benötigt und umgekehrt? Kann und darf ein Lizenz-Sharing stattfinden?

- Falls UC3a die Nutzung seiner Lizenzen dem Unternehmen UC3b untersagt, wie wird sichergestellt, dass UC3b keinen Job auf Computing Resources von UC3a startet und behauptet, der Job sei von UC3a initiiert?

Nehmen wir nun an, dass die Virtuelle Organisation selbst auch Lizenzen ebendieser Anwendung erworben hat. Dies ist möglich, da nach [Wik10g] eine Virtuelle Organisation als eigenständige rechtliche Einheit auftritt. In diesem Fall gibt es einige weitere interessante Fragestellungen:

- Wessen Lizenzen werden bei Job-Ausführung benutzt? Die des jeweiligen Unternehmens oder die der Virtuellen Organisation?
- Darf UC3a beziehungsweise UC3b bei Bedarf die Lizenzen der Virtuellen Organisation verwenden?
- Angenommen die Lizenzen der Virtuellen Organisation sind auf einer Pay-per-Use-Basis: Wie wird sichergestellt, dass zuallererst die Floating-Lizenzen der einzelnen Unternehmen benutzt werden und erst bei Bedarf die Lizenzen der Virtuellen Organisation?
- Wie erfolgt dir korrekte Abrechnung?

Dieses Fallbeispiel einer Virtuellen Organisation mit einem dazugehörigen Grid beschreibt sehr deutlich die Problematiken, die dort auftreten. Teilweise decken sich diese mit den Problematiken aus dem Fallbeispiel 2, einige sind jedoch neu hinzugekommen. Diese betrachten vor allem rechtliche sowie organisatorische Aspekte, die innerhalb einer Virtuellen Organisation auftreten. Diese Aspekte betreffen einerseits das grundlegende Accounting und Billing und andererseits die eigenen Firmen-Policies sowie die Lizenzverträge.

3.2 Anforderungen an ein Lizenzmodell

In den vorangegangenen Beispielen wurde gezeigt, dass klassische Lizenzschemata, die aus herkömmlicher non-Grid-Softwarenutzung bekannt sind, in Grid-Umgebungen nicht mehr oder nur sehr eingeschränkt funktionieren. In diesem Abschnitt werden die Anforderungen aufgezeigt, die an ein Lizenzmodell für Grid-Umgebungen gestellt werden. Zuerst werden allgemeine Anforderungen erläutert, die sinnvoll für den Einsatz in Grid-Umgebungen sind. Danach wird auf die Sicherheit insbesondere in Bezug auf Authentisierung und Autorisierung eingegangen. Es werden notwendige Schritte erläutert, die sowohl den User und seine Daten schützen, aber auch den Lizenzgeber und seine Lizenzen sowie die korrekte, legale Benutzung der Lizenzen in Hinblick auf den Lizenzvertrag. Anschließend werden die Anforderungen an die Lizenzierung herausgearbeitet und erklärt. Dabei wird auf die Technik und die Vielfältigkeit eingegangen. Selbstverständlich wird auch an den Job beziehungsweise die Seite der Ressourcen gewisse Anforderungen gestellt, die im danach folgenden Abschnitt erklärt werden. Unter Berücksichtigung der bis dahin erläuterten Anforderungen wird abschließend aufgezeigt, was bei Accounting und Billing berücksichtigt werden muss und worauf im rechtlichen Sinne geachtet werden muss.

3.3 Allgemeine Anforderungen

Dieser Abschnitt befasst sich mit allgemeinen Anforderungen an ein Lizenzmodell. Hier werden sowohl strukturelle als auch mehr oder weniger selbstverständliche Anforderungen aufgezählt.

3.3.1 Allgemein: Unabhängigkeit von der eingesetzten Middleware

Das Lizenzmodell soll unabhängig von der eingesetzten Middleware sein und auf die konzeptionellen Grundlagen der verschiedenen Middlewares wie Globus Toolkit, gLite oder ähnliche aufbauen. Damit soll gewährleistet werden, dass auch zukünftige Versionen der Middlewares oder aber auch ganz neue Middleware-Entwicklungen das Lizenzkonzept aufgreifen und einsetzen können.

3.3.2 Allgemein: Nutzung Grid-spezifischer Technologien

Ein Grid basiert auf Web Services und WS-Resources. Diese mächtigen Instrumente sollen für das Lizenzmodell eingesetzt werden.

3.3.3 Allgemein: Priorisierung von Jobs

Da mehrere User Lizenzen benötigen können oder ein User mehrere Jobs startet, die Lizenzen benötigen, muss es eine Priorisierungsmöglichkeit der einzelnen Jobs geben, damit ein Job, der zum Beispiel 20 Lizenzen benötigt, keinen anderen Job, der nur 10 Lizenzen benötigt blockiert, falls nur 10 Lizenzen frei sind. Ein umfangreicher Schedulingmechanismus kann hier helfen. Dieser Scheduler muss jedoch strenge Vorgaben berücksichtigen, damit ein gegenseitiges Pushen der Priorisierungen nicht möglich ist.

3.3.4 Allgemein: Transparenz der Lizenzzuteilung

Die Lizenzzuteilung beziehungsweise Lizenzreservierung sollte transparent innerhalb des Grid Job Managements erfolgen können. Der Benutzer sollte so wenig wie möglich involviert werden, da der technische Hintergrund für die meisten Anwender uninteressant ist, die einfach nur ihren Job auf dem Grid laufen lassen wollen.

3.4 Anforderungen an die Sicherheit

Jedes Lizenzschema beruht auf Informationen, die sicher von und zu einer eindeutig identifizierbaren Person beziehungsweise Entität übermittelt werden müssen. Außerdem muss garantiert werden, dass diese Informationen weder abgeändert werden noch mehrmals gesendet werden können. Außerdem sollte die Einsicht durch Dritte unterbunden werden.

3.4.1 Sicherheit: Firewallkonfiguration

Das Lizenzmodell darf nicht verlangen, dass Firewalls großflächig geöffnet werden. Die Kommunikation zur Lizenzvalidierung muss auf Standardports stattfinden. Außerdem muss sichergestellt sein, dass ein Lizenzserver keine Verbindungen von vielen verschiedenen Instanzen entgegen nehmen muss, sondern nur von genau den Domänen, die er betreut.

3.4.2 Sicherheit: Zuverlässigkeit in unzuverlässigen Netzwerken

Die Lizenzzuteilung und damit die Ausführung des damit verbundenen Jobs darf nicht durch ein unzuverlässiges Netz beeinträchtigt werden. Das bedeutet, dass man von Nachrichtenverlusten ausgehen muss. Es kann also sowohl die Anfrage nach einer Lizenz, als auch die Lizenz selbst verloren gehen. Diese Tatsache darf insbesondere im Sinne der Abrechnung nicht vernachlässigt werden.

3.4.3 Sicherheit: Verzahnung mit den Grid-spezifischen Sicherheitsmechanismen zur Authentisierung

In jedem Grid gibt es bereits gewisse Sicherheitsmechanismen, um Nutzer als auch Ressourcen zu identifizieren, zu authentisieren und zu autorisieren. Diese Möglichkeiten sollen transparent zur Lizenzierung angewendet werden.

3.4.4 Sicherheit: Multiple Nutzung von Lizenzen unterbinden

Lizenzen sollten zur eindeutigen Identifikation mit einem eindeutigen Zähler versehen werden, so dass sie nicht mehrmals benutzt werden können.

3.5 Anforderungen an den Lizenzierungsvorgang

Im Folgenden werden Anforderungen an den Lizenzierungsvorgang dargelegt.

3.5.1 Lizenzierung: Lizenzpools nach administrativen Domänen getrennt

Eine Virtuelle Organisation, der mehrere Firmen, sprich administrative Domänen, angehören, muss die Möglichkeit haben, Lizenzen den jeweiligen administrativen Domänen zuordnen zu können. Sollte also nur ein gemeinsamer Lizenzserver eingesetzt werden, so muss dieser die Lizenzen der jeweiligen Domänen getrennt verwalten.

3.5.2 Lizenzierung: Einsatz verschiedener Lizenztypen

Da es sehr viele unterschiedliche Lizenztypen gibt, muss der Lizenzserver die Möglichkeit haben, die verbreitetsten Typen zu verwalten und Lizenzen auf dem jeweiligen Typ basierend zuzuteilen. Die am häufigsten eingesetzten Typen sind User-Locked-Lizenzen, Floating-Lizenzen sowie Pay-per-Use-Lizenzen.

3.5.3 Lizenzierung: Auswahl der Lizenzen möglich

Sollten mehrere passende Lizenzen vorhanden sein, so muss es sowohl eine automatische Auswahl der am besten passenden Lizenz aber auch eine manuelle Auswahlmöglichkeit geben. Auswahlberechtigt sind zum Beispiel bei einem Job normale Floating-Lizenzen, wenn genügend zur Verfügung stehen, aber auch möglicherweise Pay-per-Use-Lizenzen. So kann je nach Job die eine oder die andere Lizenzart besser und wirtschaftlicher sein. Aus einem anderen Betrachtungswinkel kann sowohl die Virtuelle Organisation selbst Lizenzen besitzen als auch eine Organisation der Virtuellen Organisation. In diesem Fall muss abgewogen werden, welche dieser Lizenzpools für einen Job, der von dieser Organisation aus gestartet wird, benutzt wird.

3.5.4 Lizenzierung: Hierarchische Vergabe von Lizenznutzungsrechten

Eine hierarchische Vergabe von Lizenznutzungsrechten von einem User zu einem anderen muss möglich sein. Dies ist notwendig, damit in der hochdynamischen Umgebung der Virtuellen Organisation eine hierarchische Autorisierung durchgeführt werden kann. Damit kann bestimmten Nutzern das Recht eingeräumt werden, andere User für die Lizenznnutzung zu autorisieren.

3.5.5 Lizenzierung: Blacklists und Whitelists

Der Einsatz von Blacklists und Whitelists erlaubt eine genauere Definition der Autorisierung. Die Listen bieten sich auf verschiedenen Niveaus an. Im Folgenden einige Beispiele.

- **User**
  Sollte eine der VO angehörige Firma prinzipiell keine Lizenznutzungsrechte haben, so kann einem einzelnen Nutzer dieser Firma dennoch Lizenznutzungsrechte als Gast der Firma, die Lizenzen besitzt, eingeräumt werden.

- **Firmen / Organisationen**
  Im hochdynamischen Umfeld einer VO kann es sein, dass eine Gruppe von Firmen beziehungsweise Organisationen einer VO beitritt. Hiermit ist es möglich, speziellen Teilgruppen (sprich Einzelfirmen) Lizenznutzungsrechte einzuräumen, auch wenn die Gruppe an sich keine Lizenznutzungsrechte besitzt.

Zusätzlich zu den Whitelists, bieten sich folgende Blacklists an:

- **User**
  Einzelne User können hiermit von jeglichen Lizenznutzungsrechten ausgeschlossen werden.

- **Nodes**
  Computing Resources, die der Virtuellen Organisation angehören, sind prinzipiell berechtigt, Jobs auszuführen. Jedoch kann es erwünscht sein, dass einige spezielle Resources nicht zur Ausführung lizenzierter Anwendungen herangezogen werden dürfen.

- **Firmen / Organisationen**
  Eine Teilgruppe von Firmen beziehungsweise Organisationen die einer lizenznutzungsberechtigten Virtuellen Organisation beitritt, kann von der Lizenznutzung ausgeschlossen werden.

3.5.6 Lizenzierung: Lizenzreservierung mit Frist

Lizenzen sollten für einzelne Projekte und Batches reserviert werden können. Die Reservierungsdauer sollte dabei beschränkt werden, damit man nicht für „irgendwann“ Lizenzen blockieren kann. Dabei ist zu beachten, dass durch diese Reservierungen keine Deadlocks entstehen, wenn sich einzelne Batches gegenseitig blockieren, da sie nicht genug Lizenzen erhalten und gleichzeitig aber auch keine bereits zugeteilten Lizenzen freigeben.

3.5.7 Lizenzierung: Pausieren und Fortsetzen der Lizenznutzung

Jobs innerhalb Workflows können unter Umständen gezwungen werden, dass sie für eine bestimmte Zeit pausieren, da gewisse notwendige Ressourcen nicht zur Verfügung stehen. Eine Pay-per-Use-Lizenz sollte für diese Zeitspanne freigegeben, und erst zum Fortsetzungszeitpunkt des Jobs wieder angefordert werden können.

3.5.8 Lizenzierung: Umfassendes Logging der angeforderten, zugeteilten und reservierten Lizenzen

Serverseitig sollte ein umfassendes Logging über die angeforderten, gerade zugeteilten und für die Zukunft reservierten Lizenzen stattfinden. Dabei ist zu beachten, dass diese Datensammlung sich auf das notwendigste beschränkt und nur zu einer technischen Analyse herangezogen werden kann und nicht gegen Datenschutzbestimmungen oder sonstiges verstößt. Überlegungen über die Logginginhalte soll folgende Liste geben:

- **Wer?**
  Mittels zum Beispiel einem X.509-Zertifikat kann sich der Lizenzanforderer, -reservierer authentisieren. Außerdem sollte der anfordernde Node im Sinne einer IP beziehungsweise seinem FQDN (Fully Qualified Domain Name) geloggt werden.

- **Wann?**
  Datum und Uhrzeit sind wichtig bei Reservierungen und unumgänglich bei Pay-per-Usage-Lizenzen. Außerdem lassen sich damit potentielle Lizenzleichen auffinden.

- **Wo?**
  Logging der ausführenden Computing Resources mittels der IP oder dem zugehörigen FQDN (Fully Qualified Domain Name) sind ebenfalls eine interessante Informationsquelle.

- **Art der Lizenz**
  Informationen über den Typ der Lizenzen sind wichtig um den Lizenzpool zu optimieren.

- **Anzahl der Lizenzen**
  Die Anzahl der Lizenzen zeigt statistisch auf, wo und von wem die meisten Lizenzen verwendet werden.

3.5.9 Lizenzierung: Änderungen am Lizenzpool und den Listen ohne Neustart

Der Lizenzserver sollte so konzipiert sein, dass er nicht neu gestartet werden muss, wenn Änderungen am Lizenzpool oder den Black- und Whitelists vorgenommen werden. In einem hochdynamischen System wie einer Virtuellen Organisation und einem Grid, auf dem sehr viele Jobs laufen, die teilweise sehr lang und teilweise sehr kurz sind, würde ein temporärer Lizenzserverausfall zu nicht unerheblichen Erschwernissen führen und starke Delays nach sich ziehen können.

3.5.10 Lizenzierung: Lizenzserver unabhängig der eingesetzten Plattform

Der Lizenzserver sollte unter allen gängigen Systemplattformen laufen, insbesondere Windows und Linux. Dazu bietet sich hervorragend die Entwicklung des Lizenzservers auf Basis von Java an.

3.6 Anforderungen an die Lizenzen

In diesem Abschnitt werden die Anforderungen an die Lizenzen dargestellt. Dabei wird insbesondere berücksichtigt, dass die Lizenzen in der Regel innerhalb einer Virtuellen Organisation eingesetzt werden.

3.6.1 Lizenz: Lizenz als feste Dateistruktur

Eine Lizenz muss eine feste Struktur haben. Die darin enthaltenen Informationen müssen alle notwendigen Daten enthalten, damit die Lizenz eindeutig zuweisbar und identifizierbar ist.

3.6.2 Lizenz: Mobilität der Lizenzen

Eine Lizenz muss mobil sein. Das bedeutet, wenn ein Lizenzvertrag die Nutzung der Lizenz innerhalb einer Grid-Umgebung beziehungsweise im eigenen Nutzen auf fremden Ressourcen untersagt, so darf diese Lizenz rechtlich nicht benutzt werden. Wenn diese rechtliche Einschränkung nicht gilt, dann muss die Lizenz benutzbar sein. Zusätzlich muss überhaupt die technische Möglichkeit gegeben werden, dass Lizenzen mobil sind, da dies nicht grundlegend der Fall ist.

3.7 Anforderungen an die Durchführung des Jobs

Dieser Abschnitt befasst sich mit den Anforderungen, die gestellt werden, während der Job bearbeitet wird.

3.7.1 Job: Lizenzen verlängern oder früher aufgeben

Falls ein Job mit einer Lizenz, die nur für eine gewisse Zeitspanne gültig ist, gestartet wird, muss der Job abgebrochen werden können oder es muss die Möglichkeit gegeben werden, dass die Lizenz verlängert wird. Andererseits sollte natürlich eine Lizenz nicht länger als notwendig gehalten und dafür bezahlt werden müssen.

3.7.2 Job: Jobs beenden, die innerhalb Timeouts keine Lizenzen erhalten haben

Jobs, die keine Lizenzen erhalten haben, dürfen nicht gestartet werden. Sollte ein Job aus einem Workflow bestehen, der erst ab einem bestimmten Punkt nicht weiter ausgeführt werden kann, so muss der Job an dieser Stelle abgebrochen werden.

3.7.3 Job: Bekanntgabe, woher genutzte Lizenzen kommen

Die gerade laufenden Jobs müssen überwacht werden, damit bekanntgegeben werden kann, welche Jobs welche Lizenzen benutzen und von welchem Benutzer die Jobs gestartet wurden.

3.7.4 Job: Überwachung der Lizenzen

Es muss jederzeit erkennbar sein, woher die Lizenzen sind, die gerade in Benutzung sind. Außerdem muss es die Möglichkeit geben, die Lizenzen zurückzufordern und folglich den Job abzubrechen.

3.7.5 Job: Überprüfen der Legalität der Lizenzen

Bevor der Job tatsächlich gestartet wird, muss die Legalität der zum Job gehörenden Lizenzen überprüft werden. Dieses Validieren muss sowohl die Korrektheit der Lizenzen in Hinblick auf Fälschung berücksichtigen als auch sicherstellen, dass die Daten, die mit dem Job übermittelt wurden, mit dieser Lizenz korrekterweise gebundlet wurden. Damit ist sichergestellt, dass eine Lizenz nur für das genutzt werden kann, für das sie beantragt wurde.

3.8 Anforderungen an das Accounting und Billing

Da Lizenzen immer auf einem gewissen Nutzungsrecht basieren, müssen selbstverständlich gewisse Anforderung an das Accounting und Billing gestellt werden. Auch in Hinblick auf die dynamische Erweiterung des Lizenzpools im Sinne von Pay-per-Use-Lizenzen ist ein korrektes Abrechnungsmanagement unverzichtbar.

3.8.1 Accounting: Anzeige der erhaltenen Lizenzen

Nach einer erfolgreichen Lizenzzuteilung sollte dem Anfragesteller die Möglichkeit gegeben werden, die zugeteilte(n) Lizenz(en) jederzeit einzusehen. Auch sollte er bei einer erfolgreichen Lizenz-Reservierung jederzeit die Möglichkeit haben, die Reservierung einzusehen, zu ändern oder zu stornieren.

3.8.2 Accounting: Sicheres Accounting und Abrechnungsverfahren

Das Accounting und das Abrechnungsverfahren muss sicher sein. Das bedeutet, dass die gesamte Kommunikation sowie der Datenbestand durch State-of-the-Art-Mechanismen geschützt werden müssen.

3.8.3 Accounting: Schutz vor Leugnung des Erhalts einer Lizenz

Es muss sichergestellt sein, dass ein Benutzer den Erhalt einer Lizenz nicht leugnen kann. Ein Benutzer muss also den korrekten Erhalt einer angeforderten Lizenz bestätigen, bevor er sie tatsächlich nutzen kann.

3.8.4 Accounting: Redundantes Abspeichern aller relevanten Informationen

Zusätzlich zum sicheren Abspeichern gehört natürlich auch das redundante Abspeichern aller Informationen, die für das Abrechnungsmanagement relevant sind gemäß den üblichen Konventionen.

3.8.5 Accounting: Statistische Auswertungen ermöglichen

Im Sinne des Abrechnungsmanagements sollte es möglich sein, dass statistische Auswertungen der Informationen bezüglich der Abrechnungsdaten angefertigt werden können. Darunter fällt sowohl die Aufbereitung der konkreten Pay-per-Use-Daten als auch die Auswertung der Abrechnungsmodalitäten und der Zahlungswillen der jeweiligen Instanzen.

3.8.6 Accounting: Abrechnung quasi Echtzeit durchführen

Die Abrechnungen sollten möglichst schnell durchgeführt werden können. Dies ist inbesondere wichtig beim Einsatz von Prepaid-Pay-per-Use-Lizenzen, da in Grids starke Peaks auftreten können. Ein Peak könnte somit bereits das Kontingent überschreiten, so dass die Nutzung möglichst kurz danach eingeschränkt werden müsste.

3.8.7 Accounting: Delay der Lizenzierung berücksichtigen

Ein Job benötigt möglicherweise extrem viele Lizenzen. Die Beantragung, Übermittlung, Prüfung und das Starten des Jobs dauert eine gewisse Zeit. Das Billing sollte diese Delays berücksichtigen.

3.8.8 Accounting: Accounting mit verschiedenen Granularitätsstufen

Damit statistische Auswertungen bezüglich dem Accounting möglich sind, muss das Accounting auf verschiedenen Granularitätsstufen stattfinden. Als mögliche Granularitäten bieten sich an: User, Computing Resource, Projekt, Organisation oder gesamte Virtuelle Organisation.

3.8.9 Accounting: Kostenexplosionen vermeiden

Es sollte die Möglichkeit geboten werden, Limits zu setzen. Dies verhindert effektiv die Gefahr einer Kostenexplosion.

3.8.10 Accounting: Einsicht jederzeit möglich

Dem Lizenznehmer muss jederzeit Einsicht in die Abrechnungsinformationen ermöglicht werden, damit er quasi live seine Kosten und Nutzungsstatistiken verfolgen kann.
Selbiges gilt auch für den ISV beziehungsweise den Distributor. Dieser sollte ebenfalls jederzeit Einsicht in die Abrechnungsinformationen haben, um die Abrechnung korrekt durchführen zu können beziehungsweise auf Ungereimtheiten oder Seltenheiten aufmerksam zu werden und gegebenenfalls handeln zu können.

3.9 Rechtliche Anforderungen

Selbst wenn die technologische Hürde genommen wurde, um lizenzierte Software in Grid-Umgebungen einzusetzen, so müssen noch immer einige rechtliche Aspekte berücksichtigt werden. Diese Aspekte betreffen sowohl Vorgaben und Verpflichtungen der Organisation als auch des ISVs beziehungsweise Distributors.

3.9.1 Recht: Einbeziehung der Firmen-Policies

In jeder Firma und Organisation gibt es andere Policies bezüglich Sicherheit, Nutzerverhalten, Rechnernutzung, Softwarenutzung und so weiter. Diese Policies schränken eventuell auch die Lizenznutzung von einigen Mitarbeitern ein. Diese Policies müssen berücksichtigt werden.

3.9.2 Recht: Lizenzvertrag berücksichtigen

Ein Lizenzvertrag schränkt in der Regel wie in Kapitel 2 beschrieben die Nutzung der Lizenzen und der dazugehörigen Software ein. Außerdem sind an ihn gewisse Verpflichtungen gebunden. Eine mögliche Einschränkung dieses Vertrags könnte besagen, dass Lizenzen nur innerhalb des Hoheitsgebiets der jeweiligen Organisation benutzt werden dürfen. Damit würde auch der Einsatz von mobilen Lizenzen wegfallen. Problematisch ist dieser Vertrag jedoch, wenn sich das benutzte Grid über mehrere Organisationen erstreckt und somit die lizenzierte Anwendung „irgendwo“ laufen könnte, was im Sinne des Vertrages verboten ist.

3.9.3 Recht: Erweiterung des Lizenzvertrags bezüglich Nutzung der Lizenzen in Grids

Der Lizenzvertrag muss dahingehend erweitert werden, dass die Nutzung der Lizenzen in Grid-Umgebungen gestattet wird. Diesbezügliche rechtliche Bestimmungen und Einschränkungen müssen eventuell angepasst und abgeändert werden. Sollte ein Lizenzvertrag die Nutzung der Software in Grids grundsätzlich untersagen, so kann es keine legale Lösung für die Lizenzproblematik in Grids geben.

3.9.4 Recht: Rechtliche Konsequenzen bei Ausfall oder Nichtverfügbarkeit des Lizenzservers

Ein Lizenzvertrag musst festlegen, welche rechtlichen Konsequenzen die Nichtverfügbarkeit oder ein kompletter Ausfall der einzelnen Komponenten eines Lizenzmodells hervorruft.

3.9.5 Recht: Überprüfung der korrekten Lizenznutzung

Lizenzen können, wie bereits beschrieben, für eine bestimmte Aufgabe beschränkt sein. So gibt es zum Beispiel Lizenzen, die nur zu Forschungszwecken oder universitäre Zwecke eingesetzt werden dürfen, jedoch nicht für kommerzielle Rechenaufgaben. Deshalb sollte eine Prüfung stattfinden, die sich an den gesetzlichen Datenschutz hält, für was eine Lizenz beantragt wird und zu welchem Zweck diese Lizenz später genutzt wird.

3.10 Tabellarische Übersicht

In diesem Abschnitt werden die Anforderungen nochmals tabellarisch dargestellt. Die erste Spalte besagt, in welchem Abschnitt die jeweilige Anforderung nachzulesen ist. Außerdem wird direkt auf die Seite verwiesen.

Abs.	Kategorie	Titel	Seite
3.3.1	Allgemein	Unabhängigkeit von der eingesetzten Middleware	48
3.3.2	Allgemein	Nutzung Grid-spezifischer Technologien	48
3.3.3	Allgemein	Priorisierung von Jobs	48
3.3.4	Allgemein	Transparenz der Lizenzzuteilung	48
3.4.1	Sicherheit	Firewallkonfiguration	48
3.4.2	Sicherheit	Zuverlässigkeit in unzuverlässigen Netzwerken	49
3.4.3	Sicherheit	Verzahnung mit den Grid-spezifischen Sicherheitsmechanismen zur Authentisierung	49
3.4.4	Sicherheit	Multiple Nutzung von Lizenzen unterbinden	49
3.5.1	Lizenzierung	Lizenzpools nach administrativen Domänen getrennt	49
3.5.2	Lizenzierung	Einsatz verschiedener Lizenztypen	49
3.5.3	Lizenzierung	Auswahl der Lizenzen möglich	49
3.5.4	Lizenzierung	Hierarchische Vergabe von Lizenznutzungsrechten	50
3.5.5	Lizenzierung	Blacklists und Whitelists	50
3.5.6	Lizenzierung	Lizenzreservierung mit Frist	50
3.5.7	Lizenzierung	Pausieren und Fortsetzen der Lizenznutzung	51
3.5.8	Lizenzierung	Umfassendes Logging der angeforderten, zugeteilten und reservierten Lizenzen	51
3.5.9	Lizenzierung	Änderungen am Lizenzpool und den Listen ohne Neustart	51
3.5.10	Lizenzierung	Lizenzserver unabhängig der eingesetzten Plattform	52
3.6.1	Lizenz	Lizenz als feste Dateistruktur	52
3.6.2	Lizenz	Mobilität der Lizenzen	52

Abs.	Kategorie	Titel	Seite
3.7.1	Job	Lizenzen verlängern oder früher aufgeben	52
3.7.2	Job	Jobs beenden, die innerhalb Timeouts keine Lizenzen erhalten haben	52
3.7.3	Job	Bekanntgabe, woher genutzte Lizenzen kommen	53
3.7.4	Job	Überwachung der Lizenzen	53
3.7.5	Job	Überprüfen der Legalität der Lizenzen	53
3.8.1	Accounting	Anzeige der erhaltenen Lizenzen	53
3.8.2	Accounting	Sicheres Accounting und Abrechnungsverfahren	53
3.8.3	Accounting	Schutz vor Leugnung des Erhalts einer Lizenz	53
3.8.4	Accounting	Redundantes Abspeichern aller relevanten Informationen	54
3.8.5	Accounting	Statistische Auswertungen ermöglichen	54
3.8.6	Accounting	Abrechnung quasi Echtzeit durchführen	54
3.8.7	Accounting	Delay der Lizenzierung berücksichtigen	54
3.8.8	Accounting	Accounting mit verschiedenen Granularitätsstufen	54
3.8.9	Accounting	Kostenexplosionen vermeiden	54
3.8.10	Accounting	Einsicht jederzeit möglich	54
3.9.1	Recht	Einbeziehung der Firmen-Policies	55
3.9.2	Recht	Lizenzvertrag berücksichtigen	55
3.9.3	Recht	Erweiterung des Lizenzvertrags bezüglich Nutzung der Lizenzen in Grids	55
3.9.4	Recht	Rechtliche Konsequenzen bei Ausfall oder Nichtverfügbarkeit des Lizenzservers	55
3.9.5	Recht	Überprüfung der korrekten Lizenznutzung	55

3.11 Zusammenfassung des 3. Kapitels

Im ersten Teil des 3. Kapitels wurden einige Fallbeispiele beschrieben, um auf die Fragestellungen, die bei Lizenzierung in Grid-Umgebungen auftreten, hinzuführen. Dabei wurde zuerst ein Szenario dargestellt, für das es keine Lösung gibt. Anschließend wurde ein Beispiel erläutert, das in Unternehmen derzeit großen Anklang findet. Das dritte Szenario beschrieb ein Unternehmen, das selbst keine Hochleistungs-Computer besitzt und auf fremde, externe Computing Resources zurückgreifen muss. Im letzten Szenario wurde dann eine Virtuelle Organisation und das dazugehörige Grid erläutert und worin dort die Problematik von Software-Lizenzierungen besteht.
Der im zweiten Abschnitt entwickelte Anforderungskatalog basiert auf den Fragestellungen der Fallbeispiele. Dabei wurden die Anforderungen strukturiert und beschrieben. Es wurden sowohl prinzipielle Anforderungen als auch logische sowie technische Anforderungen aufgezählt und deren Einsatzzweck definiert. Dieser Anforderungskatalog dient in den weiteren Kapiteln als Grundlage zur Evaluierung, Konzeption und Entwicklung.

4 Aktueller Stand des Lizenzmanagements in Grids

Im vorherigen Kapitel wurde anhand einigen Use-Cases allgemeine Anforderungen an ein Lizenzmodell entwickelt. Selbstverständlich gibt es neben dieser Diplomarbeit noch weitere Ansätze, um Lizenzierung in Grid-Umgebungen zu ermöglichen. Dieses Kapitel befasst sich mit der Untersuchung und Erläuterung eben dieser Ansätze.
Zum Zeitpunkt der Entstehung dieser Diplomarbeit waren unter anderem die Ansätze von GenLM, BEinGRID und SmartLM relativ weit ausgearbeitet. Diese drei Ansätze werden in diesem Kapitel näher betrachtet und erläutert. Dabei stellt sich heraus, dass GenLM auf grundlegende Eigenschaften von BEinGRID aufsetzt, jedoch einen logisch anderen Ansatz spezifiziert. BEinGRID möchte klassische Lizenzmodelle aus non-Grid-Umgebungen in Grid-Umgebungen portieren. SmartLM definiert ein neues Geschäftsmodell und spezifiziert viele Dinge neu.

4.1 GenLM

GenLM ist ein Lizenzmodell, das großteils am Fraunhofer-Institut in Kaiserslautern entwickelt wurde. Das Schema setzt auf einen relativ einfachen Client-Server-Ansatz mit gegenseitigen Signieren von Hashwerten. Als besonders wichtig wird von GenLM die legale Durchführung einer Lizenzzuteilung und Sicherung dieser Durchführung hervorgehoben.

4.1.1 Theoretische Beschreibung des Lizenzmanagementansatzes

Aus Sicht von GenLM gibt es nach [DP09] drei Beteiligte bei einem Lizenzierungsmechanismus innerhalb einer Grid-Umgebung. Die User, die Jobs auf dem Grid ausführen wollen; die Resource Provider, die Ressourcen dem Grid beziehungsweise der VO zur Verfügung stellen; und die ISVs, die ihre Software zur Verfügung stellen und lizenzieren.
So sieht GenLM als eine der wichtigsten Prioritäten, dass diese drei beteiligten Gruppen zufrieden gestellt werden. So sollen selbstverständlich die User im Prinzip nichts von der notwendigen Softwarelizenzierung mitbekommen, sondern ihre Jobs „wie gewohnt“ auf dem Grid laufen lassen können. Die Resource Provider wollen vor allem ein umfangreiches Softwarepaket anbieten, auf das die User zugreifen können. Somit ist aus Provider-Sicht notwendig, dass möglichst viele verschiedene Softwares korrekt und legal für eine Verwendung in Grid-Umgebungen lizenziert werden können. Die ISVs hingegen wollen auf jeden Fall, dass ihre Lizenzen auf legalem Weg genutzt werden und korrekt abgerechnet werden können. Dies ist wohl der wichtigste Punkt, da ohne korrektem Abrechnungs- und Lizenzierungsverfahren die Lizenzierung an sich sinnlos wird. Daher ist die Sicherstellung der korrekten Lizenzierung aus Sicht des ISVs wohl einer der Punkte, der zum Überleben des ISVs beiträgt. Somit wollen natürlich alle ISVs ihre Software vor unautorisierter Nutzung schützen.

GenLM teilt nun seinen Ansatz auf diese drei Beteiligten auf.

- **GenLM-Client**
  Der GenLM-Client arbeitet auf der Seite der User und ist für die Anforderung von Lizenzen verantwortlich. Diese Lizenzen werden als sogenannte Lizenz-Tokens übermittelt.

- **GenLM-Server**
  Der GenLM-Server läuft auf Seite des ISV und teilt nach Anfrage Lizenzen zu, nachdem der Anfragesteller vorher authentisiert wurde und für die Lizenznutzung autorisiert ist.

- **GenLM-License-Verifier**
  Auf Seite des Resource Providers läuft der GenLM-License-Verifier als Teil der Software, die der ISV dem Resource Provider übergeben hat, damit dieser die Software auf seinen Resources von Usern ausführen lassen kann. Der License-Verifier ist dafür zuständig, um den vom User übermittelten License-Token zu verifizieren und zu überprüfen, ob die Lizenz für den angeforderten Job benutzt werden darf.

Im Prinzip setzt GenLM somit auf die Generierung des Request-Tokens anhand der Eingabedaten und Übermittlung dieses Tokens zusammen mit den Eingabedaten des Jobs. Die Informationen, die der Token beinhaltet, sollen dann dafür ausreichen, um die Legalität der Software Lizenz zu überprüfen und dem Job somit Freigabe zu erteilen.

4.1.2 Anforderungen auf die GenLM besonderen Wert legt

GenLM stellt ein Lizenzschema dar, das nicht direkt auf Virtuelle Organisationen und deren Besonderheiten eingeht. Stattdessen ist GenLM ein Ansatz, der im Prinzip auf allen institutionsübergreifenden gemeinschaftlichen Projekten mit verteilten und geteilten Ressourcen anwendbar sein soll.
GenLM hat daher auch nur einen relativ kleinen Satz an Anforderungen, die im Folgenden nach [DP09] kurz dargestellt werden.

- **Unterstützung von bereits existierenden Lizenzabkommen**
  Die Wichtigkeit, dass bereits existierende Lizenzverträge nicht „einfach so“ geändert werden können, darf nicht vernachlässigt werden. Daher will GenLM bereits existierende Lizenzverträge weiterhin unterstützen, so dass diese nicht angepasst werden müssen.

- **On-Demand Lizenzen**
  On-Demand-Lizenzen entsprechen den klassischen Pay-per-Use-Lizenzen im Postpaid-Verfahren. Diese könnten einen relativ hohen Anteil in Grid-Umgebungen ausmachen, da mit diesen Lizenzen Peaks aus wirtschaftlicher Sicht relativ günstig abzudecken sind. GenLM schlägt dazu weiterhin vor, die Möglichkeit zu bieten, auf ein „Basis-Paket“ bestehend aus klassischen Floating-Lizenzen, bei Bedarf temporär weitere Lizenzen erwerben zu können.

- **Mobilität der Lizenzen**
  Eines der zentralen Probleme der Software-Lizenzierung in Grid-Umgebungen beziehungsweise Cloud-Computing-Umgebungen ist die Tatsache, dass gewöhnliche Lizenzverträge die Nutzung der lizenzierten Software verbieten, wenn die Software in einer anderen als der lizenzierten Umgebung ausgeführt wird. Zum Beispiel ist eine Lizenz

nahezu immer auf eine bestimmte Firma ausgestellt, die dieser Firma erlaubt, die Software auf ihren Ressourcen zu nutzen. Eine Nutzung der Software *von* dieser Firma auf einer Grid-Ressource, die ganz *woanders* ist und jemand anderem gehört (also einer anderen administrativen Domäne unterliegt), ist jedoch untersagt.
GenLM fordert hierzu, dass die Nutzung der Software von der Lizenz nur daraufhin eingeschränkt wird, *wer* sie nutzt aber nicht darauf eingeschränkt ist, *wo* sie ausgeführt wird, solange der legale Benutzer die Rechte der Ausführung dort hat. GenLM spricht hierbei von einer Mobilität der Lizenzen.

- **Einfache Integration**
  Das Lizenzframework soll einfach in bereits bestehende Software Packages integriert werden können. Außerdem soll das gesamte Framework, auf allen bekannten Plattformen lauffähig sein.

- **Grid- und Cloud-Unterstützung**
  Wie bereits oben erwähnt will GenLM sowohl Grids als auch Clouds unterstützen und im grundlegenden Ansatz nicht auf Spezialfälle eingehen, sondern ein Lizenzmanagementsystem darstellen, das in verteilten Umgebungen einsetzbar ist.

- **Hohe Verfügbarkeit**
  Wohl eines der wichtigsten Features eines jeden Lizenzsystems ist die Verfügbarkeit. So müssen Lizenzanfragen in einer angemessenen Zeit durchführbar sein und die Lizenztokens rasch übermittelt werden können. Auch die Verifizierung der Tokens muss angemessen durchführbar sein.

- **Sicherheit**
  Das beste Lizenzschema nutzt nichts, wenn es jederzeit und einfach umgangen werden kann. Das bedeutet, jede Kommunikation muss sicher sein, so dass jeder Benutzer authentisiert wird und er nur bei erfolgreicher Authentisierung für die Lizenznutzung autorisiert wird. Auch müssen die Tokens verschlüsselt und signiert übermittelt und vor einer Zweitbenutzung geschützt werden. Die Lizenzüberprüfungen dürfen ebenfalls nicht ohne extremen Aufwand geknackt werden.

4.1.3 Praktischer Ablauf und technische Beschreibung

Im Folgenden wird der praktische Ablauf des Lizenmodells nach [DP09] dargestellt. Dieser Ablauf setzt voraus, dass eine Kommunikation mit dem Lizenzserver möglich ist, was nicht trivial zu lösen ist. GenLM verlässt sich dabei auf andere Ansätze und setzt die grundsätzliche mögliche Kommunikation als gegeben voraus. Ein möglicher Ansatz, die Kommunikation sicherzustellen, beschreibt der im nächsten Abschnitt vorgestellte Proxy-Ansatz von BEinGRID.
GenLM basiert auf dem Einsatz von Request-Tokens. Dieser Token wird vom GenLM-Client, der beim User läuft bei Bedarf generiert. Er beinhaltet ein HashSet, das mittels SHA-256 der Input Files sowie relevante Teile des Lizenzabkommens gebildet wird. Mittels dieser Hashfunktion werden die Eingabe-Dateien relativ kollisionsresistent identifizierbar gemacht. Außerdem wird mittels diesem Token Anforderungen an die Ressourcen übermittelt, wie zum Beispiel Anzahl der Kerne, die für den Job benutzt werden sollen, Menge des zu reservierenden Arbeitsspeichers und so weiter. Außerdem wird mitgeteilt, welche Lizenzpflichtige Software eingesetzt werden soll. Dies ist unter anderem dann notwendig, wenn der bearbeitende

Lizenzserver mehrere Softwareprodukte verwaltet. Dieser Request-Token wird anschließend mit dem X.509-Zertifikat des Benutzers signiert und an den GenLM-Server übermittelt. Der Lizenzserver teilt das Request-Token nun wieder in seine Bestandteile auf und übergibt diesen Inhalt anschließend einer überprüfenden Instanz, die mittels einem Policy Plugin realisiert wird. Dieses Plugin ist dafür zuständig, den User zu authentisieren und für seinen angefragten Job zu autorisieren oder abzulehnen. In welchem Umfang und wie genau diese Überprüfungen stattfinden, definiert GenLM nicht, sondern überlässt dies den jeweiligen ISVs. Grundlegend sollte jedoch der Funktionsumfang der Prüfung folgendes beinhalten:

- Gegenprüfung der Identität des Benutzers mit einer Kundendatenbank
  Jede Anfrage nach einer Lizenz muss zuerst authentisiert werden und nur bei erfolgreicher Authentisierung darf die anfragende Instanz eventuell autorisiert werden.

- Zuteilung eines passenden Lizenzpakets
  Je nach Job und Lizenzabkommen können verschiedene Lizenzen passend sein. So muss überprüft werden, ob der Job mit einem Floating-Lizenz-Paket abgedeckt werden kann beziehungsweise soll, oder ob ein Pay-per-User-Modell eingesetzt werden soll, oder die Lizenzen on-Demand zur Verfügung gestellt werden sollen. Welches Paket überhaupt in Frage kommt, bestimmt der zu Grunde liegende Lizenzvertrag.

- Abrechnungsdaten
  Alle Daten, die zur Abrechnung erforderlich sind, müssen gesammelt werden.

Der GenLM-Server hat nun zwei Möglichkeiten, sich zu entscheiden. Entweder er akzeptiert die Lizenzanfrage auf Grund der Prüfungen des Ergebnisses des Policy Plugins oder er lehnt die Anfrage ab. Eine Ablehnung ist von GenLM nicht definiert. Wie also damit umgegangen wird, ist nicht klar.
Eine positive Reaktion, also eine Zuteilung der angefragten Lizenzen erfolgt folgendermaßen: Der GenLM-Server signiert mit seinem X.509-Zertifikat das Request-Token, das er vom GenLM-Client, also indirekt vom User, erhalten hat und durch das Policy Plugin bearbeiten hat lassen zusammen mit einem validen, der Anfrage entsprechenden Lizenzpaket. Dieses Policy Plugin kann direkt in den GenLM-Server einkompiliert oder aber als Ruby-Modul eingebaut werden, das sogar On-the-Fly-Änderungen an der Policy zulässt. Das signierte Paket aus Request-Token und Lizenzpaket bezeichnet GenLM als License-Token, da hier nun eine valide Lizenz enthalten ist. Das License-Token wird nun dem GenLM-Client übermittelt.
Der User kann nun seinen Job an das Grid übermitteln. Die Job-Submission beinhaltet selbstverständlich die notwendigen Eingabe-Daten als auch das License-Token. Bevor der Job gestartet wird, wird der License-Token mittels dem öffentlichen Schlüssel des GenLM-Servers auf seine Gültigkeit überprüft. Anschließend werden die Hashes der übermittelten Eingabe-Daten berechnet und mit den Hashes aus dem License-Token verglichen. Sollte die Signaturüberprüfung erfolgreich sein und die Hashes übereinstimmen, kann die Berechnung des Jobs starten, da der User seine Berechtigung erfolgreich nachgewiesen hat. Sollte entweder die Signaturüberprüfung fehlschlagen oder die Hashes nicht identisch sein, so gibt GenLM keine Spezifikation bekannt, wie dem User dies mitgeteilt werden soll. GenLM geht also immer von einer positiven Überprüfung aus.

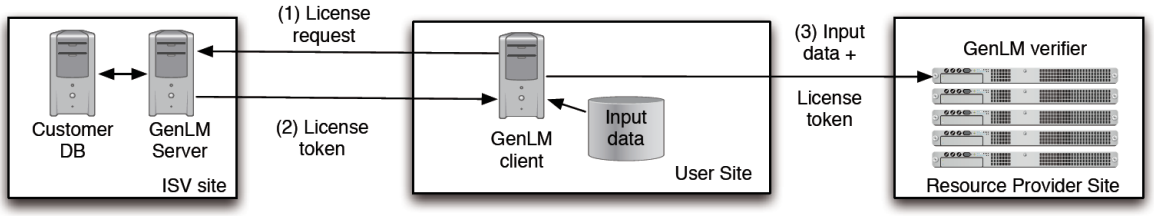

Abbildung 4.1: GenLM Komponenten nach [DP09]

Der beschriebene Ablauf setzt jedoch voraus, dass keine Nachrichten, sprich License-Tokens, verloren gehen. Bei einem Verlust eines License-Tokens wird die Lizenzanfrage berechnet, aber der User hat nie eine Lizenz erhalten. Das stellt einen finanziellen Verlust der Client-Seite dar. Andererseits kann ein User behaupten, er hätte das License-Token nie erhalten, obwohl er es sehr wohl erhalten und damit sogar seinen Job durchgeführt hat. Dies stellt indirekt einen wirtschaftlichen Verlust auf ISV-Seite dar. Daher wurde das GenLM-Modell erweitert, um diesen finanziellen und auch rechtlichen Problemen zu entgehen. Der Nachrichtenaustausch zwischen GenLM-Client und GenLM-Server wurde um weitere Nachrichten ergänzt.
Die Kommunikation zwischen GenLM-Client und GenLM-Server wurde um die Prinzipien des Zwei-Phasen-Übermittlungs-Protokolls erweitert. Der GenLM-Server übermittelt nun nicht mehr direkt das License-Token an den GenLM-Client, sondern eine verschlüsselte Version des License Tokens zusammen mit einem eindeutigen key identifier. Der Client übermittelt nun den key identifier wieder zurück an den Server. Dieser antwortet daraufhin mit dem Schlüssel zum Entschlüsseln des License Tokens. Erst jetzt beginnt auf Server-Seite der Abrechnungsprozess der Lizenzen, da der Server nun sicher sein kann, dass der Client zumindest den verschlüsselten License Token erhalten hat. Der Entschlüsselungs-Key kann jedoch immer noch verloren gehen, so dass der Client das License Token nicht entschlüsseln und verwenden kann. In diesem speziellen Fall muss sich der User an den Support des ISVs wenden, um den Entschlüsselungs-Key unabhängig von der normalen Kommunikation zu erhalten. Das Verfahren schließt jedoch aus, dass der User behauptet, dass er kein License-Token erhalten hat und stellt somit eine Verbesserung aus Sicht des ISVs dar, da Lizenzen nicht mehrmals illegalerweise genutzt werden können.

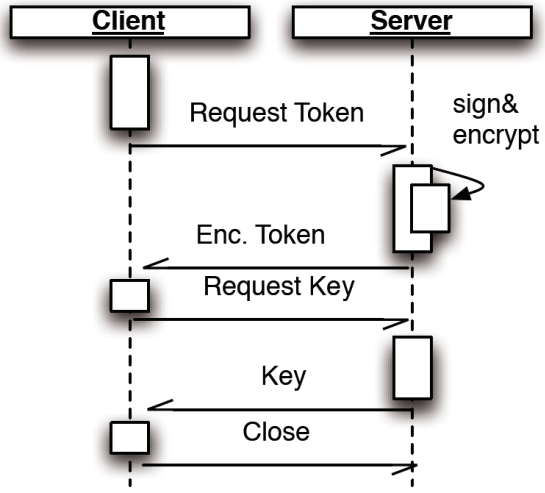

Abbildung 4.2: GenLM Message Exchange nach [DP09]

4.1.4 Skalierung und praktische Erfahrung

GenLM ist im aktuellen Stand unter C++ für Linux implementiert. Eine Windowsportierung soll jedoch auch folgen sowie eine Cross-Plattform-Nutzung von Clients und Server, die auf unterschiedlichen Plattformen laufen. Zur besseren Parallelisierung der Anfragen am GenLM-Server wurde dieser unter der Staged Event-Driven-Architecture (SEDA) nach Welsh aufgebaut. SEDA-Programme haben mehrere unabhängige Stationen definiert, die gepipelined ablaufen können. Ein SEDA-Programm ist nach [DP09] aus mehreren Stages bestehend. Jede dieser Stages besitzt eine Input Queue und einen Thread-Pool. Die Threads holen sich von der Input Queue ein Element und bearbeiten es anhand der Stage Strategy. Der gesamte Ablauf von GenLM basiert nun auf diesen einzelnen Stages.

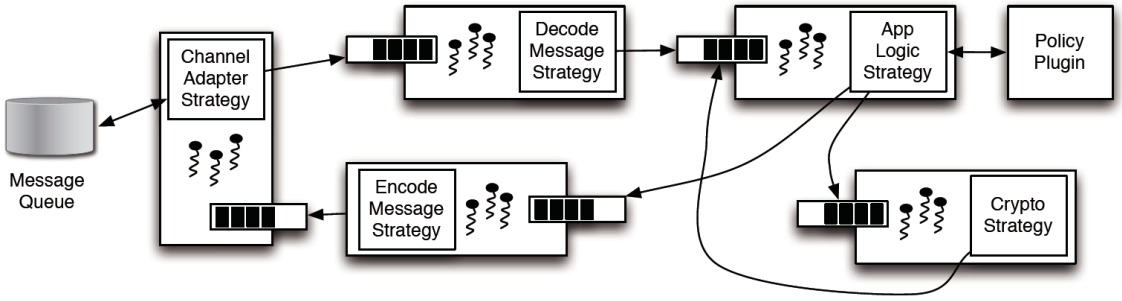

Abbildung 4.3: GenLM Seda Stages nach [DP09]

Auf Grund diesen Aufbaus, sind die GenLM-Server unabhängig voneinander. Somit treten GenLM-Server bei Bedarf einfach als Server-Pool auf, der jederzeit erweitert werden kann. Da GenLM einer Message Queue von Apache ActiveMQ ([Apa10a]) unterliegt, müssen Ser-

ver, die etwas bearbeiten wollen/können nach neuen Messages, also Request-Tokens explizit bei der Message Queue anfragen. Da nur freie Server nach unbearbeiteten Request-Tokens anfragen, skaliert die GenLM-Server-Architektur nahezu beliebig nach oben. Somit besteht auf Seite des GenLM-Servers keine Problematik bezüglich zu langsamen Abarbeitens der Lizenz-Anfragen.
Clientseitig, also den User betreffend, müssen, wie oben beschrieben, die Input-Dateien alle gehasht werden. Die verwendete SHA 256 Hashfunktion ist laut GenLM deutlich langsamer als MD5 oder SHA1, von denen GenLM aber explizit Abstand nimmt. Da die Hashes auf Clientseite „nur“ die lokale User-Maschine betreffen ist die Geschwindigkeit direkt abhängig von der CPU-Leistung der Client-Maschine. Erst wenn der lokale Arbeitsspeicher nicht ausreicht um Hashes von sehr großen Input-Dateien komplett zu beinhalten, ist der Sekundärspeicher für die Bearbeitungsgeschwindigkeit und Generierungsdauer für das Request-Token entscheidend.
Problematisch scheint hingegen die Bildung der Hashes auf Seite der Computing Resources zu sein, die durchgeführt wird, bevor der Job gestartet wird, um zu überprüfen, ob die zur Lizenzanforderung verwendeten Eingabedaten mit den übermittelten Daten übereinstimmen. Die zur Berechnung der Jobs bereitgestellten Resources werden hierbei bei jedem Job erst verwendet, um die Legalität des Jobs zu überprüfen. GenLM gibt keine Auskunft, inwieweit diese Überprüfung auf die allgemeine Performance des Grids Einfluss nimmt.

4.1.5 Sicherheitsbetrachtungen von GenLM

Ein Lizenzmanagementsystem muss vor allem drei Dinge bieten:

1. Der User darf keine Lizenzen nutzen können, für die er nicht bezahlt hat.
2. Der User darf nicht für Lizenzen belangt werden und dafür zahlen müssen, die er nicht erhalten hat beziehungsweise nicht nutzen kann.
3. Die Lizenzzuteilung muss in einem angemessenen Zeitrahmen durchführbar sein.

Der erste Punkt ist nach [DP09] durch die Kommunikation und Prüfung zwischen GenLM-Client, GenLM-Server und GenLM License Verifier prinzipiell abgedeckt. Theoretisch könnte ein Benutzer einen Job mit exakt den gleichen Inputdaten weitere Male laufen lassen, ohne dafür Lizenzen anzufordern, da License-Tokens nur User- und Eingabedaten-gebunden sind, aber keinen eindeutigen Zähler enthalten. Dies lässt sich jedoch einfach durch Hinzufügen eines solchen Zählers oder durch begrenzte Gültigkeit der License-Tokens sicherstellen. Außerdem ist diese Problematik nur theoretischer Natur, denn der User hätte dadurch keinen Vorteil, der den ISV direkt schädigen würde.
Der zweite Punkt ist sichergestellt, da der User unter Voraussetzung, dass keine Nachrichten verloren gehen können, mit Sicherheit ein benutzbares License-Token erhalten hat. Im praktischen Szenario mit Nachrichten-Verlust kann er jedoch protokoll-extern direkt beim ISV den Schlüssel für das License-Token anfordern. Somit kann er die Lizenz in jedem Fall nutzen.
Der dritte Punkt ist serverseitig auf jeden Fall durch die sehr gute Skalierung auf Grund der SEDA-Architektur gegeben. Auf Seite der Computing Resources wurde die Geschwindigkeit nicht näher untersucht, jedoch kann davon ausgegangen werden, das Hashbildung im Vergleich zur tatsächlichen Job-Abarbeitung nur eine untergeordnete Zeit benötigt.
Da der GenLM-Server direkt beim ISV untergebracht wird, ist dieser unter Voraussetzung

einer adäquat angepassten Firewall von außen nicht zu manipulieren, da mit dem Server nur per Message Queue kommuniziert wird. Da somit auch das Server-Zertifikat als sicher angesehen werden kann, sind die Daten des License-Tokens ebenfalls nicht manipulierbar. Eine Manipulation am GenLM-Client wäre daher sinnfrei. Angreifbar ist jedoch der GenLM License Verifier der auf Seite der Computing Resources beheimatet ist. Da seine Aufgabe darin besteht, zu entscheiden, ob der Job gestartet werden darf oder nicht, muss er je nach Entscheidung den Job starten lassen oder die Anfrage ablehnen. Diese Prozedur beruht auf einer Entscheidung, sprich if-Abfrage. Je nach Ergebnis der Entscheidung wird an eine andere Stelle im Code gesprungen. Genau hier liegt der Angriffspunkt: Sollte die Sprungadresse durch Reverse-Engeneering beziehungsweise Dekompilierung geknackt und manipuliert werden, so wäre auch ein Starten des Jobs bei fehlgeschlagener Lizenzverifizierung möglich.
Diese Problematik beruht jedoch auf umfangreichen Cracking-Mechanismen. Jedes Lizenzmanagementsystem ist davon betroffen, selbst ein Hardware-Dongle, da im Endeffekt jede Lizenzprüfung auf ein if-Statement hinausläuft, dessen Endsprungaddresse manipuliert werden könnte. Daher ist dies nicht als Schwachstelle speziell von GenLM zu sehen. Bekannte Schutztechniken können hierbei genauso angewendet werden, wie in jedem anderen System, das auf zu schützenden Sprungadressen basiert.

4.1.6 Evaluierung gegen den Anforderungskatalog

In diesem Abschnitt wird das Modell von GenLM gegen die Anforderungen evaluiert.

Allgemein: Unabhängigkeit von der eingesetzten Middleware (siehe 3.3.1)

GenLM gibt keinerlei Informationen preis, auf welchen Middlewares das Lizenzmodell einsetzbar ist. Die Vermutung, dass es auf Grund seiner konzeptionellen Art keine direkte Bindung an eine spezielle Middleware hat, liegt nahe.

Allgemein: Nutzung Grid-spezifischer Technologien (siehe 3.3.2)

GenLM basiert nicht auf Web Services und somit auch nicht auf Standard-Grid-Technologien.

Allgemein: Priorisierung von Jobs (siehe 3.3.3)

Es sind keine konkreten Informationen verfügbar, ob GenLM Priorisierungsmöglichkeiten bietet. Es lässt sich jedoch mutmaßen, dass dies nicht der Fall ist, da der Ansatz von GenLM konzeptioneller Art ist.

Allgemein: Transparenz der Lizenzzuteilung (siehe 3.3.4)

GenLM ist weitestgehend transparent bezüglich der Lizenzzuteilung aus Sicht des Benutzers, vorausgesetzt das gesamte System ist fertig konfiguriert und es sind genügend Lizenzen vorhanden. Die Transparenz geht verloren, wenn der Entschlüsselungskey für das License-Token verlorengegangen ist, da eine erneute Anfrage manuell beim ISV beziehungsweise Hoster des Lizenzservers stattfinden muss.

Sicherheit: Firewallkonfiguration
(siehe 3.4.1)

Da eine Kommunikation nur vom GenLM-Client zum GenLM-Server, der beim ISV gehostet wird, erfolgt, muss aus Sicht der User-Site die Firewall nicht weiter geöffnet werden, als bei Einsatz der Lizenzen in Non-Grid-Umgebungen. Auch der ISV muss seine Firewall, die zwischen GenLM-Server und dem Internet steht, nicht anders konfigurieren, als wenn er nur „normale“ Lizenzanfragen bearbeiten würde. Im Falle des Einsatzes einer Proxy-Tunnelung, die GenLM mit Verweis auf BEinGRID anspricht, gelten die Bedingungen von BEinGRID.

Sicherheit: Zuverlässigkeit in unzuverlässigen Netzwerken
(siehe 3.4.2)

GenLM berücksichtigt den Nachrichtenverlust durch gegenseitiges Bestätigen der Nachrichten zwischen GenLM-Client und GenLM-Server. Falls die Entschlüsselungsdaten auf dem Weg vom GenLM-Server zum GenLM-Client verlorengegangen sind, kann der GenLM-Client diese Daten manuell nochmals anfordern. Sollte die Job-Submission verloren gehen, so merkt dies der Benutzer direkt und kann den Job erneut übermitteln. Über den Verlust von Ergebnissen des Jobs schweigt GenLM, jedoch kann man davon ausgehen, dass auf Grid-übliche Methoden zurückgegriffen wird.

Sicherheit: Verzahnung mit den Grid-spezifischen Sicherheitsmechanismen zur Authentisierung
(siehe 3.4.3)

Die gesamte Kommunikation basiert auf X.509-Zertifikaten zur Authentisierung, jedoch nicht zur Vertraulichkeit. Der GenLM-Client verschlüsselt mit seinem privaten Schlüssel das Request-Token, somit ist der Request von jedem, der den Public-Key des Benutzers kennt, lesbar. Der GenLM-Server verschlüsselt das License-Token mit seinem privaten Schlüssel, somit ist der Inhalt ebenfalls mit dem Public-Key des Servers lesbar.

Sicherheit: Multiple Nutzung von Lizenzen unterbinden
(siehe 3.4.4)

Der Einsatz des GenLM Lizenzvalidieres unterbindet die multiple Nutzung, da der Lizenzvalidierer die Hashes der Eingabedaten kennt und somit mitloggen kann, welche Lizenzen beziehungsweise welche Eingabedatenhashes bereits benutzt wurden.

Sicherheit: Lizenzpools nach administrativen Domänen getrennt
(siehe 3.5.1)

Da der Lizenzserver, sprich der GenLM-Server beim ISV gehostet wird, muss dieser logischerweise die Lizenzpools nach seinen Kunden trennen.

Lizenzierung: Einsatz verschiedener Lizenztypen
(siehe 3.5.2)

GenLM beschreibt in [DP09] nur den Einsatz von Pay-Per-Use-Lizenzen. Floating-Lizenzen werden nicht erwähnt, doch spricht prinzipiell nichts gegen die Nutzung solcher Lizenzen.

Lizenzierung: Auswahl der Lizenzen möglich (siehe 3.5.3)

Dieses Feature ist nicht verfügbar.

Lizenzierung: Hierarchische Vergabe von Lizenznutzungsrechten (siehe 3.5.4)

Betrifft aus Sicht von GenLM nicht das Lizenzmodell, sondern eine andere Ebene, nämlich die der allgemeinen Delegation.

Lizenzierung: Blacklists und Whitelists (siehe 3.5.5)

Weder Blacklists noch Whitelists werden direkt angesprochen. GenLM spricht nur von einer „erfolgreichen Authentifizierung des Benutzers“ ([DP09]), die für eine Lizenznutzung notwendig ist.

Lizenzierung: Lizenzreservierung mit Frist (siehe 3.5.6)

Lizenzen können im GenLM-Ansatz nicht reserviert werden. Eine Lizenz wird genau dann angefragt, wenn sie benötigt wird.

Lizenzierung: Pausieren und Fortsetzen der Lizenznutzung (siehe 3.5.7)

Dieses Feature ist nicht Bestandteil des GenLM-Ansatzes.

Lizenzierung: Umfassendes Logging der angeforderten, zugeteilten und reservierten Lizenzen (siehe 3.5.8)

Mittels einem Policy-Plugin beim GenLM-Server wäre dieses Feature umsetzbar. Es ist aber nicht direkter Bestandteil von GenLM.

Lizenzierung: Änderungen am Lizenzpool und den Listen ohne Neustart (siehe 3.5.9)

Über dieses Feature sind leider keine Informationen verfügbar.

Lizenzierung: Lizenzserver unabhängig der eingesetzten Plattform (siehe 3.5.10)

Der GenLM-Client als auch der GenLM-Server und GenLM-Verifier sind sowohl mit Linux als auch Windows kompatibel.

Lizenz: Lizenz als feste Dateistruktur
(siehe 3.6.1)

Die Lizenzanfragen sowie die License-Tokens sind eindeutig definiert.

Lizenz: Mobilität der Lizenzen
(siehe 3.6.2)

Die Mobilität der Lizenzen wird durch die Aufteilung der Lizenzüberprüfung in den GenLM-Server und den GenLM-Verifier erreicht. Lizenzen können überall dort eingesetzt werden, wo ein GenLM-Verifier läuft. Diese Betrachtungsweise ist jedoch rein technischer Natur und nicht rechtlicher.

Job: Lizenzen verlängern oder früher aufgeben
(siehe 3.7.1)

Dieses Feature ist nicht dokumentiert und es wird nicht spezifiziert, ob eine Verlängerung oder ein früherer CheckIn möglich ist.

Job: Jobs beenden, die innerhalb Timeouts keine Lizenzen erhalten haben
(siehe 3.7.2)

Dieses Feature ist nicht Bestandteil des GenLM-Ansatzes, sondern befindet sich auf einer anderen Ebene.

Job: Bekanntgabe, woher genutzte Lizenzen kommen
(siehe 3.7.3)

Der GenLM-Verifier ist quasi Teil der Middleware. Ob er Informationen zur Verfolgung von Lizenzen preisgibt, ist jedoch nicht bekannt.

Job: Überwachung der Lizenzen
(siehe 3.7.4)

Ob und inwieweit Lizenzen überwacht werden, wird nicht konkret beschrieben.

Job: Überprüfen der Legalität der Lizenzen
(siehe 3.7.5)

Der GenLM-Server überprüft den Benutzer, ob er berechtigt ist, die Lizenz zu nutzen. Der GenLM-Verifier überprüft, ob der übermittelte Job ebenfalls zur Lizenzanforderung herangezogen wurde. Damit wird verhindert, dass Lizenzen für etwas benutzt werden, wofür sie nicht gedacht sind. Dies verlagert jedoch die Problematik zu einem anderen Punkt, da der GenLM-Server immer noch nicht beurteilen kann, für welche Daten die Lizenzen konkret angefordert wurden, da er nur Hashes kennt. Die Benutzung von universitären Lizenzen für einen kommerziellen Job ist damit nicht unterbindbar.

Accounting: Anzeige der erhaltenen Lizenzen (siehe 3.8.1)

Der Benutzer kann das License-Token, das er vom Server erhalten hat, untersuchen und somit die Informationen aus diesem Token beziehen.

Accounting: Sicheres Accounting und Abrechnungsverfahren (siehe 3.8.2)

Accounting und Billing findet beim Ansatz von GenLM beim GenLM-Server in Kombination mit dem GenLM-Verifier statt. Unter Voraussetzung, dass diese nicht kompromittiert wurden, ist das Accounting und Billing sicher durchzuführen.

Accounting: Schutz vor Leugnung des Erhalts einer Lizenz (siehe 3.8.3)

Dieser Schutz ist durch die notwendige Bestätigung des Erhalts der Lizenzen und dem Worst-Case-Fallback auf manuelle weitere Anfrage gegeben.

Accounting: Redundantes Abspeichern aller relevanten Informationen (siehe 3.8.4)

Über dieses Feature sind keine Informationen verfügbar.

Accounting: Statistische Auswertungen ermöglichen (siehe 3.8.5)

Da das Accounting und Billing nur rudimentär stattfindet und für erweiterte Funktionalität auf Policy-Plugins beim GenLM-Server beruht, besitzt GenLM keine direkte Möglichkeit zur statistischen Auswertung.

Accounting: Abrechnung quasi Echtzeit durchführen (siehe 3.8.6)

Da der GenLM-Server auf einer Apache ActiveMQ beruht, ist das Accounting und Billing parallel und schnell möglich.

Accounting: Delay der Lizenzierung berücksichtigen (siehe 3.8.7)

Es findet keine Berücksichtigung von Delays statt. Der GenLM-Server definiert eine Lizenz als benutzt, sobald er die Bestätigung des Erhalts der Lizenz vom GenLM-Client erhalten hat. Wie lange dann die Job-Submission und Validierung der Lizenz durch den GenLM-Verifier noch dauert, wird nicht berücksichtigt.

Accounting: Accounting mit verschiedenen Granularitätsstufen (siehe 3.8.8)

Da das Accounting und Billing nur rudimentär stattfindet und für umfangreichere Funktionalität auf ein Policy-Plugin beim GenLM-Server zurückgegriffen werden muss, ist dieses Feature nicht Teil von GenLM, kann aber mittels dieser Policies durchaus realisiert werden.

Accounting: Kostenexplosionen vermeiden (siehe 3.8.9)

Auch hier gilt, dass das Accounting und Billing nur rudimentär stattfindet und für umfangreichere Funktionalität auf ein Policy-Plugin beim GenLM-Server zurückgegriffen werden muss. Damit liese sich eine Kostenexplosion vermeiden, da einfach keine Lizenz bei einem gewissen Wert zugeteilt wird. Der GenLM-Client geht dann von einem Reject aus, wird jedoch nicht darüber informiert, warum.

Accounting: Einsicht jederzeit möglich (siehe 3.8.10)

Dieses Feature ist nicht Teil von GenLM.

Recht: Einbeziehung der Firmen-Policies (siehe 3.9.1)

GenLM definiert nur ein Lizenzmodell, das nicht auf rechtliche Aspekte eingeht. Daher wird auch keine Rücksicht auf etwaige rechtliche Inkompatibilitäten zwischen Lizenzschema und der Firmenpolitik genommen. Eine angepasste Politik wird vorausgesetzt.

Recht: Lizenzvertrag berücksichtigen (siehe 3.9.2)

Lizenzverträge können dann weiterbestehen, wenn diese die Nutzung innerhalb Grids nicht untersagen und wenn Pay-per-Use-Lizenzen angeboten werden. Eine spezielle Berücksichtigung findet von GenLM nicht statt, sondern es wird davon ausgegangen, dass die Verträge korrekt und legal ausgehandelt wurden.

Recht: Erweiterung des Lizenzvertrags bezüglich Nutzung der Lizenzen in Grids (siehe 3.9.3)

Falls ein bestehender Vertrag die Nutzung der Anwendung in einer Grid-Umgebung untersagt, so ist die Aushandlung eines neuen Vertrages notwendig. GenLM geht jedoch immer von einem korrekten Vertrag aus.

Recht: Rechtliche Konsequenzen bei Ausfall oder Nichtverfügbarkeit des Lizenzservers (siehe 3.9.4)

Da GenLM im Prinzip von einer dauerhaften Verfügbarkeit des GenLM-Servers ausgeht und keine Aussage über rechtliche Ansprüche oder Verträge macht, sind die rechtlichen Konsequenzen nicht Teil des Lizenzmodells.

Recht: Überprüfung der korrekten Lizenznutzung (siehe 3.9.5)

Eine Überprüfung der Lizenznutzung findet nur indirekt mittels den Hashes der Eingabedaten statt. Damit kann aber nicht festgestellt werden, um welche Daten es sich konkret handelt.

4.1.7 Zusammenfassung von GenLM

GenLM legt Wert darauf, dass Lizenzen genau für den Content genutzt werden, für den sie angefordert werden. Dies geschieht damit, dass zur Lizenzanfrage beim Lizenzserver die Input-Daten gehashet werden und dieser Hash an das Lizenztoken gebunden wird, der an die ausführende Instanz und den davor gelagerten Validierer übergeben wird. Damit wird die Korrektheit sichergestellt.
GenLM setzt diese Überprüfung als oberste Priorität. Dass damit aber eine der wichtigsten Problematiken einer Grid-Umgebung, nämlich die Tatsache, dass sich ein Grid über mehrere Domänen erstreckt, nicht umgangen wird, erwähnt GenLM nur mit dem Verweis auf eine weitere notwendige Technologie und verweist in [DP09] auf das Prinzip der Tunnelung von BEinGRID.
GenLM definiert einen allgemeinen Ansatz zur Lizenzierung in verteilten Umgebungen, spezifiziert aber keinen konkreten Einsatz in Grid-Umgebungen und lässt das Konzept einer Virtuellen Organisation völlig außer Acht. Die allgemeine Funktionalität beruht auf der klassischen Client-Server-Lizenzierung mit anschließender Prüfung seitens der ausführenden Instanz.
GenLM befindet sich laut [DP09] derzeit im Closed Beta Status. Ein Softwarepatent ist angemeldet, daher sind technische Details derzeit in einem closed Zustand.

4.2 BEinGRID

Abbildung 4.4: BEinGRID

BEinGRID (Business Experiments in GRID) ist das größte Projekt der Europäischen Kommission. Gegründet wurde BEinGRID laut [DSSWR10] durch die Forschungseinheit Information Society Technologies (IST), die Teil des Sechsten Forschungs-Framework Programms (Framework Programme 6 (FP6)) der Europäischen Union ist. Dieses Konsortium besteht aus 96 Partner von überall aus der EU. Diese 96 Unternehmen stellen die führenden Organisationen in Europa bezüglich Grid Computing und Service Oriented Infrastructures (SOI) dar. Außerdem decken diese Unternehmen einen Großteil der vertikalen Märkte ab und wollen unter Zuhilfenahme von Grids und Clouds ihren wirtschaftlichen und wissenschaftlichen Stand verbessern. BEinGRID sieht sich hierbei als Projekt, das Unternehmen und Organisationen in allen Belangen bezüglich Grid- und Cloud-Computing beiseitesteht und Best-Practises vermittelt.
Derzeit werden laut [BEi09] 25 Business Experiments, die aus den wichtigsten Geschäftsfeldern der EU kommen, als Showcase innerhalb des BEinGRID-Projekts dargestellt.
Die License Management Technical Area des BEinGRID Projekts hat sich damit befasst, wie man kommerzielle Software-Lizenzen in Grid-Umgebungen, speziell dem Grid des BEinGRID Projekts nutzbar machen kann. Dabei setzt der BEinGRID-Ansatz auf ein grundlegendes Pay-per-Use-Lizenzschema im Hinblick auf eine damit einhergehende Client-Server-Architektur, speziell FLEXnet von Flexera Software ([Fle10]) (ehemals Acresso). Das FLEXnet-Schema basiert in der Regel auf Floating-Lizenzen, die bei Anwendungsstart dynamisch beim Lizenzserver ausgecheckt werden und bei Beendigung der Anwendung wieder eingecheckt werden. Dieses Verfahren kann somit auch Pay-per-Use abgerechnet werden. Daher ist FLEXnet grundsätzlich geeignet für den Einsatz in Grid-Umgebungen. Jedoch nutzt FLEXnet zur Authentisierung der anfragenden Nutzer die IP-Adresse, von der die Anfrage stammt und nicht Benutzer- oder Organisationsinformationen, die auf Zertifikaten basieren. Dies stellt somit ein erhebliches Problem bei der direkten Benutzung von FLEXnet innerhalb Grids dar. Der BEinGRID-Ansatz umgeht diese Problematik durch den Einsatz von Proxies und einem PIN/TAN-Verfahren.

4.2.1 Theoretische Beschreibung des Lizenzmanagementansatzes

Der BEinGRID-Ansatz setzt laut [Sim] auf zwei grundlegende Dinge. Zum einen wird die Anfrage nach Verifizierung einer Lizenz mittels einem SOCKS-Proxy beziehungsweise sogar einer Proxy-Chain, je nach Szenario, durchgeführt. Zum anderen benötigt jeder Benutzer, der Jobs auf einem Grid ausführen möchte, einen Lizenz-Account, der in diesem Ansatz als PIN (Personal Identification Number) bezeichnet wird und außerdem einen gültigen Pool von TANs (Transaction Authentication Number), die bei Lizenzbenutzung verbraucht werden. Das PIN/TAN-Konzept setzt somit auf ein ähnliches Prinzip wie das Onlinebanking. Bei jeder Jobsubmission muss der User folglich seien PIN sowie die notwendigen TANs als zusätzliche Job-Parameter angeben. Der Resource Provider überprüft nach Erhalten eines Jobs, ob der Lizenz-Account und die TANs gültig sind. Falls dies der Fall ist, wird über eine proprietäre, verschlüsselte Kommunikation zwischen Lizenzclient (also Teil der kommerziellen Software) und Lizenzserver (beim ISV beziehungsweise Distributor) über eine Proxy-Chain geleitet. Auf den Proxy, der für die Kommunikation von Client zu Server zuständig ist, kann nur mittels gültiger TAN zugegriffen werden. Dies verhindert die Nutzung der Lizenzen durch Unberechtigte.

4.2.2 Anforderungen auf die BEinGRID besonderen Wert legt

Der Lizenzmanagement-Ansatz von BEinGRID geht einen etwas anderen Weg bezüglich den Anforderungen und auch den Zielen als GenLM. Der BEinGRID-Ansatz sieht vor allem folgende Punkte nach [Sim] und [Rae09] als wichtig an:

- **Flexibilität**
  Das gesamte Lizenzmanagementsystem soll für verschiedene Einsatzgebiete nutzbar sein. So werden mehrere Grid-Szenarien vorgestellt, für die der Ansatz nutzbar ist.

- **Middleware-Unterstützung**
  Der BEinGRID-Ansatz möchte alle denkbaren Middlewares unterstützen, daher sollen die Schnittstellen zwischen Middleware und Lizenzmanagement sehr gering ausfallen.

- **Unterstützung von verschiedenen Client-Server-Lizenzmanagementsystemen**
  Obwohl sich BEinGRID meist auf den häufig eingesetzten FLEXnet-Lizenzserver stützt und vorwiegend damit argumentiert, möchte man doch weitestgehend unabhängig der zu Grunde liegenden Client-Server-Architektur sein und im Prinzip jede Client-Server-Architektur mit nur wenigen Anpassungen unterstützen. Der BEinGRID-Ansatz möchte folglich alle bereits existierenden Client-Server-Lizenzmanagementsysteme innerhalb einer Grid-Umgebung untersützen können.

- **Bestehende Lizenzverträge bleiben erhalten**
  Da mittels der Tunnelung mit Hilfe eines SOCKS-Proxy aus Sicht des ISVs sich nichts ändert, können bestehende Lizenzverträge bestehenbleiben, solange der grundsätzliche Einsatz in verteilten Umgebungen nicht untersagt ist.

4.2.3 Technische Beschreibung

Das BEinGRID-Lizenzmanagementsystems stellt nach [Sim] folgende Begriffe vor:

- **LM-Job Description and Submission**
  Die Daten, die eine Job-Übermittlung und -Beschreibung enthalten, sind erweitert worden. Nun können die notwendigen Autorisierungsdaten (sprich PIN und TAN) bei der Job-Übermittlung übertragen werden.

- **LM-Authorization**
  Die Autorisierung erfolgt zunächst mittels einer Access Control List (ACL), die beim Resource Provider hinterlegt ist. Sollten die Daten der Liste nicht ausreichen, um dem anfragenden Benutzer Lizenzen zuzuteilen, erfolgt die Authentisierung und Autorisierung über den LM-Proxy zum Lizenzserver mittels dem bereits beschriebenen PIN/TAN-Verfahren.

- **LM-Proxy**
  Der LM-Proxy ist der Proxy, der für die Kommunikation zwischen Client und Server zwischengeschalten ist.

- **LM-Accounting**
  Das LM-Accounting-Modul dient zu Accounting-Zwecken der Benutzer, die die Lizenzen anfragen und benutzen.

- **LM-Monitor**
  Der LM-Monitor dient zur Überwachung des aktuellen Lizenzpoolzustands. Außerdem bietet er Schnittstellen, mittels dieser diese Informationen abgefragt werden können. Damit können auch Dienste höheren Levels oder bereits existierende SLA-Monitore die Informationen nutzen.

Der Ansatz von BEinGRID unterscheidet nach [Sim] grundsätzlich drei verschiedene Szenarien, bei denen Grids eingesetzt werden. Alle Szenarien basieren jedoch prinzipiell auf einer Proxy-Tunnellösung, die in etwa folgerndermaßen nach [Zie07] darzustellen ist:

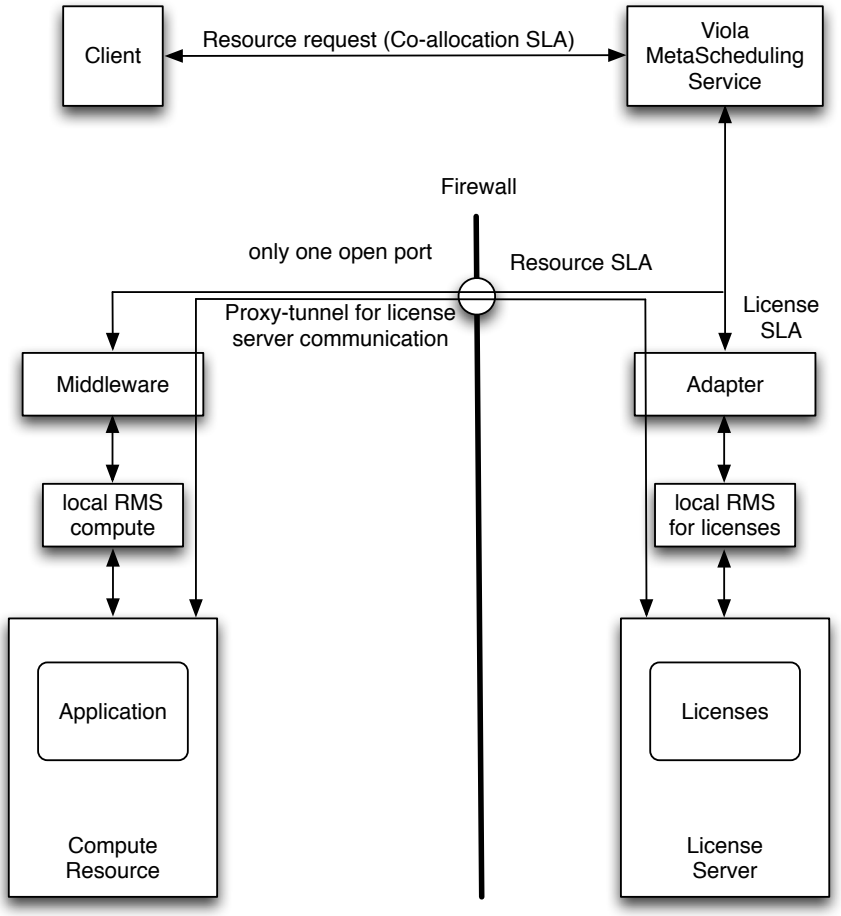

Abbildung 4.5: Basic Architecture von BEinGRID aus [LZWM08]

Die Anwendung, die auf einer Computing Resource ausgeführt werden soll und bereits mittels einem lokalen Resource Management System, das Teil der eingesetzten Middleware ist, auf diese Resource zugeteilt wurde, kommuniziert mittels einem Proxy mit der fremden Domäne, wo der Lizenzserver untergebracht ist. Da die gesamte Kommunikation zwischen Anwendung und Lizenzserver mittels diesem Proxy getunnelt wird, muss die Firewall auch nur einen Port geöffnet haben. Der Lizenzserver hostet die verfügbaren Lizenzen und entscheidet je nach Authentisierung und zugrunde liegendem SLA, ob eine Lizenz über den Proxy zurück übermittelt werden darf.

Im Folgenden werden diese Szenarien nach [Sim] näher beschrieben:

Szenario 1: Gemeinsames Grid

Dieses Szenario beschreibt den Einsatz von Ressourcen eines Resource Providers, auf die mehrere Organisation Zugriff haben. Es wird ein zentraler Lizenzserver beim Resource Provider betrieben, der für die Verwaltung aller Lizenzen aller Organisationen, die auf das Grid Zugriff haben, verantwortlich ist.
Die Anfrage nach Lizenzen beim Lizenzserver erfolgt nun über einen SOCKS-Proxy. Mittels dieser Tunnelung wird dem Lizenzserver eine valide IP der anfragenden Organisation, die tatsächlich Lizenzen besitzt, übermittelt. Bei FLEXnet ist dies zum Beispiel notwendig, da der FLEXnet-Server auf IP-Basis die Gültigkeit der Anfrage prüft.
Der BEinGRID-Ansatz nennt zwei leicht unterschiedliche Szenarien, wie der LM-Proxy die Berechtigungen überprüft. Die erste Möglichkeit ist die Prüfung basierend auf einer lokalen Access Control List (ACL). Mittels dieser Berechtigungsliste konfiguriert der Resource Manager den LM Proxy. Die Zuteilung der Berechtigungen ist also unabhängig von der Job Übermittlung. Es wird je nach Bedarf beim LM-Proxy die Konfiguration mittels dem Resource Manager aktualisiert.

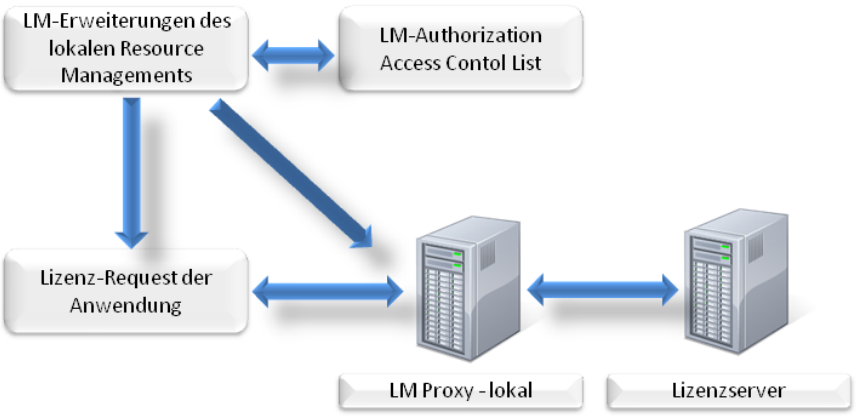

Abbildung 4.6: Szenario 1a von BEinGRID nach [Sim]

Ein etwas modifizierter Ansatz ist die Überprüfung der Berechtigung direkt bei Job Übermittlung. Dazu werden die LM-Job-Submission und LM-Job-Description-Module herangezogen, die bei jeder Job Übermittlung die Rechte erfragen.

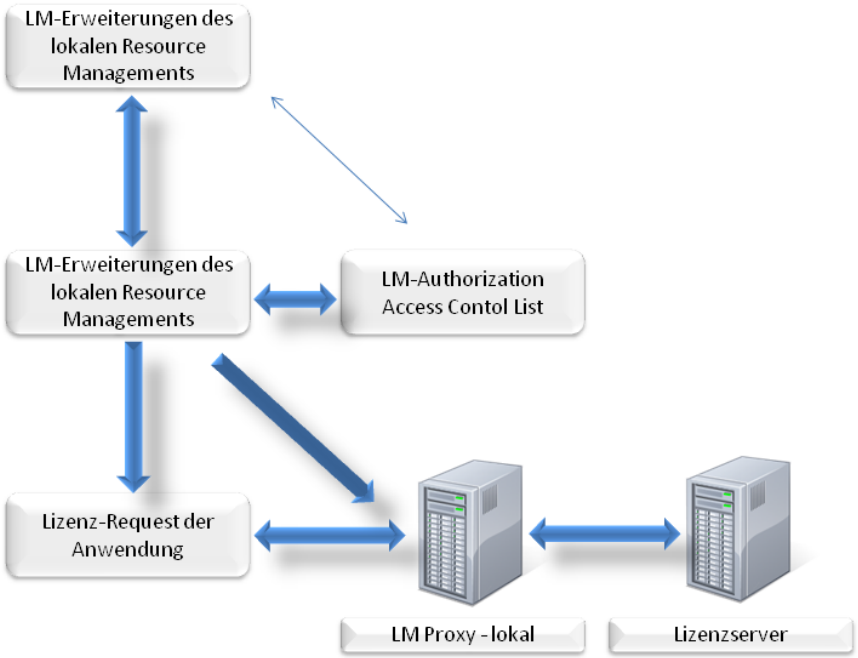

Abbildung 4.7: Szenario 1b von BEinGRID nach [Sim]

Szenario 2: Gemeinsames Grid mit Lizenzserver beim ISV

Dieses Szenario deckt sich im Prinzip mit dem ersten Szenario mit dem Unterschied, dass der Lizenzserver nicht mehr direkt beim Resource Provider untergebracht ist, sondern nun extern vom ISV verwaltet wird. Hierbei kommt nun eine Proxy-Chain zum Einsatz. Der LM-Proxy ist weiterhin unter Obhut des Resource Providers und dient zur Kommunikation mit dem LM-Upstream-Proxy, der beim ISV untergebracht ist. Erst dieser Upstream-Proxy kommuniziert mit dem eigentlichen Lizenzserver, der vom ISV betreut und verwaltet wird. Dieses Szenario erfordert explizite „Delegation of Trust“ zwischen den beiden Proxies, da hier die größte Angriffsmöglichkeit besteht.

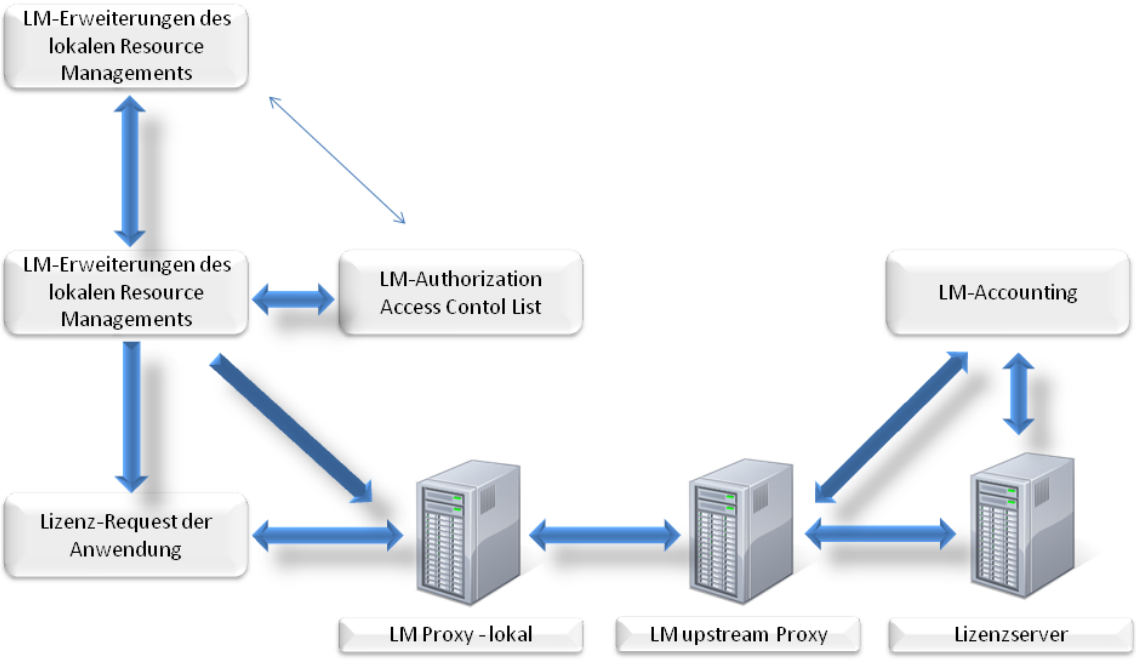

Abbildung 4.8: Szenario 2 von BEinGRID nach [Sim]

Dieses Szenario deckt insbesondere Nicht-FLEXnet-Client-Server-Lizenzmanagementschemata ab, da solche Architekturen oft darauf beruhen, dass der Lizenzserver beim ISV beziehungsweise beim Distributor untergebracht ist und somit Kommunikation mit externen Komponenten notwendig ist.

Szenario 3: Externes Grid bei einem zufälligen Provider

Dieses Szenario beschreibt eine Organisation, die kein lokales Grid betreibt. Diese Organisation möchte jedoch einen Job auf einem externen Grid ausführen. Der Job verwendet eine kommerzielle Software, die mittels FLEXnet-Lizenzen lizenziert wird. Der zugehörige FLEXnet-Server wird lokal bei der Organisation betrieben. Die Problematik dieses Szenarios besteht darin, dass die Organisation nun Zugriff von allen möglichen externen Resource-Providern auf den FLEXnet-Server ermöglichen müsste, da vor Job Übermittlung nicht klar ist, welche Ressourcen benutzt werden und eventuell sogar relativ kurzfristig der Resource-Provider bestimmt wurde. Dies würde ein erhebliches Sicherheitsrisiko darstellen.
Der Ansatz nach BEinGRID definiert hierfür wiederum einen FLEXnet-Server, der als Web/SOCKS-Server enkapsuliert ist und mittels einem PIN/TAN-Verfahren erreichbar ist. Diese Kombination erlaubt zusätzlich auch Monitoring-Aspekte.

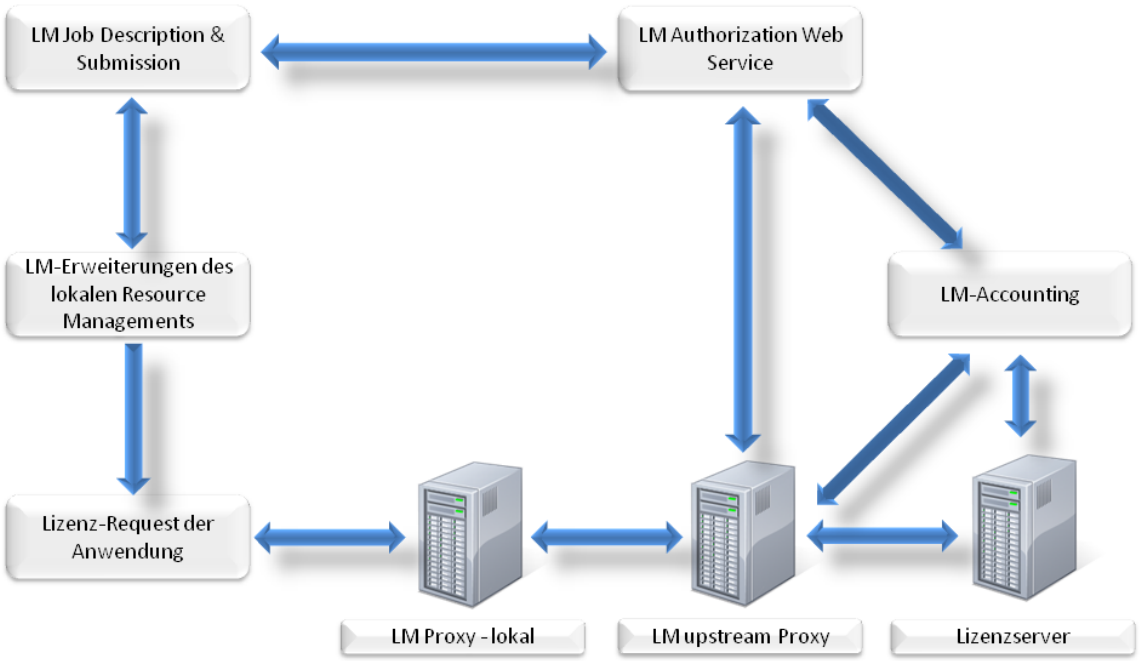

Abbildung 4.9: Szenario 3 von BEinGRID nach [Sim]

Szenario 4: Lokales Grid und License Service Provider

Dieses Szenario beschreibt eine Organisation, die ein eigenes Grid besitzt, aber keine Lizenzen. Da sie jedoch kurzzeitig einen Job auf dem lokalen Grid mit einer kommerziellen Anwendung starten möchte, wofür Lizenzen von einem License Service Provider angeboten werden und je nach Bedarf abgerechnet werden können. Für dieses System ist es notwendig, dass die Organisation beim License Service Provider einen Abrechnungsaccount besitzt. Über diesen Account können dann die notwendigen Lizenzen, in Form eines License-Accounts (PIN) und TANs, angefordert werden.
Beim Start der Anwendung auf dem Grid muss nun eine Kommunikation diese Anwendung vom Lizenzserver auf Seite des License Service Provider mit den notwendigen Lizenzen bedient werden können. Die Anfrage nach Lizenzen wird also über einen lokalen Proxy umgeleitet. Anschließend wird sie über einen Upstream-Proxy, der beim License Service Provider läuft, zum Lizenzserver weitergegeben.
Konkret gesagt wird zunächst die License-Account Information von einer Umgebungsvariable des Jobs ausgelesen und an die Lizenzanfrage angehängt. Diese Lizenzanfrage wird dann dem lokalen Proxy übergeben, der außerdem dafür zuständig ist, dass gültige, der Anfrage entsprechende TANs ausgewählt werden und bei der Übermittlung an den externen Upstream-Proxy mit übergeben werden. Dieser Upstream-Proxy entpackt die Informationen und übergibt sie dem Lizenzserver, der nun dafür verantwortlich ist, die korrekten Lizenzen auszuchecken. Das Lizenzpaket wird anschließend wieder dem Upstream-Proxy übergeben, der nun ein Billing-Verfahren starten kann. Anschließend wird das Lizenzpaket über die Proxychain zurückgeleitet und kann somit von der Organisation genutzt werden.

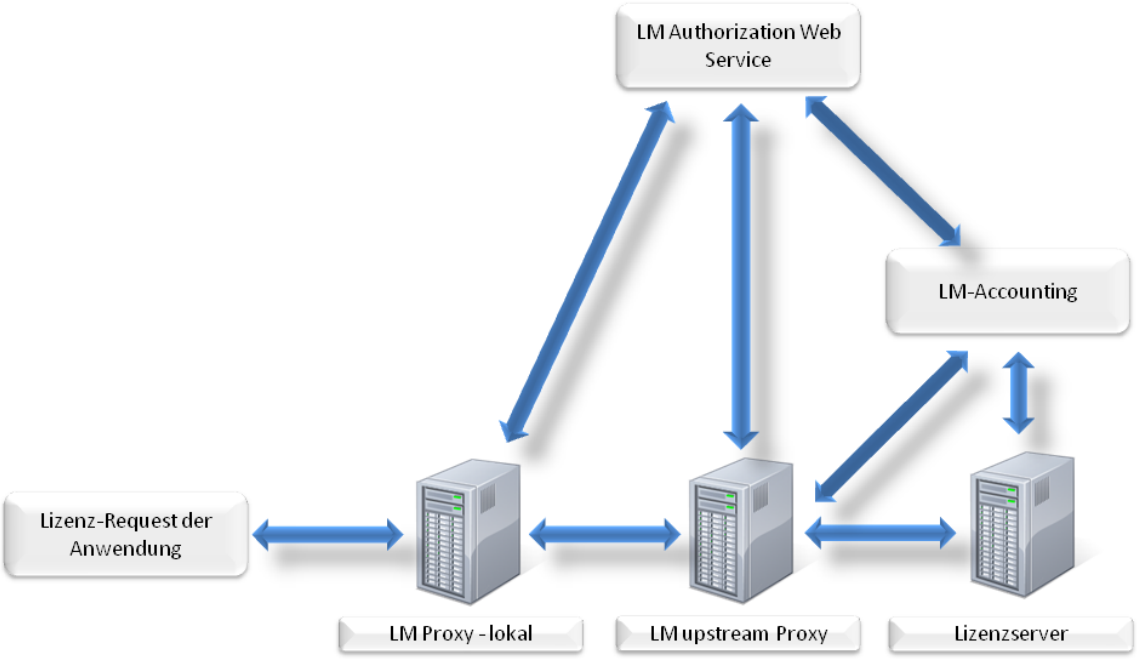

Abbildung 4.10: Szenario 4 von BEinGRID nach [Sim]

4.2.4 Skalierung und praktische Betrachtungen

Der Flaschenhals des BEinGRID-Ansatzes ist der Einsatz der Proxies. Wichtigste Voraussetzung ist somit eine potente Hardware, auf der die Proxies untergebracht sind, so dass möglichst viele Verbindungen gleichzeitig und mit geringem Delay gehandhabt werden können.
In den Business Experiments von BEinGRID wird das Verfahren bereits teilweise eingesetzt und funktioniert dort auch den Umständen entsprechend gut. Da heutzutage Proxies bereits bis zu 2500 Verbindungen pro Sekunde annehmen und bearbeiten können, sollten selbst größere Grid-Umgebungen kein Performance-Problem haben. Fraglich ist jedoch, ob die Performance und Skalierbarkeit in Zukunft genauso schnell zunimmt, wie die Größe von Grids ansteigt.

4.2.5 Sicherheitsbetrachtungen von BEinGRID

Der BEinGRID-Ansatz ist immer um den Einsatz von Proxies angesiedelt. Die Kommunikation zwischen den beiden Parteien erfolgt daher stets sicher verschlüsselt und authentisiert sowie autorisiert mittels dieser Proxies. Man muss also diese Proxies mit allen herkömmlich bekannten Sicherheitsmaßnahmen schützen. Eine Kompromittierung eines der Proxies bricht somit sämtliche Sicherheit, Vertraulichkeit und Integrität. Die Proxies sind außerdem einem Denial-of-Service-Angriff ausgeliefert. DoS-Attacken müssen also bereits davor, falls möglich, abgefangen werden, da sonst keine Lizenzen angefordert oder eingecheckt werden können.
Alle anderen Kommunikationsabläufe finden intern innerhalb einer administrativen Domäne statt und müssen mit standartisierten Sicherheitsmechanismen geschützt werden.

4.2.6 Evaluierung gegen den Anforderungskatalog

In diesem Abschnitt wird das Modell von BEinGRID gegen die Anforderungen evaluiert.

Allgemein: Unabhängigkeit von der eingesetzten Middleware (siehe 3.3.1)

BEinGRID ist auf den verschiedenen Standard-Middlewares einsetzbar. Welche Anpassungen konkret jedoch vorgenommen werden müssen ist nicht bekannt. Aktuell hat BEinGRID jedoch nur seine Komponenten für Globus Toolkit 4 sowie Unicore 6 entwickelt. Weitere Grid-Middlewares sind noch nicht verfügbar aber eventuell geplant.

Allgemein: Nutzung Grid-spezifischer Technologien (siehe 3.3.2)

BEinGRID basiert grundlegend nicht auf Web Services und somit auch nicht auf Standard-Grid-Technologien. Nur das Accounting wird mit einem Dienst oberflächlich durchgeführt.

Allgemein: Priorisierung von Jobs (siehe 3.3.3)

Es sind keine konkreten Informationen verfügbar, ob der Ansatz von BEinGRID Priorisierungsmöglichkeiten bietet. Da der Ansatz von BEinGRID auf standard-Lizenzserver wie zum Beispiel FLEXnet basiert, ist davon auszugehen, dass eine Priorisierung genau dann

geboten wird, wenn der Lizenzserver dies unterstützt. Somit ist dieser Punkt nicht Teil des Lizenzmodells von BEinGRID, sondern Teil eines unabhängigen Lizenzservers.

Allgemein: Transparenz der Lizenzzuteilung (siehe 3.3.4)

Der Ansatz von BEinGRID läuft weitestgehend transparent ab, nachdem alle erforderlichen Proxies installiert und konfiguriert wurden. Die Proxies sind jedoch in der Hinsicht umfangreich konfigurierbar, welche Benutzer welche Lizenzrechte haben. Ist diese Konfiguration jedoch korrekt vorgenommen, so erfolgt die eigentliche Lizenzzuteilung für den Anwender transparent, wenn das zugrunde liegende Serversystem (wie zum Beispiel FLEXnet) transparent arbeitet. Zu beachten ist jedoch, dass der Benutzer laut [Rae09] einen Lizenz-Account und eine TAN-Liste benötigt. Bei einer Job-Submission muss er diese angeben.

Sicherheit: Firewallkonfiguration (siehe 3.4.1)

Die gesamte Kommunikation erfolgt über einen oder mehrere Proxies. Daher muss die Firewall so geöffnet werden, dass die Kommunikation mit beziehungsweise zwischen den Proxies nicht blockiert wird.

Sicherheit: Zuverlässigkeit in unzuverlässigen Netzwerken (siehe 3.4.2)

BEinGRID ermöglicht die Nutzung bestehender Client-Server-Lizenzmodelle mittels Proxy-Tunnels. Da diese Modelle bereits in non-Grid-Umgebungen eingesetzt werden, muss eine Zuverlässigkeit in unzuverlässigen Netzen gewährleistet sein. Der Tunnel selbst kann mit herkömmlichen Mitteln, wie sie allgemein zur Tunnelung verwendet werden, gesichert werden.

Sicherheit: Verzahnung mit den Grid-spezifischen Sicherheitsmechanismen zur Authentisierung (siehe 3.4.3)

Da dem Ansatz von BEinGRID ein klassisches Client-Server-Lizenz-Modell zugrunde liegt, gelten die dafür definierten Sicherheitsvorkehrungen. Ob X.509-Zertifikate eingesetzt werden oder nicht, ist somit nicht von BEinGRID festgelegt.
Die Authentisierung an den Proxies erfolgt mittels einem PIN/TAN-Konzept laut [Sim] auf Basis des One-Time-Password-Verfahrens (OTP).

Sicherheit: Multiple Nutzung von Lizenzen unterbinden (siehe 3.4.4)

Da nur eine Tunnelung des darunterliegenden Client-Server-Lizenz-Modells stattfindet, unterliegt diese Aufgabe genau diesem. Allerdings dürfen die Proxies nicht kompromittiert sein.

Lizenzierung: Lizenzpools nach administrativen Domänen getrennt (siehe 3.5.1)

Inwieweit die zugrunde liegende Client-Server-Struktur wie zum Beispiel FLEXnet dies handhabt, ist nicht Teil des BEinGRID-Ansatzes, sondern betrifft eine andere Ebene.

Lizenzierung: Einsatz verschiedener Lizenztypen (siehe 3.5.2)

Welche Lizenztypen konkret angeboten werden, ist direkt vom darunterliegenden Modell abhängig und nicht von BEinGRID. Jedoch sind Node-Locked-Lizenzen sinnfrei.

Lizenzierung: Auswahl der Lizenzen möglich (siehe 3.5.3)

Da die Anwendung, die ausgeführt werden soll, relativ autonom die Lizenzanfrage basierend auf dem zugrunde liegenden Client-Server-Modell wie zum Beispiel FLEXnet ausführt, hat der Benutzer nur insoweit Einfluss auf die Lizenzauswahl, wie es das Modell zulässt.

Lizenzierung: Hierarchische Vergabe von Lizenznutzungsrechten (siehe 3.5.4)

Aus Sicht von BEinGRID ist die Vergabe der Nutzungsrechte nicht Teil des Lizenzschemas, sondern betrifft die allgemeine Delegation.

Lizenzierung: Blacklists und Whitelists (siehe 3.5.5)

Da jeder Benutzer über einen Abrechnungsaccount verfügen muss, damit er eine PIN zur Kommunikation mit den Proxies erhält, ist eine Black- und Whitelist auf dieser Ebene denkbar.

Lizenzierung: Lizenzreservierung mit Frist (siehe 3.5.6)

Reservierungen sind nicht direkt möglich.

Lizenzierung: Pausieren und Fortsetzen der Lizenznutzung (siehe 3.5.7)

Dieses Feature ist nicht Bestandteil des BEinGRID-Ansatzes.

Lizenzierung: Umfassendes Logging der angeforderten, zugeteilten und reservierten Lizenzen (siehe 3.5.8)

Logging findet sowohl beim Lizenzserver, wie zum Beispiel FLEXnet, als auch bei der Job Submission statt.

Lizenzierung: Änderungen am Lizenzpool und den Listen ohne Neustart (siehe 3.5.9)

Da der Lizenzserver grundsätzlich unabhängig vom BEinGRID-Ansatz ist, hat BEinGRID auch keinen Einfluss auf dieses Feature. Es kommt nur darauf an, welcher Lizenzserver (FLEXnet etc.) eingesetzt wird und inwieweit dieser die Änderungen am Lizenzpool ohne Neustart unterstützt.

Lizenzierung: Lizenzserver unabhängig der eingesetzten Plattform (siehe 3.5.10)

Der BEinGRID-Ansatz beruht auf dem Einsatz von Proxies. Daher gehen die Einschränkungen vom zugrundeliegenden Client-Server-Modell aus. Die Proxies basieren auf SOCKS SS5 und sind daher an diese Einschränkungen gebunden.

Lizenz: Lizenz als feste Dateistruktur (siehe 3.6.1)

Die Dateistruktur der Lizenzen ist abhängig von der zugrunde liegenden Technik (FLEXnet etc.). Daher ist keine einheitliche Lizenzstruktur vorhanden.

Lizenz: Mobilität der Lizenzen (siehe 3.6.2)

Durch das Tunneln mittels Proxies wird eine Mobilität der Lizenzen ereicht. Diese Mobilität ist jedoch nur technischer Natur. Rechtliche Aspekte werden hierbei nicht betrachtet.

Job: Lizenzen verlängern oder früher aufgeben (siehe 3.7.1)

Dieses Feature ist vom darüberliegenden Modell abhängig und daher nicht direkter Teil des BEinGRID-Ansatzes.

Job: Jobs beenden, die innerhalb Timeouts keine Lizenzen erhalten haben (siehe 3.7.2)

Dieses Feature ist nicht Bestandteil des BEinGRID-Ansatzes, sondern befindet sich auf einer anderen Ebene.

Job: Bekanntgabe, woher genutzte Lizenzen kommen (siehe 3.7.3)

Aus Sicht der Middleware wird eine Anwendung gestartet, die sich selbstständig mittels der Proxy-Tunnelung um die notwendigen Lizenzen kümmert. Die Middleware selbst weiß somit nicht über die Lizenzen bescheid, sie kann nur beurteilen, ob eine Anwendung wie gewünscht startete oder ob sie blockierte, da keine Lizenzen zugeteilt wurden. Der License Accounting Web Service, der laut [Rae09] bestandteil des BEinGRID-Ansatzes ist, kann jedoch darüber Informationen geben.

Job: Überwachung der Lizenzen
(siehe 3.7.4)

Der Lizenzserver sollte wissen, wo seine Lizenzen momentan sind. Dies ist jedoch abhängig vom eingesetzten Client-Server-Modell und dessen Features. Unabhängig davon weiß der License Accounting Web Service über die momentan zugeteilten Lizenzen bescheid.

Job: Überprüfen der Legalität der Lizenzen
(siehe 3.7.5)

Unter Voraussetzung, dass die Proxies nicht kompromittiert sind, ist eine Überprüfung der Lizenzen direkt mittels dem darunterliegenden Client-Server-Modell wie zum Beispiel FLEXnet möglich. Der Ansatz von BEinGRID verlässt sich dahingehend also auf dieses Modell. Eine Kompromittierung eines der Proxies kann jedoch dazu führen, dass Lizenzen illegal genutzt werden.

Accounting: Anzeige der erhaltenen Lizenzen
(siehe 3.8.1)

Der Benutzer kann beim License Accounting Web Service seine Abrechnungsdaten einsehen und bekommt dort auch Informationen, welche Lizenzen er erhalten hat.

Accounting: Sicheres Accounting und Abrechnungsverfahren
(siehe 3.8.2)

Der License Accounting Web Service ist für das Accounting und Billing in Kombination mit dem Lizenzserver verantwortlich. Es gilt somit diesen Service vor falschen Angaben zu schützen. Inwieweit der Server sicher ist, ist abhängig vom eingesetzten Client-Server-Modell.

Accounting: Schutz vor Leugnung des Erhalts einer Lizenz
(siehe 3.8.3)

Dieses Feature ist nicht Teil des Ansatzes von BEinGRID, sondern Aufgabe des darunterliegenden Client-Server-Modells. Je nach eingesetztem Lizenz-Server und zugehörigem Modell ist dieser Schutz vorhanden oder nicht.

Accounting: Redundantes Abspeichern aller relevanten Informationen
(siehe 3.8.4)

Da das Logging Aufgabe des zugrunde liegenden Client-Server-Modells ist, ist dieses Feature abhängig vom eingesetzten Lizenz-Server.

Accounting: Statistische Auswertungen ermöglichen
(siehe 3.8.5)

Dieses Feature ist ebenfalls Aufgabe des zugrunde liegenden Client-Server-Modells und nicht Teil der Komponenten von BEinGRID.

Accounting: Abrechnung quasi Echtzeit durchführen (siehe 3.8.6)

Die Abrechnung erfolgt indirekt mittels dem License Accounting Web Service in Kombination mit dem eingesetzten Lizenzserver. Inwieweit der Web Service unter hoher Last skaliert, ist nicht bekannt. Außerdem müssen die Proxies vor DoS-Attacken geschützt sein, damit dort keine großen Verzögerungen auftreten.

Accounting: Delay der Lizenzierung berücksichtigen (siehe 3.8.7)

Die Delays werden nicht berücksichtigt, was zu einer Problematik führen kann, wenn ein Proxy mittels zum Beispiel einer DoS-Attacke überlastet wird oder der License Accounting Web Service überlastet ist, was zu erheblichen Verzögerungen führen kann.

Accounting: Accounting mit verschiedenen Granularitätsstufen (siehe 3.8.8)

Inwieweit Accounting auf verschiedenen Granularitätsstufen stattfindet, ist nicht näher dokumentiert.

Accounting: Kostenexplosionen vermeiden (siehe 3.8.9)

Ob dieses Feature vorhanden ist, ist nicht näher dokumentiert, jedoch liese es sich einfach als Teil des License Accounting Web Services umsetzen.

Accounting: Einsicht jederzeit möglich (siehe 3.8.10)

Der License Accounting Web Service hält umfangreiche Informationen bereit. Es ist davon auszugehen, dass sowohl der Benutzer als auch der Administrator somit jederzeit Zugriff auf die relevanten Informationen erhält.

Recht: Einbeziehung der Firmen-Policies (siehe 3.9.1)

Solange die individuellen Firmenpolitiken die Nutzung der Komponenten des BEinGRID-Ansatzes nicht untersagen und die bestehenden Lizenzverträge eine Nutzung in Grids erlauben, ist der Einsatz unproblematisch. Falls dies jedoch nicht der Fall ist, so bedarf es einer Änderung.

Recht: Lizenzvertrag berücksichtigen (siehe 3.9.2)

Lizenzverträge können in der Regel bestehen bleiben, wenn die Nutzung der Anwendung nicht in Grid-Umgebungen untersagt ist. Außerdem müssen Pay-per-Use oder Floating-Lizenzen verfügbar sein. Eine spezielle Berücksichtigung dieser Tatsachen findet jedoch nicht statt. Es wird davon ausgegangen, dass die Verträge korrekt ausgehandelt wurden.

Recht: Erweiterung des Lizenzvertrags bezüglich Nutzung der Lizenzen in Grids (siehe 3.9.3)

Der Ansatz von BEinGRID beruht darauf, dass ein korrekter Lizenzvertrag ausgehandelt wurde. BEinGRID geht jedoch davon aus, dass dies der Fall ist.

Recht: Rechtliche Konsequenzen bei Ausfall oder Nichtverfügbarkeit des Lizenzservers (siehe 3.9.4)

Da der Ansatz von BEinGRID von einer dauerhaften Verfügbarkeit des Lizenz-Servers und des License Accounting Web Services ausgeht und keine Aussage über rechtliche Ansprüche oder Verträge macht, sind die rechtlichen Konsequenzen nicht Teil des Lizenzmodells beziehungsweise nicht näher spezifiziert.

Recht: Überprüfung der korrekten Lizenznutzung (siehe 3.9.5)

Dieses Feature wird nicht näher dokumentiert.

4.2.7 Zusammenfassung von BEinGRID

Der BEinGRID-Ansatz beschreibt technische Möglichkeiten zur sicheren Kommunikation zwischen den jeweiligen Instanzen. Jedoch bleibt die Definition sehr technisch und geht keinesfalls auf logische Strukturen ein. Es wird niemals der Fall angesprochen, dass eine Virtuelle Organisation vorhanden ist, die dem Fallbeispiel dieser Diplomarbeit gerecht wird.
Der Ansatz setzt somit technisch immer auf den Einsatz eines oder mehrerer Proxies, womit IP-basierte Authentisierungsverfahren innerhalb eines Grids möglich werden. Außerdem ermöglicht der Einsatz eines Proxies, dass Firewalls nur für eine Verbindung geöffnet werden müssen. Da BEinGRID auf SOCKS-Proxies aufsetzt, wird damit auch sicherheitsrelevante Verschlüsselung sowie Authentisierung und Autorisierung ermöglicht. Der Lizenzserver wird als SOCKS/Web Service enkapsuliert. Außerdem werden Accounting- und Abrechungsfunktionen sowie das Monitoring der aktuell genutzten Lizenzen geboten.
BEinGRID dient dazu, klassische Client-Server-Modelle wie zum Beispiel FLEXnet in einer Grid-Umgebung lauffähig zu machen und diese mittels Proxies einzusetzen.
Eine Prüfung, ob die angeforderten Lizenzen tatsächlich dafür genutzt werden, wofür sie angefordert wurden, findet allerdings nicht statt.
Der BEinGRID-Ansatz wird derzeit implementiert und setzt dabei auf einen SOCKS SS5-Proxy ([Ric10] sowie das One-Time-Password (OTP) Verfahren als PIN/TAN-Schema. Der gesamte Ansatz wird unter der GPL veröffentlicht und lauffähig unter Globus Toolkit 4 sowie Unicore 6 sein.
Der BEinGRID-Ansatz zeigt in den oben angesprochenen Szenarios durchaus, dass er relativ generisch ist und in verschiedenen Use-Cases einsetzbar ist, wenn man nur auf die gebotenen Funktionalitäten Wert legt. Jedoch ist der Ansatz von BEinGRID, wie der Name schon sagt, auf „Business Experiments“ vorwiegend zugeschnitten. In diesem Experimentierumfeld gibt es in der Regel keine rechtlich einschränkenden Verträge, da der Einsatz der auf dem Grid auszuführenden Anwendung und Entwicklung dieser Anwendung nahe beieinander liegen.

4.3 SmartLM

Abbildung 4.11: SmartLM

SmartLM ([Sma10]) ist laut [IT10] ein Projekt, dem mehrere Organisationen angehören. Diese sind Atos Origin, Fraunhofer SCAI, Jülich Research Center, Cineca, INTES GmbH, Ansys, LMS International, The 451 Group, T-Systems-SfR, Centro de Supercomputación de Galicia und Gridcore AB. Das Projekt startete im Februar 2008 und ist auf eine Dauer von 30 Monaten angelegt.
SmartLM soll dabei am Ende seiner Entwicklung (die zum Zeitpunkt der Diplomarbeit noch lange nicht abgeschlossen ist), nicht nur ein Lizenzmodell für Grid-Umgebungen bieten, sondern einige weitere Aufgaben erfüllen. SmartLM ist also deutlich mehr als nur ein Lizenzmodell für Grids. Es ist konkret gesagt ein Rahmenwerk für neue Geschäftsmodelle im allgemeinen Bezug des Distributed Computings. Damit wird natürlich auch Lizenzmanagement in Grids betrachtet und ein dafür geeignetes Modell konzipiert. Dieses Modell ist Teil des gesamten SmartLM-Konzepts und nur in gewisser Hinsicht einzeln zu betrachten.

4.3.1 Theoretische Beschreibung des Lizenzmanagementansatzes

Der SmartLM-Ansatz basiert laut [IT09] darauf, dass Software-Lizenzen als Services angesehen werden. Darüber liegt als Kernelement der SmartLM License Service. Alle Lizenzen werden von diesem Dienst verwaltet. Der License-Dienst ist nicht monolithisch, sondern vielmehr eine Sammlung von einzelnen Diensten, die untereinander auf Tokens basierend kommunizieren. Der License Service kann allgemein in verteilten Systemen sowie in lose gekoppelten Systemen eingesetzt werden, da die Software-Lizenzen selbst Web Service Resources sind und somit dynamisch sind.

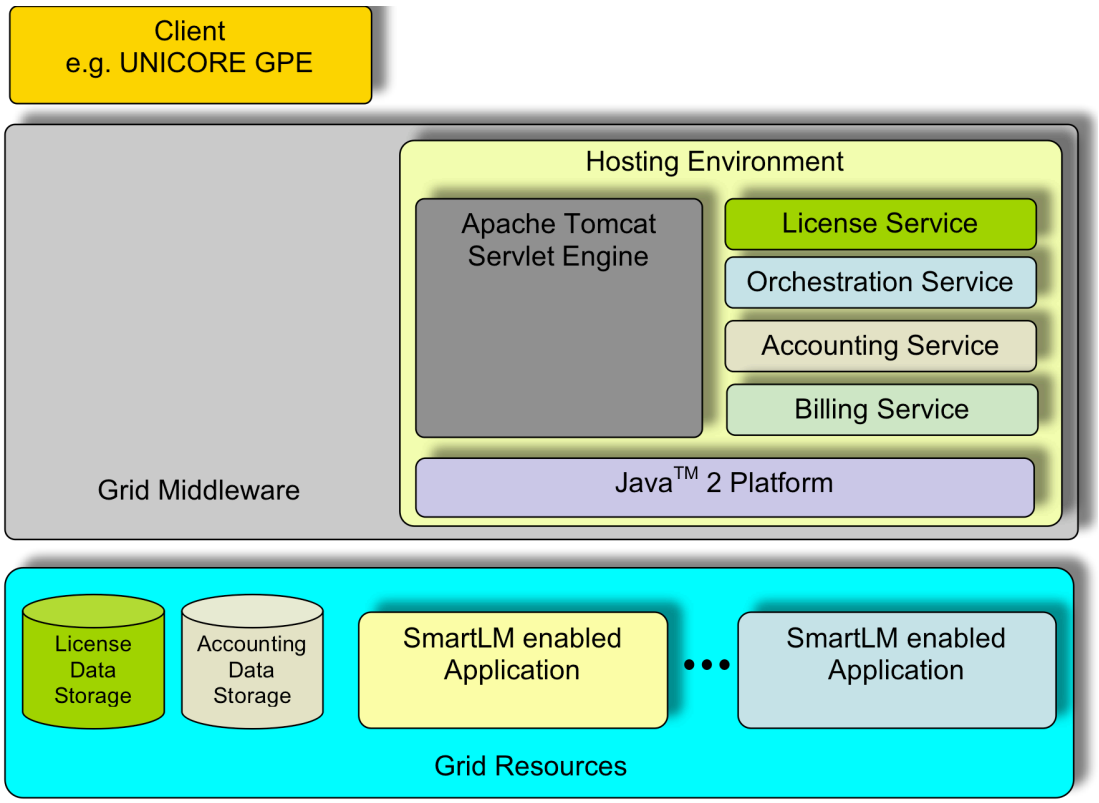

Abbildung 4.12: SmartLM Architektur nach [Zie07]

Die Lizenzen werden laut [IT09] als Agreements gemanaged. Diese Agreements sind erweiterte Service-Level-Agreements (SLA), die zwischen Verkäufer und Käufer ausgehandelt werden müssen (Negotiation). Außerdem sind diese Agreements dynamisch, so dass sie später je nach Bedarf angepasst werden können (Re-Negotiation).
Darüberhinaus bietet SmartLM die Möglichkeit, Lizenzen vorab zu reservieren. Laut [IT09] sogar so, dass Lizenzen verfügbar sind, wenn benötigt, aber nicht wegen einer Reservierung blockiert werden, falls eine andere Anwendung sie benötigt. Dieses Prinzip vermischt das On-Demand-Schema mit anderen Lizenzschemata.
SmartLM bietet außerdem die Möglichkeit ein umfangreiches Logging für das Accounting und Billing. Ausführliche Analysen des Lizenzgebrauchs sowie Möglichkeiten der Budget-Kontrolle sind vorgesehen.
SmartLM entkoppelt die Ausführungs-Umgebung von der Seite, die für den Lizenzserver verantwortlich ist. Die Implementierung des Lizenzservers soll Teil des Grids beziehungsweise der Cloud sein.

4.3.2 Anforderungen auf die SmartLM besonderen Wert legt

Laut [IT10] sind das vor allem folgende:

- SmartLM möchte ein umfassendes Verständnis erreichen in Bezug auf die Anforderungen für ein Lizenzmodell sowohl in Grids als auch in Clouds im kommerziellen Umfeld. Dazu hat SmartLM sowohl Software-Vendors, Application Service Providers (ASP), IT Integrators, Resource Providers und Endbenutzer befragt und miteinbezogen.
- SmartLM möchte Service-orientierte Geschäftsmodelle genauer untersuchen, insbesondere in Hinblick auf über mehrere Organisationen verteilte Szenarios.
- SmartLM soll am Ende ein sicheres Plattform-unabhängiges Lizenzmanagement-Framework ergeben.
- SmartLM möchte Modelle und Technologien zum Accounting und Billing in Bezug auf Lizenzen anbieten.
- Die entwickelten Management-Tools sollen anhand kommerzieller Anwendungen in Grid-Umgebungen überprüft werden.

SmartLM ist folglich nicht nur ein Lizenzmodell für Grid-Umgebungen sondern ein gesamtes Rahmenwerk für kommerzielle Anwendungen und deren Umgebungsfeld innerhalb Grids.

4.3.3 Technische Beschreibung

Das konkrete Produkt, das SmartLM entwickelt, nennt sich *elasticLM*. Die wichtigsten Features dieses Produkts sind laut [IT09] folgende:

- Die Anwendungen, die die Lizenzen benötigen, können in den Grid-Umgebungen und Clouds auch dann laufen, wenn während der Ausführung keine Netzwerkverbindung zur Seite besteht, die den zugehörigen Lizenzserver hostet, der die Lizenzen zugeteilt hat.
- Budget-Grenzen werden bei Anforderung einer Lizenz geprüft und durchgesetzt.
- Die Lizenznutzung kann einfach überwacht werden, da elasticLM Zugriff auf und Management von allen Lizenzen erlaubt, die einer Site gehören.
- Es können lokale Policies bezüglich Site-spezifischen Lizenznutzungsberechtigungen definiert werden. Diese Policies bauen auf den zugrundeliegenden eingebetteten Policies des ISVs auf.
- elasticLM unterstützt das Co-Scheduling von Lizenzen und Computing Resources. Das bedeutet, es gibt die Möglichkeit, Lizenzen genau dann zu beantragen, wenn bestimmte Ressourcen verfügbar werden. Außerdem wir die Re-Negotiation der Lizenzbedingungen während der Laufzeit unterstützt.
- Ein genauer und benutzerspezifischer Preis kann vorab berechnet werden. Diese Berechnung basiert auf einer Vielzahl von konfigurierbaren Parametern.
- Lizenzen können reserviert werden für einen späteren Gebrauch. Desweiteren werden Lizenzen anhand der Verfügbarkeit der Ressourcen koordiniert.

- Mittels dem Accounting und Billing System, das auf der tatsächlichen Lizenznutzung basiert, ist es möglich, die Accounting Informationen nach dem Lizenzgebrauch anzupassen.
- ASPs können den Benutzern Anwendungen auf Basis von On-Demand-Lizenzen anbieten. Das bedeutet, der Benutzer muss selbst keine Lizenzen erworben haben.

Allgemein liegt jedem Lizenz-Checkout ein Service Level Agreement zugrunde. Dieser basiert auf den darüber liegenden Service Level Agreements. Bei jedem Lizenz-Checkout wird überprüft ob der anfragende User berechtigt ist, solch ein SLA einzugehen.
Somit lässt sich das Produkt elasticLM nicht als einzelnes Konzept sehen. Erst in Kombination mit allen anderen Aspekten von SmartLM kann es als Konzept betrachtet werden.
Die komplette Architektur beruht auf mehreren einzelnen Komponenten, die großteils als Services realisiert sind.

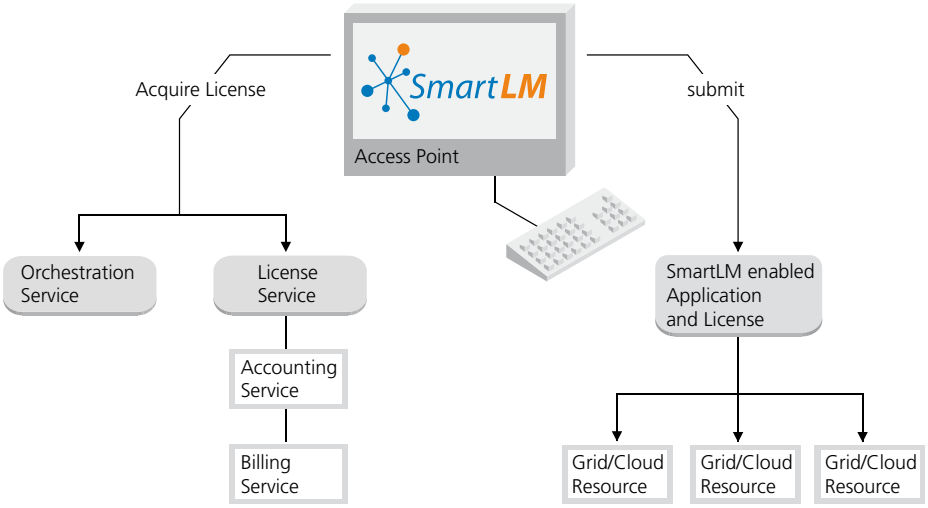

Abbildung 4.13: SmartLM Architektur aus [MZ09]

Diese Grafik beschreibt die allgemeine Kommunikation von SmartLM. SmartLM besitzt einen zentralen Access Point. Der Benutzer kommuniziert mit diesem zur Anfrage nach Lizenzen wie auch zum Submitten von Jobs. Zuerst muss sich der Benutzer gegenüber dem Access Point mittels seinem Zertifikat authentisieren. Ist diese Authentisierung positiv, so wird geprüft, ob der Benutzer für die Lizenznutzung und implizit für das Abschließen eines temporären SLAs autorisiert ist. Wenn dies der Fall ist, so kommuniziert der Access Point mit dem Orchestration Service und dem License Service, um die angefragten Lizenzen zu erhalten und abzurechnen. Nach Erhalt der Lizenzen und Bindung der Eingabedaten an die Lizenzen mittels Hashes wird der Job des Benutzers, also die Anwendung, die mit SmartLM kompatibel ist und die gerade vom License Service erhaltenen Lizenzen, an die jeweiligen Computing Resources übergeben.

SmartLM beschreibt, wie oben bereits erwähnt, das gesamte Framework für ein neues Geschäftsmodell. Das eigentliche Produkt, das für das Lizenzmanagement verantwortlich ist, wird elasticLM genannt, ist aber als Teil des gesamten Konzepts zu sehen.
Die folgenden beiden Bilder klassifizieren zwei verschiedene Möglichkeiten zur Kommunikation zwischen lokalem Server und der remote Computing Site.

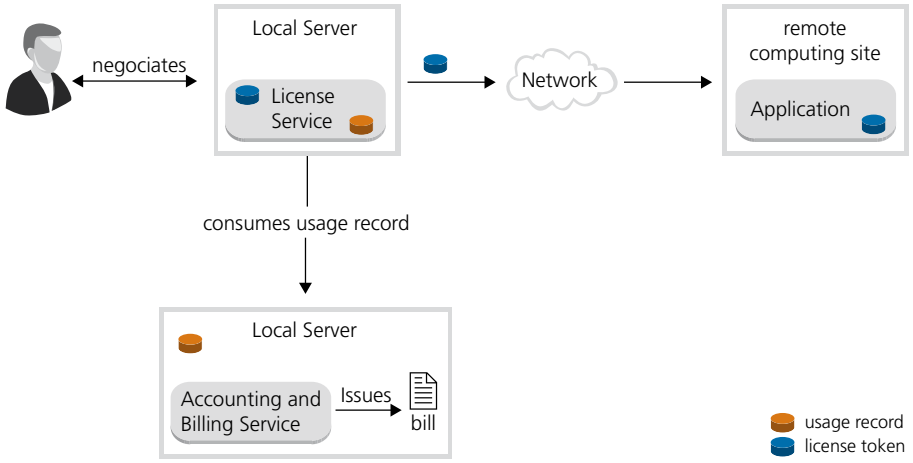

Abbildung 4.14: elasticLM Basic Scenario aus [MZ09]

Das sogenannte Basic Scenario ermöglicht keine bidirektionale Verbindung zwischen lokalem Server und der externen Computing Site. Daraus resultiert, dass keine vertrauliche Instanz von SmartLM dort untergebracht ist. Somit kann keine vertrauliche Rückmeldung erfolgen.

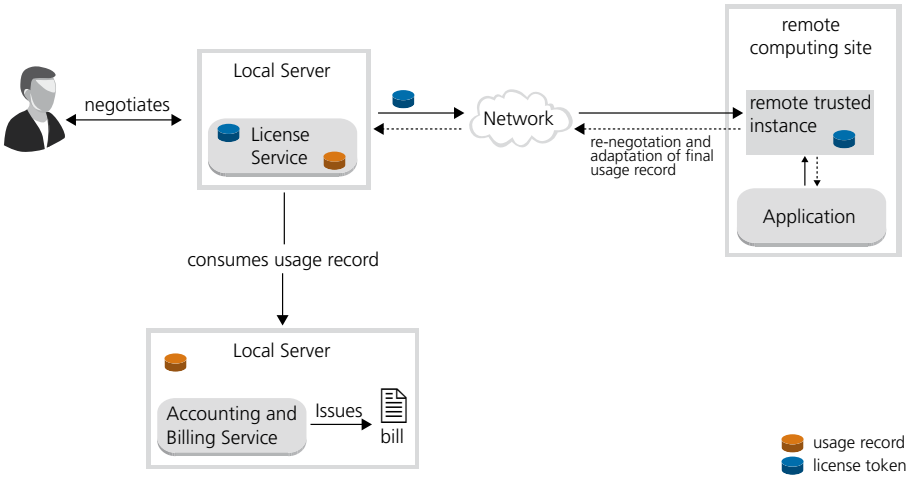

Abbildung 4.15: elasticLM Advanced Scenario aus [MZ09]

In manchen Fällen ist jedoch eine bidirektionale Verbindung möglich, da eine vertrauliche Instanz auf Seite der Computing Site vorhanden ist. elasticLM spricht dabei dann vom Advanced Scenario. In diesem Szenario können die Re-Negotiations der SLAs, auf denen die Lizenzen basieren, erfolgen. Außerdem ermöglicht diese bidirektionale Kommunikation, dass Lizenzen auch wieder vorzeitig abgegeben werden können. Die endgültige Abrechnung kann somit auch erst nach der eigentlichen Job-Durchführung stattfinden.

4.3.4 Skalierung und praktische Erfahrung

Zum Zeitpunkt der Diplomarbeit war der Prototyp von SmartLM noch nicht fertiggestellt. Laut [IT10] ist dieser jedoch noch für das Frühjahr 2010 geplant. Daher sind auch noch keine Informationen beziehungsweise Dokumentationen über den praktischen Einsatz des SmartLM-Ansatzes verfügbar. Aus Sicht dieser Diplomarbeit handelt sich also bis jetzt „nur“ um ein theoretisches Lizenzmodell, das sich in seiner praktischen Durchsetzbarkeit und insbesondere Schnelligkeit unter großer Last und möglicher Skalierbarkeit erst noch beweisen muss.

4.3.5 Sicherheitsbetrachtungen von SmartLM

Laut Aussage von [MZ09] setzt SmartLM auf den derzeitigen State-of-the-Art-Sicherheitstechniken auf und benutzt unter anderem X.509-Zertifikate sowie XACML. Diese Techniken werden für die folgenden sicherheitsrelevanten Punkte eingesetzt:

- Authentisierung und Autorisierung der Benutzer, Dienste und Server.
- Sichere und vertrauliche Kommunikation zwischen den Komponenten.
- Sicherheit beim Delegations-Prozess bei Benutzung eines Portals oder Orchestrators.
- Offenlegung von wichtigen Informationen wie zum Beispiel kompromittierte Lizenzen.
- Sicherstellung der Integrität des gesamten Prozesses, damit eine Lizenzzuteilung nicht geleugnet werden kann.
- Sicherung des Lizenzierungsmechanismuses und die daran beteiligten Komponenten wie zum Beispiel Lizenzgeneratoren.
- Schutz der ausführbaren Dateien, die bei der Lizenzvergabe beteiligt sind.
- Sicherstellung, dass Lizenzen nicht unter Vorgabe falscher Uhrzeiten oder Dati angefragt werden.

4.3.6 Evaluierung gegen den Anforderungskatalog

In diesem Abschnitt wird das Modell von SmartLM gegen die Anforderungen evaluiert.

Allgemein: Unabhängigkeit von der eingesetzten Middleware (siehe 3.3.1)

Laut [IT09] soll der Ansatz von SmartLM in allen „major Grid middleware“-Umgebungen nach den jeweiligen spezifischen Anpassungen zur Umsetzung des gesamten Geschäftsmodells einsatzbereit sein.

Allgemein: Nutzung Grid-spezifischer Technologien (siehe 3.3.2)

SmartLM basiert großteils auf Services und setzt somit grundsätzlich auf Standard-Grid-Technologien auf.

Allgemein: Priorisierung von Jobs (siehe 3.3.3)

Es sind keine konkreten Informationen verfügbar, ob SmartLM beziehungsweise elasticLM Priorisierungsmöglichkeiten bietet. Im Zuge der SLAs lässt sich dies jedoch vermuten.

Allgemein: Transparenz der Lizenzzuteilung (siehe 3.3.4)

Der Ansatz von SmartLM ist bekanntlich nicht nur ein Lizenzmodell, sondern ein gesamtes Geschäftsmodell. Dieses Geschäftsmodell ist so umfangreich und ändert viele de-facto-Standards, so dass die Nutzung des Grids „anders“ ist als früher. Ist jedoch der Wechsel in das neue Geschäftsmodell und Betriebsmodell vollzogen, so erfolgt die Lizenzzuteilung ansich transparent.

Sicherheit: Firewallkonfiguration (siehe 3.4.1)

SmartLM und das dazugehörige elasticLM sind zusammen ein komplett neuer Ansatz und ein neues Geschäftsmodell. Firewalls und Firmen-Policies müssen zwangsweise an die Funktionalität und Kommunikation angepasst werden.

Sicherheit: Zuverlässigkeit in unzuverlässigen Netzwerken (siehe 3.4.2)

SmartLM beruht auf allen gängigen Vorkehrungen, die für diesen Einsatzzweck notwendig sind.

Sicherheit: Verzahnung mit den Grid-spezifischen Sicherheitsmechanismen zur Authentisierung (siehe 3.4.3)

Der Ansatz von SmartLM beruht zur Authentisierung auf X.509-Zertifikaten laut [Zie08]. Wenn keine Zertifizierungs-Infrastruktur vorhanden ist, kann die Authentisierung auch mittels userID und Passwort stattfinden. Außerdem wird auch ein Shibboleth-ähnlicher Ansatz angeboten. Verschlüsselung ist somit durchsetzbar.
Der Client benutzt X.509-Zertifikate zur Authentisierung. Wie der Server sich gegenüber dem Client authentisiert ist nicht öffentlich dokumentiert. Ob eine Verschlüsselung des Datentransfers mit dem jeweiligen Public-Key des Empfängers zur Sicherstellung der Vertraulichkeit stattfindet, ist ebenfalls nicht öffentlich dokumentiert, jedoch ist davon auszugehen.

Sicherheit: Multiple Nutzung von Lizenzen unterbinden (siehe 3.4.4)

Das SmartLM-Produkt elasticLM unterbindet die multiple Nutzung mittels aktueller State-of-the-Art Sicherheitsmechanismen laut [MZ09]. Unter anderem werden die Lizenzen ähnlich zum Ansatz von GenLM an die Daten gebunden.

Lizenzierung: Lizenzpools nach administrativen Domänen getrennt (siehe 3.5.1)

SmartLM ist ein sehr breit gefächertes Geschäftsmodell, das für die unterschiedlichsten Einsätze gedacht ist. Daraus resultierend kann der Lizenz-Server bei verschiedenen Instanzen untergebracht sein. Im Falle des Hostings bei einem ASP muss dieser selbstverständlich die Lizenzpools nach seinen Kunden beziehungsweise Organisationen trennen, damit eine Organisation nicht die Lizenzen einer anderen Organisation nutzen kann.

Lizenzierung: Einsatz verschiedener Lizenztypen (siehe 3.5.2)

Auf der Tatsache beruhend, dass SmartLM Lizenzverträge auf Basis von Service Level Agreements darstellt, sind im Prinzip viele Möglichkeiten von Lizenztypen denkbar. Prinzipiell spricht SmartLM von Lizenzen auf Basis von Pay-per-Use.

Lizenzierung: Auswahl der Lizenzen möglich (siehe 3.5.3)

In SmartLM erfolgt die Auswahl der Lizenzen durchwegs transparent. Das bedeutet, dass der Benutzer keinen Einfluss darauf hat, welche Lizenzen verwendet werden. Dem User wird jedoch die Möglichkeit geboten, zu entscheiden, wann und wie er die Lizenzen beantragen möchte. So wird laut [MZ09] die Möglichkeit geboten, Lizenzen über ein Portal, per Grid Scheduler, direkt mittels der Anwendung oder über ein Command Line Interface zu beziehen.

Lizenzierung: Hierarchische Vergabe von Lizenznutzungsrechten (siehe 3.5.4)

Dieses Feature wird nicht konkret angesprochen. Da SmartLM jedoch ein gesamtes Geschäftsmodell darstellt, betrifft die Delegation von Rechten, im Gegensatz zum Ansatz von GenLM oder BEinGRID, direkt den Ansatz von SmartLM. Eine Umsetzung dieses Features ist somit denkbar, jedoch nicht explizit erwähnt.

Lizenzierung: Blacklists und Whitelists (siehe 3.5.5)

SmartLM bietet laut [MZ09] die Möglichkeit, lokale Policies zur Regulierung der Lizenznutzung zu definieren. Diese Policies bieten die Möglichkeit, Blacklists und Whitelists auf verschiedenen Ebenen oder aber auch deutlich komplexere Konstrukte zu definieren.

Lizenzierung: Lizenzreservierung mit Frist (siehe 3.5.6)

SmartLM bietet einen komplexen Reservierungsmechanismus. So ist es möglich, Lizenzen für eine bestimmte Zeitspanne zu reservieren, aber auch zum Beispiel Lizenzen kombiniert mit den Computing Resources zu reservieren.

Lizenzierung: Pausieren und Fortsetzen der Lizenznutzung (siehe 3.5.7)

Dieses Feature ist zwar nicht direkt Bestandteil des SmartLM-Ansatzes, jedoch lässt es sich theoretisch über Umwege erreichen: Mittels Verlängerung der reservierten, gerade in Benutzung befindlichen Lizenzen und Re-Negotiation des zugrunde liegenden SLAs könnte dieses Feature erreicht werden. Praktisch gesehen ist es jedoch nicht vorhanden.

Lizenzierung: Umfassendes Logging der angeforderten, zugeteilten und reservierten Lizenzen (siehe 3.5.8)

Der gesamte Ansatz von SmartLM bietet umfangreiches Logging an. Da SmartLM das gesamte Geschäftsmodell darstellt, muss es auch Logging der Lizenzen direkt anbieten, sonst wäre es nicht vollständig. Gerade da SmartLM sich als Modell gegenüber ASPs, ISVs und RPs gut verkaufen lassen muss, muss eine solch grundlegende Funktionalität der Vollständigkeit halber enthalten sein.

Lizenzierung: Änderungen am Lizenzpool und den Listen ohne Neustart (siehe 3.5.9)

Bezüglich diesem Feature sind keine konkreten Informationen verfügbar. Jedoch ist auf Grund der Tatsache, dass die Lizenzbedingungen jederzeit live geändert werden können, die Vermutung berechtigt, dass Änderungen am Lizenzpool ebenfalls on-the-fly durchgeführt werden können.

Lizenzierung: Lizenzserver unabhängig der eingesetzten Plattform (siehe 3.5.10)

Als Geschäftsmodell sollte SmartLM zumindest den Einsatz auf den gängigsten Plattformen wie Linux und Windows unterstützen. Ob dies jedoch tatsächlich der Fall ist, lässt sich aus den öffentlich zugänglichen Beschreibungen nicht explizit herausfinden.

Lizenz: Lizenz als feste Dateistruktur (siehe 3.6.1)

Eine Lizenz ist im Umfeld von SmartLM eine sehr komplexe Struktur, die sogar stetigem Wandel mittels Re-Negotiation unterworfen sein kann. Jedoch basieren die Lizenzen alle auf einer festen Struktur, da SmartLM nur SmartLM-Lizenzen zulässt, also speziell für SmartLM angepasste Lizenzen. Eine Anwendung muss also auch speziell für SmartLM angepasst werden.

Lizenz: Mobilität der Lizenzen
(siehe 3.6.2)

SmartLM-Lizenzen beruhen auf License-Tokens, die auf SLAs basieren. Diese Tokens sind unabhängig von der generierenden Instanz (aka Lizenz-Server) im gesamten Geschäftsumfeld einsetzbar. Es handelt sich also um mobile Lizenzen. Die SLAs und Lizenzverträge müssen im Sinne von SmartLM an die Mobilität der Lizenzen angepasst werden.

Job: Lizenzen verlängern oder früher aufgeben
(siehe 3.7.1)

Dieses Feature ist in SmartLM laut [MZ09] umfangreich mittels Re-Negotiation zur Laufzeit umgesetzt. So ist es möglich, Lizenzen frühzeitig wieder abzugeben beziehungsweise Lizenzen je nach Bedarf zu verlängern oder sogar zu erweitern.

Job: Jobs beenden, die innerhalb Timeouts keine Lizenzen erhalten haben
(siehe 3.7.2)

Über dieses Feature ist keine konkrete Information verfügbar. SmartLM präferiert ein Abrechungsverfahren auf Basis von Pay-per-Use und insbesondere On-Demand. Somit sind im Prinzip immer „genug“ Lizenzen verfügbar. Limitieren könnte jedoch eine gesetzte Budget-Grenze zur Vermeidung einer Kostenexplosion, was zu einem Abbruch des Jobs führt. Was in diesem Fall konkret passiert ist jedoch nicht genauer aus den veröffentlichten Informationen ermittelbar.

Job: Bekanntgabe, woher genutzte Lizenzen kommen
(siehe 3.7.3)

Im Sinne des komplexen und umfangreichen Accounting- und Logging-Verfahren und insbesondere dem Single Point für das Management der Lizenzen nach [MZ09], sind umfangreiche Informationen bezüglich der Lizenzen verfügbar.

Job: Überwachung der Lizenzen
(siehe 3.7.4)

Da SmartLM als Geschäftsmodell alle Informationen sammeln muss, ist ein umfangreiches und komplexes Accounting- und Logging-Verfahren verfügbar. Insbesondere mittels dem Single Point für das Management der Lizenzen nach [MZ09] sind umfangreiche Informationen bezüglich der Lizenzen verfügbar.

Job: Überprüfen der Legalität der Lizenzen
(siehe 3.7.5)

Lizenzen werden ähnlich wie im Ansatz von GenLM an die Daten gebunden und mittels State-of-the-Art Sicherheitsmechanismen verifiziert. Daher gilt für den SmartLM-Ansatz dieselbe Betrachtung wie für den GenLM-Ansatz. Darüber hinaus wird auch der Code der validierenden Instanzen mittels XACML ([OAS10b]) geschützt und laut [MZ09] Maßnahmen zur Sicherstellung der korrekten Uhrzeit und somit Gültigkeit einer Lizenz vorgenommen.

Accounting: Anzeige der erhaltenen Lizenzen (siehe 3.8.1)

Der Benutzer hat jederzeit Einsicht in seine gerade benutzten beziehungsweise reservierten Lizenzen und kann diese auch im Sinne der Re-Negotiation jederzeit ändern, verlängern, erweitern oder stornieren.

Accounting: Sicheres Accounting und Abrechnungsverfahren (siehe 3.8.2)

Im Sinne des umfangreichen Accountings und Billings, das SmartLM bietet, ist dieses laut [MZ09] auch mit den klassischen Sicherheitsmechanismen gegen Fälschungen geschützt.

Accounting: Schutz vor Leugnung des Erhalts einer Lizenz (siehe 3.8.3)

Der Ansatz von SmartLM beinhaltet klassische Sicherheitsmaßnahmen zur Sicherstellung, dass der Erhalt beziehungsweise die Benutzung von Lizenzen nicht geleugnet werden kann.

Accounting: Redundantes Abspeichern aller relevanten Informationen (siehe 3.8.4)

SmartLM bietet keine konkreten Informationen über dieses Feature. Es lässt sich jedoch mutmaßen, dass alle relevanten Informationen redundant erfasst werden, da SmartLM sonst nicht als vollständiges Geschäftsmodell auftreten könnte, da gefälschte oder verlorengegangene Abrechnungsinformationen das wirtschaftliche Aus eines Unternehmens bedeuten können.

Accounting: Statistische Auswertungen ermöglichen (siehe 3.8.5)

Im Zuge des umfangreichen Accountings und Billings werden die verschiedensten Parameter und Usage-Statistiken geloggt und für eine Auswertung zur Verfügung gestellt.

Accounting: Abrechnung quasi Echtzeit durchführen (siehe 3.8.6)

Inwieweit die Abrechnung in Echtzeit bei SmartLM durchgeführt werden kann, ist nicht klar dokumentiert.

Accounting: Delay der Lizenzierung berücksichtigen (siehe 3.8.7)

Inwieweit und ob überhaupt diese Verzögerung berücksichtigt wird, ist nicht klar dokumentiert. Es wird auch nicht dokumentiert wie schnell auf eine Re-Negotiation der SLAs der Lizenzen eingegangen werden kann, wie lange eine solche Bearbeitung dauert und wie diese Verzögerungen in die Abrechnung mit einfließen.

Accounting: Accounting mit verschiedenen Granularitätsstufen (siehe 3.8.8)

Accounting und Billing ist ein wichtiger Bestandteil des gesamten Geschäftsmodells von SmartLM. Inwieweit das Accounting jedoch konfiguriert werden kann und ob folglich das Accounting auf verschiedenen Granularitätsstufen stattfinden kann, ist jedoch nicht konkret dokumentiert.

Accounting: Kostenexplosionen vermeiden (siehe 3.8.9)

Budget-Begrenzungen können beim Einsatz von SmartLM gesetzt werden und werden bei jeder Anforderung nach einer Lizenz geprüft und durchgesetzt.

Accounting: Einsicht jederzeit möglich (siehe 3.8.10)

Nach [MZ09] bietet SmartLM die Möglichkeit jederzeit up-to-date Informationen über die Lizenzen und zugehörige Informationen.

Accounting: Einbeziehung der Firmen-Policies (siehe 3.9.1)

Firmen-Policies können jederzeit zur Definition der Lizenznutzungsrechte mit einbezogen werden. Jedoch wird von SmartLM vorausgesetzt, dass die Firmenpolitiken kompatibel mit dem gesamten Geschäftsmodell sind und nicht die Nutzung von elasticLM oder andere Teile von SmartLM ausschließen oder unmöglich machen.

Recht: Lizenzvertrag berücksichtigen (siehe 3.9.2)

SmartLM setzt komplett neue Lizenzen ein. Folglich sind auch alle Lizenzverträge Verträge, die auf SmartLM und das zugehörige Geschäftsmodell zugeschnitten sind. Eine Berücksichtigung von bereits bestehenden Verträgen findet daher nicht statt, da diese Verträge alle abgeändert werden müssten. SmartLM setzt diese neuen Verträge voraus, damit alle beteiligten Instanzen an das Geschäftsmodell angepasst werden können.

Recht: Erweiterung des Lizenzvertrags bezüglich Nutzung der Lizenzen in Grids (siehe 3.9.3)

Bestehende Verträge werden nicht erweitert, sondern komplett neu, an SmartLM angepasst, erstellt. Diese Verträge müssen dann selbstverständlich die Nutzung der Anwendung, auf die sich die Verträge beziehen, in Grids erlauben.

Recht: Rechtliche Konsequenzen bei Ausfall oder Nichtverfügbarkeit des Lizenzservers (siehe 3.9.4)

Alle Verträge von SmartLM basieren auf Service Level Agreements. Diese SLAs definieren immer rechtliche Konsequenzen bei Nichteinhalten bestimmter zugesagter Features oder Dienstgüten. Die rechtlichen Konsequenzen werden also in diesen SLAs definiert.

Recht: Überprüfung der korrekten Lizenznutzung (siehe 3.9.5)

SmartLM beinhaltet eine umfangreiche Untersuchung der Validität einer Lizenz. So wird jede Lizenzanfrage und jede Lizenz mittels X.509-Zertifikaten signiert. Außerdem wird eine Lizenz an Eingabedaten gebunden. Um welche Daten es sich jedoch konkret handelt kann auf Grund des One-Way-Hashes nicht festgestellt werden. Da jedoch jede Lizenz einem SLA unterliegt, können zumindest in diesem SLA rechtliche Konsequenzen bei Falschbenutzung festgehalten werden, wenn diese mittels anderweitiger Methoden erwiesen ist.

4.3.7 Zusammenfassung von SmartLM

Der Ansatz von SmartLM ist nicht nur ein Lizenzmodell sondern ein umfangreiches Modell für Geschäftslösungen. SmartLM stellt dabei hohe Anforderungen sowohl an den ISV als auch die Grid-Umgebung. Um das Lizenzmodell elasticLM einzusetzen, muss das gesamte Umfeld an das SmartLM-Umfeld angepasst werden. So müssen alle Lizenzverträge zur Benutzung innerhalb einer SmartLM-Umgebung abgeändert, sowie alle davon betroffenen Anwendungen an SmartLM angepasst werden. Klassische Lizenzen und Lizenzschema können nicht länger eingesetzt werden. SmartLM möchte das Lizenz-Management nicht nur evolutionieren, sondern mit einem kompletten neuen Geschäftsmodell revolutioneren. Es stellt sich daher berechtigterweise die Frage, ob Unternehmen beziehungsweise ISVs ihr komplettes Umfeld umstrukturieren wollen, nur um eine Lizenzierung der Software in Grid-Umgebungen zu erreichen.
Jedoch bietet SmartLM ein sehr umfangreiches und relativ vollständiges Lizenzmodell innerhalb des gesamten Geschäftsmodells. Dieses durchdachte Gesamtkonzept könnte durchaus eine Revolution für Grid-Umgebungen darstellen.

4.4 Tabellarische Übersicht

In diesem Abschnitt werden die drei vorgestellten Modelle nochmals tabellarisch gegenübergestellt. Dabei wird zur Bewertung das Maß der deutschen Schulnoten (1 = „sehr gut“ bis 5 = „mangelhaft“) herangezogen. Ein nicht vorhandenes Feature wird mit einem „-“ gekennzeichnet. Wenn keine Informationen verfügbar sind, wird ein „?“ gesetzt. Die erste Spalte verweist auf den Abschnitt, in dem die Anforderung beschrieben wird.

Abs.	Kategorie	Titel	GenLM	BEinG.	S.LM
3.3.1	Allgemein	Unabhängigkeit von der eingesetzten Middleware	?	3	1
3.3.2	Allgemein	Nutzung Grid-spezifischer Technologien	-	4	1
3.3.3	Allgemein	Priorisierung von Jobs	-	-	1
3.3.4	Allgemein	Transparenz der Lizenzzuteilung	2	3	1
3.4.1	Sicherheit	Firewallkonfiguration	1	1	1
3.4.2	Sicherheit	Zuverlässigkeit in unzuverlässigen Netzwerken	2	1	1
3.4.3	Sicherheit	Verzahnung mit den Grid-spezifischen Sicherheitsmechanismen zur Authentisierung	2	2	2
3.4.4	Sicherheit	Multiple Nutzung von Lizenzen unterbinden	1	1	1
3.5.1	Lizenzierung	Lizenzpools nach administrativen Domänen getrennt	1	1	1
3.5.2	Lizenzierung	Einsatz verschiedener Lizenztypen	4	1	3
3.5.3	Lizenzierung	Auswahl der Lizenzen möglich	-	-	-
3.5.4	Lizenzierung	Hierarchische Vergabe von Lizenznutzungsrechten	-	-	?
3.5.5	Lizenzierung	Blacklists und Whitelists	-	2	1
3.5.6	Lizenzierung	Lizenzreservierung mit Frist	-	-	1
3.5.7	Lizenzierung	Pausieren und Fortsetzen der Lizenznutzung	-	-	4
3.5.8	Lizenzierung	Umfassendes Logging der angeforderten, zugeteilten und reservierten Lizenzen	4	1	1
3.5.9	Lizenzierung	Änderungen am Lizenzpool und den Listen ohne Neustart	?	?	1
3.5.10	Lizenzierung	Lizenzserver unabhängig der eingesetzten Plattform	2	-	2
3.6.1	Lizenz	Lizenz als feste Dateistruktur	1	-	1
3.6.2	Lizenz	Mobilität der Lizenzen	2	1	1

Abs.	Kategorie	Titel	GenLM	BEinG.	S.LM
3.7.1	Job	Lizenzen verlängern oder früher aufgeben	-	-	2
3.7.2	Job	Jobs beenden, die innerhalb Timeouts keine Lizenzen erhalten haben	-	-	-
3.7.3	Job	Bekanntgabe, woher genutzte Lizenzen kommen	-	2	1
3.7.4	Job	Überwachung der Lizenzen	-	1	1
3.7.5	Job	Überprüfen der Legalität der Lizenzen	2	2	1
3.8.1	Accounting	Anzeige der erhaltenen Lizenzen	1	1	1
3.8.2	Accounting	Sicheres Accounting und Abrechnungsverfahren	1	1	1
3.8.3	Accounting	Schutz vor Leugnung des Erhalts einer Lizenz	1	2	1
3.8.4	Accounting	Redundantes Abspeichern aller relevanten Informationen	-	?	?
3.8.5	Accounting	Statistische Auswertungen ermöglichen	4	?	1
3.8.6	Accounting	Abrechnung quasi Echtzeit durchführen	1	2	2
3.8.7	Accounting	Delay der Lizenzierung berücksichtigen	-	-	-
3.8.8	Accounting	Accounting mit verschiedenen Granularitätsstufen	4	-	1
3.8.9	Accounting	Kostenexplosionen vermeiden	4	2	1
3.8.10	Accounting	Einsicht jederzeit möglich	-	1	1
3.9.1	Recht	Einbeziehung der Firmen-Policies	-	-	1
3.9.2	Recht	Lizenzvertrag berücksichtigen	-	-	-
3.9.3	Recht	Erweiterung des Lizenzvertrags bezüglich Nutzung der Lizenzen in Grids	1	1	1
3.9.4	Recht	Rechtliche Konsequenzen bei Ausfall oder Nichtverfügbarkeit des Lizenzservers	-	-	1
3.9.5	Recht	Überprüfung der korrekten Lizenznutzung	3	-	2

4.5 Zusammenfassung des 4. Kapitels

In diesem Kapitel wurden drei Lizenzmodelle für Grid-Umgebungen vorgestellt und näher betrachtet. Diese drei ausgewählten Ansätze waren zum Zeitpunkt der Entstehung der Diplomarbeit relativ weit fortgeschritten und ausgearbeitet. Trotzdem darf nicht außer Acht gelassen werden, dass noch weitere Ansätze derzeit in Entwicklung oder Konzeption sind, die jedoch meist sehr spezifisch ausfallen und daher kaum als allgemeiner Ansatz betrachtet werden können.
Alle drei vorgestellten Ansätze sind grundlegend verschieden voneinander und zeigen daher auf, dass es viele Möglichkeiten gibt, um eine Lizenzierung in Grid-Umgebungen durchzuführen.
Im ersten Abschnitt dieses Kapitels wurde GenLM als Ansatz betrachtet und dabei gezeigt, dass GenLM insbesondere Wert legt auf die korrekte Benutzung von Forschungslizenzen und kommerziellen Lizenzen sowie auf die sichere Kommunikation.
Der Ansatz von GenLM hat die geringste Komplexität, ist aber dennoch meist ausreichend in seiner Funktionalität. Darauf basierend, dass Lizenzen umfassend bei Checkout sowie Nutzung validiert werden, erfüllt das Modell bereits die wichtigsten relevanten Features für den Einsatz in Grid-Umgebungen. Für den Einsatz bei Resource Providern beziehungsweise Application Service Providern ist das Modell GenLM jedoch nur mittels einem Fallback auf Proxy-Tunnelung nach dem Ansatz von BEinGRID einsetzbar.
Der Ansatz von GenLM befindet sich laut [DP09] im Status „Patent Pending“ und ist folglich derzeit technisch absolut geschlossen und unzugänglich. [DP09] nennt als Release-Datum 2009, jedoch sind diesbezüglich keine aktuellen Informationen verfügbar.
Der zweite Abschnitt befasste sich mit dem Ansatz von BEinGRID. Dieser Ansatz setzt vor allem auf die Tunnelung der Lizenzanfragen mittels einem Proxy sowie den Einsatz von FLEXnet als Lizenzserverstruktur. Das grundlegende Konzept von BEinGRID beruht somit darauf, non-Grid-Applikationen zu gridifizieren und bestehende Lizenzmodelle innerhalb Grid-Umgebungen einsetzbar zu machen.
Der Ansatz von BEinGRID ist laut [Sim] Open-Source und wird unter der GPL ([GNU10]) veröffentlicht. Der komplette Ansatz ist bereits relativ weit fortgeschritten und wird in den Business Experiments von BEinGRID teilweise schon eingesetzt. Auch eine konkrete Veröffentlichung unter der GPL wird angestrebt.
Im letzten Abschnitt wurde die Lösung von SmartLM vorgestellt. SmartLM ist kein alleiniges Lizenzmodell, sondern vielmehr ein hochkomplexes Geschäftsmodell, das den gesamten Softwarelizenzierungsvorgang neu organisieren will. SmartLM setzt zur Aushandlung von Lizenzrechten sowie Autorisierung der Benutzer auf WS-Agreement, einem Web Services Standard. Problematisch ist insbesondere die Voraussetzung zum Einsatz von SmartLM, dass alles für SmartLM angepasst werden muss, sogar die einzusetzende Software.
SmartLM ist komplex und möchte allumfassend sein. Es ist nicht nur ein Lizenzmodell, sondern ein gesamtes Geschäftsmodell. Wenn man SmartLM als Lizenzschema einsetzen möchte, so muss man seine gesamte Umgebung daran anpassen. Lizenzverträge sind keine klassischen Lizenzverträge mehr, sondern basieren auf Service Level Agreements und sind hochdynamisch und nicht mehr länger quasi-statisch. Das Konzepz von SmartLM ist extrem umfangreich und zwingt alle beteiligten Instanzen zu umfangreichen Änderungen in ihrem bisherigen Geschäftsmodell. Eine Anwendung, die mittels SmartLM lizenziert werden soll, muss damit kompatibel sein beziehungsweise gemacht werden. Auch müssen ASPs beziehungsweise Resource Provider ihre Verwaltungssoftware zu SmartLM kompatibel machen,

damit das gesamte SmartLM umgesetzt werden kann. SmartLM möchte also nicht nur eine Evolution bekannter Lizenzschemata sondern eine komplette Revolution.
SmartLM befindet sich laut [inS08] bereits seit Februar 2008 in Konzeption und Entwicklung mit einer geplanten Dauer von 30 Monaten. Im Dokument [IT09] wird beschrieben, dass ein erster Prototyp im Frühjahr 2010 fertig sein wird.
Alle drei vorgestellten Ansätze befinden sich zum Zeitpunkt der Entwicklung dieser Diplomarbeit noch nicht in einem marktreifen Zustand.

5 Entwicklung eines Lizenzschemas

In den vorangegangenen Kapiteln wurde aufgezeigt, warum und inwieweit bisher entwickelte Lizenzansätze für Grid-Umgebungen nicht allen Anforderungen aus dem Anforderungskatalog genügen. Es hat sich dabei herausgestellt, dass es jedoch grundlegend verschiedene konzeptionelle Ansätze gibt. Jedes dieser Modelle zielt auf andere Gesichtspunkte ab. Daher sind die Ansätze auch nicht direkt miteinander vergleichbar.
In diesem Kapitel wird nun ein Lizenzkonzept vorgestellt, das im Rahmen dieser Diplomarbeit entworfen wurde. Es deckt den Anforderungskatalog bestmöglich ab. Das Lizenzmodell ist als Definition eines gesamten Konzepts zu sehen und dient nicht dazu, bereits bestehende Schutz- und Lizenzierungsmethoden zu „gridifizieren".
Während der Konzeption des Lizenzmodells wurde besonderen Wert darauf gelegt, dass das Modell generisch genug für den allgemeinen Einsatz innerhalb Grid-Umgebungen von Virtuellen Organisationen geeignet ist. Außerdem soll das Modell auf Standard-Technologien zurückgreifen, die in jeder Grid-Umgebung anzutreffen sind. Dies betrifft insbesondere die allgemeine Benutzer-Authentisierung und -Autorisierung sowie den Einsatz von Web Services.

5.1 Beschreibung des Lizenzmodells

Dieser Abschnitt befasst sich mit dem grundsätzlichen Ablauf des Lizenzmodells. Das Lizenzmodell soll die Möglichkeit geben, eine kommerzielle Anwendung innerhalb Grid-Umgebungen auszuführen. Das Grid, das in dieser Diplomarbeit betrachtet wird, gehört einer Virtuellen Organisation.
Zunächst wird ein Überblick über das gesamte Modell gegeben und einige grundsätzliche Zusammenhänge dargelegt. Anschließend wird das Modell in seine logischen Bestandteile aufgeteilt und die Abläufe dieser Teile erklärt. Dabei ist zu beachten, dass noch nicht auf die technischen Gegebenheiten und Feinheiten eingegangen wird. Diese werden erst anschließend bei der genauen Definition der einzelnen Komponenten und Services genau erläutert.
Bei der Erklärung der einzelnen Abläufe wird daher häufig auf spätere Abschnitte dieses Kapitels verwiesen.

5.1.1 Darstellung des Konzepts

Dieser Abschnitt stellt das Lizenzmodell in seiner Gesamtheit dar. Dabei ist ersichtlich, welche Entitäten zum Informationsaustausch miteinander in Kontakt stehen.

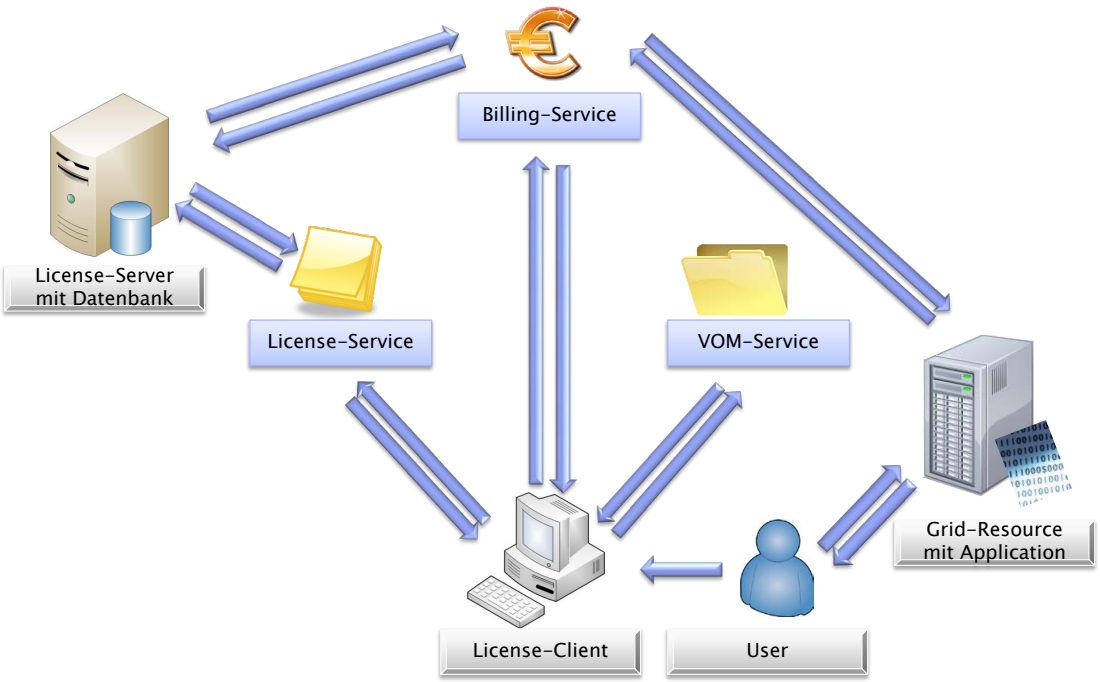

Abbildung 5.1: License-Modell: Gesamtübersicht

Im Folgenden werden die Grundzüge des Konzepts genannt:

- Das Lizenzkonzept beruht auf einer Kombination von Web Services in Verbindung mit einem *License-Client* und einem *License-Server*, der als WS-Resource implementiert ist.
- Die Authentisierung erfolgt grundsätzlich mittels X.509-Zertifikaten.
- Lizenzen sind Floating-Lizenzen oder Pay-per-Use-Lizenzen.
- Die Berechtigung, eine Lizenz zu beantragen ist in der VOMS-Datenbank abgespeichert.
- Die Berechtigung, eine Lizenz zu nutzen, entscheidet der *License-Server* anhand seiner Kundendatenbank.
- Es gibt einen *License-Client*, der den Benutzer vertritt.
- Der *License-Client* dient zur Kommunikation mit den Web Services.
- Der *License-Service* nimmt ausschließlich Anfragen vom *License-Client* entgehen.

- Der *License-Server* nimmt Anfragen sowohl vom *License-Service* als auch vom *Billing-Service* entgegen.
- Die *Application*, die auf den Grid-Ressourcen ausgeführt werden soll, prüft ihre zugeteilte Lizenz mittels verschiedenen Mechanismen.
- Der *License-Server* dient für das Zuteilen und Verwalten von Lizenzen. Er prüft außerdem, ob die anfragende Person Lizenzen besitzt. Zusätzlich ist er für das Logging zuständig.
- Der *License-Client*, der *License-Server* und die *Application* müssen mit gängigen Mechanismen wie zum Beispiel Obfuscation und Verschlüsselungen gegen Manipulationen geschützt sein.
- Der *Billing-Service* nimmt Anfragen vom *License-Client* sowie von der *Application* entgegen.
- *License-Service*, *Billing-Service* und *License-Server* sind WS-Resources. Sie bestehen also aus einem Web Service in Verbindung mit Java-Klassen.
- *License-Service* und *Billing-Service* können in der Grid-Umgebung mehrfach auftreten. Damit kann ein Pfad bereitgestellt werden.
- Der Virtual Organisation Membership Service dient zur Verwaltung der Benutzerberechtigungen.

5.1.2 Voraussetzungen für den Einsatz des Lizenzschemas

Das Lizenzschema benötigt einige Voraussetzungen, damit es eingesetzt werden kann. Im Folgenden werden diese Dependencies dargestellt:

- Eine Grid-Umgebung ist obligatorisch.
- Eine zur Middleware kompatible Java-Version muss installiert sein (siehe [Sun10a]).
- Benutzer müssen über Zertifikate verfügen.
- VOMS muss eingerichtet und konfiguriert sein.
- Alle Benutzer, die Lizenznutzungsrechte haben sollen, müssen in VOMS eingebunden sein und für die Nutzung autorisiert werden.
- Die Services müssen in die Grid-Umgebung deployed werden.
- Der Benutzer muss Zugriff auf einen *License-Client* haben.
- Der *License-Server* muss installiert, konfiguriert und als WS-Resource gestartet werden.
- Es muss ein Lizenzvertrag auf Basis des Lizenzmodells existieren.

5.1.3 Aufbau der Anwendung

Sollte sich ein ISV beziehungsweise Distributor dafür entscheiden, eine Anwendung mittels diesem Lizenzschema zu lizenzieren, so wird diese Anwendung, wie bei jedem Lizenzschema üblich, mit dem Lizenzschema verschmolzen.
Dieses Lizenzschema ist somit eine Neudefinition und dient nicht zur Verwendung bereits bestehender mit bestimmten Mitteln geschützter Software, sondern ist vielmehr als Definition eines gesamten Konzepts zu sehen.

5.2 Beschreibung des Ablaufes

In diesem Abschnitt wird erläutert, wie die gesamte Lizenzierung grundsätzlich abläuft. Dabei wird dargelegt, welche Komponente welche Aufgabe übernimmt und wie die Kommunikation und der Informationsaustausch untereinander stattfindet. Die Aufgaben werden konkret bei der Beschreibung der Komponenten erläutert.

5.2.1 Authentisierung des Benutzers

Zuallererst muss sich der Benutzer selbstverständlich gegenüber dem Grid authentisieren. Dazu bieten sich je nach Umgebung verschiedene Mechanismen an. In der Regel hat ein Benutzer ein Zertifikat, das von einer Certificate Authority signiert und somit geprüft ist. Der Einsatz von VOMS bietet sich an.

5.2.2 Beantragung einer Lizenz

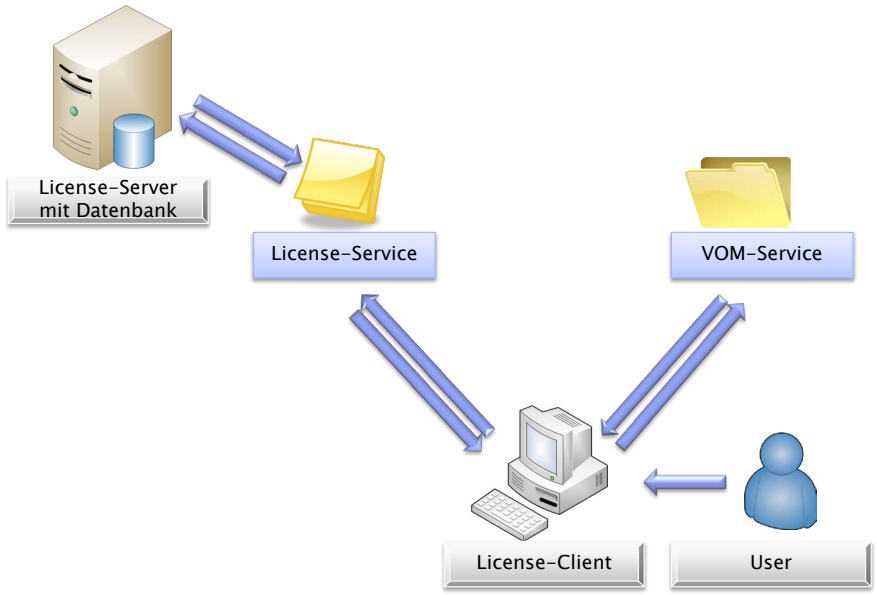

Abbildung 5.2: License-Modell: Beantragung einer Lizenz (Ausschnitt aus Abbildung 5.1)

- **Initialisierung des Vorgangs**
  Der Benutzer initialisiert den Vorgang zur Beantragung einer Lizenz (siehe 5.3.1).
- **Discovery der Services**
  Der *License-Client* sucht nach den notwendigen Web Services (siehe 5.3.2).
- **Generierung des Request-Tokens**
  Der Benutzer übermittelt dem *License-Client* alle notwendigen Informationen, damit dieser ein *Request-Token* generieren kann (siehe 5.3.3).

- **Aufruf des License-Services**
  Der *License-Client* ruft nun den *License-Service* auf und übergibt diesem das *Request-Token*.

- **Bearbeitung des Request-Tokens durch den License-Service**
  Der *License-Service* bearbeitet das *Request-Token* (siehe 5.5.1) und übergibt es anschließend dem *License-Server*.

- **Bearbeitung des Request-Tokens durch den License-Server**
  Der *License-Server* ist nun verantwortlich für die Bearbeitung des *Request-Tokens* (siehe 5.4.1).

- **Generierung des License-Tokens beziehungsweise des Reject-Tokens**
  Der *License-Server* hat bei der Bearbeitung des *Request-Tokens* entschieden, ob er im Folgeschritt ein *License-Token* oder ein *Reject-Token* generiert (siehe 5.4.2 beziehungsweise 5.4.3).

- **Übermittlung des jeweiligen Tokens**
  Der *License-Server* übermittelt nun das jeweilige Token dem *License-Service* als Antwort. Anschließend übergibt der *License-Service* das Token dem *License-Client* als Antwort für den Service-Aufruf (siehe 5.5.2 beziehungsweise 5.5.3).

- **License-Client wartet auf Antwort vom License-Service**
  Der *License-Client* befindet sich bis jetzt immer noch in einem Wartezustand auf Antwort des *License-Services*. Sollte es zu einem Timeout kommen, reagiert er entsprechend (siehe 5.3.4).

- **Bearbeitung des License-Tokens beziehungsweise des Reject-Tokens**
  Wenn der *License-Client* eine Antwort vom *License-Service* erhalten hat, so ist dies entweder ein *License-Token* oder ein *Reject-Token*. Je nachdem führt der *License-Client* die Bearbeitung durch (siehe 5.3.5 beziehungsweise 5.3.6).

- **Abschluss der Beantragung**
  Der *License-Client* und der Benutzer sind nun im Besitz eines *License-Tokens*, also einer Lizenz beziehungsweise wissen bescheid, dass eine Lizenzanfrage gescheitert ist. Die Beantragung der Lizenz ist somit abgeschlossen.

5.2.3 Starten des Jobs

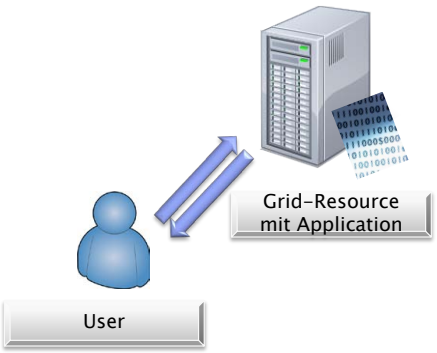

Abbildung 5.3: License-Modell: Job-Submission (Ausschnitt aus Abbildung 5.1)

- **Job-Submission**
  Der Benutzer kann nun seinen Job auf die Grid-Ressourcen übermitteln. Die Input-Daten des Jobs bestehen nun aus den „eigentlichen“ Input-Daten sowie dem *License-Token*.
- **Scheduling der Anwendung**
  Aus Sicht der Grid-Middleware handelt es sich um eine „normale“ Anwendung, die auf Ressourcen zugeteilt werden muss und dann gestartet werden kann. Dies ist Aufgabe des Grid-Schedulers.
- **Warten auf das Ergebnis**
  Der Benutzer wartet nun auf das Ergebnis. Dies kann entweder die erfolgreiche Durchführung des Jobs sein, dann bekommt der User wie gewohnt die Ergebnisdaten. Andererseits kann aber auch das *License-Token* abgelehnt oder die Lizenz im Laufe der Zeit ungültig werden, dies würde dem Benutzer ebenfalls als Ergebnis mitgeteilt.

5.2.4 Aktivierung der Lizenz und Durchführung des Jobs

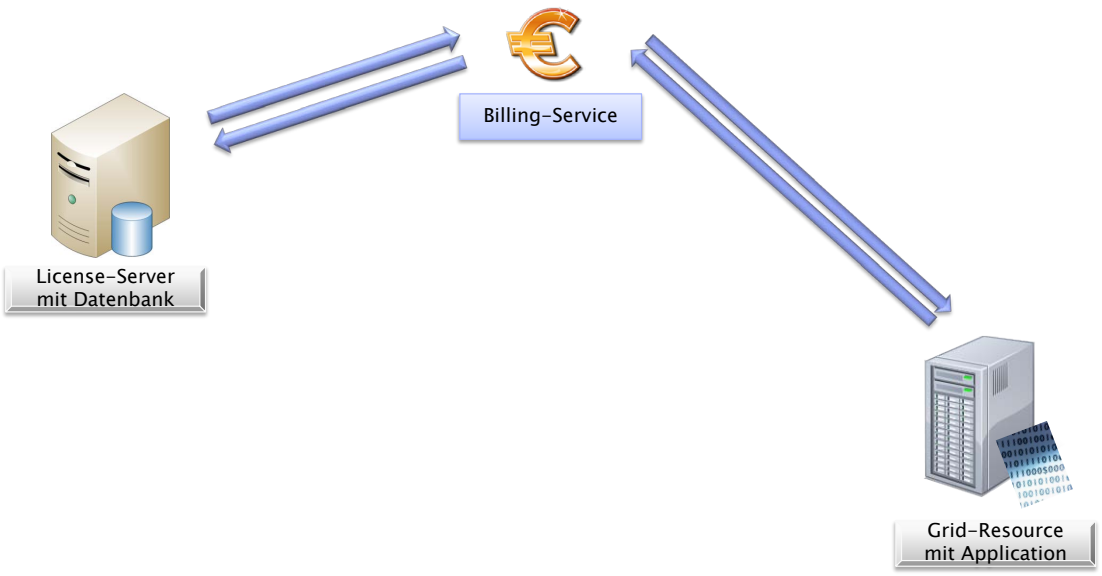

Abbildung 5.4: License-Modell: Activation (Ausschnitt aus Abbildung 5.1)

- **Start der Anwendung**
  Der Grid-Scheduler hat nun die *Application* gestartet.
- **Bearbeitung des License-Tokens**
  Die *Application* prüft nun zunächst die Gültigkeit des zugeteilten *License-Tokens* (siehe 5.6.1). Die Anwendung wird abgebrochen, wenn die Lizenz ungültig ist.
- **Generierung des Activation-Requests**
  Da die Lizenz erst noch aktiviert werden muss, generiert die *Application* nun den *Activation-Request* (siehe 5.6.2). Anschließend ruft sie den *Billing-Service* auf und übergibt diesem den *Activation-Request*.
- **Bearbeitung des Activation-Requests durch den Billing-Service**
  Nach Bearbeitung des *Activation-Requests* (siehe 5.7.1) leitet der *Billing-Service* den *Activation-Request* an den *License-Server* weiter.
- **Bearbeitung des Activation-Requests durch den License-Server**
  Der *License-Server* kümmert sich nun um die Validierung des *Activation-Requests* und die Durchführung beziehungsweise Ablehnung des Requests (siehe 5.4.4).
- **Generierung des Activation-Tokens beziehungsweise Activation-Rejects**
  Der *License-Server* generiert je nach Ergebnis einen *Activation-Token* beziehungsweise einen *Activation-Reject* (siehe 5.4.5 beziehungsweise 5.4.6) und übermittelt diesen an den *Billing-Service*.

- **Bearbeitung des Activation-Token / -Rejects durch den Billing-Service**
  Der *Billing-Service* bearbeitet nun den *Activation-Token* beziehungsweise den *Activation-Reject* (siehe 5.7.2) und leitet diesen danach an die aufrufende *Application* als Antwort weiter.

- **Bearbeitung des Activation-Token / -Rejects durch die Application**
  Die *Application* hat nun entweder einen *Activation-Token* oder einen *Activation-Reject* erhalten und überprüft die erhaltenen Informationen (siehe 5.6.3 beziehungsweise 5.6.4).

- **Unlock und Generierung des CheckIn-Requests**
  Wenn die *Application* einen gültigen *Activation-Token* erhalten hat, wird ein *CheckIn-Request* zur Rückgabe der Lizenzen generiert. Danach wird die eigentliche Anwendung entsperrt und die Durchführung des Jobs kann beginnen (siehe 5.6.5). Dieser Punkt entspricht nun dem klassischen Ausführen einer Anwendung auf Grid-Ressourcen.

- **Laufende Prüfung der Lizenz**
  Die *Application* prüft laufend und in bestimmten Zeitintervallen die Gültigkeit der zur Verfügung gestellten Lizenzen (siehe 5.6.6). Im Falle, dass keine Gültigkeit mehr vorliegt, wird die Berechnung abgebrochen und der Job beendet.

- **Ende und Abbruch des Jobs**
  Wenn der Job zu Ende ist beziehungsweise abgebrochen wurde, wird als Teil des Ergebnisses der *CheckIn-Request* ausgegeben, der zu Fallback-Zwecken dient. Außerdem wird ein automatisches CheckIn der Lizenzen mittels dem *Billing-Service* versucht (siehe 5.6.7).

- **Bearbeitung des CheckIn-Requests durch den Billing-Service**
  Der *Billing-Service* bearbeitet den *CheckIn-Request* und übermittelt ihn an den *License-Server* (siehe 5.7.3).

- **Bearbeitung des CheckIn-Requests durch den License-Server**
  Der *License-Server* bearbeitet den *CheckIn-Request* und generiert nach erfolgreicher Durchführung eine *CheckIn-Response* (siehe 5.4.7). Diese übermittelt er als Antwort an den *Billing-Service*.

- **Bearbeitung der CheckIn-Response durch den Billing-Service**
  Nach Bearbeitung der *CheckIn-Response* (siehe 5.7.4) leitet der *Billing-Service* diese als Antwort an die aufrufende *Application* weiter.

- **Erfolgreiches Ende des Jobs**
  Zu diesem Punkt ist der Job nun erfolgreich durchgeführt worden und die Lizenzen wurden zum Ende wieder beim *License-Server* eingecheckt.

5.2.5 Abfrage von Status-Informationen

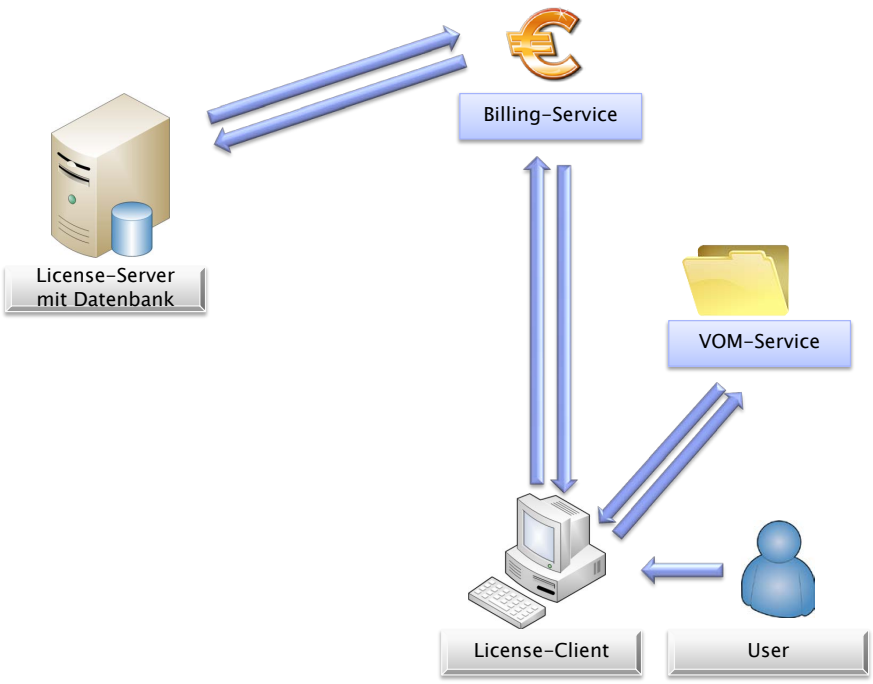

Abbildung 5.5: License-Modell: Abfrage von Informationen (Ausschnitt aus Abbildung 5.1)

- **Abfrage von Status-Informationen jederzeit möglich**
  Der Benutzer kann jederzeit mittels dem *License-Client* Status-Informationen zu seinen aktuell reservierten Lizenzen, gerade benutzten Lizenzen und ehemals benutzten Lizenzen erhalten. Dazu wendet er sich, nachdem er sich erfolgreich authentisiert hat, bezüglich einer Statusabfrage an den *License-Client*.

- **Generierung des License-Information-Requests**
  Der *License-Client* bearbeitet die Anfrage des Benutzers und generiert aus den zur Verfügung gestellten Daten einen *License-Information-Request* (siehe 5.3.9) und ruft den *Billing-Service* damit auf.

- **Bearbeitung des License-Information-Requests durch den Billing-Service**
  Der *Billing-Service* leitet den *License-Information-Request* an den *License-Server* weiter (siehe 5.7.5).

- **Bearbeitung des License-Information-Requests durch den License-Server**
  Der *License-Server* bearbeitet den *License-Information-Request*. Er antwortet dem *Billing-Service* entweder mit den angeforderten Informationen als *License-Information-Response* oder lehnt die Anfrage ab (siehe 5.4.8).

- **Erhalt der License-Information-Response**
  Über den *Billing-Service* erhält der *License-Client* die *License-Information-Response* als Antwort. Die Informationen stellt er dem Benutzer zur Verfügung.

5.2.6 Manueller CheckIn von Lizenzen

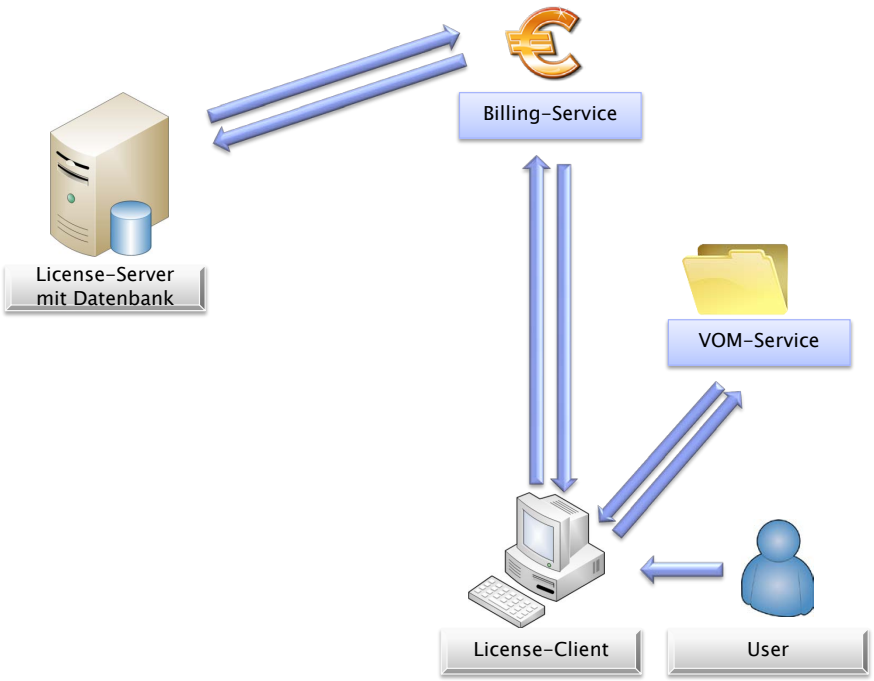

Abbildung 5.6: License-Modell: Manueller CheckIn (Ausschnitt aus Abbildung 5.1)

- **Fallback auf manuelles CheckIn**
  Wenn ein automatisches Einchecken der Lizenzen bei Ende oder Abbruch des Jobs nicht funktionierte, wurde dem Benutzer ein *Manual-CheckIn-Request* übermittelt. Mit diesem kann er nun den manuellen Eincheck-Vorgang initiieren.

- **Start des Vorgangs**
  Der Benutzer wendet sich an den *License-Client* mit dem *Manual-CheckIn-Request*.

- **Bearbeitung des Manual-CheckIn-Requests**
  Der *License-Client* bearbeitet den *Manual-CheckIn-Request* (siehe 5.3.7) und übermittelt diesen an den *Billing-Service*.

- **CheckIn-Vorgang analog zum automatischen CheckIn**
  Ab hier läuft der manuelle CheckIn genauso ab, wie der automatische, sprich über den *Billing-Service* zum *License-Server* und wieder zurück.

- **Bearbeitung der Manual-CheckIn-Response**
  Der *License-Client* erhält als Antwort vom *Billing-Service* die *Manual-CheckIn-Response*, die der *License-Client* dem Benutzer zur Verfügung stellt (siehe 5.3.8). Der CheckIn ist damit abgeschlossen oder kann bei Fehler nochmals wiederholt werden.

5.2.7 Reservierungen ändern

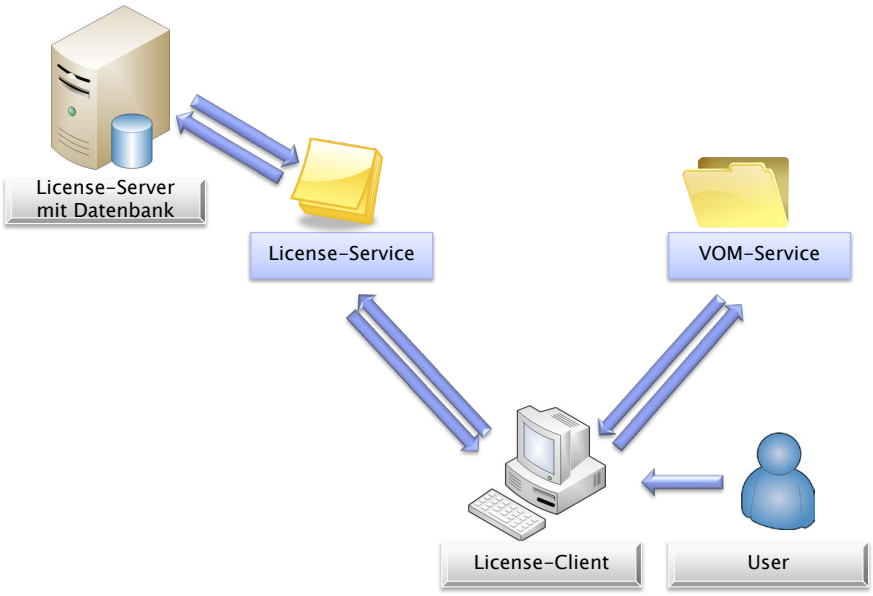

Abbildung 5.7: License-Modell: Reservierungen ändern (Ausschnitt aus 5.1)

- **Initialisierung des Vorgangs**
  Der Benutzer initialisiert den Vorgang zum Update einer Lizenz (siehe 5.3.1).
- **Generierung des License-Update-Requess**
  Der Benutzer übermittelt dem *License-Client* alle notwendigen Informationen, damit dieser einen *License-Update-Request* generieren kann (siehe 5.3.10).
- **Aufruf des License-Services**
  Der *License-Client* ruft nun den *License-Service* auf und übergibt diesem den *License-Update-Request*.
- **Bearbeitung des License-Update-Requests durch den License-Service**
  Der *License-Service* bearbeitet den *License-Update-Request* (siehe 5.5.4) und übergibt diesen anschließend dem *License-Server*.
- **Bearbeitung des License-Update-Requests durch den License-Server**
  Der *License-Server* ist nun verantwortlich für die Bearbeitung des *License-Update-Requests* (siehe 5.4.9).
- **Generierung des License-Update-Tokens beziehungsweise des -Rejects**
  Der *License-Server* hat bei der Bearbeitung des *License-Update-Requests* entschieden, ob er im Folgeschritt ein *License-Update-Token* oder einen *License-Update-Reject* generiert (siehe 5.4.10 beziehungsweise 5.4.11).

- **Übermittlung des jeweiligen Tokens**
  Der *License-Server* übermittelt nun das jeweilige Token dem *License-Service* als Antwort. Anschließend übergibt der *License-Service* das Token dem *License-Client* als Antwort für den Service-Aufruf (siehe 5.5.5 beziehungsweise 5.5.6).

- **License-Client wartet auf Antwort vom License-Service**
  Der *License-Client* befindet sich bis jetzt immer noch in einem Wartezustand auf Antwort des *License-Services*. Sollte es zu einem Timeout kommen, reagiert er entsprechend (siehe 5.3.4).

- **Bearbeitung des License-Update-Tokens beziehungsweise des -Rejects**
  Wenn der *License-Client* eine Antwort vom *License-Service* erhalten hat, so ist dies entweder ein *License-Update-Token* oder ein *License-Update-Reject*. Je nachdem führt der *License-Client* die Bearbeitung durch (siehe 5.3.11 beziehungsweise 5.3.12).

- **Abschluss des Updates**
  Der *License-Client* und der Benutzer sind nun im Besitz eines *License-Update-Tokens*, also einer upgedateten Lizenz. Falls das Update abgelehnt wurde, kann der *License-Client* dies dem Benutzer mitteilen.
  Das *License-Update-Token* ist analog zu einem *License-Token* zu benutzen.
  Ein *License-Update-Reject* ist in der Regel dann übermittelt worden, wenn der Job bereits läuft und eine Verkürzung beziehungsweise komplette Stornierung nicht mehr möglich ist. In diesem Fall kann der Benutzer manuell den Job abbrechen und danach erneut einen *License-Update-Request* beantragen.

5.3 Der License-Client

Der *License-Client* ist die Komponente, mit der der User kommuniziert um seine notwendigen Lizenzen zu erhalten und zu verwalten sowie Informationen über bereits zugeteilte Lizenzen anzufordern. Der *License-Client* ist daher selbst eine Anwendung und für grundlegende Features verantwortlich.
Der *License-Client* besitzt ein Zertifikat, das er für Signaturen benötigt. Außerdem hat er einen Pool mit Zertifikaten von vertraulichen *License-Servers* und *Applications*.
Die Aufgaben des *License-Clients* sind in mehrere Gebiete unterteilt.

5.3.1 Initialisierung des Vorgangs

Der *License-Client* bietet insbesondere die Möglichkeit, den gesamten Lizenzierungsvorgang zu starten. Er ermöglicht daher dem Nutzer die Übermittlung der Daten, die für eine Lizenzierung notwendig sind. Der Benutzer muss dafür die zur Generierung des *Request-Tokens* beziehungsweise *License-Update-Requests* benötigten Informationen bereitstellen.

5.3.2 Discovery der Services

Der *License-Client* ermittelt die URIs des *License-Services* sowie des *Billing-Services*.

5.3.3 Generierung des Request-Tokens

Aus den vom Benutzer erhaltenen Daten generiert der *License-Client* ein *Request-Token*. Der Benutzer hat folgende Daten bereitzustellen:

- **Authentisierungsinformationen**
  Damit sich der Benutzer gegenüber dem *License-Client* authentisieren kann, muss er ein gültiges Zertifikat übermitteln.
- **Input-Daten des Jobs**
  Die Anwendung, die auf dem Grid gestartet werden soll, benötigt in der Regel Daten zum Bearbeiten. Eine Lizenz wird an diese Daten gebunden. Daher übermittelt der Benutzer die Daten an den *License-Client*.
- **Application**
  Da ein *License-Client* für mehrere Anwendungen zuständig sein kann, muss natürlich mitgeteilt werden, für welche Anwendung Lizenzen angefordert werden.
- **Lizenzinhaber**
  Der Benutzer übermittelt, welche Lizenzen benutzt werden sollen. Dies hat zum Beispiel dann Sinn, wenn sowohl seine Organisation als auch die Virtuelle Organisation Lizenzen besitzt.
- **Anzahl der Lizenzen**
  Eine obligatorische Information.
- **Typ der Lizenzen**
  Da das Lizenzmodell sowohl Pay-per-Use als auch Floating-Lizenzen unterstützt, kann der Benutzer eine Auswahl für seinen Job treffen, falls er dazu berechtigt ist.

- **Start und Dauer der Gültigkeit**
  Zu Reservierungszwecken kann die Gültigkeit der Lizenzen auf eine in der Zukunft liegende Zeitspanne gesetzt werden.

Der *License-Client* generiert daraus das *Request-Token*. Dieses beinhaltet jedoch nicht die konkreten Input-Daten, sondern einen Hash dieser Daten. Alle Parameter bilden gemeinsam mit einer Zufallszahl und signiert vom *License-Client* das *Request-Token*.
Dieses Token wird dann an den *License-Service* übergeben.

5.3.4 Timeout

Sollte innerhalb eines festgelegten Timeouts keine Antwort vom *License-Service* kommen, da ein Request, Token oder Reject verlorengegangen ist, so schickt der *License-Client* den exakt selben *Request-Token* beziehungsweise *License-Update-Request* nochmals.

5.3.5 Bearbeitung des License-Tokens

Nach einer erfolgreichen Anfrage erhält der *License-Client* ein *License-Token* vom *License-Service*. Dieses Token beinhaltet eine Lizenz für die angefragte Anwendung für den gewünschten Zeitraum. Der *License-Client* prüft die Signatur des *License-Servers* und stellt somit sicher, dass das Token tatsächlich von einem vertraulichen *License-Server* generiert wurde. Der *License-Client* stellt dieses Token nun dem Benutzer zur Verfügung.

5.3.6 Bearbeitung des Reject-Tokens

Nach einer Ablehnung der Anfrage erhält der *License-Client* ein *Reject-Token*. Dieses Token enthält Informationen warum die Anfrage scheiterte. Der *License-Client* stellt diese Informationen nach einer Prüfung der Signatur des Tokens dem Benutzer zur Verfügung.

5.3.7 Bearbeitung des Manual-CheckIn-Requests

Der *License-Client* prüft den Inhalt des *Manual-CheckIn-Requests* und prüft den Status des Jobs, den der CheckIn betrifft. Nur wenn dieser Job abgeschlossen ist und die Signatur des Requests gültig ist, wird ein *CheckIn-Request* generiert und vom *License-Client* signiert. Dieser Request wird dann dem *Billing-Service* zum nochmaligen Versuch übermittelt.

5.3.8 Bearbeitung der Manual-CheckIn-Response

Der *License-Client* erhält die *Manual-CheckIn-Response* vom *Billing-Service* und leitet diese weiter an den Benutzer.

5.3.9 Bearbeitung eines Status-Requests

Der Benutzer kann den *License-Client* dazu verwenden, Status-Informationen zu einer angeforderten oder reservierten Lizenz anzufordern. Dazu muss er sich gegenüber dem *License-Client* authentisieren und die ID der Lizenzanfrage übergeben, die er aus dem *License-Token* erhält. Der *License-Client* generiert daraufhin einen *License-Information-Request*. Dieses besteht aus der ID sowie den Benutzerinformationen und ist signiert vom *License-Client*. Damit stellt er eine Anfrage beim *Billing-Service*.

Als Antwort erhält der *License-Client* die gewünschten Informationen als *License-Information-Response*. Nach einer Prüfung der Signatur stellt er sie dem Benutzer zur Verfügung.

5.3.10 Generierung des License-Update-Requests

Der Benutzer kann den *License-Client* dazu verwenden, bereits reservierte Lizenzen zu verlängern oder eine Reservierung zu stornieren. Der Benutzer hat bereits in der Vergangenheit ein *License-Token* erhalten. Dieses übermittelt er zusammen mit seinen Update-Wünschen an den *License-Client* mit der Beauftragung eines Updates. Der *License-Client* prüft die Validität des Tokens und generiert daraus einen *License-Update-Request*, das aus den übermittelten Daten zusammen mit einer Signatur des *License-Clients* besteht.

5.3.11 Bearbeitung des License-Update-Tokens

Nach einem erfolgreichen Update erhält der *License-Client* ein *License-Update-Token* vom *License-Service*. Dieses Token beinhaltet die upgedatete Lizenz. Der *License-Client* prüft die Signatur des *License-Servers* und stellt somit sicher, dass das Token tatsächlich von einem vertraulichen *License-Server* generiert wurde.
Der *License-Client* stellt dieses Token nun dem Benutzer zur Verfügung.

5.3.12 Bearbeitung des License-Update-Rejects

Nach einer Ablehnung eines Updates erhält der *License-Client* einen *License-Update-Reject*. Dieser enthält Informationen warum das Update scheiterte. Der *License-Client* stellt diese Informationen nach einer Prüfung der Signatur dem Benutzer zur Verfügung.

5.4 Der License-Server

Der *License-Server* verwaltet alle Lizenzen. Er entspricht prinzipiell einem normalen Lizenzserver mit den gleichen Aufgaben, wie er in jedem Client-Server-basierten Lizenzschema Einsatz findet. Er besitzt also ein eigenes Zertifikat sowie einen Pool, der die vertraulichen Zertifikate der kommunizierenden Entitäten enthält. Er unterscheidet sich jedoch in der Hinsicht, dass Web Services bei ihm die Aufrufe durchführen und Informationen übermitteln beziehungsweise anfordern. Außerdem ist er selbst eine WS-Resource und daher innerhalb eines Containers deployed. Jedoch sind aus technischer Sicht alle Möglichkeiten zur Lastverteilung, wie zum Beispiel eine Clusterung, ausschöpfbar. Zur Abspeicherung der Daten bietet sich eine Datenbank wie zum Beispiel MySQL ([Sun10c]) an. Die Datenbank benötigt sinnvollerweise mindestens drei Tables. Eine Table zur Verwaltung der Kundendaten sowie eine Table zur Verwaltung der Anwendungen. Außerdem eine weitere Table zur Verwaltung der gegenwärtig reservierten, zugeteilten und ehemals zugeteilten Lizenzen. Diese dritte Table ist insbesondere für das Billing und zur statistischen Auswertung relevant.
Die erste Table, die die Kundendaten enthält, kann beliebig aussehen, muss aber zwingend das Attribut user-ID enthalten. Selbiges gilt für die zweite Table, die auf jedenfall das Attribut Application-ID beinhalten muss. Auf diese beiden Attribute referenziert die dritte Table, die als *License-Table* folgende Attribute benötigt:

- ID (eindeutige ID in der Table)
- user-ID
- Application-ID
- Hash der Input-Daten
- Anzahl der Lizenzen
- Typ der Lizenzen
- Lizenzinhaber
- Zeitpunkt des Beginns der Reservierung
- Zeitpunkt des Endes der Reservierung
- Zeitpunkt des Starts der Abrechnung
- Zeitpunkt des Endes der Abrechnung
- Status (reserved, in use, used)
- Zufallszahl
- Zähler der identischen Requests (zur Überwachung der Verluste des *License-Tokens*)
- Job-ID
- Referenz auf eine weitere Table zur Auflistung der ausführenden Computing Resources

5.4.1 Bearbeitung des Request-Tokens

Der *License-Server* erhält das *Request-Token* durch den *License-Service*. Nach erfolgreicher Prüfung der Signatur des Requests prüft der *License-Server* die Identität des Anfragestellers sowie die angeforderte Anwendung gegen die Kundendatenbank. Außerdem wird mittels der *License-Table* überprüft, ob das angeforderte Lizenzpaket zugeteilt werden kann, oder ob bereits, im Falle von Floating-Lizenzen, das Kontingent erschöpft ist. Der *License-Server* entscheidet dann, ob er die Anfrage bearbeitet oder ablehnt. Eine Ablehnung wird dem *License-Service* mitgeteilt. Im Falle eines positiven Ergebnisses bearbeitet der *License-Server* die Anfrage. Es kann jedoch passieren, dass für die gewünschte Zeitspanne keine Lizenzen zur Verfügung stehen. Dies wird bei der Generierung des *License-Tokens* berücksichtigt.
Der *License-Server* trägt nun die Daten des Requests in die *License-Table* ein. Er kann dabei folgende Attribute füllen: ID, user-ID, Application-ID, Hash der Input-Daten, Anzahl der Lizenzen, Typ der Lizenzen, Lizenzinhaber, Zeitpunkt des Beginns der Reservierung, Zeitpunkt des Endes der Reservierung, die Zufallszahl und den Zähler. Außerdem wird der Status auf reserved gesetzt. Die übrigen Attribute werden erst später bearbeitet.
Bevor er jedoch die Daten einträgt, prüft er anhand der Daten aus der Datenbank und der Daten aus dem *Request-Token*, ob es sich bei dem erhaltenen *Request-Token* um eine erneute Anfrage handelt, da das *License-Token* verlorengegangen ist. In diesem Fall wird nochmals ein neues *License-Token* generiert, und der Zähler der Requests für diese Lizenzen in der Datenbank inkrementiert.

5.4.2 Generierung des License-Tokens

Das *License-Token* wird nach positiver Prüfung des *Request-Tokens* generiert. Zusätzlich zu den Daten des Requests beinhaltet das *License-Token* folgende Attribute:

- **Token-Typ**
  Zur Identifizierung, ob es sich um ein *License-Token* oder um ein *Reject-Token* handelt.
- **Crypto-Hash**
  Aus allen Daten des *Request-Tokens* zusammen mit der Zufallszahl wird ein Hash gebildet und verschlüsselt.
- **Signatur des License-Servers**
  Der *License-Server* signiert das gesamte *License-Token*.

Dieses *License-Token* dient nun als vollwertige Lizenz, kann jedoch noch nicht direkt verwendet werden, sondern muss später erst aktiviert werden. Das *License-Token* wird dem *License-Service* als Antwort für die Anfrage übermittelt.

5.4.3 Generierung des Reject-Tokens

Im Falle einer Ablehnung aus diversen Gründen wird ein *Reject-Token* generiert. Dieses Token beinhaltet zusätzlich zu den Attributen des Requests folgende Informationen:

- **Token-Type**
  Zur Identifizierung, ob es sich um ein *License-Token* oder um ein *Reject-Token* handelt.
- **Reject-Grund**
  Dieses Attribut beschreibt den Grund einer Ablehnung. Die Ablehnung kann stattgefunden haben, weil der Benutzer keine Lizenznutzungsrechte besitzt oder weil keine Lizenzen zum gewünschten Zeitpunkt zur Verfügung stehen. Außerdem kann es natürlich ein allgemeines Authentisierungsproblem oder aber auch ein technisches Problem wie zum Beispiel ein Timeout bei der Bearbeitung gegeben haben.
- **Crypto-Hash**
  Aus allen Daten des *Request-Tokens* zusammen mit der Zufallszahl wird ein Hash gebildet und verschlüsselt.
- **Signatur des License-Servers**
  Der *License-Server* signiert das gesamte *Reject-Token*.

Dieses *Reject-Token* wird dem *License-Service* als Antwort für die Anfrage übermittelt.

5.4.4 Bearbeitung des Activation-Requests

Den *Activation-Request* erhält der *License-Server* vom *Billing-Service*. Zuallererst prüft er die Gültigkeit der Signaturen des Requests um sicherzugehen, dass der *Activation-Request* nicht manipuliert wurde. Danach verbindet er sich mit der Datenbank und sucht in der *License-Table* nach dem Eintrag des Requests. Außerdem prüft der *License-Server*, ob die Uhrzeit und das Datum des *Activation-Requests* innerhalb eines Toleranzbereichs um die eigene Uhrzeit und Datum liegt. Zusätzlich generiert er nochmals den Crypto-Hash aus den Daten der Datenbank und prüft ihn gegen den Crypto-Hash aus dem *Activation-Request*.
Bei einem negativen Ergebnis wird ein *Activation-Reject* generiert. Bei einem positiven Ergebnis ein *Activation-Token*.
Zusätzlich werden bei positiver Prüfung in der Datenbank folgende Attribute gesetzt:

- Zeitpunkt des Starts der Abrechnung
- Status auf *in use*
- Auflistung der Computing Resources
- Job-ID

5.4.5 Generierung des Activation-Tokens

Das *Activation-Token* beinhaltet die Informationen des *Activation-Requests* und zusätzlich folgende:

- **Activation-Typ**
  Zur Identifizierung, ob es sich um ein *Activation-Token* oder um ein *Activation-Reject* handelt.

- **neuer Crypto-Hash**
  Zufallszahl um 1 inkrementiert als neuen Crypto-Hash.

- **Signatur des License-Servers**
  Der *License-Server* signiert das gesamten *Activation-Token*.

Das *Activation-Token* wird dem *Billing-Service* als Antwort übermittelt.

5.4.6 Generierung des Activation-Rejects

Der *Activation-Reject* beinhaltet die Informationen des *Activation-Requests* und zusätzlich folgende:

- **Activation-Typ**
  Zur Identifizierung, ob es sich um ein *Activation-Token* oder um ein *Activation-Reject* handelt.

- **Reject-Grund**
  Dieses Attribut beschreibt den Grund einer Ablehnung. Die Ablehnung kann stattgefunden haben, weil die Signaturen ungültig sind, die Daten nachweislich manipuliert wurden oder die Uhrzeit nicht im Toleranzbereich liegt. Es kann auch sein, dass das *License-Token* bereits nicht mehr gültig ist. Außerdem kann eine Ablehnung aus einem technischen Problem wie zum Beispiel einem Timeout bei der Bearbeitung resultieren.

- **Signatur des License-Servers**
  Der *License-Server* signiert den gesamten *Activation-Reject*.

Der *Activation-Reject* wird dem *Billing-Service* als Antwort übermittelt.

5.4.7 CheckIn nicht mehr benötigter Lizenzen

Der *Billing-Service* übermittelt dem *License-Server* einen *CheckIn-Request*. Der *License-Server* prüft zunächst die Signatur dieses Requests und führt bei erfolgreicher Prüfung den Request durch. Dabei setzt er die dem Request zugehörige Lizenzinformation in der Datenbank auf *used* und deaktiviert damit die Gültigkeit der Lizenz. Außerdem setzt er das Attribut bezüglich dem Ende der Benutzung auf die aktuelle Uhrzeit.
Anschließend generiert der *License-Server* eine *CheckIn-Response* und übermittelt diese an den *Billing-Service* als Antwort. Diese Response ist ein vom Server signierter *CheckIn-Request*.

5.4.8 Bearbeitung des License-Information-Requests

Der *Billing-Service* kann mittels einem *License-Information-Request* nach Informationen über eine bestimmte Lizenz verlangen. Der *License-Server* prüft die Signatur dieser Anfrage und ermittelt bei erfolgreicher Validierung die gewünschten Daten aus der Datenbank. Anschließend generiert er eine *License-Information-Response*, die die Daten aus dem *License-Information-Request* zusammen mit den zugehörigen Daten aus der Datenbank enthält und vom *License-Server* signiert wird. Diese Response übermittelt er dem anfragenden *Billing-Service*.

5.4.9 Bearbeitung des License-Update-Requests

Der *License-Server* erhält den *License-Update-Request* durch den *License-Service*. Nach erfolgreicher Prüfung der Signaturen des Requests prüft der *License-Server*, ob er die Änderung durchführen kann. Falls der Status auf reserved gesetzt ist, ist prinzipiell eine Änderung möglich. Falls der Status auf in-use oder used gesetzt ist, kann eine Stornierung oder Kürzung der Gültigkeit nicht mehr erfolgen. Wenn eine Verlängerung beantragt wurde, kann diese genehmigt werden, wenn das Kontingent nicht erschöpft ist.
Wenn der Request genehmigt wurde, updatet der *License-Server* die Daten in der *License-Table*.
Je nachdem ob der Request genehmigt wurde oder nicht, wird ein *License-Update-Token* oder ein *License-Update-Reject* generiert.

5.4.10 Generierung des License-Update-Tokens

Dieser Vorgang ist analog zur Generierung des *License-Tokens* (siehe 5.4.2).

5.4.11 Generierung des License-Update-Rejects

Sollte der *License-Update-Request* abgelehnt worden sein, so generiert der *License-Server* ein *License-Update-Reject*. Dieser Reject wird analog zum *Reject-Token* generiert (siehe 5.4.3). Als Reject-Grund wird hierbei jedoch vorwiegend auf eine Lizenz verwiesen, die bereits in Benutzung ist. Der *License-Update-Reject* wird dem aufrufenden *License-Service* als Antwort übergeben.

5.5 Der License-Service

Der *License-Service*, der als Service mit dem *License-Client* sowie mit dem *License-Server* kommuniziert, wird in diesem Abschnitt vorgestellt.

5.5.1 Bearbeitung des Request-Tokens

Der *License-Service* erhält vom *License-Client* das *Request-Token*. Aus diesem Token werden relevante Informationen zur Autorisierung entnommen. Zuerst wird das Zertifikat des *License-Clients* geprüft zur Sicherstellung, dass es sich um eine korrekte Anfrage handelt. Anschließend prüft der *License-Service* mittels dem Virtual Organisation Membership Service, ob der anfragende Benutzer grundsätzlich berechtigt ist, die angeforderten Lizenzen zu nutzen. Umfangreiche Black- und Whitelists sind hier denkbar.
Der *License-Service* entscheidet nun, ob das *Request-Token* abgelehnt oder akzeptiert wird. Bei einer Ablehnung, bekommt der anfragende *License-Client* eine negative Antwort mit der Begründung der Ablehnung.
Nach einer positiven Autorisierungs-Prüfung übermittelt der *License-Service* das *Request-Token* an den *License-Server* zur endgültigen Bearbeitung.

5.5.2 Bearbeitung des License-Tokens

Im Falle einer positiven Antwort vom *License-Server* erhält der *License-Service* ein *License-Token*, das die Lizenz beinhaltet. Dieses Token übermittelt der *License-Service* als Antwort dem *License-Client*.

5.5.3 Bearbeitung des Reject-Tokens

Im Falle einer negativen Antwort vom *License-Server* erhält der *License-Service* ein *Reject-Token*. Dieses Token übermittelt der *License-Service* als Antwort dem *License-Client*.

5.5.4 Bearbeitung des License-Update-Requests

Der Vorgang der Bearbeitung des *License-Update-Requests* und Weiterleitung dessen, ist analog zur Bearbeitung des *Request-Tokens* (siehe 5.5.1).

5.5.5 Bearbeitung des License-Update-Tokens

Im Falle einer positiven Antwort vom *License-Server* erhält der *License-Service* ein *License-Update-Token*, das die überarbeitete Lizenz beinhaltet. Dieses Token übermittelt der *License-Service* als Antwort dem *License-Client*.

5.5.6 Bearbeitung des License-Update-Rejects

Im Falle einer negativen Antwort vom *License-Server* erhält der *License-Service* einen *License-Update-Reject*. Diesen übermittelt der *License-Service* als Antwort dem *License-Client*.

5.6 Die Application

Die *Application* wird auf den Grid-Ressourcen ausgeführt und benötigt eine gültige Lizenz zum Start. Damit die Anwendung die eigentliche Berechnung durchführt, müssen einige Aufgaben erledigt werden. Ein Teil dieser Aufgaben läuft auch während der Ausführung.

5.6.1 Bearbeitung des License-Tokens

Die *Application* wurde an die Grid-Ressourcen als Job übermittelt. Dabei wurden gewisse Input-Daten angegeben. Außerdem ist das *License-Token*, das dem Benutzer vom *License-Client* zugeteilt wurde, Teil des Jobs und muss somit ebenfalls übermittelt werden.
Zuallererst prüft die Anwendung das übermittelte *License-Token* auf Gültigkeit. So wird sowohl das *License-Client*-Zertifikat als auch das *License-Server*-Zertifikat auf Vertraulichkeit untersucht. Anschließend wird ein Hash der Input-Daten gebildet und mit dem Hash der Input-Daten aus dem *License-Token* verglichen. Somit kann die Anwendung sichergehen, dass die Lizenz gültig ist und für die Input-Daten angefordert wurde.
Bei negativem Ergebnis der Prüfung wird die Anwendung gestoppt und der Job abgebrochen, da kein gültiges *License-Token* vorhanden ist. Der Benutzer erkennt einen frühzeitig abgebrochen Job anhand des Error-Levels.
Bei positivem Ergebnis der Prüfung wird nun ein *Activation-Request* generiert. Dieser *Activation-Request* ist notwendig zur Aktivierung der Lizenz.

5.6.2 Generierung des Activation-Requests

Der *Activation-Request* entspricht dem *License-Token* versehen mit einigen zusätzlichen Informationen:

- Datum und Uhrzeit
- Identifikation der Computing Resource
- Job-ID

Nach Generierung des *Activation-Requests* und Signierung nimmt die *Application* Kontakt zum *Billing-Service* auf und übermittelt den *Activation-Request*.

5.6.3 Bearbeitung des Activation-Tokens

Die *Application* prüft die Signatur des *Activation-Tokens* und startet bei erfolgreicher Validierung einen Unlock der eigentlichen Anwendung.

5.6.4 Bearbeitung des Activation-Rejects

Bei Erhalt eines *Activation-Rejects* wird der Job abgebrochen und dem Benutzer werden als Ergebnis des Jobs die Informationen aus dem *Activation-Reject* mitgeteilt.

5.6.5 Unlock der Anwendung

Der Unlock der Anwendung kann dann stattfinden, wenn ein erhaltenes *Activation-Token* als gültig eingestuft wurde. Bevor die Berechnung des Jobs gestartet wird, wird noch ein *CheckIn-Request* generiert. Der *CheckIn-Request* beinhaltet das *Activation-Token* versehen mit weiteren Informationen:

- Aktuelle Uhrzeit und Datum
- Job-ID

Der *CheckIn-Request* wird von der *Application* signiert und gilt als Teil des Ergebnisses des Jobs. Er dient zu Fallback-Zwecken bei unvorhergesehen Abbrüchen der Anwendung.
Nach dem Unlock wird die Anwendung direkt gestartet und die bei Job-Submission übermittelten Input-Daten können bearbeitet werden.

5.6.6 Laufende Prüfung der Gültigkeit der Lizenz

Da eine Lizenz auf eine bestimmte Zeitspanne beschränkt ist, kann es passieren, dass ein Job länger läuft als die Lizenz gültig ist. Ist die Gültigkeit abgelaufen, stellt die *Application* erneut einen *Activation-Request*, da es sein kann, dass die Lizenz mittlerweile verlängert wurde. Sollte dieser Request abgelehnt werden, wird die Berechnung des Jobs abgebrochen.

5.6.7 Ende oder Abbruch des Jobs

Bei Ende oder Abbruch des Jobs wird die Lizenz wieder zurückgegeben und ungültig gemacht. Dies ist ein kritischer Moment, denn wenn das Einchecken einer Lizenz nicht klappt, wird dem Benutzer weiterhin die Lizenz in Rechnung gestellt, beziehungsweise die Lizenz als belegt angesehen. Daher gibt es für das Einchecken einen Fallback-Mechanismus, der nach einem Timeout beziehungsweise einem Fehler ausgelöst wird.
Prinzipiell läuft das CheckIn jedoch folgendermaßen ab: Die *Application* sendet den, bei Unlock generierten *CheckIn-Request*, an den *Billing-Service*. Der *CheckIn-Request* wird vor Übermittlung an den *Billing-Service* von der *Application* signiert.
Wird nun eine *CheckIn-Response* als Antwort erhalten, so wird deren Signatur geprüft und als Teil der Output-Daten des Jobs an das Job-Ergebnis angehängt.
Im Falle eines Fehlers oder Timeouts wird der *CheckIn-Request* als Teil der Output-Daten des Jobs an das Job-Ergebnis angehängt. Der Benutzer erfährt damit, dass seine Lizenz noch *nicht* wieder eingecheckt wurde und er sich manuell darum kümmern muss.

5.7 Der Billing-Service

Der *Billing-Service* ist vor allem für die Verwaltung der Aktivierungen der Lizenzen zuständig. Außerdem nimmt er Anfragen vom *License-Client* entgegen, um statistische Auswertungen zu ermöglichen sowie den aktuellen Stand der eigenen Lizenzen zu ermitteln. Die Aufgaben des *Billing-Services* sind folgende:

5.7.1 Bearbeitung des Activation-Requests

Der *Billing-Service* merkt sich Informationen über den *Activation-Request*. Anschließend leitet er den *Activation-Request* an den *License-Server* weiter.

5.7.2 Bearbeitung des Activation-Tokens und des Activation-Rejects

Als Antwort erhält der *Billing-Service* vom *License-Server* entweder ein *Activation-Token* oder ein *Activation-Reject*. Diese leitet er an die aufrufende *Application* direkt weiter, nachdem er den *Activation-Request* in seiner Informationssammlung als bearbeitet markiert hat.

5.7.3 Bearbeitung des CheckIn-Requests

Der Billing-Service merkt sich Informationen über den *CheckIn-Request*. Anschließend leitet er den *CheckIn-Request* an den *License-Server* weiter. Sollte es dabei zu einem Timeout oder Fehler kommen, wird dies als Antwort der *Application* mitgeteilt.

5.7.4 Bearbeitung der CheckIn-Response

Die vom *License-Server* erhaltene *CheckIn-Response* wird in der Informationssammlung geloggt und anschließend an die aufrufende *Application* beziehungsweise den *License-Client* als Antwort übermittelt.

5.7.5 Bearbeitung des License-Information-Requests

Der *License-Client* kann den *Billing-Service* dazu benutzen, um Informationen über eine Lizenz herauszubekommen. Diesen *License-Information-Request* übermittelt der *Billing-Service* zur Bearbeitung an den *License-Server*.
Die erhaltene Antwort wird als Antwort dem *License-Client* übergeben.

5.8 Definition der rechtlichen Aspekte

Wie bereits mehrfach erwähnt, nutzt das beste Lizenzschema nichts, wenn es rechtlich nicht tragbar oder der zugrunde liegende Lizenzvertrag nicht stimmig ist. Somit befasst sich dieser Abschnitt mit den rechtlichen Bestimmungen und notwendigen Klauseln der Lizenzierung.

5.8.1 Grundlagen der Lizenzierung

Damit dieses Lizenzschema nutzbar ist, muss die Anwendung selbstverständlich auf diesem Schema beruhen und darf nicht mit einem anderen Lizenzierungsmechanismus lizenziert und geschützt werden. Die Anwendung muss exakt mit diesem Lizenzschema lizenziert werden.

5.8.2 Verantwortlichkeiten

In diesem Abschnitt werden die jeweiligen Verantwortlichkeiten dargestellt.

- Der ISV beziehungsweise Distributor stellt den *License-Server*, den *License-Client* sowie die Web Services zur Verfügung. Außerdem wird selbstverständlich die *Application* bei Kauf beziehungsweise Erwerb zur Verfügung gestellt.
- Die Organisation, die die *Application* nutzen möchte, muss sich darum kümmern, dass der *License-Server* als WS-Resource deployed wird und dass die Web Services ebenfalls aktiv geschalten werden.
- Für die Konfiguration der Komponenten ist ebenfalls die Organisation verantwortlich.
- Die Virtuelle Organisation, die das Grid betreibt, ist verantwortlich dafür, dass alle Web Services sowie der *License-Server* verfügbar und erreichbar sind.
- Der *License-Server* erhält seine Stammdatenbank vom ISV beziehungsweise Distributor. Die Datenbank kann auch extern untergebracht sein und ein geschützter Remote-Zugriff stattfinden.
- Die korrekte Bedienung des *License-Clients* obliegt dem User.
- Bei einem Fallback auf manuelles CheckIn der Lizenzen, ist der Benutzer für die Durchführung verantwortlich.
- Darüberhinaus gelten selbstverständlich alle klassischen Verantwortlichkeiten einer verteilten Umgebung und eines Lizenzvertrags.

5.8.3 Der Lizenzvertrag

Der Lizenzvertrag, auf dem dieses Lizenzmodell basiert, entspricht weitestgehend einem klassischen Lizenzvertrag. Es werden Bestimmungen, Verantwortlichkeiten, Haftungen, Haftungsausschlüsse, Konsequenzen und so weiter festgelegt.
Zu beachten ist jedoch, dass die oben genannten Verantwortlichkeiten klar erwähnt sind. Außerdem darf der Lizenzvertrag weder die Nutzung der *Application* in Grid-Umgebungen verbieten noch darf er das Ausführen auf „fremden“ (also einer anderen Organisation der Virtuellen Organisation gehörenden) Computing Resources untersagen.
Für eine explizite Formulierung des Lizenzvertrags sind selbstverständlich die jeweiligen Rechtsabteilungen verantwortlich.

5.9 Evaluierung des Lizenzmodells gegen den Anforderungskatalog

In diesem Abschnitt wird das Lizenzmodell gegen die Anforderungen evaluiert.

Allgemein: Unabhängigkeit von der eingesetzten Middleware (siehe 3.3.1)

Das Lizenzmodell wurde konzeptionell entwickelt und ist bezüglich der Middlewares sehr flexibel. Die prototypische Implementierung aus Kapitel 6 erfolgt jedoch auf Basis des Globus Toolkit 4.0.8.

Allgemein: Nutzung Grid-spezifischer Technologien (siehe 3.3.2)

Das Lizenzmodell verwendet Grid-typische Methoden und Funktionalitäten. So werden insbesondere die Web Services-Technologie sowie WS-Resources zur Lizenzierung eingesetzt.

Allgemein: Priorisierung von Jobs (siehe 3.3.3)

Die Priorisierung von Jobs wird nicht angeboten, sondern Jobs werden der Reihe nach, wie vom Grid-Scheduler vorgegeben, abgearbeitet.

Allgemein: Transparenz der Lizenzzuteilung (siehe 3.3.4)

Die Lizenzzuteilung erfolgt nicht vollständig transparent. Der User muss eine Lizenz mittels dem *License-Client* anfordern. Dies muss manuell ausgelöst werden. Der Vorgang selbst erfolgt jedoch transparent. Anschließend übermittelt der Benutzer die Lizenz als Teil der Input-Daten für den Job. Die Überprüfung der Gültigkeit sowie der CheckIn der Lizenz bei Ende des Jobs erfolgt transparent, wenn es zu keinem Fehler kommt.

Sicherheit: Firewallkonfiguration (siehe 3.4.1)

Die Firewallkonfiguration muss in Kleinigkeiten angepasst werden, die auch für eine klassische Lizenzierung notwendig wären.
Es muss gewährleistet werden, dass der *License-Client* auf die Services zugreifen kann und dass die Services mittels einem Pfad, bestehend aus Services, auf den *License-Server* zugreifen können. Sollte eine remote-Datenbank eingesetzt werden, so muss für die Kommunikation zwischen *License-Server* und Datenbank eventuell die Firewall angepasst werden. Außerdem ist zu beachten, dass der ISV beziehungsweise der zuständige Distributor Zugriff auf die Datenbank des *License-Servers* bekommt.

Sicherheit: Zuverlässigkeit in unzuverlässigen Netzwerken (siehe 3.4.2)

Zuverlässigkeit in unzuverlässigen Netzen spielt eine große Rolle bei Softwarelizenzierungen. Das Lizenzmodell kann man in dieser Hinsicht in drei Teile unterteilen.

- **Beantragung einer Lizenz**
  Wenn der *Request-Token* verlorengeht, merkt dies der *License-Client* anhand eines Timeouts. Er wiederholt die Anfrage.
  Wenn das *License-Token* verlorengeht, ist dies aus Sicht des Clients identisch zum Verlust des *Request-Tokens*. Die gesendete Wiederholung wird beim *License-Server* als Wiederholung identifiziert anhand der Daten, die der *Request-Token* innehat und die Zustellung des *License-Tokens* wird erneut durchgeführt.

- **Aktivierung der Lizenz**
  Bei Verlust des *Activation-Requests* merkt dies die *Application* anhand eines Timeouts. Der Übermittlungsversuch kann wiederholt werden. Der Verlust des *Activation-Tokens* ist aus Sicht der *Application* identisch. Somit kommt es zu einer Wiederholung der Anfrage. Der *License-Server* erkennt anhand des *Activation-Requests*, dass es sich um eine Wiederholung handelt und übermittelt nochmals den *Activation-Token*.

- **Rückgabe der Lizenz**
  Wenn die automatische Rückgabe der Lizenz scheitert, kann der Benutzer manuell mittels dem *License-Client* und dem *Manual-CheckIn-Request* die Rückgabe auslösen. Dies ist auch mehrmals möglich. Bei Verlust der *Manual-CheckIn-Response* und nochmaliger Übermittlung des *Manual-CheckIn-Requests* erkennt der *License-Server* dies als Wiederholung.

Accounting und Billing haben somit keine Probleme bei Verlust von Tokens und das Lizenzschema ist auch in nicht zuverlässigen Netzen einsetzbar.

Sicherheit: Verzahnung mit den Grid-spezifischen Sicherheitsmechanismen zur Authentisierung (siehe 3.4.3)

Die Authentisierung von Benutzern erfolgt mittels seiner Zertifikate, die er sowieso zur Benutzung von Grid-Ressourcen besitzt. Der *License-Client*, *License-Server* und die *Application* authentisieren sich gegenseitig ebenfalls mit X.509-Zertifikaten.
Die Autorisierung, ob ein bestimmter Nutzer Lizenzen anfordern darf (aus Sicht der Organisation und Virtuellen Organisation), ist mittels dem Virtual Organisation Membership Service abgedeckt. Die Autorisierung gegenüber einer Kundendatenbank, um eine Anfrage grundsätzlich zu genehmigen, wird klassisch vom *License-Server* durchgeführt.

Sicherheit: Multiple Nutzung von Lizenzen unterbinden (siehe 3.4.4)

Multiple Nutzung ist mittels der Generierung der individuellen Tokens unter Einbeziehung vieler Daten sowie insbesondere Bindung der Lizenz an die Input-Daten sowie der Aktivierung der Tokens unterbunden.

Sicherheit: Lizenzpools nach administrativen Domänen getrennt (siehe 3.5.1)

Die Lizenzpools sind sowohl in der VOMS-Datenbank als auch in der *License-Server*-Datenbank frei trennbar.

Lizenzierung: Einsatz verschiedener Lizenztypen (siehe 3.5.2)

Das Lizenzmodell bietet den Einsatz von Floating-Lizenzen und Pay-per-Use-Lizenzen an.

Lizenzierung: Auswahl der Lizenzen möglich (siehe 3.5.3)

Der Benutzer kann den Typ auswählen, wenn er für beide Typen autorisiert ist.

Lizenzierung: Hierarchische Vergabe von Lizenznutzungsrechten (siehe 3.5.4)

Dieses Feature wird indirekt mittels VOMS unterstützt und bezieht sich somit auf die allgemeine Delegation von Rechten.

Lizenzierung: Blacklists und Whitelists (siehe 3.5.5)

Die Datenbank des *License-Servers* enthält nur grundsätzliche Informationen, ob ein Benutzer Kunde ist oder nicht.
Mittels der VOMS-Datenbank lassen sich jedoch Black- und Whitelists umsetzen.

Lizenzierung: Lizenzreservierung mit Frist (siehe 3.5.6)

Lizenzen können für eine zukünftige Zeitspanne reserviert werden und sind nur in dieser Zeitspanne aktivierbar.

Lizenzierung: Pausieren und Fortsetzen der Lizenznutzung (siehe 3.5.7)

Dieses Feature ist nicht in vollem Umfang vorhanden.

Lizenzierung: Umfassendes Logging der angeforderten, zugeteilten und reservierten Lizenzen (siehe 3.5.8)

Das Logging übernimmt der *License-Server*. Er hat eine umfangreiche Datenbank hinter sich, in der genau ersichtlich ist, wer wann auf welchen Ressourcen welche Lizenzen eingesetzt hat. Außerdem ist hier ersichtlich, welche Lizenzen von wem für welche Zeitspanne reserviert sind und welche Lizenzen angefordert aber noch nicht aktiviert sind.

Lizenzierung: Änderungen am Lizenzpool und den Listen ohne Neustart (siehe 3.5.9)

Administrative Änderungen sind jederzeit möglich, da die Daten auf einer Datenbank basieren und bei jedem Vorgang neu gelesen werden.

Lizenzierung: Lizenzserver unabhängig der eingesetzten Plattform (siehe 3.5.10)

Der *License-Server* sowie der *License-Client* basieren auf Java. Somit sind sie auf allen Plattformen einsetzbar, für die Java zur Verfügung steht.

Lizenz: Lizenz als feste Dateistruktur (siehe 3.6.1)

Alle Tokens sind eindeutig definiert.

Lizenz: Mobilität der Lizenzen (siehe 3.6.2)

Die Lizenzen sind mobil und nicht an Hardware gebunden. Vielmehr sind sie an Software und vor allem an Daten gebunden.

Job: Lizenzen verlängern oder früher aufgeben (siehe 3.7.1)

Wenn ein Job beendet ist, wird die Lizenz automatisch zurückgegeben. Sollte ein Job länger als ursprünglich gedacht laufen, so kann der Benutzer manuell die Lizenzen verlängern.

Job: Jobs beenden, die innerhalb Timeouts keine Lizenzen erhalten haben (siehe 3.7.2)

Erhält eine *Application* innerhalb eines Timeouts kein *Activation-Token*, so wiederholt sie die Anfrage. Sollte nach einer bestimmten Anzahl von Wiederholungen immer noch kein *Activation-Token* erhalten worden sein, wird die *Application* und damit der Job beendet.

Job: Bekanntgabe, woher genutzte Lizenzen kommen (siehe 3.7.3)

Da der Benutzer bereits weiß, welche Lizenzen er der *Application* mitgegeben hat, ist dieses Feature irrelevant und die Information auf einem anderen Weg erreichbar.

Job: Überwachung der Lizenzen (siehe 3.7.4)

Die *Application* führt laufend und in bestimmten Zeitintervallen Überwachungen der Lizenzen aus und prüft diese, ob sie immer noch gültig sind.

Job: Überprüfen der Legalität der Lizenzen (siehe 3.7.5)

Die *Application* prüft die Legalität des *License-Tokens* einerseits mittels Signaturen andererseits lässt sie das *License-Token* vor Benutzung erst noch aktivieren. Somit wird sichergestellt, unter der Voraussetzung, dass keine Zertifikate erfolgreich gefälscht wurden, dass die übermittelte Lizenz legal ist.

Accounting: Anzeige der erhaltenen Lizenzen (siehe 3.8.1)

Der Benutzer kann jederzeit mittels dem *License-Client* den Status seiner Lizenzen anfragen.

Accounting: Sicheres Accounting und Abrechnungsverfahren (siehe 3.8.2)

Das Accounting ist sicher unter der Voraussetzung, dass keine korrupten Zertifikate eingesetzt werden beziehungsweise der Pool der vertraulichen Zertifikate ungültige Zertifikate enthält. Auch das Abrechnungsverfahren beruht auf den gleichen Sicherheitsmechanismen.

Accounting: Schutz vor Leugnung des Erhalts einer Lizenz (siehe 3.8.3)

Ohne Übermittlung einer Lizenz wird ein Job nicht ausgeführt. Eine Leugnung des Erhalts kombiniert mit einer Nutzung der geleugneten Lizenz ist nicht möglich, da jede Lizenz vor Start des Jobs noch automatisch aktiviert wird. Mittels dieser Aktivierung ist aus Sicht des *License-Servers* sichergestellt, dass der Benutzer die Lizenz tatsächlich erhalten hat. Er kann es nicht mehr abstreiten.

Accounting: Redundantes Abspeichern aller relevanten Informationen (siehe 3.8.4)

Alle relevanten Daten können redundant gespeichert werden. Für die Datenbanken bieten sich dafür standardisierte Methoden an.

Accounting: Statistische Auswertungen ermöglichen (siehe 3.8.5)

Statistische Auswertungen sind indirekt möglich durch Abfragen der Stati der Lizenzen. Der *License-Server* kann darüberhinaus ein Interface für globale statistische Auswertungen anbieten.

Accounting: Abrechnung quasi Echtzeit durchführen (siehe 3.8.6)

Die Abrechnung erfolgt live zu den Beantragungen und Aktivierungen.

Accounting: Delay der Lizenzierung berücksichtigen (siehe 3.8.7)

Das Delay der Lizenzzuteilung wird insofern berücksichtigt, dass eine Lizenz erst noch direkt bei Nutzung aktiviert werden muss und somit die Abrechnung erst zu diesem Zeitpunkt beginnt. Die Übermittlung des *Activation-Tokens* wird jedoch nicht mehr berücksichtigt. Sie nimmt aber den geringsten Teil der Aktivierungsdauer ein.

Accounting: Accounting mit verschiedenen Granularitätsstufen (siehe 3.8.8)

Das Accounting auf verschiedenen Granularitätsstufen findet indirekt statt, da der Benutzer bei Beantragung einer Lizenz angibt, wessen Lizenzen er benutzen möchte (eigene, Organisation, Virtuelle Organisation; falls verfügbar).

Accounting: Kostenexplosionen vermeiden (siehe 3.8.9)

Dieses Feature kann der *License-Client* indirekt bei einer Anfrage nach Lizenzen umsetzen, indem er zuerst eine Auswertung der bereits reservierten und benutzen Lizenzen durchführt und gegen eine Budget-Limitierung aus der VOMS-Datenbank prüft.

Accounting: Einsicht jederzeit möglich (siehe 3.8.10)

Einsicht in die aktuell reservierten, zugeteilten, benutzen und ehemals benutzten und reservierten Lizenzen ist jederzeit möglich.

Recht: Einbeziehung der Firmen-Policies (siehe 3.9.1)

Firmen-Policies werden nicht mit einbezogen. Jedoch werden nur Standard-Funktionen und -Komponenten eingesetzt, die in Grid-Umgebungen üblich sind. Daher sollte der Einsatz des Lizenzschemas in keinster Weise eine Firmen-Policy verletzen.

Recht: Lizenzvertrag berücksichtigen (siehe 3.9.2)

Das Lizenzschema beruht auf seinem eigenen Lizenzvertrag. Bestehende Verträge müssen erneuert und abgeändert werden.

Recht: Erweiterung des Lizenzvertrags bezüglich Nutzung der Lizenzen in Grids (siehe 3.9.3)

Da das Lizenzschema auf seinem eigenen Vertrag beruht, ist die Nutzung in Grids selbstverständlich nicht untersagt.

Recht: Rechtliche Konsequenzen bei Ausfall oder Nichtverfügbarkeit des Lizenzservers (siehe 3.9.4)

Rechtliche Konsequenzen können individuell bei jedem Vertrag festgelegt werden, wobei jedoch zu beachten ist, dass die Virtuelle Organisation großteils selbst für die Verfügbarkeit verantwortlich ist, da die Komponenten auf den eigenen Ressourcen betrieben werden.

Recht: Überprüfung der korrekten Lizenznutzung (siehe 3.9.5)

Eine Lizenz ist immer an die Input-Daten gebunden, jedoch hat der *License-Server*, da er nur den Hash der Daten kennt, keine konkreten Informationen über die tatsächlichen Daten.

5.10 Tabellarische Übersicht

In diesem Abschnitt wird das entwickelte Lizenzmodell nochmals tabellarisch gegen den Anforderungskatalog dargestellt. Dabei wird zur Bewertung das Maß der deutschen Schulnoten (1 = „sehr gut“ bis 5 = „mangelhaft“) herangezogen. Ein nicht vorhandenes Feature wird mit einem „-“ gekennzeichnet. Die erste Spalte verweist auf den Abschnitt, in dem die Anforderung beschrieben wird.

Abs.	Kategorie	Titel	Erfüllung
3.3.1	Allgemein	Unabhängigkeit von der eingesetzten Middleware	1
3.3.2	Allgemein	Nutzung Grid-spezifischer Technologien	1
3.3.3	Allgemein	Priorisierung von Jobs	-
3.3.4	Allgemein	Transparenz der Lizenzzuteilung	2
3.4.1	Sicherheit	Firewallkonfiguration	1
3.4.2	Sicherheit	Zuverlässigkeit in unzuverlässigen Netzwerken	1
3.4.3	Sicherheit	Verzahnung mit den Grid-spezifischen Sicherheitsmechanismen zur Authentisierung	1
3.4.4	Sicherheit	Multiple Nutzung von Lizenzen unterbinden	1
3.5.1	Lizenzierung	Lizenzpools nach administrativen Domänen getrennt	1
3.5.2	Lizenzierung	Einsatz verschiedener Lizenztypen	1
3.5.3	Lizenzierung	Auswahl der Lizenzen möglich	1
3.5.4	Lizenzierung	Hierarchische Vergabe von Lizenznutzungsrechten	2
3.5.5	Lizenzierung	Blacklists und Whitelists	2
3.5.6	Lizenzierung	Lizenzreservierung mit Frist	1
3.5.7	Lizenzierung	Pausieren und Fortsetzen der Lizenznutzung	-
3.5.8	Lizenzierung	Umfassendes Logging der angeforderten, zugeteilten und reservierten Lizenzen	1
3.5.9	Lizenzierung	Änderungen am Lizenzpool und den Listen ohne Neustart	1
3.5.10	Lizenzierung	Lizenzserver unabhängig der eingesetzten Plattform	2
3.6.1	Lizenz	Lizenz als feste Dateistruktur	1
3.6.2	Lizenz	Mobilität der Lizenzen	1

Abs.	Kategorie	Titel	Erfüllung
3.7.1	Job	Lizenzen verlängern oder früher aufgeben	1
3.7.2	Job	Jobs beenden, die innerhalb Timeouts keine Lizenzen erhalten haben	1
3.7.3	Job	Bekanntgabe, woher genutzte Lizenzen kommen	1
3.7.4	Job	Überwachung der Lizenzen	1
3.7.5	Job	Überprüfen der Legalität der Lizenzen	1
3.8.1	Accounting	Anzeige der erhaltenen Lizenzen	1
3.8.2	Accounting	Sicheres Accounting und Abrechnungsverfahren	1
3.8.3	Accounting	Schutz vor Leugnung des Erhalts einer Lizenz	1
3.8.4	Accounting	Redundantes Abspeichern aller relevanten Informationen	1
3.8.5	Accounting	Statistische Auswertungen ermöglichen	1
3.8.6	Accounting	Abrechnung quasi Echtzeit durchführen	1
3.8.7	Accounting	Delay der Lizenzierung berücksichtigen	2
3.8.8	Accounting	Accounting mit verschiedenen Granularitätsstufen	2
3.8.9	Accounting	Kostenexplosionen vermeiden	3
3.8.10	Accounting	Einsicht jederzeit möglich	1
3.9.1	Recht	Einbeziehung der Firmen-Policies	3
3.9.2	Recht	Lizenzvertrag berücksichtigen	-
3.9.3	Recht	Erweiterung des Lizenzvertrags bezüglich Nutzung der Lizenzen in Grids	1
3.9.4	Recht	Rechtliche Konsequenzen bei Ausfall oder Nichtverfügbarkeit des Lizenzservers	1
3.9.5	Recht	Überprüfung der korrekten Lizenznutzung	2

5.11 Zusammenfassung des Lizenzmodells

In den vorangegangenen Abschnitten wurde das Lizenzmodell, das in dieser Diplomarbeit entwickelt wurde, umfangreich dargestellt und beschrieben. Das Konzept beruht auf dem Einsatz von Web Services und WS-Resources sowie auf einem Client. Alle beteiligten Komponenten lassen sich gegen Manipulation mit State-of-the-Art-Mechanismen schützen. Dieser Schutz ist jedoch nicht Bestandteil des Lizenzmodells, darf aber, sobald das Konzept umgesetzt wird, nicht vernachlässigt werden. So ist insbesondere wichtig, dass der *License-Client* und der *License-Server* sowie die *Application* geschützt sind und nicht ohne weiteres manipuliert werden können. Ein absoluter Schutz ist jedoch niemals möglich, da Entscheidungen auf Sprungadressen im Code beruhen. Diese Sprungadressen sind mittels Reverse-Engineering sowie anderen Cracking-Methoden zu knacken. Ein gewisser Kopierschutz muss also zusätzlich eingeführt werden. Dieser ist aber nicht Teil der Diplomarbeit und vor allem nicht Teil eines Konzepts, sondern betrifft die praktische nicht-prototypische Umsetzung des Konzepts.

5.12 Anwendungsszenarien für das Lizenzmodell

In diesem Abschnitt werden einige Szenarien vorgestellt, in denen das entwickelte Lizenzmodell eingesetzt werden kann und wofür es gedacht ist.

5.12.1 Szenario 1: Privates Grid

In diesem Szenario existiert ein Unternehmen, nennen wir es SZ1, das eine rechenintensive Anwendung ausführen möchte zu kommerziellen Zwecken zum eigenen Nutzen. Damit SZ1 die gewünschten Rechenergebnisse schnell bekommt, besitzt SZ1 mehrere Computing Resources auf denen die Anwendung parallel ausgeführt werden kann. Diese Computing Resources sind innerhalb eines Grids angesiedelt, das von SZ1 aufgesetzt wurde und auf das nur SZ1 Zugriff hat. Es handelt sich also um ein privates Grid.
Die Software, die SZ1 einsetzen möchte, wird mittels dem in der Diplomarbeit entwickelten Lizenzmodell lizenziert und vertrieben. Nachdem SZ1 den Lizenzvertrag mit seinem Distributor ausgehandelt hat, erhält SZ1 die Software, den *License-Client*, den *License-Server* sowie die deploybaren Web Services, die zum Lizenzmodell gehören. SZ1 ist nun verantwortlich dafür, dass die Services in den Container der eingesetzten Middleware deployt werden und der *License-Server* als WS-Resource gestartet wird. Außerdem muss SZ1 seinen Mitarbeitern einen *License-Client* zur Verfügung stellen. Anschließend muss SZ1 seine VOMS-Datenbank konfigurieren und Benutzer-Berechtigungen eintragen. Dabei kann SZ1 bestimmen, welche Benutzer, Abteilungen oder sonstige Entitäten Lizenzen anfordern dürfen. Diese Berechtigungszuteilung ist analog zu anderen VOMS-Autorisierungen und -Gruppierungen. Als letztes muss SZ1 sicherstellen, dass der ISV beziehungsweise der zuständige Distributor Zugriff auf die Datenbank des *License-Servers* bekommt, damit er dort die Lizenzinformationen hinterlegen kann.
Nachdem dies alles erledigt ist, kann die Anwendung benutzt werden und der Lizenzierungsvorgang läuft wie in diesem Kapitel beschrieben ab.

5.12.2 Szenario 2: Verteiltes Grid einer Virtuellen Organisation

Dieses Szenario beschreibt eine Virtuelle Organisation, die aus Personen und Ressourcen verschiedener realer Organisationen besteht. Aus den in Kapitel 2 beschriebenen Gründen wurde diese Virtuelle Organisation gegründet. Ein Unternehmen dieser Virtuellen Organisation bezeichnen wir als SZ2a, ein anderes Unternehmen als SZ2b.
Diese Virtuelle Organisation bietet nun jeder Person der beteiligten Unternehmen die Möglichkeit, alle beteiligten Computing Resources zu nutzen.
SZ2a möchte dieses gemeinsame Grid nun für eine rechenintensive Anwendung nutzen. Die Software, die SZ2a einsetzen möchte, wird mittels dem in der Diplomarbeit entwickelten Lizenzmodell lizenziert und vertrieben. Nachdem SZ2a den Lizenzvertrag mit seinem Distributor ausgehandelt hat, erhält SZ2a die Software, den *License-Client*, den *License-Server* sowie die deploybaren Web Services, die zum Lizenzmodell gehören. SZ2a ist nun verantwortlich dafür, dass die Services in den Container der eingesetzten Middleware des Grids der VO deployt werden und der *License-Server* als WS-Resource gestartet wird. Außerdem muss SZ2a seinen Mitarbeitern, die der VO angehören, einen *License-Client* zur Verfügung stellen. Anschließend muss SZ2a seine VOMS-Datenbank konfigurieren und Benutzer-Berechtigungen eintragen. Dabei kann SZ2a bestimmen, welche Benutzer, Abteilungen oder sonstige Entitäten Lizenzen anfordern dürfen. Diese Berechtigungszuteilung ist analog zu anderen VOMS-Autorisierungen und -Gruppierungen. Als letztes muss SZ2a sicherstellen, dass der ISV beziehungsweise der zuständige Distributor Zugriff auf die Datenbank des *License-Servers* bekommt, damit er dort die Lizenzinformationen hinterlegen kann. Nachdem dies alles erledigt ist, kann die Anwendung benutzt werden und der Lizenzierungsvorgang läuft wie in diesem Kapitel beschrieben ab.
SZ2b hat keine Möglichkeit die Lizenzen zu nutzen, da es nicht per VOMS autorisiert ist. Sollte Mitarbeitern aus SZ2b Rechte gegeben werden, so wird die Lizenzierung dennoch vom *License-Server* abgelehnt, da SZ2b kein Kunde ist und somit keine Lizenzrechte besitzt.

5.12.3 Szenario 3: Mehrere Lizenzpools in einer Virtuellen Organisation

Dieses Szenario ist ähnlich zu Szenario 2. Jedoch erwirbt Unternehmen SZ2b nun ebenfalls Lizenzen derselben Anwendung. Da bereits die grundlegende Infrastruktur in diesem Grid vorhanden ist, kann der ISV beziehungsweise der Distributor nun diese weiteren Lizenzen einfach in der Datenbank des *License-Servers* hinzufügen. SZ2b ist nun für die korrekte Konfiguration von VOMS verantwortlich, so dass seine Mitarbeiter, die Teil der Virtuellen Organisation sind, für die Lizenznutzung autorisiert werden.
Zu diesem Zeitpunkt kann sowohl SZ2a als auch SZ2b seine eigenen Lizenzen im gemeinsamen Grid nutzen. Man kann dieses Szenario nun noch weiter ausbauen und SZ2a und SZ2b gehen einen gemeinsamen Vertrag ein im Sinne der Virtuellen Organisation und bestimmen, dass beide Unternehmen auch die Lizenzen des jeweils anderen Unternehmen (unter bestimmten Bedingungen, zum Beispiel Zeitintervallen) nutzen darf. SZ2a und SZ2b müssen dann beim ISV beziehungsweise Distributor eine Zusammenlegung der Lizenzpools beantragen. Anschließend treten die beiden Unternehmen nur noch als *ein* gemeinsames Unternehmen gegenüber dem ISV beziehungsweise Distributor auf. Sie handeln also als Virtuelle Organisation. Welches Unternehmen nun wessen Lizenzen zu welchem Zeitpunkt nutzen darf, muss die Virtuelle Organisation bestimmen. Das gemeinsame Lizenzpaket ist mit dem in dieser Diplomarbeit entwickelten Lizenzmodell lizenzierbar.

5.13 Zusammenfassung des 5. Kapitels

In diesem Kapitel wurde ein Lizenzmodell entworfen, das den Anforderungen des Anforderungskatalogs weitestgehend entspricht und für den Einsatz in Grid-Umgebungen einer Virtuellen Organisation entworfen wurde.
Zuerst wurde das Lizenzmodell in seinen Grundzügen dargestellt und kurz die Eigenheiten und Voraussetzungen für den Einsatz dargelegt. Dabei wurde insbesondere erklärt, dass sich dieses Lizenzmodell für den Einsatz in privaten Grids als auch in verteilen Grids von Virtuellen Organisation einsetzen lässt. Außerdem wurde das Lizenzmodell in seiner Gesamtheit dargestellt.
Anschließend wurde das Modell in seine logischen Bestandteile zerlegt und der Ablauf innerhalb dieser Bestandteile erläutert, so dass ein globaler Überblick über das Lizenzmodell hergestellt wurde. Zuerst wurde hier erklärt, wie der Benutzer eine Lizenz für seine Anwendung beantragen kann, welche Komponenten daran beteiligt sind und wie die Kommunikation prinzipiell abläuft. Das Starten des Jobs läuft analog zur klassischen Grid-Nutzung ab und die zuvor erhaltene Lizenz wird als Teil der Input-Daten des Jobs übergeben. Die Aktivierung der Lizenz und die Durchführung des Jobs läuft automatisiert ab.
Anschließend wurde die Möglichkeit zur Abfrage von Status-Informationen erklärt. Zu guter Letzt wurde die Funktionalität zur Reservierungsänderung oder Lizenzerweiterung dargetellt und deren Ablauf definiert.
Die beteiligten Komponenten wurden ausführlich dargestellt.
Der *License-Client* dient als Schnittstelle zwischen Benutzer und Lizenzmodell und übernimmt daher vielfältige Aufgaben.
Der *License-Server* ist als WS-Resource im Grid deployed und ist insbesondere für die Verwaltung und Zuteilung von Lizenzen verantwortlich. Außerdem übernimmt er die Aktivierung von Lizenzen.
Der *License-Service* sowie der *Billing-Service* sind als Web Services innerhalb des Grids deployed. Sie können auch mehrfach auftreten und somit als Pfad agieren, um die Anfragen von *License-Client* zum *License-Server* beziehungsweise *Application* zum *License-Server* über mehrere Ressourcen zu leiten.
Die *Application* übernimmt die Aktivierung und das CheckIn von Lizenzen und kümmert sich außerdem darum, dass die eigentliche Anwendung als Job gestartet wird. Außerdem ist die *Application* für die ständige Überwachung der Gültigkeit der Lizenzen verantwortlich.
Nach der genauen Funktionsbeschreibung und Definition der einzelnen Komponenten, wurden die rechtlichen Aspekte des Lizenzmodells erklärt und darauf eingegangen, was es für rechtliche Voraussetzungen für den Einsatz des Modells gibt.
Bei der abschließenden Evaluierung des Lizenzkonzepts gegen den Anforderungskatalog hat sich gezeigt, dass nahezu alle Anforderungen erfüllbar sind und man nur sehr geringe Abstriche machen muss. Dieser Abschnitt endete mit einer tabellarischen Übersicht.
Abschließend wurden einige Anwendungsszenarien für das Lizenzmodell beschrieben und der Einsatz des Modells in diesen Szenarien dargestellt.
Es hat sich also gezeigt, dass es möglich ist, ein Lizenzmodell für Grid-Umgebungen zu konzeptionieren, das auf Grid-spezifischen Technologien beruht. Der gesamte logische Ablauf basiert auf Web Services und den dazugehörigen Web Service Resources, also stateful Web Services mit Java-Klassen.

6 Entwicklung eines Prototypen

Im vorangegangenen Kapitel wurde ein Lizenzmodell auf Basis des Anforderungskatalogs konzeptioniert. Dieses Lizenzmodell soll nun prototypisch implementiert werden. Dieses Kapitel befasst sich mit der Dokumentation der Entwicklung dieses Prototypen sowie der Erläuterung seiner Funktionsweise.
Die gesamte Implementierung, sprich der Eclipse-Workspace und die Web Service Resources, liegen als CD dieser Diplomarbeit bei. Der gesamte Quellcode ist viel zu umfangreich, um ihn in der Diplomarbeit abzudrucken.

6.1 Allgemeines

Dieser Abschnitt befasst sich damit, wie an die Entwicklung des Prototypen herangegangen wurde und welche Voraussetzungen und Vorbereitungen getroffen wurden.

6.1.1 Beschreibung der Umgebung

Die Entwicklung des Prototypen wurde auf privaten Computern mit einem privaten Grid auf Basis von GT4 durchgeführt, da die von der Universität bereitgestellte Virtuelle Maschine unvorhersehbare Probleme aufgeworfen hat (siehe 6.10.2).
Die eingesetzte Umgebung basiert auf folgenden Komponenten:

- **openSuse 11.2**
  Das Betriebssystem, mit dem sowohl der Computer zur Entwicklung als auch die Computer des Grids betrieben werden, ist *openSuse 11.2* ([ope10b]) mit den aktuellsten Sicherheits- und Produktupdates.

- **Java**
  Als Java-Umgebung wird sowohl die Laufzeit- (JRE) als auch die Entwicklungsumgebung (JDK) von Sun in der aktuellsten Version 1.6 Update 18 ([Sun10a]) eingesetzt.

- **Globus Toolkit 4.0.8**
  Das Grid basiert auf der Middleware Globus Toolkit 4.0.8 ([The10]).

- **PostgreSQL**
  Da das Globus Toolkit 4 eine Datenbank im Hintergrund benötigt, wurde dafür PostgreSQL 8.4.2-1.1.1 ([Pos10]) eingesetzt und für die GT4-spezifischen Anforderungen konfiguriert und die notwendigen Tabellen angelegt.

- **Eclipse**
  Als Entwicklungsumgebung für die Java-Klassen, Web Services und Web Service Resources wurde das Eclipse IDE for Java EE Developers eingesetzt. Die benutzte Version basiert auf Eclipse 3.5 SR2 und ist zu beziehen von [Ecl10b].

- **g-Eclipse**
  Zum Arbeiten mit dem Grid wurde g-Eclipse 1.0 RC3 verwendet. Diese Version erhält man von [g-E09].

- **MySQL**
  Als Datanbank für den *License-Server* wurde MySQL 5.1.36-6.7.2 eingesetzt. Der MySQL-Server ist zu beziehen von [MyS10a].

6.1.2 Eingesetzte Tools

Dieser Abschnitt beschreibt die bei der Entwicklung zusätzlich eingesetzten Tools.

- **Keytool**
  Das Keytool ist ein Tool zur Generierung von (selbstsignierten) Zertifikaten. Es liegt dem Java-Paket von Sun bei. Mehr Informationen sind unter [Sun10b] nachzulesen.

- **Globus Service Build Tools**
  Zum Packen aller Dateien einer WS-Resource in ein Grid-Archiv (.gar-File) wurden die Globus Service Build Tools benutzt. Diese Tools dienen dazu, dass alle Dateien in ihrer notwendigen Ordner-Struktur innerhalb des Archivs gespeichert werden, so dass dieses Archiv danach direkt in die Grid-Umgebung deployed werden kann. Die Globus Service Build Tools sind downloadbar von [SLS10].

6.2 Der Eclipse-Workspace

In diesem Abschnitt wird die Package-Struktur des gesamten Projekts beschrieben.

6.2.1 Package-Struktur

/
Im Wurzelverzeichnis befinden sich die Globus Service Build Tools bestehend aus globus-build-service.sh, globus-build-service.py, build.xml und LICENSE. Außerdem sind in diesem Verzeichnis einige Shellscripte zur einfacheren Generierung der .gar-Files und zur Durchführung der Deployments angelegt worden.
Wichtig zu erwähnen ist noch die namespace2package.mappings-Datei, die dazu dient, die Namespaces der WSDL-Dateien in einfachere Package-Strukturen abzubilden. Außerdem sind hier die fertigen deploybaren Grid-Archive abgespeichert.

- **/application**
  Dieses Package beinhaltet die Java-Klassen der Anwendung, die lizenziert werden soll.
- **/build**
  - **/build/classes**
    Dieses Package beinhaltet die kompilierten Klassen der WS-Resources.
  - **/build/lib**
    Hier sind die gepackten Grid Archive untergebracht.
  - **/build/schema**
    In diesem Ordner sind alle notwendigen WSDL- und XSD-Dateien zu finden, die die Web Services beziehungsweise WS-Resources benötigen.
  - **/build/stubs**
    Dieses Package beinhaltet alle kompilierten Klassen der WS-Resources.
  - **/build/stubs-org_globus_BillingService**
    Dieser Ordner dient im Prinzip als temporärer Ordner zur Ablage der Java-Dateien. Diese Dateien werden beim Packen in ein Grid-Archiv kompiliert. Die Package-Definition in den jeweiligen Java-Dateien ist daher relativ zu sehen und nicht absolut.
  - **/build/stubs-org_globus_LicenseServer**
    Selbige Funktion wie stubs-org_globus_BillingService.
  - **/build/stubs-org_globus_LicenseService**
    Selbige Funktion wie stubs-org_globus_BillingService.
- **/licenseClient**
  Dieses Package beinhaltet die Klassen des *License-Clients*.
- **/licenseLogger**
  Der *License-Logger* ist eine Klasse, die der *License-Client* für das Logging benutzt.

- **/org/globus**
  - **/org/globus/BillingService**
    Hier sind notwendige Dateien für das Deployment des *Billing-Services* wie zum Beispiel deploy-jndi-config.xml und deploy-server.wsdd unterbracht.
    - **/org/globus/BillingService/impl**
      In diesem Package befinden sich die Java-Dateien des *Billing-Services*.
  - **/org/globus/LicenseServer**
    Die Dateien für das Deployment des *License-Servers* wie zum Beispiel deploy-jndi-config.xml und deploy-server.wsdd sind hier gespeichert.
    - **/org/globus/LicenseServer/impl**
      In diesem Package befinden sich die Java-Dateien des *License-Servers*.
  - **/org/globus/LicenseService**
    Die Dateien deploy-jndi-config.xml und deploy-server.wsdd sowie weitere für das Deployment zuständige Dateien des *License-Services* sind hier unterbracht.
    - **/org/globus/LicenseService/impl**
      In diesem Package befinden sich die Java-Dateien des *License-Services*.
  - **/stubs**
    - **/stubs/BillingService_instance**
      Hier befinden sich die Java-Dateien der Tokens und Properties, die der *Billing-Service* bearbeitet beziehungsweise besitzt. Außerdem sind in einigen Unterordnern noch Java-Dateien zur Adressierung der WS-Resource sowie für das SOAP-Binding untergebracht.
    - **/stubs/LicenseServer_instance**
      Analog zu BillingService_instance.
    - **/stubs/BillingService_instance**
      Analog zu BillingService_instance.
- **/schema**
  Hier sind in in den jeweiligen Packages die WSDL-Dateien für den *Billing-Service*, *License-Server* und *License-Service* untergebracht.

6.2.2 Vorbereitung des Eclipse-Workspaces

Die Entwicklung des Prototypen wurde unter der Einbeziehung einiger fremder Packages vorgenommen. Diese müssen bei Eclipse als Build-Path sowie als Class-Path konfiguriert werden. Der, dieser Diplomarbeit auf CD beiliegende Workspace, hat diese Pakete bereits richtig eingebunden, jedoch ist es möglich, dass auf anderen Computern die Pfade anders lauten, so dass diese angepasst werden müssen, damit die WS-Resources und die anderen Java-Anwendungen kompilieren.
Folgende fremde Jar-Archive wurden in die Entwicklung miteinbezogen:

- **Globus-Toolkit 4 Bibliotheken**
  Die GT4-Bibliotheken sind im Unterverzeichnis lib im GT4-Installationsordner zu finden.

- **Apache Axis Bibliotheken**
  Die Adressierung der WS-Resources basiert bei Verwendung von GT4 auf Apache Axis. Die Bibliotheken wurden daher verwendet.

- **MySQL Connector**
  Zur Anbindung des *License-Servers* an die MySQL-Datenbank wurde der MySQL Connector 5.1.6 verwendet. Dieser ist von [MyS10b] zu beziehen. Dieses Jar-Archiv muss in das lib-Verzeichnis der GT4-Installation kopiert werden.

- **JavaMail**
  Die Bibliothek JavaMail 1.4.3, die von [Ora10] zu beziehen ist, wurde ebenfalls verwendet.

Zusätzlich ist zu beachten, dass das GT4-Installationsverzeichnis zur Kompilierung in den Class-Path miteinbezogen wird.
Wenn dies alles berücksichtigt ist, kann der Workspace erfolgreich kompilieren.

6.3 Allgemeine Vorbereitungen

In diesem Abschnitt werden allgemeine Vorbereitungen beschrieben. Diese sind vor der eigentlichen Entwicklung zu treffen, da der Prototyp darauf basiert.

6.3.1 Installation und Konfiguration der Zertifikate

Da sowohl der *License-Client*, der *License-Server* als auch die *Application* die jeweiligen Tokens signieren und die erhaltenen Signaturen prüfen, benötigen alle jeweils (mindestens) ein Zertifikat. Das eigene muss in dem jeweiligen KeyStore gespeichert werden. Die fremden vertraulichen Zertifikate müssen in den zugehörigen TrustStores gespeichert werden. Das Generieren und Abspeichern der (selbstsignierten) Zertifikate erfolgt mittels dem Javabeiliegenden Tool „keytool“ ([Sun10b]).
Zuerst wird ein Schlüsselpaar für den *License-Client* erstellt mittels

```
keytool -genkey -alias licenseClient -keyalg DSA
-validity 365 -keystore clientkeystore
```

Man gibt die geforderten Daten ein und exportiert anschließend das Zertifikat mittels

```
keytool -export -alias licenseClient -file clientcert.cer
-keystore clientkeystore
```

Als Passwort wurde *clientpass* und *clientkeystorepass* gewählt.
Analog wird ein Schlüsselpaar und ein Zertifikat für den *License-Server* erstellt:

```
keytool -genkey -alias licenseServer -keyalg DSA
-validity 365 -keystore serverkeystore

keytool -export -alias licenseServer -file servercert.cer
-keystore serverkeystore
```

Für die *Application* wird analog ebenfalls ein Schlüsselpaar und Zertifikat erstellt.

```
keytool -genkey -alias Application -keyalg DSA
-validity 365 -keystore appkeystore

keytool -export -alias Application -file appcert.cer
-keystore appkeystore
```

Anschließend werden die jeweiligen TrustStores gefüllt.

```
keytool -import -file appcert.cer -alias Application
-keystore clienttruststore -storepass clienttruststorepass

keytool -import -file servercert.cer -alias licenseServer
-keystore clienttruststore -storepass clienttruststorepass

keytool -import -file appcert.cer -alias Application
-keystore servertruststore -storepass servertruststorepass
```

```
keytool -import -file clientcert.cer -alias licenseClient
-keystore apptruststore -storepass apptruststorepass

keytool -import -file clientcert.cer -alias licenseClient
-keystore servertruststore -storepass servertruststorepass

keytool -import -file servercert.cer -alias licenseServer
-keystore apptruststore -storepass apptruststorepass
```

Das Host-Zertifikat der Grid-Computing-Resource muss den TrustStores ebenfalls hinzugefügt werden. Da es sich dabei um ein Zertifikat im PEM-Format handelt und die TrustStores nur DER-Formate akzeptieren, muss das Host-Zertifikat erst umgewandelt werden.

```
openssl x509 -in /etc/grid-security/hostcert.pem
-inform PEM -out ./hostcert.der -outform DER
```

Danach kann es den TrustStores hinzugefügt werden.

```
keytool -import -file hostcert.der -alias LicenseService
-keystore clienttruststore -storepass clienttruststorepass

keytool -import -file hostcert.der -alias LicenseService
-keystore servertruststore -storepass servertruststorepass

keytool -import -file hostcert.der -alias LicenseService
-keystore apptruststore -storepass apptruststorepass
```

Da der Benutzer ebenfalls ein Zertifikat benötigt, wird für diesen auch ein Schlüsselpaar und Zertifikat angelegt.

```
keytool -genkey -alias hoeber -keyalg DSA -validity 365
-keystore userkeystore

keytool -export -alias hoeber -file usercert.cer
-keystore userkeystore
```

Nun sind die notwendigen Zertifikate generiert. Zu beachten ist, dass die jeweiligen Anwendungen und WS-Resources Zugriff auf „ihre“ Zertifikate, KeyStores und TrustStores haben und dass die Pfade und Passwörter, die prototypischerweise hart in den Java-Dateien gesetzt sind, korrekt sind.
Selbstverständlich darf in einer Produktivumgebung nicht mit selbstsignierten Zertifikaten gearbeitet werden, sondern die Zertifikate müssen mindestens von einer eigenen betrieblichen CA unterzeichnet sein! Besser wäre jedoch sogar ein Zertifikat ausgehend von einer global anerkannten vertraulichen Zertifizierungsstelle.

6.3.2 Vorbereitung der MySQL-Datenbank

Der *License-Server* benötigt seine MySQL-Datenbank zur Verwaltung der Kundenstammdaten sowie der reservierten, zugeteilten, stornierten und benutzten Lizenzen. Dazu verwendet der Prototyp die Datenbank „LicenseServer“ in einem MySQL-Server.
Der MySQL-Server muss folglich dahingehend administriert werden, dass diese Datenbank angelegt wird und ein Benutzer konfiguriert wird, der Zugriff auf diese Datenbank hat. Im Prototyp verwendet der *License-Server* einen Benutzer, der hart in der Java-Klasse festgelegt ist.
Die Datenbank besitzt zwei simple Tabellen.

- **user**
  Diese einfache Tabelle repräsentiert die Kundendaten und beschreibt, wieviele Lizenzen der jeweilige Kunde besitzt. Sie besteht aus dem Schema:
  - userID (int)
  - Name (varchar)
  - ApplicationID (varchar)
  - LicenseAmount (varchar)
- **LicenseTable**
  Diese Tabelle beinhaltet alle Lizenzen, die reserviert, benutzt, storniert sind und ehemals in Benutzung waren.
  Sie besteht aus dem Schema:
  - ID (int)
  - userID (varchar)
  - ApplicationID (varchar)
  - InputDataHashCode (int)
  - amount (int)
  - licenseOwner (varchar)
  - begin (datetime)
  - ending (datetime)
  - status (varchar)
  - crypto (int)
  - counter (int)
  - jobID (varchar)
  - licenseType (int)
  - beginUse (datetime)
  - endingUse (datetime)
  - computingResource (varchar)

Diese Datenbank-Struktur wird vom *License-Server*-Prototyp vorausgesetzt. Dabei gilt zu beachten, dass die Datenbankstruktur weder in einer Normalform noch vollständig für eine Produktivumgebung ist, sondern nur eine prototypische Struktur darstellt.

6.4 Allgemeine Hinweise zum Prototypen

Die gesamte Implementierung stellt einen Prototypen dar. Das bedeutet, sie greift in einigen Stellen auf rudimentäre Methoden und Dummies zurück. Außerdem deckt sie keinesfalls alle notwendigen Vorkehrungen für den Einsatz in einer Produktivumgebung ab.
Einige Konfigurationen sind *hard-coded* und nur in den jeweiligen Klassen änderbar. Damit das gesamte Projekt läuft, müssen eventuell Pfade zu eingebundenen Jars oder der Inhalt der Config-Files angepasst werden. Jedoch ist zu beachten, dass der *License-Client*, der *License-Server* und die *Application* hart auf ihre Config-Dateien sowie auf ihre Zertifikate, KeyStores und TrustStores linken. Dies ist für einen Prototypen denkbar, jedoch für eine Produktivumgebung nicht sinnvoll.
Der Zugriff auf die MySQL-Datenbank erfolgt unverschlüsselt! Die Datenbank ist nicht gegen Einsicht und Manipulation Dritter geschützt. Die Zugriffsdaten auf die Datenbank sind in der Config-Datei des *License-Servers* gespeichert und nicht geschützt! Eine nicht-prototypische Version muss auf gängige Sicherheitsmaßnahmen in dieser Hinsicht zurückgreifen.
Alle Java-Klassen sind ungeschützt und daher leicht dekompilierbar, was ein erhebliches Sicherheitsrisiko für den Lizenzierungsvorgang darstellt. Dies ist für einen Einsatz in einer realen Umgebung nicht vertretbar. Die Klassen müssen mit bekannten Mitteln wie zum Beispiel Obfuscation geschützt werden.
Der *License-Client* kommt als Prototyp komplett ohne GUI aus. Alle notwendigen Daten eines Auftrags werden als Eingabeparameter bei Aufruf des *License-Clients* übergeben und basieren auf einer festen eineindeutigen Syntax.

6.5 Die Implementierung des License-Clients

Der *License-Client* besteht aus zwei einzelnen Java-Dateien. Zum einen aus seiner in der LicenseClient.java implementierten Logik, zum anderen aus der LicenseLogger.java, die für einfaches Logging zuständig ist.

6.5.1 Die Logik

Die LicenseClient-Klasse importiert zunächst alle notwendigen Packages. Darunter fallen die Java-internen Packages zum Erstellen und Lesen von Dateien aus dem Dateisystem und die Packages zum Zugriff und der Verwaltung auf Zertifikate, KeyStores und TrustStores. Außerdem benötigt die LicenseClient-Klasse selbstverständlich einige Packages von GT4 und Axis zum Zugriff auf Web Services und WS-Resources innerhalb dem Grid. Damit die LicenseClient-Klasse die Tokens generieren und versenden beziehungsweise empfangen und interpretieren kann, müssen die Tokens ebenfalls importiert werden.
Beim Start des *License-Clients* werden zunächst die Eingabeparameter geprüft. Bei falschen oder ungenügenden Eingabepararametern oder wenn die Parameter in falscher Reihenfolge eingegeben wurden, wird der *License-Client* beendet und eine kleine Hilfestellung ausgeben (siehe 6.5.2).
Bei positiver Prüfung wird eine Selbstinitialisierung vorgenommen. Dabei werden die EndpointReferences des *License-Services* und des *Billing-Services* generiert und Referenzen auf die Service-Ports angelegt.
Anschließend werden einige Variablen gesetzt und die Selbstinitialisierung mit dem Einlesen der KeyStores und TrustStores und der Gewinnung der jeweiligen Keys abgeschlossen.
Nach erfolgreich abgeschlossener Selbstinitialisierung führt der *License-Client* den Wunsch des Benutzers aus. Je nach Aufgabe wird das zugehörige Token-Objekt initialisiert und die Parameter basierend auf den Eingabeparameter gesetzt. Anschließend werden die Daten des Tokens algorithmisch miteinander zu einem weiteren Objekt verknüpft. Dieses Objekt wird mit dem privaten Schlüssel des *License-Clients* signiert. Das signierte Objekt und die Signatur werden dem jeweiligen Token-Objekt übergeben. Damit ist die Generierung des Token-Objekts abgeschlossen.
Anschließend erfolgt der Aufruf des zuständigen Web Services und seinem Dienst mit der Übergabe des Tokens als Eingabeobjekt. Der *License-Client* wartet nun auf Antwort des beauftragten Web Services.
Der *License-Client* erhält vom beauftragten Web Service eine Antwort (wenn kein Fehler aufgetreten ist). Das erhaltene Objekt ist ein Token-Objekt. Der *License-Client* prüft alle zu prüfenden Signaturen auf Gültigkeit und analysiert somit die Echtheit des Token-Objekts. Wenn das Token-Objekt als unverfälscht eingestuft wurde und von einer vertraulichen Instanz signiert wurde, schreibt der *License-Client* das Token-Objekt mittels einem ObjectOutputStream in das /tmp-Verzeichnis zur weiteren Verwendung seitens des Benutzers.

6.5.2 Bedienung

Der *License-Client* lässt sich folgendermaßen bedienen:

- **Beauftragung zur Beschaffung eines License-Tokens**
  Folgende Parameter müssen in exakt dieser Reihenfolge übergeben werden:
  - getLicense
  - Pfad und Dateiname zum eigenen Zertifikat
  - Eingabedaten für den Grid-Job
  - Anwendungs-ID
  - Lizenz-Eigentümer
  - Anzahl der gewünschten Lizenzen
  - Typ der Lizenzen (1 = Floating; 2= PayPerUse)
  - Datum und Uhrzeit des Reservierungsbeginns (YYYY-MM-DD-HH-MM)
  - Datum und Uhrzeit des Reservierungsende (YYYY-MM-DD-HH-MM)
- **Beauftragung zum Update einer reservierten Lizenz**
  Folgende Parameter müssen in exakt dieser Reihenfolge übergeben werden:
  - updateLicense
  - Absoluter Pfad und Dateiname zum eigenen Zertifikat
  - Anzahl der gewünschten Lizenzen (updatefähig)
  - Typ der Lizenzen (1 = Floating; 2= PayPerUse) (updatefähig)
  - Datum und Uhrzeit des Reservierungsbeginns (YYYY-MM-DD-HH-MM) (updatefähig)
  - Datum und Uhrzeit des Reservierungsende (YYYY-MM-DD-HH-MM) (updatefähig)
  - Absoluter Pfad zum bereits erhaltenen *License-Token*
- **Beauftragung zum manuellen CheckIn**
  Folgende Parameter müssen in exakt dieser Reihenfolge übergeben werden:
  - manualCheckIn
  - Absoluter Pfad und Dateiname zum eigenen Zertifikat
  - Absoluter Pfad zum einzucheckenden CheckInRequest-Objekt
- **Beauftragung zur Beschaffung von Informationen**
  Folgende Parameter müssen in exakt dieser Reihenfolge übergeben werden:
  - getInformation
  - Absoluter Pfad und Dateiname zum eigenen Zertifikat
  - Absoluter Pfad zu einem LicenseToken-Objekt, von dem aktuelle Informationen beschafft werden sollen

6.6 Die Implementierung des License-Servers

Der *License-Server* ist als WS-Resource implementiert. Das heißt, er besteht aus einem Web Service mit zugehörigen Java-Klassen und ist stateful. Grundsätzlich besteht der *License-Server* somit aus Web Service-spezifischen Dateien (WSDL-File, JNDI-Descriptor, WSDD-Definitions) zur Beschreibung des Web Services sowie aus mehreren Java-Dateien, die die Logik der WS-Resource implementieren. Außerdem gehören zum *License-Server* die Implementierungen der Tokens, die er interpretiert und generiert (siehe Package-Struktur in 6.2.1).

6.6.1 WSDL, WSDD und JNDI

Das WSDL-File des *License-Servers* definiert die Schnittstellen (Dienste) und Properties.

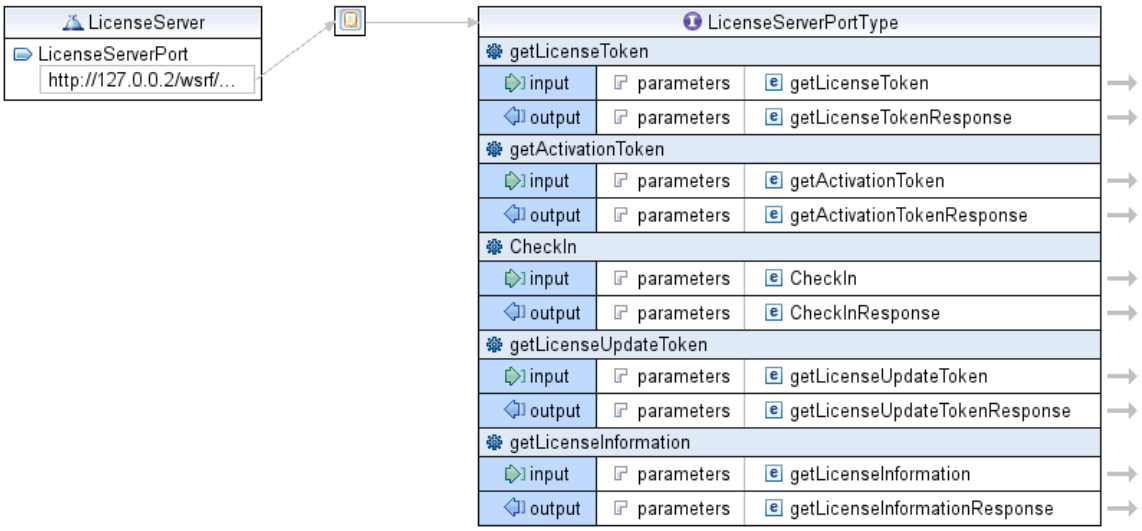

Abbildung 6.1: Grafische Darstellung des WSDL-Files des License-Servers

Obiges Bild zeigt eine grafische Übersicht über das WSDL-File des *License-Servers*. Es ist ersichtlich, dass der *License-Server* eine Vielzahl von Tokens entgegen nimmt und wieder ausgibt.

Beispielhaft für die anderen sehr ähnlichen Tokens ist im Folgenden das *License-Token* dargestellt. Man erkennt die Parameter des *License-Tokens* und deren jeweiligen Datentypen. Alle anderen Tokens sind in ihrer Struktur und Aufbau analog zum *License-Token*, haben jedoch eventuell weitere Attribute.

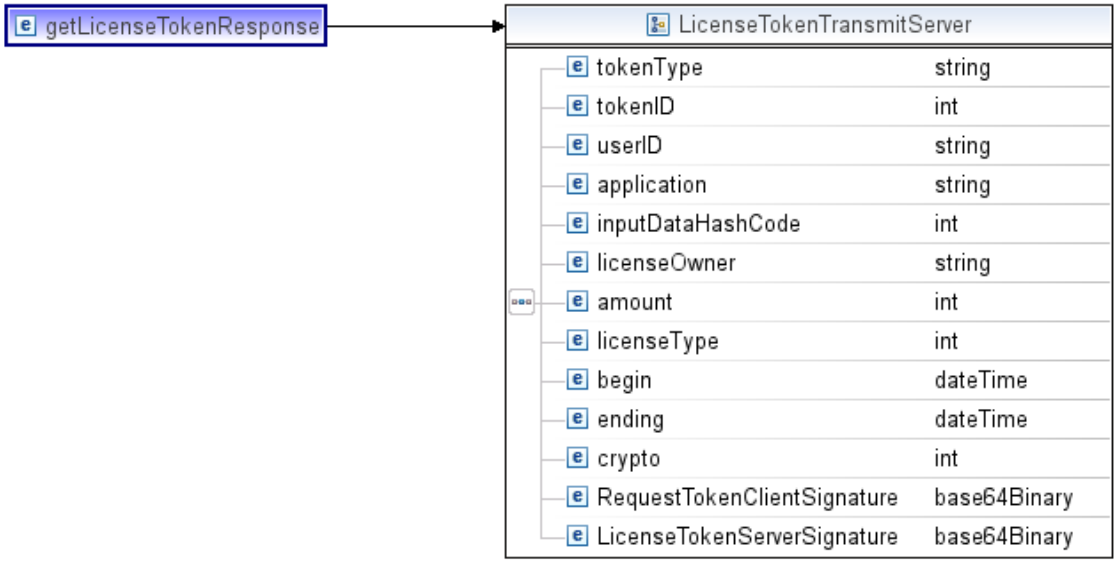

Abbildung 6.2: Grafische Darstellung des License-Tokens

Das WSDL-File wurde in der Grundstruktur mit dem Editor von Eclipse generiert. Einige Feinheiten jedoch wurden manuell im Code angepasst und erweitert.
Damit der *License-Server* in die Grid-Umgebung deployed werden kann, sind noch ein JNDI-Descriptor sowie eine WSDD-Datei notwendig. Diese beinhalten Konfigurationsparameter und sind relativ straight-forward zu schreiben.

6.6.2 Die Logik

Die Logik des *License-Servers* ist in mehrere Java-Klassen aufgeteilt. Die eigentliche Logik und Funktionalität ist in der LicenseServer.java-Datei untergebracht.
Als erstes werden alle benötigten Packages und Klassen importiert. Darunter fallen die Java-internen Packages zum Erstellen und Lesen von Dateien aus dem Dateisystem und die Packages zum Zugriff und der Verwaltung auf Zertifikate, KeyStores und TrustStores. Selbstverständlich benötigt die LicenseServer-Klasse einige Packages des Globus Toolkit 4 sowie von Axis, da es sich um eine WS-Resource handelt. Damit die LicenseServer-Klasse die Tokens generieren und versenden beziehungsweise empfangen und interpretieren kann, müssen die Tokens ebenfalls importiert werden. Außerdem ist für die Kommunikation und den Zugriff auf die MySQL-Datenbank der Import der dafür verwendeten Packages notwendig.
Beim Deployment des *License-Servers* und Start des GT4-Containers, der den *License-Server* hostet, werden zunächst die Konfigurationsparameter eingelesen. Anschließend wird eine Selbstinitialisierung vorgenommen. Diese beinhaltet die Konfiguration der Datenbank-

schnittstellenobjekte sowie das Setzen einiger Variablen. Die Selbstinitialisierung wird mit dem Einlesen der KeyStores und TrustStores und der Gewinnung der jeweiligen Keys abgeschlossen.
Nach erfolgreich abgeschlossener Selbstinitialisierung ist die WS-Resource bereit und wartet auf Aufgaben, die sie von der WS-Resource des *License-Services* oder von der WS-Resource des *Billing-Services* erhält. Dieser Zustand ist der StandBy-Zustand der *License-Server*-WS-Resource.
Der *License-Server* als WS-Resource bietet verschiedene Dienste an (siehe 6.6.1). Diese Dienste sind WS-typisch als Java-Methoden implementiert.

getLicenseToken

Bei diesem Serviceaufruf wird ein RequestToken-Objekt als Parameter übergeben.
Das RequestToken-Objekt wird initialisiert und die Signatur und somit der Inhalt des Tokens auf Gültigkeit geprüft. Im Falle eines gültigen Tokens wird mittels einem SQL-Aufruf in der MySQL-Datenbank nachgeschaut, ob der anfragende Benutzer berechtigt ist, Lizenzen zu erhalten. Wenn er das ist, wird ein neues SQL-Statement generiert, das dazu dient, die Daten des Tokens in die Datenbank einzutragen. Der Request ist somit gespeichert. Abschließend wird ein LicenseToken-Objekt generiert, das zusätzlich zum Inhalt des RequestToken-Objekts die Signatur des *License-Servers* enthält.
Sollte es nicht möglich sein, dem Benutzer seinen Wunsch zu erfüllen, wird ein RejectToken-Objekt generiert, das prinzipiell identisch zum LicenseToken-Objekt ist, jedoch das tokenType-Flag anders gesetzt hat.
Das generierte Objekt wird als Antwort der aufrufenden *License-Service*-WS-Resource übergeben.

getActivationToken

Bei diesem Serviceaufruf wird ein ActivationRequest-Objekt als Parameter übergeben.
Das ActivationRequest-Objekt wird initialisiert und die Signatur des *License-Servers* geprüft, der das zugehörige *License-Token* ausgestellt hat. Zusätzlich wir die Signatur der *Application* geprüft, die den *Activation-Request* generiert hat. Folglich wird der Inhalt des Tokens auf Gültigkeit geprüft. Im Falle eines gültigen Tokens wird mittels einem SQL-Aufruf in der MySQL-Datenbank nachgeschaut, ob es zu dem *Activation-Request* einen zugehörigen Eintrag gibt. Wenn dies der Fall ist, wird ein neues SQL-Statement generiert, das in der Datenbank die startTime des zugehörigen Datensatzes auf die aktuelle Uhrzeit und Datum setzt.
Abschließend wird ein ActivationToken-Objekt generiert, das zusätzlich zum Inhalt des ActivationRequest-Objekts die Signatur des *License-Servers* enthält.
Sollte es nicht möglich sein, die Lizenz zu aktivieren oder ein Fehler mit der Datenbank auftreten, wird ein ActivationReject-Objekt generiert. Dieses ist prinzipiell identisch zum ActivationToken-Objekt, hat jedoch das tokenType-Flag anders gesetzt.
Das generierte Objekt wird als Antwort der aufrufenden *Billing-Service*-WS-Resource übergeben.

CheckIn

Bei diesem Serviceaufruf wird ein CheckInRequest-Objekt als Parameter übergeben.
Das CheckInRequest-Objekt wird initialisiert und die Signatur des *License-Servers* geprüft, der das zugehörige *License-Token* ausgestellt hat. Zusätzlich wird die Signatur der *Application* geprüft, die den *CheckIn-Request* generiert hat. Folglich wird der Inhalt des Tokens auf Gültigkeit geprüft. Wenn das Token als gültig erkannt wurde und somit nicht gefälscht ist, wird mittels einem SQL-Aufruf in der MySQL-Datenbank nachgeschaut, ob es zu dem *CheckIn-Request* einen zugehörigen Eintrag gibt. Wenn dies der Fall ist, wird ein neues SQL-Statement generiert, das in der Datenbank die endingTime des zugehörigen Datensatzes auf die aktuelle Uhrzeit und Datum setzt.
Im Anschluss daran wird ein CheckInResponse-Objekt generiert, das zusätzlich zum Inhalt des CheckInRequest-Objekts die Signatur des *License-Servers* enthält.
Das generierte Objekt wird als Antwort der aufrufenden *Billing-Service*-WS-Resource übergeben.

getLicenseUpdateToken

Bei diesem Serviceaufruf wird ein LicenseUpdateRequest-Objekt als Parameter übergeben.
Das LicenseUpdateRequest-Objekt wird initialisiert und die Signatur des *License-Servers* geprüft, der das zugehörige *License-Token* ausgestellt hat. Zusätzlich wir die Signatur des *License-Clients* geprüft, der den *License-Update-Request* generiert hat. Folglich wird der Inhalt des Tokens auf Gültigkeit geprüft. Im Falle eines gültigen Tokens wird mittels einem SQL-Aufruf in der MySQL-Datenbank nachgeschaut, ob es zu dem *License-Update-Request* einen zugehörigen Eintrag gibt. In diesem Eintrag wird geprüft, ob die zugehörige Lizenz nicht im Status „used“ oder „inUse“ ist, da sonst eventuell das Update nicht gewährt werden kann. Wenn jedoch das Update möglich ist, wird ein neues SQL-Statement generiert, das den Eintrag in der Datenbank auf den aktualisierten Stand bringt.
Abschließend wird ein LicenseUpdateResponse-Objekt generiert, das zusätzlich zum Inhalt des LicenseUpdateRequest-Objekts die Signatur des *License-Servers* enthält.
Das generierte Objekt wird als Antwort der aufrufenden *License-Service*-WS-Resource übergeben.

getLicenseInformation

Bei diesem Serviceaufruf wird ein LicenseInformationRequest-Objekt als Parameter übergeben.
Das LicenseInformationRequest-Objekt wird initialisiert und die Signatur des *License-Clients* geprüft, der den *License-Information-Request* generiert hat. Folglich wird der Inhalt des Tokens auf Gültigkeit geprüft. Im Falle eines gültigen Tokens wird mittels einem SQL-Aufruf in der MySQL-Datenbank nachgeschaut, ob es zu dem *License-Information-Request* einen zugehörigen Eintrag gibt. Die Informationen werden genommen und ein neues LicenseInformationResponse-Objekt generiert und anschließend vom *License-Server* signiert.
Das generierte Objekt wird als Antwort der aufrufenden *Billing-Service*-WS-Resource übergeben.

6.7 Die Implementierung des License-Services

Der *License-Service* ist ebenfalls als WS-Resource implementiert. Das heißt, er besteht aus einem Web Service mit zugehörigen Java-Klassen und ist stateful. Folglich besteht der *License-Service*, wie auch der *License-Server*, aus Web Service-spezifischen Dateien (WSDL-File, JNDI-Descriptor, WSDD-Definitions) zur Beschreibung des Web Services sowie aus mehreren Java-Dateien zur Implementierung der Logik der WS-Resource. Für genauere Package-Beschreibungen siehe 6.2.1.

6.7.1 WSDL, WSDD und JNDI

Das WSDL-File des *License-Services* definiert die Schnittstellen (Dienste) und Properties.

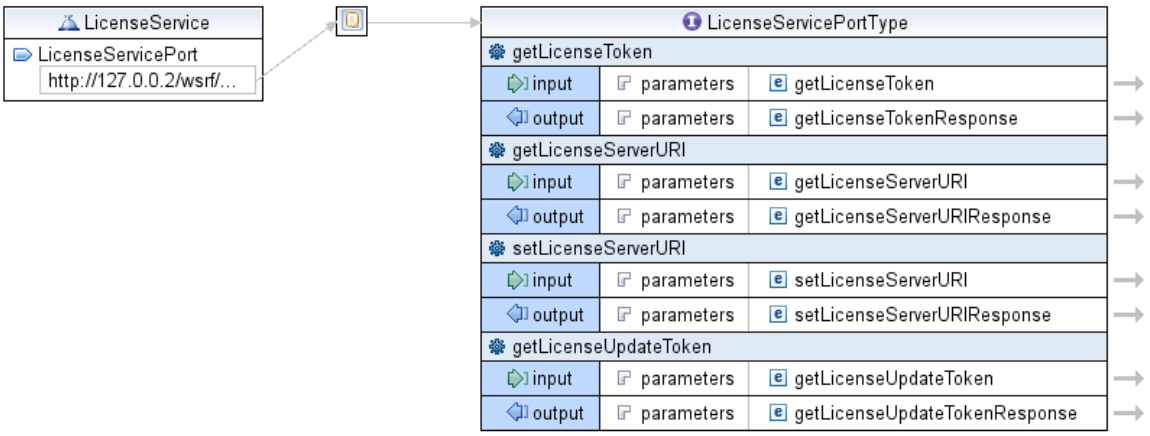

Abbildung 6.3: Grafische Darstellung des WSDL-Files des License-Services

Dieses Bild zeigt eine grafische Übersicht über das WSDL-File des *License-Services*. Auf den ersten Blick sieht es sehr ähnlich zu dem des *License-Servers* aus. Jedoch erkennt man, dass der *License-Service* insgesamt nur vier Tokens beherrscht. Dafür hat er zusätzlich die WS-Resource-Property *LicenseServerURI*, die die URI des zuständigen *License-Servers* beziehungsweise die URI des nächsten *License-Services* innerhalb eines Pfades beschreibt.
Für eine Darstellung der einzelnen Tokens sei auf 6.6.1 verwiesen, in dem beispielhaft das *License-Token* dargestellt ist, zu dem alle Tokens ähnlich sind.
Das WSDL-File wurde ebenfalls in der Grundstruktur mit dem Editor von Eclipse generiert. Es war jedoch analog zur Entwicklung des WSDL-Files für den *License-Server* notwendig, einige Feinheiten direkt im Code anzupassen und zu definieren.
Damit der *License-Service* in die Grid-Umgebung deployed werden kann, sind noch ein JNDI-Descriptor sowie eine WSDD-Datei zur Konfiguration notwendig. Diese sind analog zu den Dateien des *License-Servers*.

6.7.2 Die Logik

Die Logik des *License-Services* ist in Java-Klassen untergebracht. Die eigentliche Logik und Funktionalität ist in der LicenseService.java-Datei implementiert.
Diese Klasse importiert zunächst alle notwendigen Packages wie zum Beispiel die Bibliotheken von GT4 oder Axis. Damit die LicenseService-Klasse die Tokens versenden beziehungsweise empfangen kann, müssen die Tokens ebenfalls importiert werden. Außerdem ist für die Kommunikation mit der *License-Server*-WS-Resource der Import der dafür zuständigen Packages notwendig.
Beim Deployment des *License-Services* und Start des GT4-Containers, der den *License-Service* hostet, wird eine Selbstinitialisierung vorgenommen. Diese beinhaltet insbesondere das Generieren eines ServiceEndpoint-Objekts zum Zugriff auf die *License-Server*-WS-Resource.
Nach erfolgreich abgeschlossener Selbstinitialisierung ist die WS-Resource bereit und wartet auf Aufgaben, die sie vom *License-Client* erhält. Dieser Zustand ist der StandBy-Zustand.
Die angebotenen Dienste (siehe 6.7.1) sind WS-typisch als Java-Methoden implementiert.

getLicenseToken

Bei diesem Serviceaufruf wird ein RequestToken-Objekt als Parameter übergeben.
Die *License-Service*-WS-Resource ermittelt mittels ihrer *LicenseServerURI*-Property die zuständige *License-Server*-WS-Resource und ruft dort den Service getLicenseToken auf. Das von diesem Service erhaltene Token-Objekt leitet die *License-Service*-WS-Resource als Antwort an den aufrufenden *License-Client* weiter.

getLicenseUpdateToken

Dieser Service-Aufruf ist analog zum getLicenseToken-Aufruf implementiert.

getLicenseServerURI

Dieser Aufruf basiert auf den WS-Resource-Properties, nimmt keinen Parameter entgegen und gibt als Antwort die *LicenseServerURI*-Property an den aufrufenden *License-Client* zurück.

setLicenseServerURI

Dieser Aufruf basiert auf den WS-Resource-Properties und erwartet als Eingabe einen String, der die URI des *License-Servers* repräsentiert. Als Antwort wird ein Boolean zurückgegeben, der den erfolgreichen Aufruf darstellt.

6.8 Die Implementierung des Billing-Services

Der *Billing-Service* ist analog zum *License-Service* als WS-Resource implementiert. Das heißt, er besteht ebenfalls aus einem Web Service mit zugehörigen Java-Klassen und ist stateful. Er besitzt analog ein WSDL-File, einen JNDI-Descriptor und WSDD-Definitions zur Beschreibung des Web Services sowie aus mehreren Java-Dateien, die die Logik implementieren. Für genauere Package-Beschreibungen siehe 6.2.1.

6.8.1 WSDL, WSDD und JNDI

Das WSDL-File des *Billing-Services* definiert die Schnittstellen (Dienste) und Properties.

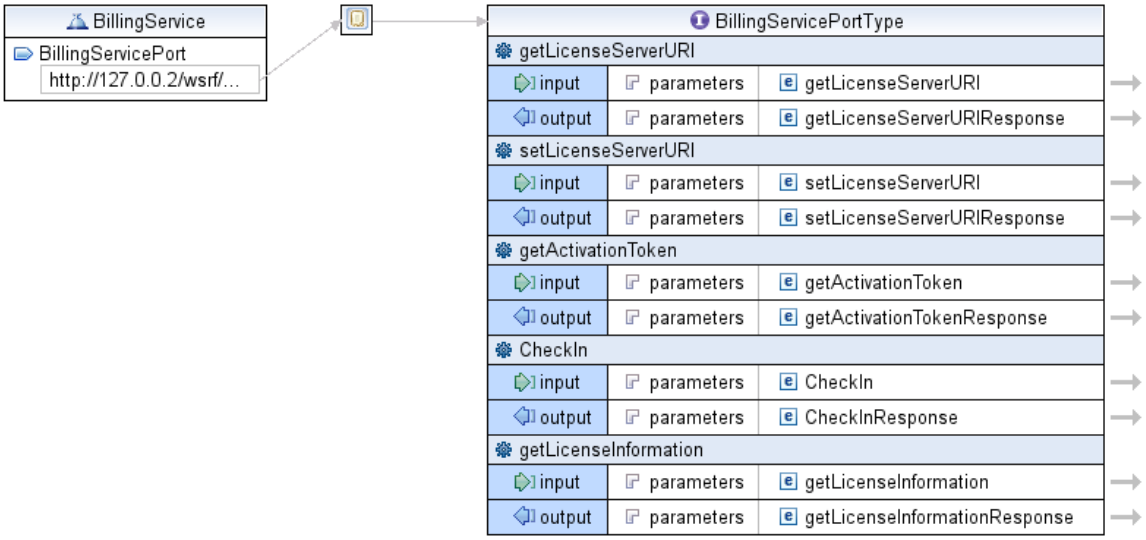

Abbildung 6.4: Grafische Darstellung des WSDL-Files des Billing-Services

In diesem Bild ist die grafische Übersicht des WSDL-Files des *Billing-Services* dargestellt. Es ist ersichtlich, dass der *Billing-Service* insgesamt sechs verschiedene Tokens beherrscht. Außerdem hat der *Billing-Service* analog zum *License-Service* die WS-Resource-Property *LicenseServerURI*, die die URI des zuständigen *License-Servers* beschreibt beziehungsweise die URI des nächsten *Billing-Services* innerhalb eines Pfades darstellt.
Für eine Darstellung der einzelnen Tokens sei auf 6.6.1 verwiesen, in dem beispielhaft das *License-Token* gezeigt ist, zu dem alle Tokens ähnlich sind.
Die Entwicklung des WSDL-Files erfolgte analog zur Entwicklung des WSDL-Files des *License-Services*.
Auch der JNDI-Descriptor sowie die WSDD-Datei waren relativ straight-forward zu schreiben.

6.8.2 Die Logik

Die Logik des *Billing-Services* ist in mehrere Java-Klassen aufgeteilt. Die eigentliche Logik und Funktionalität ist in der BillingService.java-Datei untergebracht.
Die grundsätzliche Struktur und Funktionsweise ist identisch zu der des *License-Services*. Zuerst erfolgen alle notwendigen Imports gefolgt von einer Selbstinitialisierung.
Nach erfolgreich abgeschlossener Selbstinitialisierung ist die WS-Resource bereit und wartet auf Aufgaben, die sie vom *License-Client* und der *Application* erhält. Dieser Zustand ist der StandBy-Zustand.
Die Logik der Dienste, die der *Billing-Service* als WS-Resource bietet (siehe 6.8.1), wurden WS-typisch als Java-Methoden implementiert.

getActivationToken

Bei diesem Serviceaufruf wird ein ActivationToken-Objekt als Parameter übergeben.
Der *Billing-Service*-WS-Resource ermittelt mittels ihrer *LicenseServerURI*-Property die zuständige *License-Server*-WS-Resource und ruft dort den Service getActivationToken auf. Das von diesem Service erhaltene Token-Objekt leitet die *Billing-Service*-WS-Resource als Antwort an die aufrufende *Application* weiter.

CheckIn

Die Abarbeitung dieses Service-Aufrufs erfolgt analog zur Bearbeitung des getActivationToken-Aufrufs.

getLicenseServerURI

Dieser Aufruf bezieht sich auf die ResourceProperties und wird analog zu dem identischen Aufruf innerhalb des *License-Services* bearbeitet.

setLicenseServerURI

Auch dieser Aufruf bezieht sich auf die ResourceProperties und wird ebenfalls analog zum Aufruf innerhalb des *License-Services* bearbeitet.

6.9 Die Implementierung der Application

Die *Application* besteht aus einer einzelnen Java-Datei, nämlich der Application.java, die die gesamte implementierte Logik darstellt.

6.9.1 Die Logik

Die Application-Klasse importiert zunächst alle notwendigen Packages. Darunter fallen die Java-internen Packages zum Erstellen und Lesen von Dateien aus dem Dateisystem und die Packages zum Zugriff und der Verwaltung auf Zertifikate, KeyStores und TrustStores. Außerdem benötigt die Application-Klasse selbstverständlich einige Packages von GT4 und Axis zum Zugriff auf Web Services und WS-Resources innerhalb dem Grid. Damit die Application-Klasse die Tokens generieren und versenden beziehungsweise empfangen und interpretieren kann, müssen die Tokens ebenfalls importiert werden.
Beim Start der *Application* wird zunächst eine Selbstinitialisierung vorgenommen. Dabei wird das EndpointReference-Objekt des *Billing-Services* generiert und Referenzen auf den Service-Port angelegt.
Anschließend werden einige Variablen gesetzt und die Selbstinitialisierung mit dem Einlesen der KeyStores und TrustStores und der Gewinnung der jeweiligen Keys abgeschlossen.
Anschließend werden die Eingabeparameter geprüft. Die *Application* erwartet als Anwendung, die in einem Grid ausgeführt wird beziehungsweise als Job gestartet wird zwei Parameter. Der erste Parameter stellt einen Pfad auf das zu benutzende *License-Token* dar. Der zweite Parameter entspricht dem normalen Input für einen Grid-Job, wird aber prototypischerweise nur als String angenommen und nicht als binäre Daten.
Bei falschen oder ungenügenden Eingabeparametern oder wenn die Parameter in falscher Reihenfolge eingeben wurden, wird die *Application* beendet und das Ergebnis des Jobs ist eine kleine Hilfestellung, die die Eingabeparameter beschreibt.
Nach erfolgreicher positiver Prüfung wird der *License-Token* mittels einem ObjectInputStream-Objekt eingelesen. Das *License-Token* wird initialisiert, die Daten des Tokens algorithmisch miteinander verknüpft und mit den übermittelten Daten verglichen. Anschließend wird die Signatur des *License-Servers*, der das *License-Token* ausgestellt hat, auf Vertraulichkeit geprüft und der Inhalt des Tokens verifiziert. Als letztes werden die übermittelten Input-Daten für den Job gehasht und mit dem Hash aus dem *License-Token* verglichen. Ist einer dieser Schritte nicht positiv verlaufen, wird der Job abgebrochen.
Sind jedoch, wie erwartet, alle Prüfungen positiv verlaufen, generiert die *Application* ein ActivationRequest-Objekt. Dies beinhaltet alle Daten aus dem *License-Token*, die aktuelle Zeit, die Identifikation der Computing-Resource (FQDN, wenn möglich, mittels InetAddress.getLocalHost().getHostName() ausgelesen) sowie eine JobID (prototypisch nur ein simpler String) und die Signatur der *Application*.
Anschließend erfolgt der Aufruf des Dienstes getAktivationToken beim *Billing-Service* mit dem ActivationRequest-Objekt als Eingabeobjekt. Die *Application* wartet nun auf Antwort des *Billing-Service*.
Die *Application* erhält vom beauftragten *Billing-Service* eine Antwort (wenn kein Fehler aufgetreten ist). Das erhaltene Objekt ist ein ActivationToken-Objekt. Die *Application* prüft alle zu prüfenden Signaturen auf Gültigkeit und analysiert somit die Echtheit des Token-Objekts. Wenn das Token-Objekt als unverfälscht eingestuft wurde und von einer vertraulichen Instanz signiert wurde, wird ein CheckInRequest-Objekt generiert, das die aktuelle

Uhrzeit/Datum und die JobID (wiederum als einfachen String) enthält und anschließend von der *Application* signiert wird. Prototypischerweise wird dieses Objekt in /tmp mittels einem ObjectOutputStream-Objekt abgespeichert und nicht der Grid Middleware als Output übergeben.
Anschließend beginnt die *Application* ihre Threads zu spawnen.
Es handelt sich dabei um fünf Threads, jeder mit einer anderen Aufgabe betreut.

- **Unlock-Thread**
  Dieser Thread befindet sich dauerhaft in einer Endlosschleife. In gewissen Zeitabständen wird die Variable isActivated auf ihren Wert überprüft. Diese Variable bezeichnet den Status der Lizenz. Zum jetzigen Zeitpunkt ist diese Variable also auf true gesetzt (nach erfolgreicher Prüfung des *Activation-Tokens* geschehen), da ein *Activation-Token* erst vor wenigen Augenblicken für valide befunden wurde. In einem gewissen Zeitintervall wird die Variable geprüft. Solange sie auf true gesetzt ist, wird nichts unternommen, sobald sie jedoch auf false gesetzt wurde, initiiert der Unlock-Thread einen Abbruch der Berechnung.

- **CheckIn-Request-Update-Thread**
  Dieser Thread kümmert sich um die Generierung von CheckInRequest-Objekten und deren Abspeicherung im /tmp-Ordner. Diese Prüfung findet in einer Endlosschleife in gewissen Zeitabständen statt.

- **Live-LicenseCheck-Thread**
  Der Live-LicenseCheck-Thread prüft ständig die aktuelle Uhrzeit gegen die Gültigkeit der Lizenz. Außerdem kennt er die Zeitspanne der Lizenzgültigkeit, die er ebenfalls zur Überprüfung der Dauer heranzieht. Sollte die Lizenz ungültig werden, so setzt er die isActivated-Variable auf false, was den Unlock-Thread zum Initiieren eines Abbruchs auffordert.

- **JobStatusMonitor-Thread**
  Dieser Thread prüft in einem bestimmten Zeitintervall, in welchem Zustand der Job gerade ist. Dies ist notwendig, um zu erkennen, ob der Job noch läuft, oder bereits abgeschlossen ist.

- **Job-Thread**
  Dies ist der Thread, in dem der eigentliche Job läuft.

Bei Ende der Berechnung initiiert die *Application* einen Serviceaufruf beim *Billing-Service* zur Durchführung des CheckIns der Lizenzen.
Die Output-Daten der Berechnung werden prototypischerweise im /tmp-Verzeichnis abgespeichert.

6.10 Probleme bei der Entwicklung

In diesem Abschnitt werden Probleme aufgezeigt, die bei der Entwicklung und Implementierung des Prototypen aufgetreten sind.

6.10.1 Technische Probleme

Schwierigkeiten bei der Entwicklung gab es einige, die jedoch alle im Laufe der Zeit lösbar waren. Folgende Schwierigkeiten sollen jedoch nicht unerwähnt bleiben:

- **Kommunikation mit der MySQL-Datenbank**
  Die Kommunikation mit der MySQL-Datenbank erfolgt mittels einer Connector-Bibliothek, die selbstverständlich eine sehr exakte Syntax zur Umwandlung des SQL-Statements als String in ein „echtes“ SQL-Statement voraussetzt. Das Lernen dieser Syntax war anfangs etwas kompliziert und teilweise nicht logisch.

- **WSDL-Entwicklung**
  Die Entwicklung der WSDL-Files mittels Eclipse ist prinzipiell schön gelöst. Jedoch trüben einige Problematiken den Eindruck. Einige Definitionen sind nicht mittels dem GUI durchführbar, sondern können nur im Code definiert werden. Ein kleiner Tippfehler oder Gedankenfehler führt dazu, dass beim weiteren Bearbeiten mittels dem GUI das komplette WSDL fehlerhaft wird, da an falschen Positionen der Code automatisch angepasst wird. Beim Kopieren von Tags innerhalb des Codes kann es auch dazu führen, dass man das WSDL-File später nicht mehr mittels dem GUI bearbeiten kann, obwohl das WSDL-File valide ist.

- **WS-Resource-Entwicklung**
  Die WS-Resources wurden schrittweise entwickelt. Jedoch müssen bereits alle Methodenrümpfe passend zu dem zugehörigen WSDL-File implementiert sein, bevor eine WS-Resource deployed werden kann. Ist dies nicht der Fall, führt das zu teilweise subtilen Fehlermeldungen.

- **Analyse von Laufzeitfehlern der WS-Resources**
  Laufzeitfehler der WS-Resources sind teilweise sehr schwer nachzuvollziehen und zu interpretieren, obwohl der GT4-Container mittels -debug gestartet wurde.

- **If-Then-Else**
  Alle entwickelten Java-Klassen basieren auf If-Then-Else-Statements. Teilweise wurden diese sehr komplex und vielstufig. Diese Tatsache hat hin und wieder unvorhergesehene und auf den ersten Blick nicht nachvollziehbare Probleme hervorgerufen.

- **Tippfehler**
  Das leidige Thema eines jeden Programmierers: Ein kleiner Tippfehler führt zu einem kompletten Scheitern eines komplexen Vorgangs. Solche kleinen Tippfehler sind selten aufgetreten, aber kosten bei der Entwicklung enorm viele Nerven, da die hervorgerufenen Laufzeitfehler subtil und oft nicht nachvollziehbar waren.

6.10.2 Spezielle Probleme auf Grund der Umgebung der Universität

Das Testbed, das für diese Diplomarbeit zur Verfügung gestellt wurde, hat leider im Laufe der Zeit mehr Fragen und Probleme aufgeworfen, als es genutzt hat. Daher wurde auch auf die Benutzung dieses Testbeds verzichtet, leider erst nachdem schon viele Probleme aufgetreten waren. Ein Großteil der Entwicklung erfolgte auf privaten Computern mit einem privaten Grid, die hardwaretechnisch und in ihrer Software-Ausstattung auf aktuellem Niveau sind und eine sinnvolle Prototyp-Entwicklung ermöglichen.
Folgende, unter anderem, aufgetretene Probleme des Universitäts-Testbeds sollen genannt werden:

- **Zugriff nur per SSH über einen anderen Rechner**
  Zugriff auf das bereitgestellte Testbed konnte nur per SSH von einem Rechner aus erfolgen, auf dem man ebenfalls nur per SSH Zugriff hatte. Ein SSH-Tunnel mittels Port-Forwarding auf den localhost war nur unzureichend: der Tunnel brach häufig zusammen.

- **Firewall**
  Leider war die Firewall des Testbeds rigoros und lies nur SSH durch. Ein Zugriff auf die Web Services waren nur aus einer Shell des Testbeds möglich. Da die Entwicklungsumgebung jedoch Eclipse beziehungsweise g-Eclipse war, war es höchst umständlich, auf die WS-Resources zuzugreifen. Auch wurde während der Entwicklung der Diplomarbeit mitgeteilt, dass es Probleme mit der Firewall gab.

- **Langsame Verbindung**
  Teilweise war der Zugriff mittels SSH extrem langsam. Man konnte zusehen, wie die Zeichen in der Shell nacheinander angezeigt wurden. Dies war zwar nur hinundwieder der Fall, jedoch war ein sinnvolles Arbeiten zu diesen Zeitpunkten ausgeschlossen.

- **Unzureichende Softwareprodukte**
  VOMS war zwar installiert, aber kein notwendiges aktuelles Python.
  Python konnte nicht ohne weiteres in der Debian-Umgebung kompiliert und installiert werden, wie man es gewohnt ist.

- **Java veraltet, nicht ohne weiteres updatebar**
  Java lag in einer veralteten 1.5-Version vor. Dies führte zu obskuren Fehlern wie zum Beispiel, dass eine WS-Resource nicht deployed werden konnte. Erst nach langer Recherche hat sich herausgestellt, dass es an Java 1.5 lag. Ein Update auf 1.6 mit umfangreichen Anpassungen von Konfigurationen wäre jedoch zeitlich nicht tragbar gewesen.

- **MySQL-Zugriff nicht möglich**
  MySQL war zwar installiert, aber in keinster Weise konfiguriert. Einzig rigorose Zugriffsbeschränkungen lagen vor.

- **Globus-User zum Starten des GT4-Containers: keine Rechte**
  Standardmäßig wird der GT4-Container von einem Benutzer der Gruppe globus gestartet. Meist nennt sich der Benutzer ebenfalls globus. Leider konnte diese Identität nicht verwendet werden. Ein Starten des GT4-Containers als root kann jedoch zu unvorhersehbaren Komplikationen führen, da eventuell Dateien mit zu stark beschränkten Zugriffsrechten geschrieben werden.

- **Arbeiten ohne GUI**
  Arbeiten und insbesondere Entwickeln komplett ohne GUI sind in der heutigen Zeit keinesfalls Stand der Technik.

6.11 Kritische Betrachtung des Prototypen

Dieser Abschnitt befasst sich mit einer kritischen Betrachtung des Prototypen.
Der Prototyp implementiert einen Großteil der Features des Lizenzmodells, das in Kapitel 5 entwickelt wurde. Einige Funktionen wurden jedoch vereinfacht implementiert oder mit Hilfe von Dummies gestaltet.
Ein Prototyp ist eben ein Prototyp: er soll die grundlegende Funktionalität des Konzepts darstellen und zeigen, in welche Richtung eine Implementierung gehen kann. Er ist niemals vollständig, da es sonst kein Prototyp mehr wäre, sondern ein finales Produkt. Unter diesem Gesichtspunkt wird der Prototyp in diesem Abschnitt abschließend beleuchtet.

6.11.1 Prototyp versus Lizenzmodell

Das Lizenzmodell definiert einige Feinheiten, die der Prototyp nicht umsetzt:

- **VOMS-Anbindung**
  Die wohl größte Differenz zwischen Prototyp und Lizenzmodell ist die Anbindung an den Virtual Organisation Membership Service.
  Der Prototyp verzichtet auf diese Anbindung gänzlich. Er benutzt der Einfachheit halber „nur“ ein gewöhnliches Zertifikat, das den Benutzer authentisiert. Folglich sind auch einige Features nicht im Prototypen implementiert, die direkt auf VOMS beruhen: Es können keine Black- und Whitelists definiert werden. Die einzige Prüfung, ob ein Benutzer eine Lizenz erhält ist somit die Prüfung seitens des *License-Servers*.

- **Linux-only-Kompatibilität**
  Aufgrund der hart gecodeten Pfad-Variablen zum Abspeichern der Tokens im /tmp-Ordner sowie die hart gecodeten Pfade zu den KeyStores und TrustStores in Linux-konformen Format ist der Prototyp nur unter Linux-Umgebungen einsetzbar. Für eine Windows-Kompatibilität müssen die Pfade angepasst werden. Auch im Einsatz in einer Linux-Umgebung müssen die Pfade eventuell angepasst werden, wenn die Stores nicht an exakt dem gecodeten Platz zu finden sind.

- **Vereinfachte Funktionen**
  Nicht jede Funktionalität ist im Prototyp vollständig umgesetzt. Die Informationsbeschaffung ist nur rudimentär implementiert. Außerdem sind statistische Auswertungen nicht direkt möglich.

6.11.2 Prototyp versus reale Welt

Ein Prototyp ist per Definition „nur“ ein Prototyp und daher nicht ohne weiteres direkt in der „realen Welt“ einsetzbar. Dieses Kapitel zeigt die offensichtlichsten Missstände in diesem Hinblick auf und erläutert, wo am Prototypen noch grundsätzlich Arbeit notwendig ist. Folgende Gesichtspunkte dürfen nicht außer acht gelassen werden:

- **Vervollständigung der Funktionalität**
  Alle in Abschnitt 6.11.1 genannten Differenzen müssen implementiert werden.
- **Schutz der Klassen**
  Alle Java-Klassen müssen gegen Dekompilierung und Reverse-Engineering mittels klassischer Mittel geschützt werden.
- **Keine hard-coded Variablen**
  Keine Variable darf hard-coded sein. Alle Variablen müssen über Config-Files eingelesen werden können und somit verändert werden können.
- **Schutz der privaten Schlüssel und Passwörter**
  Die privaten Schlüssel und Passwörter müssen geschützt werden. Sie dürfen keinesfalls durch Dritte einsehbar sein.
- **Verwendung echter Zertifikate**
  Das gesamte Lizenzmodell muss auf echten Zertifikaten basieren und nicht auf selbstsignierten Zertifikaten.
- **Schutz der Datenbank**
  Die Datenbank muss gegen Manipulation und Einsicht der Daten durch Unbefugte geschützt werden. Außerdem müssen die Zugriffsparameter der Datenbank anderweitig abgespeichert und geschützt werden.
- **Input-Daten generisieren**
  Die Input-Daten müssen als binäre Daten akzeptiert werden.
- **Application-Output**
  Der Output der Application, sprich die Job-Ergebnisse müssen auf „normalem“ Weg zurückgegeben werden.
- **Fehlermanagement**
  Der Prototyp hat nur rudimentäres Fehlermanagement. Das bedeutet, es werden zwar alle bekannten Fehler mittels einem Try-Catch-Block gefangen. Der Umgang mit einem aufgetretenen Fehler ist jedoch nur sehr einfach implementiert. In einer realen Umgebung müsste das Fehlermanagement deutlich umfangreicher ausfallen.

6.12 Zusammenfassung des 6. Kapitels

Das 6. Kapitel befasste sich mit der Entwicklung eines Prototypen für das in dieser Diplomarbeit konzeptionierte Lizenzmodell.
Die Implementierung erfolgte in einer openSuse-11.2-Umgebung mit den aktuellsten Sicherheits und Softwareupdates. Als Entwicklungswerkzeug kam vorwiegend das Eclipse IDE for Java EE Developers zum Einsatz.
Mit dieser Basis wurde der *License-Client* als Java-Applikation entwickelt. Diese Java-Anwendung ist für die Ausführung der Requests des Benutzers zuständig. Als Prototyp verzichtet der *License-Client* jedoch auf ein GUI, sondern bietet nur die Interpretation von Kommandozeilenparameter an.
Der *License-Server* wurde als WS-Resource implementiert. Er besteht also aus einem Web Service zusammen mit Java-Klassen und ist stateful. Der Web Service wurde zuerst mit einem WSDL-File beschrieben, das in den Grundzügen mit dem GUI von Eclipse entwickelt wurde. Die Feinheiten wurden direkt im WSDL-Code definiert. Die Entwicklung der Java-Klassen der *License-Server*-WS-Resource erfolgte wiederum mit Eclipse.
Die Implementierung des *License-Services* und des *Billing-Services* erfolgte ebenfalls als WS-Resources. Die Vorgehensweise war ähnlich zur Implementierung der *License-Server*-WS-Resource, jedoch weit weniger komplex auf Grund der geringeren Funktionalität im Vergleich zum *License-Server*.
Als letztes wurde die *Application* als Java-Applikation entwickelt, die in Grids ausgeführt werden kann. Die Entwicklung dieser Java-Anwendung gestaltete sich ähnlich zur Entwicklung des *License-Clients*, sprich die Implementierung wurde großteils in Eclipse durchgeführt.
Im danach folgenden Abschnitt wurden einige Probleme aufgezählt, die während der Entwicklung aufgetreten sind. Diese wurden unterteilt in logische und technische Schwierigkeiten sowie Probleme, die mit dem von der Universität bereitgestellten Testbed aufgetreten sind. Die Gründe, warum die Entwicklungsarbeit schließlich auf privaten Computern und einem privaten Grid durchgeführt wurden, sind vielfältig.
Die gesamte Entwicklung ist prototypischer Art. Das bedeutet, die Java-Anwendungen und WS-Resources greifen teilweise auf Dummies zurück beziehungsweise besitzen hart gecodete Variablen. Auch sind keine Schutzmaßnahmen gegen Manipulationen der Java-Klassen getroffen. Ein Prototyp ist eben nur ein Prototyp und kein finales Produkt. Er soll nur zeigen, in welche Richtung eine Implementierung eines Konzepts gehen kann und auf welche Probleme und Ideen man dabei stoßen kann.
Die Tatsache, dass es sich um einen Prototypen handelt, wurde als Abschluss dieses Kapitels kritisch betrachtet. Der Vergleich Prototyp versus Lizenzmodell und Prototyp versus reale Welt sollen hierbei die wichtigsten Differenzen aufzeigen.
Während der Entwicklung hat sich gezeigt, dass die Umsetzung eines komplexen Konzepts sehr umfangreich ist und nicht jeder Punkt, unter Berücksichtigung des relativ kurzen Zeitkontingents einer Diplomarbeit, eins-zu-eins implementiert werden kann.

7 Zusammenfassung und Ausblick

Dies ist das letzte Kapitel der Diplomarbeit. Es fasst die Arbeit zusammen und dient zur Interpretation der Ergebnisse. Außerdem wird auf allgemeine Probleme bei der Entstehung der Arbeit eingegangen. Abschließend werden in einem Ausblick in die Zukunft persönliche Gedanken zum möglichen weiteren Verlauf der Lizenzierung in Grid-Umgebungen erläutert. Die Arbeit endet mit einigen allgemeinen Worten und einem kurzen Fazit.

7.1 Zusammenfassung der Arbeit

In der Einleitung der Arbeit wurde beschrieben, dass die Diplomarbeit einer Vier-Phasen-Entstehung unterliegt. Die erste, das Brainstorming, war deutlich länger als ursprünglich geplant. Dies lag daran, dass es unzählige verschiedene Mechanismen zur Softwarelizenzierung gibt. Es existieren zwar einige Konzepte, die weit verbreitet sind und von vielen Independent Software Vendors eingesetzt werden, es gibt aber auch immens viele unabhängige Konzepte. Die Untersuchung vieler verschiedener Modelle zeigte jedoch schon sehr bald, dass alle auf einem gemeinsamen Nenner beruhen: Lizenz anfordern, Anforderung prüfen, Lizenz zuteilen, lizenzierte Anwendung zusammen mit Lizenz starten, Lizenz auf Gültigkeit prüfen. Dieser Vorgang ist Grundlage jedes Lizenzmodells und daher in jedem Modell in einer gewissen Weise wiederzufinden. Auch die in dieser Arbeit dargestellten Ansätze zur Lizenzierung in Grid-Umgebungen, sowie das in dieser Arbeit entwickelte Lizenzmodell, beruhen grundsätzlich auf diesem Schema.
Das Brainstorming und die Informationssammlung der ersten Phase bezog sich auch auf die Untersuchung heutiger Grid-Umgebungen, insbesondere unter Einsatz von Globus Toolkit 4. Die erkannten speziellen Eigenheiten, Bedingungen und Verpflichtungen trugen im weiteren Verlauf der Diplomarbeit für die Entwicklung des Anforderungskatalog wesentlich bei.
In der zweiten Phase wurden die gewonnenen Information abstrahiert, um daraus einen Anforderungskatalog zu definieren. Es hat sich dabei gezeigt, dass die Anforderungen an ein Lizenzmodell für Grid-Umgebungen sich nicht extrem von den Anforderungen an ein klassisches Lizenzmodell unterscheiden. Die prinzipiellen Grundzüge sind identisch. Erst die Eigenheiten einer Grid-Umgebung führen zu abgewandelten oder zusätzlichen Anforderungen. Genau diese Erweiterungen disqualifizieren den Einsatz klassischer Lizenzmodelle. Es gibt jedoch bereits einige konzeptionellen Entwicklungen im Bereich Lizenzmodelle für Grid-Umgebungen. Drei Modelle sind in ihrer Konzeption beziehungsweise Entwicklung relativ weit fortgeschritten und sind durchaus interessant in ihrer Funktionsweise. Zu diesen drei Modellen wurden Informationen beschafft (soweit diese nicht unter Verschluss gehalten werden) und untersucht. Dabei hat sich herausgestellt, dass die Ansätze jeweils eigene Ziele definieren, sich in ihrer Funktionalität stark unterscheiden, jedoch alle auf dem gemeinsamen Nenner einer Lizenzierung beruhen. Keines der Modelle kann jedoch allen Anforderungen gerecht werden, nur SmartLM sticht hier sehr positiv hervor. Die Tatsache, dass der Anforderungskatalog nicht erfüllt wird, hat den weiteren Verlauf der Entwicklung der Diplomarbeit positiv beeinflusst, da die Notwendigkeit eines umfassenderen Lizenzmodells gegeben ist.

An diesem Punkt begann die dritte Phase der Entstehung.
In dieser Phase wurde ein neues Lizenzmodell für Grid-Umgebungen konzeptioniert. Dieses Modell deckt möglichst alle Punkte des Anforderungskatalogs ab. Bei der Definition des Modells wurde insbesondere Wert darauf gelegt, dass Grid-typische Konzepte Verwendung finden. So wurde untersucht, inwieweit Web Services und WS-Resources als Teil des Lizenzmodells eingesetzt werden können. Es hat sich bei den Überlegungen gezeigt, dass dies durchaus möglich ist und der Einsatz von WS-Resources als Dienstleister für die Lizenzierung sinnvoll ist. Der Anforderungskatalog kann mit dem entwickelten Lizenzmodell nahezu vollständig abgedeckt werden.
Die letzte Phase der Entstehung der Diplomarbeit war vorwiegend praktischer Natur. Das konzeptionierte Lizenzmodell wurde auf Basis von Java, Web Services und WS-Resources praktisch implementiert. Die Implementierung stellt einen Prototyp dar, der die grundlegende Funktionalität des Lizenzmodells abdeckt. Er kann aber nicht als finales Produkt gesehen werden, die prototypische Natur ist erkennbar. Der Prototyp zeigt jedoch in welche Richtung eine Implementierung des Lizenzmodells gehen kann. Eine umfangreiche und vollständige Implementierung des Lizenzmodells unter Berücksichtigung aller Schutzmaßnahmen und Funktionalitäten würde den zeitlichen Rahmen einer Diplomarbeit sprengen.
Mit der Entwicklung des Prototypen ist die Vier-Phasen-Entstehung abgeschlossen.

7.2 Interpretation der Ergebnisse

Das Konzept des Lizenzmodells, das in dieser Diplomarbeit entwickelt wurde, soll als Basis für weitere Konzeptionierungen dienen und die grundlegende Funktionalität eines Lizenzmodells für Grid-Umgebungen auf Basis von Web Services und WS-Resources bieten. Das Modell ist keinesfalls auf die beschriebene Funktionalität beschränkt, sondern ermöglicht die Integration weiterer Features. Somit kann das definierte Konzept als Ergebnis dieser Arbeit gelten und für weitere Forschungs- und Entwicklungszwecke herangezogen werden.

7.3 Schwierigkeiten bei der Entstehung

Während der Entstehung der Diplomarbeit traten verschiedene Schwierigkeiten auf. Eine der größten Problematiken während der Informationsbeschaffung ist die Tatsache, dass sehr viele technische Dokumente zu den verschiedenen Lizenzmodellen (sowohl non-Grid als auch Grid) unter Verschluss gehalten werden. Dies ist insoweit verständlich, da die jeweiligen Entwickler zum Schutz ihres Lizenzmodells und folglich der Software, die darauf beruht, häufig „Schutz durch Unwissen“ einsetzen. Das bedeutet in etwa: „Wenn man nicht weiß, wie etwas funktioniert, weiß man auch nicht, wo man anfangen soll, es zu knacken oder zu umgehen.“ Dieses Versteckenspielen führt jedoch dazu, dass man häufig nur auf Dokumente trifft, die nichts anderes sind, als Werbung für das eigene Konzept beziehungsweise Produkt. Erst mit vielem Suchen und nach Kontaktaufnahme zu verschiedenen Instituten, war es möglich, konkretere Informationen zu erhalten. Somit lässt sich allgemein sagen, dass die Informationsbeschaffung und -auswertung sehr schwierig war.
Im weiteren Verlauf der Entstehung der Arbeit traten bei der Implementierung des Prototypen ebenfalls Schwierigkeiten und Probleme auf. Diese waren jedoch vorwiegend technischer Natur und sind bereits in Kapitel 6 ausführlich beschrieben.

7.4 Ausblick in die Zukunft

Lizenzierung in Grid-Umgebung steckt immer noch in den Kinderschuhen.
Dies liegt unter anderem daran, dass noch keine große Nachfrage besteht und somit aus wirtschaftlicher Sicht die Entwicklung eines allumfassenden Lizenzmodells nicht tragbar ist. Erst in den letzten ein bis zwei Jahren setzen Unternehmen private Grids ein oder schließen sich zu (bis jetzt noch) relativ kleinen Virtuellen Organisation mit einem gemeinsamen Grid zusammen. Das Potential von Grids wird also gerade erst entdeckt, oft sind noch klassische Computing-Cluster im Einsatz, die wohl erst abgelöst werden, wenn ihre Rechenleistung nicht mehr ausreicht. Durchsucht man die einschlägigen Foren und Mailinglisten zum Thema Grids und Clouds, so gibt es nahezu ausschließlich Diskussionsteilnehmer aus dem universitären Umfeld. Einige Leute beteiligen sich ebenfalls an Diskussionen, weil sie entweder „mal von Grids gehört haben“ oder weil sie sich für „neueste Konzepte und Technologien interessieren“. Nur äußerst selten findet man Diskussionsteilnehmer, die sich als Mitarbeiter einer Firma oder Organisation zu erkennen geben, die explizit Grids für den produktiven Betrieb einsetzen. Dies zeigt, dass Grids noch nicht den, vor einigen Jahren prophezeiten, Durchbruch in Unternehmen feiern können. Erst wenn die Nachfrage nach dem Einsatz von kommerzieller Software in Grid-Umgebungen steigt, wird auch das Engagement in die Entwicklung eines allumfassenden Lizenzmodells für Grid-Umgebungen steigen. Andererseits betrachtet wird jedoch der Einsatz von Grids in Unternehmen nicht stark zunehmen, solange es kaum nutzbare kommerzielle Software gibt. Diese Tatsache stellt somit einen gewissen Teufelskreis dar, der trotz aller Bemühungen von Forschungseinrichtungen und Universitäten noch etwas länger ungebrochen bleiben wird.
Doch nicht nur die Tatsache, dass Grids sich nicht so schnell verbreiten, wie vorhergesagt, sondern auch Überlegungen, wie genau ein allumfassendes Lizenzmodell aussehen kann, müssen noch abgeschlossen werden und sowohl von den Independent Software Vendors als auch von den Unternehmen, die eine derart lizenzierte Anwendung einsetzen wollen, angenommen werden.
Die in dieser Arbeit betrachteten Lizenzmodelle sind zum jetzigen Zeitpunkt, einige Monate nach der genauen Untersuchung, immer noch nicht weiter fortgeschritten, sondern auf dem gleichen Stand. Das zeigt auch, dass scheinbar kein großes Interesse seitens der Unternehmen herrscht, so dass momentan Forschungs- und Entwicklungsgelder wohl eher in andere Gebiete abfließen. Ob dies eventuell auch in der Wirtschaftskrise des letzten Jahres begründet ist, ist nicht nachvollziehbar.
Definitiv lässt sich sagen, dass in den nächsten Monaten (oder ein bis zwei Jahren) der Einsatz von Grids zunehmen wird und folglich auch das Verlangen nach kommerzieller Software für Grids steigen wird. Somit ist die Konzeptionierung und Entwicklung eines umfassenden Lizenzmodells durchaus notwendig. Eventuell ist bereits in einigen Monaten mit einem marktreifen Prototypen von SmartLM zu rechnen. Inwieweit sich das in der Diplomarbeit entwickelte Lizenzmodell in der Zukunft weiterentwickelt, ist noch nicht vorhersehbar.

7.5 Eigene Worte und Fazit

Die Untersuchung und Erforschung verschiedenster Lizenzmodelle stellte sich als sehr komplex und umfangreich heraus. Jedoch war es sehr interessant zu sehen, dass es eine Menge unterschiedlicher Modelle gibt, die jedoch alle auf dem exakt gleichen Nenner beruhen.
Überrascht hat mich hingegen die Tatsache, dass Grids sich immer noch kaum weiter verbreitet haben, als sie es vor zwei Jahren waren. Das Lesen in vielen Diskussionsforen und Mailinglisten war ebenfalls interessant, um den aktuellen Stand der Grid-Communities zu erfahren. Grids sind momentan vorwiegend in Forschungsinstituten und Universitäten beheimatet. Das ist einerseits schade, da das Konzept von Grids sehr viele Möglichkeiten für ein Shared Computing bietet. Andererseits aber auch verständlich, denn viele Unternehmen sind mit ihren klassischen Clustern zufrieden und handeln ganz nach dem Motto: „Never change a running System!“
Bis Grids zum Standard werden, wird noch einige Jahre dauern. Auch wird sich erst in einigen Jahren zeigen, ob sich Grids überhaupt durchsetzen. Die Technologie von Clouds, die durchaus als Konkurrenz von Grids gesehen werden kann, ist ebenfalls sehr interessant. Clouds bieten den großen Vorteil, dass sie intuitiver zu bedienen sind und die Zeit in die Einarbeitung deutlich geringer ist als bei Grids. Da Zeit bekanntlich Geld ist, werden Clouds eventuell aus wirtschaftlicher Sicht bevorzugt. Bis Grids oder Clouds sich aber global durchsetzen, wird sich auch noch viel im Sinne der Lizenzierung von Software ändern. Eventuell springen auch noch bekannte Entwickler wie zum Beispiel Flexera Software (FLEXnet) auf den Zug auf und präsentieren ein Lizenzmodell für Grid-Umgebungen.

Abschließend möchte ich festhalten, dass die letzten Monate eine Vielzahl von Gefühlen bei mir hervorgerufen haben, die die Entstehung der Diplomarbeit unterschiedlich beschreiben. Zum einen war es eine informative Zeit, ich konnte viel neues Wissen erlangen. Zum anderen war es eine stressige Zeit, da viel zu tun war und weder das Ende noch der Weg zum Ziel erkennbar war. Auch Verzweiflung war manchmal spürbar, wenn ich vor den beschriebenen Problemen stand.
Zu guter Letzt steht jedoch das Glück im Vordergrund: Die Diplomarbeit ist abgeschlossen, ich habe viel dabei erfahren und ein Konzept entwickelt, das vielleicht irgendwann in der freien Wirtschaft anzutreffen ist.

8 Anhang

Abbildungsverzeichnis

Literaturverzeichnis

[AB03] Andresen, Fred und Ulrich Bantle: *Ich seh dich nicht - Fallstricke im End User License Agreement*. Linux-Magazin, 2007/03. http://www.linux-magazin.de/Heft-Abo/Ausgaben/2007/03/Ich-seh-dich-nicht.

[ACC+] Alfieri, R., R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lörentey und F. Spataro: *VOMS, an Authorization System for Virtual Organizations*. Technischer Bericht, INFN and Department of Physics, Parma, . http://grid-auth.infn.it/docs/VOMS-Santiago.pdf.

[ACD+07] Andrieux, Alain, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steven Tuecke und Ming Xu: *Web Services Agreement Specification (WS-Agreement)*. Technischer Bericht, Open Grid Forum, 2007. http://www.ogf.org/documents/GFD.107.pdf.

[Apa10a] Apache: *ActiveMQ*. Website, 2010. http://activemq.apache.org/.

[Apa10b] Apache: *Axis*. Website, 2010. http://ws.apache.org/axis/.

[Apa10c] Apache: *HTTP Server Project*. Website, 2010. http://httpd.apache.org/.

[Apa10d] Apache: *Tomcat*. Website, 2010. http://tomcat.apache.org/.

[BEi09] BEinGRID. Website, 2009. http://www.beingrid.eu/.

[BOI10] BOINC. Website, 2010. http://boinc.berkeley.edu/.

[Cec08] Ceccanti, Andrea: *VOMS Admin User's Guide*, 2008.

[DP09] Dalheimer, Matthias und Franz-Josef Pfreundt: *GenLM: License Management for Grid and Cloud Computing Environments*. Technischer Bericht, Fraunhofer Institut für Techno- und Wirtschaftsmathematik, 2009.

[DSSWR10] Dimitrakos, Theo, K. Stanoevska-Slabeva, Th. Wozniak und S. Ristol: *The BEinGRID Project*. In: *Grid and Cloud Computing: A Business Perspective on Technology and Applications*. Springer-Verlag Berlin Heidelberg, 2010.

[Ecl10a] Eclipse Foundation: *Eclipse*. Website, 2010. http://www.eclipse.org.

[Ecl10b] Eclipse Foundation: *Eclipse IDE for Java EE Developers*. Website, 2010. http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/galileosr2.

[FK98] Foster, Ian und Carl Kesselman: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[FK03] Foster, Ian und Carl Kesselman: *The Grid: Blueprint for a New Computing Infrastructure: Second Edition*. Morgan Kaufmann, 2003.

[FKNT02] Foster, Ian, Carl Kesselman, Jeffrey M. Nick und Steven Tuecke: *The Physiology of the Grid*. Technischer Bericht, Argonne National Laboratory, University of Chicago, University of Southern California, IBM Corporation, 2002. http://www.globus.org/alliance/publications/papers/ogsa.pdf.

[FKS+05] Foster, I., H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell und J. Von Reich: *The Open Grid Services Architecture*, 2005. http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf.

[FKT01] Foster, Ian, Carl Kesselman und Steven Tuecke: *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. Intl J. Supercomputer Applications, 2001. http://www.globus.org/alliance/publications/papers/anatomy.pdf.

[Fle10] Flexera Software. Website, 2010. http://www.flexerasoftware.com/.

[Fos02] Foster, Ian: *What is the Grid? A Three Point Checklist*. Technischer Bericht, Argonne National Laboratory, University of Chicago, 2002. http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf.

[g-E09] g-Eclipse. Website, 2009. http://www.geclipse.eu.

[GHP+08] Grimm, Christian, Benjamin Henne, Stefan Piger, Michael Schiffers und Wolfgang Ziegler: *Generische VO-Strukturen für das D-Grid*. Technischer Bericht, D-Grid, 2008.

[GNU10] GNU: *General Public License*. Website, 2010. http://www.gnu.org/licenses/gpl.html.

[Gri08] GridKA School: *g-Eclipse Tutorial @ GridKA School 2008*. Technischer Bericht, g-Eclipse, 2008. http://www.geclipse.eu/fileadmin/Documents/Documentation/gks08-handout.pdf.

[inS08] inSiDE - Innovatives Supercomputing in Deutschland: *Grid-friendly Software Licensing for Location independent Application Execution*. inSiDE, Vol. 6 No. 1 Spring 2008. http://inside.hlrs.de/htm/Edition_01_08/article_12.htm.

[IT09] IT-Tude: *SmartLM's flexible licensing virtualization technology makes licenses mobile objects*. Website, 2009. http://www.it-tude.com/smartlm-licenses-mobile-objects.html.

[IT10] IT-Tude: *SmartLM: Grid-friendly software licensing for location independent application execution*. Website, 2010. http://www.it-tude.com/smartlm.html.

[Leh09] Lehr- und Forschungseinheit für Kommunikationssysteme und Systemprogrammierung: *Ausschreibung der Diplomarbeit*. Website, 2009.

http://www.nm.ifi.lmu.de/teaching/Ausschreibungen/Diplomarbeiten/da-lizenz/.

[LZWM08] Li, Jiadao, Wolfgang Ziegler, Oliver Wäldrich und Daniel Mallmann: *Towards SLA Based Software License Management in Grid Computing*. Technischer Bericht, ERCIM, Fraunhofer Insitut, Research Centre Jülich (Jülich Supercomputing Centre), CoreGRID, 2008. http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0136.pdf.

[MBC+08] Mallmann, Daniel, Sergio Bernardi, Claudia Cacciari, Francesco D'Andria, Roberto d'Ippolito, Henning Eickenbusch, Naji El Masri, Andrés Gómez, Björn Hagemeier, Jiadao Li, Daniel Mallmann, Josep Martrat Sotil, Jose Carlos Mourino Gallego, Angela Rumpl, Eberhard Sekler, Christian Simmendinger, Devarajan Subramanian, Wolfgang Ziegler und Csilla Zsigri: *SmartLM - Funcional requirements of the new licensing architecture for Grids*. Technischer Bericht, SmartLM, 2008.

[MSZ06] Milke, Jens-Michael, Michael Schiffers und Wolfgang Ziegler: *Virtuelle Organisationen in Grids: Charakterisierung und Management*. Technischer Bericht, Forschungszentrum Karlsruhe, Ludwig-Maximilians-Universität München, Fraunhofer Institut SCAI, 2006. http://www.nm.ifi.lmu.de/pub/Publikationen/msz06/PDF-Version/msz06.pdf.

[MyS10a] MySQL: *Download*. Website, 2010. http://dev.mysql.com/downloads/mysql/.

[MyS10b] MySQL: *MySQL Connectors*. Website, 2010. http://dev.mysql.com/downloads/connector/.

[MZ09] Martrat, Josep und Wolgang Ziegler: *SmartLM - Broschüre*. Technischer Bericht, SmartLM, 2009. http://www.smartlm.eu/fileadmin/SmartLM/Download/smartLM_Broschuere_web.pdf.

[OAS10a] OASIS: *Advanced Open Standards For The Information Society*. Website, 2010. http://www.oasis-open.org/home/index.php.

[OAS10b] OASIS: *OASIS eXtensible Access Control Markup Language (XACML) TC*. Website, 2010. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[OAS10c] OASIS: *Web Services Resource Framework*. Website, 2010. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[Ope10a] Open Grid Forum: *Open Forum Open Standards*. Website, 2010. http://www.ogf.org/.

[ope10b] openSuse: *openSuse 11.2*. Website, 2010. http://de.opensuse.org/Willkommen_auf_openSUSE.org.

[Ora10] Oracle - Sun Developer Network: *JavaMail 1.4.3*. Website, 2010. http://java.sun.com/products/javamail/downloads/index.html.

[Pos10] PostgreSQL: *PostgreSQL 8.4.2-1.1.1*. Website, 2010. http://www.postgresql.org/download/.

[Rae09] Raekow, Yona: *Lizenzmanagement im GRID - Ein Ergebnis aus BEinGRID*, 2009. http://www.rrzn.uni-hannover.de/fileadmin/ful/mitarbeiter/wiebel/1ter_LizenzenWs/Folien_Lizenzmanagement_BEinGRID.pdf.

[Ric10] Ricchetti, Matteo: *SS5 Official Web Site*. Website, 2010. http://ss5.sourceforge.net/index.htm.

[RSKF09] Raekow, Yona, Christian Simmendinger und Otmar Krämer-Fuhrmann: *License management in grid and high performance computing*. Computer Science - Research and Development, Volume 23, Numbers 3-4 / Juni 2009. http://www.springerlink.com/content/4853871310322584/.

[SC06] Sotomayor, Borja und Lisa Childers: *Globus Toolkit 4 - Programming Java Services*. Morgan Kaufmann, 2006.

[Sch07] Schiffers, Michael: *Management dynamischer Virtueller Organisationen in Grids*. Doktorarbeit, Ludwig-Maximilans-Universität München, 2007.

[SET10] SETI@home. Website, 2010. http://setiathome.berkeley.edu/.

[SG08] Stümpert, Mathias und Ariel García: *g-Eclipse final Architecture*. Technischer Bericht, g-Eclipse, 2008. http://www.geclipse.eu/fileadmin/Documents/Deliverables/D1.8.pdf.

[Sim] Simmendinger, Christian: *Support for Client-Server based Licese Management Schemes in the Grid*. Technischer Bericht, T-Systems, BEinGRID, . http://www.it-tude.com/fileadmin/gridipedia/lm_cluster/LM_WHITEPAPER_V0.1-1.pdf.

[SLS10] Sotomayor, Borja, Marcos López und Txus Sánchez: *Globus Service Build Tools*. Website, 2010. http://gsbt.sourceforge.net.

[Sma10] SmartLM. Website, 2010. http://www.smartlm.eu/.

[Sun10a] Sun: *Java*. Website, 2010. http://developers.sun.com/downloads/top.jsp.

[Sun10b] Sun: *keytool - Key and Certificate Management Tool*. Website, 2010. http://java.sun.com/javase/6/docs/technotes/tools/solaris/keytool.html.

[Sun10c] Sun: *MySQL*. Website, 2010. http://www.mysql.de/.

[TCF+03] Tuecke, S., K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling und P. Vanderbilt: *Open Grid Services Infrastructure (OGSI) Version 1.0*, 2003. http://www.ggf.org/documents/GFD.15.pdf.

[The09] The Globus Alliance: *Globus Toolkit 4*. Website, 2009. http://www.globus.org/.

[The10] The Globus Alliance: *Globus Toolkit 4.0.8*. Website, 2010. http://www.globus.org/toolkit/downloads/4.0.8/.

[WGG+09] Wolniewicz, Pawel, Harald Gjermundrød, Sylva Girtelschmid, Nicholas Loulloudes, Thomas Köckerbauer, Jie Tao, Mariusz Wojtysiak und Ashish Thandavan: *g-Eclipse Technical Documentation*. Technischer Bericht, g-Eclipse, 2009. http://www.geclipse.eu/fileadmin/Documents/Deliverables/D2.9_D3.8_D4.8-TechnicalDocumentation.pdf.

[Wik10a] Wikipedia: *Endbenutzer-Lizenzvertrag*. Website, 2010. http://de.wikipedia.org/wiki/Endbenutzer-Lizenzvertrag.

[Wik10b] Wikipedia: *g-Eclipse*. Website, 2010. http://de.wikipedia.org/wiki/G-Eclipse.

[Wik10c] Wikipedia: *Grid-Computing*. Website, 2010. http://de.wikipedia.org/wiki/Grid-Computing.

[Wik10d] Wikipedia: *Lizenz*. Website, 2010. http://de.wikipedia.org/wiki/Lizenz.

[Wik10e] Wikipedia: *Service Level Agreement*. Website, 2010. http://de.wikipedia.org/wiki/Service-Level-Agreement.

[Wik10f] Wikipedia: *Virtual Organization Membership Service*. Website, 2010. http://en.wikipedia.org/wiki/Voms.

[Wik10g] Wikipedia: *Virtuelle Organisation*. Website, 2010. http://de.wikipedia.org/wiki/Virtuelle_Organisation.

[Wik10h] Wikipedia: *Webservice*. Website, 2010. http://de.wikipedia.org/wiki/Webservice.

[Wol09a] Wolniewicz, Pawel: *g-Eclipse Developer Guide*. Technischer Bericht, g-Eclipse, 2009. http://www.geclipse.eu/fileadmin/Documents/Deliverables/D2.9_D3.8_D4.8-DeveloperGuide.pdf.

[Wol09b] Wolniewicz, Pawel: *g-Eclipse User Guide*. Technischer Bericht, g-Eclipse, 2009. http://www.geclipse.eu/fileadmin/Documents/Deliverables/D2.9_D3.8_D4.8-UserGuide.pdf.

[Wor10] World Wide Web Consortium (W3C): *Web of Services*. Website, 2010. http://www.w3.org/standards/webofservices/.

[Zie07] Ziegler, Wolfgang: *Lizenzmodelle in Gridumgebungen*. Workshop, 2007. http://sugi.d-grid.de/de/schulungsinhalte/details.html?tx_sugi_pi2%5Bcontentid%5D=12.

[Zie08] Ziegler, Wolfgang: *SmartLM - SLA in Software License Technology for Grids, Clouds, SOA*. Technischer Bericht, SmartLM, Fraunhofer, 2008. http://www.dagstuhl.de/Materials/Files/09/09131/09131.ZieglerWolfgang.Slides.pdf.