

- Projektarbeit -

Entwicklung und Evaluation eines Raytracing-Dienstes für das Venice Service Grid

Alexander Arimond

14. Oktober 2009

Betreuer : Prof. Dr. Paul Müller
Dr. Markus Hillenbrand

AG Integrierte Kommunikationssysteme
Fachbereich Informatik
Technische Universität Kaiserslautern

Zusammenfassung

Das Rendern von 3D-Szenen mithilfe des Raytracing-Algorithmus ist ein aufwändiger Prozess, der auf einzelnen Rechnern unter Umständen mehrere Stunden oder Tage dauern kann. Eine Parallelisierung und Verteilung dieses Vorgangs verspricht eine Beschleunigung und eine damit einhergehende, höhere Produktivität des Renderns. Die parallele Verarbeitung kann durch Einbeziehung von Grid Computing realisiert und unterstützt werden. Grid Computing ermöglicht den einfachen Zugriff auf zusätzliche Hardware-Ressourcen wie CPUs, welche für das Rendern verwendet werden können.

Diese Projektarbeit beschäftigt sich mit der Entwicklung und Evaluation eines Raytracing-Dienstes für das Venice Service Grid, ein Framework zur Bereitstellung von Webservice-basierten Diensten, welches an der Technischen Universität Kaiserslautern entwickelt wird. Einen Schwerpunkt der Arbeit bildet die Einbeziehung von Grid Computing zur Parallelisierung der Raytracing-Funktionalität.

Inhaltsverzeichnis

1	Einleitung	5
1.1	Hintergrund	5
1.2	Ziele	5
1.3	Aufbau	5
2	Grundlagen	7
2.1	Grid Computing	7
2.1.1	Definition	7
2.1.2	Beispiele	8
2.1.3	Klassifizierungen	8
2.1.4	Allgemeine Architektur	8
2.2	Venice Service Grid	10
2.2.1	Überblick	10
2.2.2	Architektur	10
2.2.3	Entwicklung neuer Dienste	12
2.3	Globus Toolkit	13
2.3.1	Überblick	13
2.3.2	Architektur	13
2.3.3	Grid Resource Allocation Management	14
2.3.4	GridFTP	15
2.3.5	Sicherheit	15
2.4	Raytracing	16
2.4.1	Überblick	16
2.4.2	Algorithmus	17
2.4.3	Verteiltes Rendern	17
3	Entwicklung des Raytracing-Dienstes	19
3.1	Verwendete Werkzeuge und Software	19
3.1.1	Venice	19
3.1.2	Globus Toolkit	19
3.1.3	POV-Ray	19
3.1.4	Sonstige	20
3.2	Anforderungen an den Dienst	20
3.3	Entwurf des Dienstes	22
3.3.1	Entwurf der Operationen	22
3.3.2	Datentypen	22
3.3.3	Asynchrone Auftragsbearbeitung	25
3.3.4	Benachrichtigung des Benutzers	26
3.3.5	Status und Vorschau	27
3.3.6	Grundsätzlicher Ablauf des Renderns	27
3.3.7	Anbindung an Globus Toolkit 4	28
3.3.8	Dienst für Globus Toolkit 4	30
3.3.9	Authentifizierung am Grid	32
3.3.10	Fehlerbehandlung	35
3.3.11	Benutzerschnittstelle	36

4	Evaluation	38
4.0.12	Parameter	38
4.0.13	Umgebung	40
4.0.14	Durchführung	40
4.0.15	Ergebnisse	40
5	Related Work - Instant Grid	44
6	Zusammenfassung und Ausblick	45
A	Code-Beispiele	47
B	Experimentelle Ergebnisse	48
	Literaturverzeichnis	49
	Abkürzungsverzeichnis	51

1 Einleitung

1.1 Hintergrund

Heutzutage stellen Wissenschaft, Industrie, aber auch der gewöhnliche Nutzer, hohe Anforderungen an verfügbare Rechenkapazitäten. Immer aufwändigere Berechnungen und Anwendungen sowie enorme Mengen an zu verarbeitenden Daten machen die Nutzung hochskalierbarer, vernetzter Systeme notwendig. Eine mögliche Lösung für dieses Problem bietet das Grid Computing. Durch geeignete Middleware können große Rechen- und Speicherkapazitäten vernetzt und auf einfache Weise zugänglich gemacht werden.

Auch das Rendern von dreidimensionalen Szenen mithilfe des Raytracing-Algorithmus stellt ein aufwändiges Verfahren dar. Komplexe Beschreibungen von 3D-Szenen führen zu Laufzeiten des Renderns, die auf einzelnen Rechnern unter Umständen mehrere Tage benötigen können. Auch hier kann Grid-Computing helfen, indem es eine parallele Ausführung des Renderns auf vielen Rechenknoten im Grid ermöglicht, und dadurch die Ausführung deutlich beschleunigt.

1.2 Ziele

Diese Projektarbeit beschäftigt sich mit der Entwicklung und Evaluation eines Raytracing-Dienstes für das Venice Service Grid. Das Venice Service Grid ist ein Framework zur Bereitstellung von Webservice-basierten Diensten, welches von der Arbeitsgruppe Integrierte Kommunikationssysteme (ICSY) des Fachbereichs Informatik an der Technischen Universität Kaiserslautern entwickelt wird. Besonderer Schwerpunkt der Projektarbeit ist die Integration des entwickelten Dienstes mit der Grid Computing Middleware Globus Toolkit 4. Ziel dabei ist es, die Funktionalität des Dienstes durch Nutzung externer Grid-Ressourcen performanter zu machen.

1.3 Aufbau

Im ersten Kapitel der Projektarbeit werden die Grundlagen erklärt. Es wird eine allgemeine Einführung in das Thema Grid-Computing sowie einen Überblick über das Venice Service Grid als auch über das Globus Toolkit 4 geben. Schließlich wird der Algorithmus des Raytracings im Allgemeinen beleuchtet.

Im dritten Kapitel wird die Entwicklung des Raytracing-Dienstes beschrieben. Zunächst werden kurz die verwendete Software und Werkzeuge genannt. Dann werden die Anforderungen an den Dienst definiert. Diese führen dann zum Entwurf des Dienstes. Es werden die Operationen und Datentypen des Dienstes erläutert, der grundsätzliche interne Ablauf des Dienstes skizziert, sowie auf die Realisierung einzelner Anforderungen eingegangen. Ein Schwerpunkt bildet die Integration mit dem Globus Toolkit 4. Hierzu wird ein eigener Dienst spezifiziert, welcher hier ebenfalls kurz vorgestellt wird. Ein weiteres wichtiges Thema stellt die Authentifizierung von Benutzern im Rahmen des Globus Toolkit 4 dar. Letzlich wird beispielhaft die entwickelte Benutzerschnittstelle vorgestellt. Diese vermittelt nochmal einen Gesamteindruck der Funktionalität des Dienstes aus Benutzersicht.

Im vierten Kapitel wird der Dienst hinsichtlich seiner Performanz evaluiert. Es werden Experimente beschrieben, welche die Möglichkeit der Beschleunigung des Renderns durch externe Grid-Ressourcen aufzeigen.

Das fünfte Kapitel gibt einen kurzen Einblick in ein Projekt namens Instant Grid. In diesem Projekt wird ähnliche Funktionalität bereitgestellt wie im hier vorgestellten Raytracing-Dienst. Auch wird hier ein kleiner Vergleich der beiden Ansätze unternommen.

Schließlich gibt es im letzten Kapitel eine Zusammenfassung der vorliegenden Projektarbeit und ihrer wichtigen Aspekte sowie einen Ausblick, in dem Möglichkeiten der Erweiterung der Dienste benannt werden.

2 Grundlagen

In diesem Kapitel werden Grundlagen geschaffen, die zum Verständnis der Arbeit notwendig oder hilfreich sind. Es gibt eine Einführung in das Thema Grid Computing. In diesem Zusammenhang werden die zwei wesentlichen Technologien, um die es in dieser Projektarbeit geht, beschrieben: Das Venice Service Grid sowie das Globus Toolkit. Ausserdem wird der für das Anwendungsszenario verwendete Algorithmus Ray Tracing erläutert.

2.1 Grid Computing

2.1.1 Definition

Grid Computing ist ein relativ neues Paradigma der Datenverarbeitung und ein aktueller Gegenstand der Wissenschaft. Auf die Frage, was Grid Computing ist, gibt es viele Antworten und Definitionen. Auch werden diese Definitionen teilweise noch angepasst, aufgrund moderner Entwicklungen und der Aktualität des Themas. Grundsätzlich hat Grid Computing zum Ziel, eine Integration von Netzwerken, Kommunikation, Datenverarbeitung und Information zu erreichen, um so eine virtuelle Plattform für den Zugang zu Datenverarbeitung und Datenverwaltung zu schaffen, ähnlich wie das Internet, welches Ressourcen integriert, um eine virtuelle Plattform für den Zugang zu Information bereitzustellen [1, 2].

Die entstehende Plattform, also "Das Grid", soll die Infrastruktur für Datenverarbeitung und -verwaltung bieten, welche als Grundlage für eine globale Gesellschaft und einen Wandel in Industrie, Regierung, Forschung & Wissenschaft sowie in der Unterhaltung dienen soll.

Einer früheren Definition von Ian Foster und Carl Kesselman zufolge ist ein Grid eine Hardware- und Software-Infrastruktur, die einen zuverlässigen, konsistenten, von überall erreichbaren und preiswerten Zugriff auf die Kapazitäten von Hochleistungsrechnern ermöglicht [1].

Zusammen mit Steven Tuecke erweiterten sie dieses Konzept unter besonderer Betrachtung sogenannter *Virtueller Organisationen* (VO), und definierten das Grid-Problem, anhand dessen die Anforderungen an ein Grid deutlich werden: Es soll eine flexible, sichere, und koordinierte Verteilung von Ressourcen zwischen dynamischen Sammlungen von Individuen, Institutionen und Ressourcen (was sie dann als virtuelle Organisationen bezeichnen) ermöglicht werden [3]. Eine weitere Aussage ist, dass beim Grid Computing, anders als beim konventionellen verteilten Rechnen, großer Fokus auf hochskalierbares Verwalten von Ressourcen, innovative Applikationen und Hochperformanz gelegt wird.

Später grenzte Ian Foster Grid Computing gegenüber konventionellem verteilten Rechnen ab, indem er eine "Three Point Checklist" [4] aufstellt:

Ein Grid ist ein System, welches ...

1. ... Ressourcen koordiniert, die nicht unter zentralisierter Kontrolle stehen
2. ... offene Standards, allgemeingebrauchliche Protokolle und Schnittstellen verwendet
3. ... nicht-triviale Quality of Services ermöglicht (z.B. bezüglich Antwortzeiten, Durchsatz, Verfügbarkeit, Sicherheit etc.)

2.1.2 Beispiele

Ein besseres Verständnis des Begriffes Grid Computing erhält man bei der Betrachtung konkreter Anwendungsszenarien. So halten Methoden und Technologien des Grid Computings Einzug in verschiedensten Bereichen wissenschaftlicher und kommerzieller Natur.

Ein Paradebeispiel für die Notwendigkeit des Einsatzes von Grid Computing bieten die enormen rechnerischen Anforderungen des Large Hadron Colliders (LHC) am CERN in der Schweiz. Der LHC ist ein Teilchenbeschleuniger mit einem Umfang von 27 Kilometern, und ist die größte Maschine, die bisher von Menschen gebaut worden ist. Man schätzt, dass ab ihrer Inbetriebnahme jährlich ca. 15 Petabytes an Daten physikalischer Experimente anfallen, die verarbeitet und analysiert werden müssen. Zu diesem Zweck wurde das World LHC Computing Grid Projekt (WLCG) ins Leben gerufen, welches die Ressourcen von 170 Rechenzentren aus 34 Ländern verknüpft, um die anfallenden Berechnungen zu verteilen [5].

Ein Beispiel, mit dem sich diese Projektarbeit auseinandersetzt, ist das Rendern von 3D-Szenen. Bei der verwendeten Technik des Raytracings ist es möglich, das Rendern zu parallelisieren und dadurch zu beschleunigen. Die Allokation der Ressourcen, die zum Rendern benötigt werden, übernimmt dabei ein Grid-Dienst. Ein möglicher Anwendungsbereich für dieses Szenario bietet die Unterhaltungsindustrie. Im Bereich der Animationsfilme fallen stets große Mengen an 3D-Szenen an, die gerendert werden müssen. Der Bedarf an entsprechenden Ressourcen zum Rendern ist daher sehr hoch. Grid-Computing ist ein mögliches Hilfsmittel zur Unterstützung dieser Prozesse, indem es entsprechende Ressourcen auf einfache Art bereitstellt.

2.1.3 Klassifizierungen

Man kann im Allgemeinen drei verschiedene Typen von Grids unterscheiden: *Compute Grids*, *Data Grids* und *Service Grids* [6]. Ein Compute Grid stellt Ressourcen zur Verfügung, um zeitaufwändige Berechnungen mit hohem Durchsatz durchführen zu können. Ein Data Grid ermöglicht das sichere, verteilte Speichern großer Datenmengen. Ein Service Grid nutzt letztlich diese Grid-Typen und schafft Mehrwert für den Endbenutzer, indem es spezielle Dienste bereitstellt, wie zum Beispiel virtualisierte Applikationen oder entsprechende Algorithmen, und für diese Dienste dann auch intuitive graphische Schnittstellen anbietet.

Im Folgenden wird das Venice Service Grid, sowie das Globus Toolkit vorgestellt. Wie wir später sehen werden, wird das Venice Service Grid in dieser Projektarbeit hauptsächlich als Service Grid genutzt, während das Globus Toolkit als Middleware dient, um auf Compute Grid Ressourcen zuzugreifen.

2.1.4 Allgemeine Architektur

In *The Anatomy Of The Grid* [3] beschreiben Foster, Kesselman und Tuecke eine abstrakte Architektur für den Aufbau von Grids. Dabei gehen sie mehr auf die generellen Klassen von Komponenten ein, die für ein Grid benötigt werden, anstatt konkrete Protokolle oder Technologien aufzuzählen. Die Architektur folgt einem Schichtenmodell, welches aus fünf Komponenten besteht. Diese werden im Folgenden kurz erläutert. Diese Einteilung wird in der Projektarbeit später referenziert, um die entwickelten und genutzten Komponenten einzuordnen.

Fabric Layer. Diese Schicht stellt die eigentlichen Ressourcen bereit, auf die der Zugriff durch die Protokolle des Grids ermöglicht wird. Dies sind beispielsweise Rechner, Speichersys-

teme, Katalogsysteme, Netzwerkressourcen oder Sensoren. Auch logische Entitäten wie Verteilte Speichersysteme (DFS), Cluster oder Verteilte Rechner-Pools zählen dazu.

Connectivity Layer. Diese Schicht definiert die Kommunikations- und Authentifizierungsprotokolle für Grid-spezifische Netzwerktransaktionen. Sie ermöglicht damit Kommunikation zwischen den Ressourcen des Fabric Layers und bietet kryptographische Sicherheitsmechanismen zur Verifizierung der Identitäten von Benutzern und Ressourcen.

Resource Layer. Aufbauend auf den Protokollen des Connectivity Layers definiert der Resource Layer Protokolle, um u.a. das sichere Aushandeln, Initiieren, Überwachen und Kontrollieren von Operationen auf individuellen Ressourcen zu ermöglichen. Die Implementierungen dieser Protokolle nutzen Funktionen des Fabric Layer, um Zugriff und Kontrolle über dessen lokale Ressourcen zu erhalten. Wichtig ist, dass der Resource Layer nur individuelle Ressourcen betrachtet, und nicht den globalen Zustand oder den atomaren Zugriff auf verteilte Sammlungen von Ressourcen.

Collective Layer. Während der Resource Layer nur einzelne Ressourcen betrachtet, beinhaltet der Collective Layer Protokolle und Dienste, die eher globaler Natur sind und Interaktionen zwischen ganzen Sammlungen von Ressourcen abdecken. Darunter fallen beispielsweise *Directory Services, Monitoring and Diagnostics Services (MDS)* oder *Data Replication Services*.

Application Layer. In dieser Schicht befinden sich alle Applikationen einer Virtuellen Organisation. Diese bauen auf den Diensten aller anderen Schichten auf. Einen Überblick über den Zusammenhang der Schichten gibt Abbildung 1.

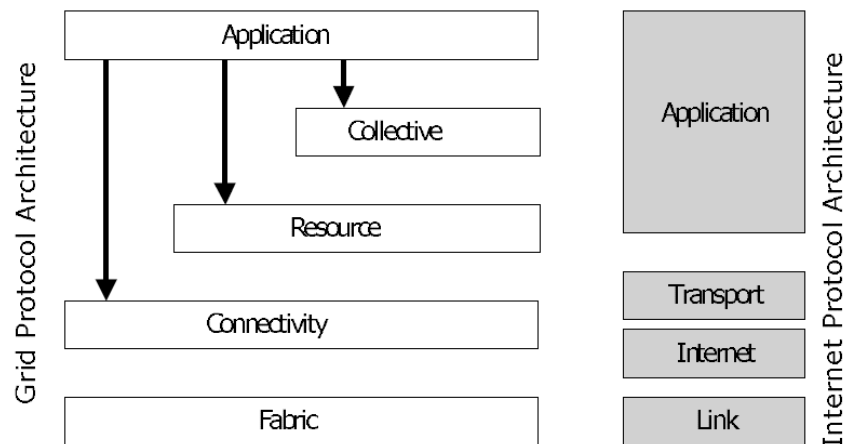


Abbildung 1: Architektur eines Grids (aus [3])

2.2 Venice Service Grid

2.2.1 Überblick

Das Venice Service Grid [7] wird von der Arbeitsgruppe Integrierte Kommunikationssysteme (ICSY) des Fachbereichs Informatik an der Technischen Universität Kaiserslautern entwickelt. Es entstand aus einem Projekt über Voice-over-IP (VoIP) in Zusammenarbeit mit der Siemens AG.

Ziel ist es, ein leichtgewichtiges Service Grid [6](siehe auch Kapitel 2.1.3) zu entwickeln. Nur wenige solcher Service Grids existieren zurzeit und ihre Entwicklung steckt meist noch in den Kinderschuhen. Das Globus Toolkit 4 (siehe Kapitel 2.3) bietet zwar schon Dienst-Funktionalitäten an, ist aber keineswegs leichtgewichtig. Leichtgewichtig meint hierbei, dass ein Grid leicht einzusetzen, zu warten und zu benutzen sein sollte. Außerdem liegt der Fokus beim Globus Toolkit mehr auf der Verwendung als Compute-Grid.

Das Venice Service Grid ist ein offenes Framework für verteilte Applikationen, welches ein leichtes Erstellen, Entwickeln, Integrieren und Benutzen von Diensten ermöglicht. Dies soll erreicht werden, indem die nutzbaren Ressourcen virtualisiert werden und von den verwendeten Technologien abstrahiert wird.

2.2.2 Architektur

Die Dienste im Venice Service Grid sind als Webservices implementiert, aufbauend auf dem Tomcat¹ Servlet Container und der Axis² SOAP engine. Die Schnittstellen der Dienste werden beschrieben mit Hilfe der Web Services Description Language (WSDL) und kommunizieren mit SOAP über HTTP. Alle Datentypen, die zum Austausch von Daten zwischen den Diensten verwendet werden können, sind in öffentlich zugänglichen XML Schemata spezifiziert. Dadurch können die WSDL und XML Schemata in den Dienstbeschreibungen wiederverwendet werden. Durch die Nutzung von Web Services werden Standardisierung, Flexibilität und Interoperabilität ermöglicht, sowie plattform- und sprachunabhängige Implementierungen.

Die Dienste, die das Venice Service Grid anbietet, lassen sich im Wesentlichen in drei Kategorien aufteilen: Da wären die *Management Services*, die zur allgemeinen Verwaltung von Benutzern, Ressourcen und Diensten notwendig sind. Die *Information Services*, die für das Sammeln, Verwalten und Verteilen von Daten verantwortlich sind. Und die *Application Services* sind letztendlich alle horizontalen und vertikalen Dienste. Horizontal bedeutet dabei, dass alle Applikationen, die auf dem Framework basieren, diese Dienste nutzen können, während vertikal bedeutet, dass die Dienste eher domänenspezifisch, d.h. nur für bestimmte Applikationen aus dem entsprechenden Anwendungsgebiet nützlich sind (siehe auch Abbildung 2). Im Folgenden sollen einige der Dienste beispielhaft vorgestellt werden.

- **SSO** Eine typische Verwaltungsaufgabe für ein Grid betrifft die Authentifizierung der Benutzer. Im Venice Service Grid wird dies durch eine Token-basierte Single Sign-on (SSO) Strategie gelöst. Der Benutzer authentifiziert sich einmal an dem entsprechenden Dienst und erhält im Gegenzug ein sogenanntes Token, mit dem er sich bei weiterer Kommunikation mit den Diensten des Grid ausweisen kann. Dabei dient das Token nicht nur der Authentifizierung, sondern auch der Bestimmung von Benutzungsrechten für einzelne Dienste, also Authorisierungsaspekten. Da in einem Anwendungsszenario oft mehrere

¹Apache Tomcat: <http://tomcat.apache.org>

²Apache Axis: <http://ws.apache.org/axis>

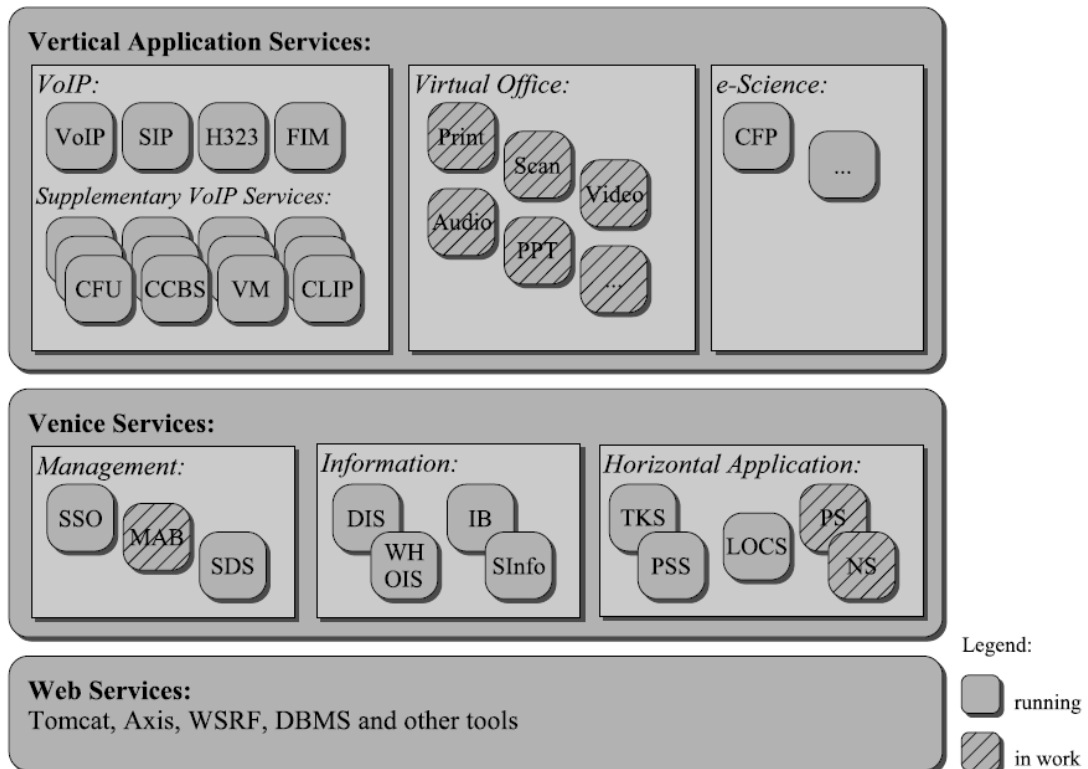


Abbildung 2: Überblick über die Venice Architektur (aus [6])

Dienste involviert sind, ist eine Single Sign-on Strategie von Vorteil, da sich der Benutzer bei der Nutzung verschiedener Dienste nicht jedesmal neu authentifizieren muss, indem er beispielsweise ein Passwort angibt, sondern das Token die Identität des Client sicherstellt.

- **DIS und WHOIS** Ein Beispiel für einen Informationsdienst ist der *Domain Information Service* (DIS). Dieser bietet sämtliche Informationen an, die für die Nutzung von Diensten in der entsprechenden Domäne notwendig sind. Unter anderem beinhaltet dies auch Informationen über den Single Sign-on Dienst und die Domäne selbst. Informationen über die Nutzer innerhalb einer Domäne erhält man über den *Who Is Service* (WHOIS). Um die Privatsphäre zu gewährleisten, muss der entsprechende Nutzer vorher sein Einverständnis geben, damit man dessen Daten einsehen kann.
- **PSS** Ein horizontaler Dienst ist der *Property Storage Service* (PSS). Dieser erlaubt es, Schlüssel/Wert-Paare zu speichern. Dieser Dienst bietet großes Potenzial als Hilfsdienst, beispielsweise um Benutzer- oder Dienstinstellungen zu sichern, oder um sonstige beliebigen Daten abzuspeichern.
- **NS** Ein weiterer nützlicher Dienst ist der *Notification Service* (NS). Dieser bietet einen publish/subscribe-Mechanismus für Ereignisse. Dienste können Ereignisse definieren und sich für bestimmte Ereignisse eintragen lassen. Tritt ein Ereignis dann ein, werden alle eingetragenen Dienste benachrichtigt. Durch den Notification Dienst wird sogenanntes

Polling reduziert, d.h. Dienste müssen nicht ständig nachfragen, ob ein Ereignis eingetroffen ist, sondern werden automatisch bei Eintritt des Ereignisses benachrichtigt.

- **E-Mail und SMS** In Sachen Kommunikation bietet Venice Dienste an, die über einfache Schnittstellen das Versenden von E-Mails und SMS ermöglichen.
- **TimeKeeper** Der TimeKeeper Service bietet eine einfache Schnittstelle, um zeitliche Messungen durchzuführen. So kann die Dauer von Dienstauführungen persistent gespeichert und über einen längeren Zeitraum statistisch ausgewertet werden, um u.a. das arithmetische Mittel, den Median oder die Standardabweichung zu erhalten. Auch grafische Darstellungen der Statistiken sind möglich, beispielsweise in Form einer kumulativen Verteilungsfunktion über die Ausführungszeit eines Dienstes. Anhand dieser lässt sich einschätzen, mit welcher Wahrscheinlichkeit ein Dienstaufwurf nach einer bestimmten Zeitspanne beendet ist.

2.2.3 Entwicklung neuer Dienste

Venice verfolgt bei der Entwicklung von Diensten den “Contract-first”-Ansatz³ [8]. Das bedeutet, dass nach der Anforderungsanalyse und der dienst-orientierten Entwicklungsphase die Schnittstellen und die Funktionalität des Dienstes wohldefiniert sind. Dies betrifft sowohl die Operationen, als auch die entsprechenden Datentypen. Danach sind die wesentlichen Schritte bei der Schaffung neuer Dienste das Erstellen und Kompilieren der benötigten Schnittstellen, Bindungen und Klassen, die eigentliche Implementierung und Konfiguration des Dienstes, sowie schließlich die Verwendung und Verwaltung des Dienstes (siehe auch Abbildung 3).

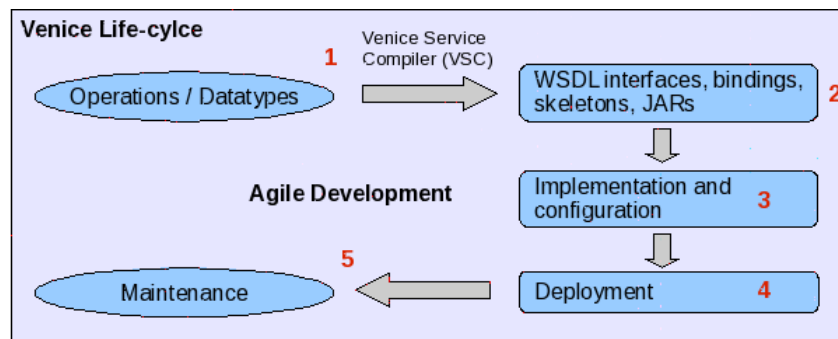


Abbildung 3: Der Lebenszyklus eines Dienstes (aus [7])

Dieser Lebenszyklus von Diensten wird im Venice Service Grid vom sogenannten *Venice Service Compiler* unterstützt. Dieser ist Teil einer Abstraktionsschicht innerhalb von Venice, die zum Ziel hat, die Entwicklungsphase eines Dienstes unabhängig von der zugrundeliegenden Webservice-Technologie zu machen. So findet die Definition der Dienstoperationen in einer eigens entwickelten abstrakten Syntax statt. Aus dieser abstrakten Definition des Dienstes generiert der Venice Service Compiler Dienstbeschreibungen im WSDL-Format, und darauf aufbauend mit Hilfe des in Axis enthaltenen Werkzeugs *WSDL2Java* entsprechende Java-Klassen

³Ein Kontrakt im Sinne eines Dienstes ist die Menge aller Daten, die veröffentlicht werden, um den Dienst für Andere verfügbar und zugreifbar zu machen.

als Grundlage für die Implementierung und Integration der Dienste und Datentypen innerhalb des Axis-Frameworks⁴.

Der Venice Service Compiler findet in dieser Projektarbeit bei der Entwicklung des Ray-tracing-Dienstes Verwendung. Besonderes Augenmerk wird hier auf die Definition der Operationen sowie der für den Dienst benötigten Datentypen gerichtet.

2.3 Globus Toolkit

2.3.1 Überblick

Das Globus Toolkit [9] ist eine Middleware für die Schaffung von Grid-Systemen, die seit den späten 90er Jahren entwickelt wird, um wiederum die Entwicklung von dienst-orientierten, verteilten Applikationen und Infrastrukturen zu unterstützen [10]. Die Komponenten des Globus Toolkit beschäftigen sich dabei unter anderem mit wichtigen Kernpunkten verteilter Systeme wie Sicherheit, Zugang zu Ressourcen, Verwaltung von Ressourcen, das Finden von Ressourcen und die Verteilung von Daten. Aufbauend auf oder interagierend mit diesen Komponenten findet sich ein breiteres "Globus Ökosystem" von Werkzeugen und weiteren Komponenten, das eine Fülle von nützlichen Funktionalitäten bereitstellt. Mit Hilfe dieser Werkzeuge wurden bereits verschiedene Arten von Grid-Infrastrukturen und verteilten Applikationen entwickelt.

Das Globus Toolkit, aktuell in seiner vierten Version (GT4), basiert dabei auch auf Web Services Technologie, was gegenüber vorherigeren Versionen höhere Robustheit, Performanz, Benutzbarkeit, Dokumentation, Kompatibilität und Funktionalität verspricht. Es nutzt Web Services, um Schnittstellen zu definieren und die internen Komponenten zu strukturieren. Web Services nutzen flexible, erweiterbare und angenommene XML-basierte Mechanismen zur Beschreibung, zum Finden und zum Aufruf von Netzwerkdiensten.

2.3.2 Architektur

In Abbildung 4 erhält man einen Überblick über einige Aspekte der GT4 Architektur. Man kann drei Mengen von Komponenten ausmachen:

- Eine Menge von Diensten implementiert nützliche Infrastrukturfunktionalitäten. Dabei geht es um Ausführungsverwaltung (GRAM), Datenzugriff und Datenübertragung (Grid-FTP, RFT, OGSA-DAI), Replikationsmanagement (RLS, DRS), Monitoring und Auffindung (Index, Trigger, WebMDS), Rechtemanagement (MyProxy, Delegation). Die meisten Dienste sind dabei als Java Web Services implementiert. Auf GRAM und GridFTP, sowie das Rechtemanagement, wird in den Kapiteln 2.3.3, 2.3.4 und 2.3.5 näher eingegangen, da diese Funktionalitäten auch in der Projektarbeit verwendet werden.
- Es existieren drei sogenannte Container, in denen man selbst entwickelte Dienste in Java, C oder Python anbieten kann. Die Container an sich unterstützen dabei Aspekte wie Sicherheit, Verwaltung, Auffindung von Diensten und andere Mechanismen, die häufig beim Aufbau von Diensten benötigt werden.

⁴Außerdem werden auch noch WSDD-Dateien für Axis, sowie Dokumentation in Form von HTML und Java-Doc generiert.

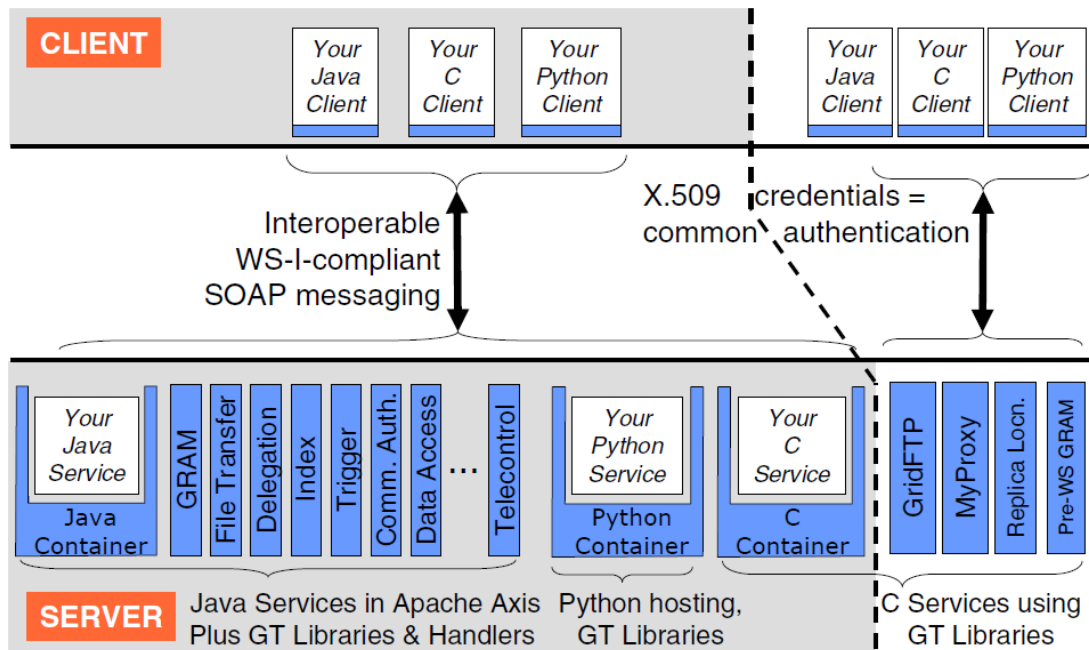


Abbildung 4: Aspekte der GT4 Architektur (aus [10])

- Eine Menge von Client-Bibliotheken erlaubt es Programmen, die beispielsweise in Java, C oder Python geschrieben sind, GT4-Dienste oder von Benutzern entwickelte Dienste zu nutzen.

2.3.3 Grid Resource Allocation Management

Wenn man einen Auftrag auf einem Rechner ausführen will, so ist es nötig, diesen zu konfigurieren und an die Bedürfnisse des Auftrages anzupassen, ein ausführbares Programm bereitzustellen, die Ausführung in Gang zu setzen, und letztendlich die Ausführung zu kontrollieren und die Ergebnisse zu verwalten.

In GT4 können diese Aufgaben vom Grid Resource Allocation Management (GRAM) unterstützt werden. Das GRAM bietet eine Web Service Schnittstelle, über die man beliebige Berechnungen auf entfernten Rechnern initiieren, beobachten und verwalten kann. Ausserdem ist es möglich über diese Schnittstelle verschiedenste Bedingungen der Ausführung auszudrücken, wie beispielsweise die Art und Menge der benötigten Ressourcen, welche Daten zum Ort der Ausführung transportiert werden müssen, welches Programm genau ausgeführt werden soll und wie dessen Argumente aussehen, und welche Zugangsberechtigungen benutzt werden. Über andere Operationen können Clients den Status eines Auftrages abfragen.

Das GRAM, als ein Protokoll zur Allokation von Rechenressourcen und für das Überwachen und Kontrollieren von Berechnungen auf diesen Ressourcen, bettet sich im Sinne der in Kapitel 2.1.4 beschriebenen Architektur in den Ressource Layer ein [3].

2.3.4 GridFTP

Globus-Applikationen müssen oft große Datenmengen verwalten und integrieren, die an verschiedensten Orten vorliegen können. Dieses Datenproblem ist sehr breit und komplex. Es existieren jedoch einige GT4-Komponenten, die in diesem Zusammenhang nützliche Funktionalität bereitstellen, um dieses Problem zu lösen. Einer dieser Mechanismen ist GridFTP. Die Implementierung der GridFTP-Spezifikation stellt Bibliotheken und Werkzeuge für sichere, hoch-performante Datenübertragung [11] bereit. So wurden Übertragungsraten von 27 Gbit/s über WANs gemessen. Ausserdem kann GridFTP mit konventionellen FTP Clients und Servern interoperieren. Ebenso wie das GRAM findet sich GridFTP als Management-Protokoll für Datenzugriff im Ressource Layer wieder [3].

2.3.5 Sicherheit

Die Sicherheit im Globus Toolkit 4 wird mit Hilfe der *Grid Security Infrastructure* (GSI) umgesetzt. Im Wesentlichen basiert diese auf dem TLS-Protokoll [12], Public-Key-Kryptographie⁵ und digital signierten Zertifikaten, um so die Sicherheit auf Transport-Ebene gewährleisten zu können [14]. GSI ordnet sich damit in den in Kapitel 2.1.4 beschriebenen Connectivity Layer ein [3].

Jeder Nutzer von Grid-Ressourcen kann sich durch ein eigenes Zertifikat im X.509-Format⁶ ausweisen und so an einer Grid-Ressource authentifizieren. Ein solches Zertifikat beinhaltet Informationen über die Identität des Nutzers, einen öffentlichen Schlüssel, der zu dieser Identität gehört, Informationen über die zuständige *Certificate Authority* (CA) sowie deren digitale Signatur. Die CA ist eine übergeordnete Instanz, meist eine Organisation, die Zertifikate ausstellt und signiert. "Vertraut" der Besitzer der Grid-Ressource der CA, so kann er anhand der Signatur des Zertifikats die Echtheit des Zertifikats feststellen, und dann anhand des Zertifikats die Identität des Nutzers sicherstellen⁷.

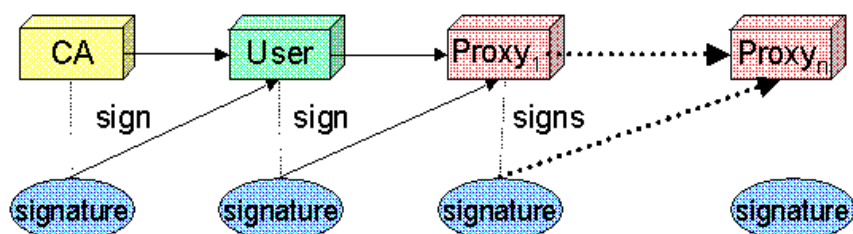


Abbildung 5: Signieren von Zertifikaten⁷

Um das Delegieren von Aufgaben und Daten, als auch Single Sign-On zu unterstützen, nutzt die GSI sogenannte Proxy-Zertifikate. Diese können aus einfachen Zertifikaten generiert und signiert werden und besitzen nur eine begrenzte zeitliche Gültigkeit. Dadurch können Rechte des Benutzers auf einfache Weise an zusätzliche Ressourcen weitergereicht werden, ohne dass dies jedesmal durch den Zertifikatsbesitzer (beispielsweise durch Eingabe eines Passwortes) bestätigt werden muss. Entfernte Instanzen können also mit einem Proxy-Zertifikat im Namen

⁵Funktionsweise zum Beispiel nachzulesen in [13]

⁶X.509: <http://tools.ietf.org/html/rfc3280>

⁷Details dazu unter <http://www-unix.globus.org/toolkit/docs/4.0/security/>. Siehe Key Concepts.

des Benutzers auf weitere Ressourcen zugreifen. Dies nennt man Delegation. Nach Ablauf der zeitlichen Begrenzung ist das Proxy-Zertifikat wertlos, sodass eine unzulässige Aneignung oder Weiterverwendung des Proxy-Zertifikats (mitsamt privatem Schlüssel) seitens Dritter nicht die sicherheitstechnischen Konsequenzen hat, die eine Aneignung des Original-Zertifikats mit sich brächte. Zusätzlich kann ein Proxy-Zertifikat auch noch bestimmte Informationen enthalten, welche die Rechte bei der Nutzung desselbigen weiter einschränken. Abbildung 5 demonstriert den Vorgang des Signierens von Zertifikaten.

In dieser Projektarbeit wird die Verwendung von Proxy-Zertifikaten im Kontext des entwickelten Raytracing-Dienstes näher beleuchtet und verschiedene Arten der Bereitstellung des Proxy-Zertifikats erläutert.

2.4 Raytracing

2.4.1 Überblick

Rendern oder Bildsynthese bezeichnet in der 3D-Computergrafik die Erzeugung eines Bildes aus einer Szene. Eine Szene ist ein virtuelles räumliches Modell, das Objekte und deren Materialeigenschaften, Lichtquellen, sowie die Position und Blickrichtung eines Betrachters definiert. Computerprogramme zum Rendern von Bildern werden **Renderer** genannt. [15]

Hauptaufgaben des Renderns sind die Ermittlung der vom virtuellen Betrachter aus sichtbaren Objekte (Verdeckungsberechnung), die Simulation des Aussehens von Oberflächen, beeinflusst durch deren Materialeigenschaften (Shading), sowie die Berechnung der Lichtverteilung innerhalb der Szene, die sich unter anderem durch die indirekte Beleuchtung zwischen Körpern äußert.

Welche Techniken beim Rendern eingesetzt werden, hängt vor allem von der Anwendungsdomäne ab. Ein wesentlicher Punkt dabei ist, ob das Rendern in Echtzeit stattfinden muss, oder ob die verwendete Rechenzeit eine untergeordnete Rolle spielt.

Beim fotorealistischen Rendern von 3D-Szenen kommt häufig die gängige Technik des **Raytracing** (dt. Strahlenverfolgung) zum Einsatz. Diese zeichnet sich durch hohe Einfachheit und Eleganz in der Idee und durch leichte Implementierung aus [16].

Raytracing ist ein Algorithmus zur Berechnung der Sichtbarkeit von dreidimensionalen Objekten von einem bestimmten Punkt im Raum aus. Das Verfahren basiert auf der Rückverfolgung von Lichtstrahlen, und wird vorrangig in der Computergrafik (aber auch anderen Anwendungsgebieten) eingesetzt, um 3D-Szenen auf zweidimensionale Bildflächen zu projizieren. Dabei wird berechnet, welche Objekte der 3D-Szene sichtbar sind und welche nicht. Außerdem werden verschiedene Merkmale der Oberflächen der Objekte ermittelt, wie zum Beispiel Farbe, Textur, Beschaffenheit, sowie die Beleuchtung der Objekte durch in der Szenerie vorhandene Lichtquellen. Insgesamt werden diese Eigenschaften dann auf die einzelnen Pixel der Bildebene abgebildet.

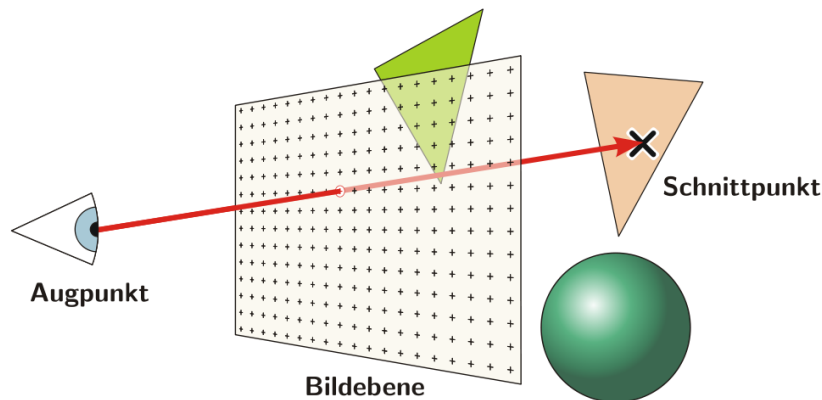


Abbildung 6: Die Lichtstrahlen werden rückwärts bis hin zur Lichtquelle verfolgt (aus [17])

2.4.2 Algorithmus

Der grundlegende Algorithmus basiert auf der Rückverfolgung (engl. backward ray tracing⁸) von Lichtstrahlen, die von einer vorgegebenen 3D-Szene, respektive ihrer Lichtquellen, ausgesandt werden, und letztendlich im Auge des Betrachters ankommen und ein Bild der Szene erzeugen.

Die Modellierung des Beobachters als einzelner Augpunkt im Raum basiert dabei auf dem Lochkameramodell, mit dem Unterschied, dass Fokus (also der Augpunkt) und Bildebene vertauscht sind. Abbildung 6 demonstriert dies.

Für jedes Pixel der Bildebene wird nun ein Lichtstrahl berechnet, der vom Augpunkt ausgehend in die 3D-Szene ausgeht. Dieser Lichtstrahl entspricht einer Geraden, für die nun Schnittpunkte mit Objekten der 3D-Szene ermittelt werden. Die jeweils zum Augpunkt nächstliegenden Schnittpunkte der Geraden bestimmen die sichtbaren Objekte der Szene. An den Schnittpunkten wird nun ein lokales Beleuchtungsmodell für die entsprechenden Objekte berechnet. Dies geschieht zum Beispiel dadurch, dass weitere Strahlen, die von der am Schnittpunkt anliegenden Oberfläche reflektiert werden, rekursiv rückverfolgt werden. Aber auch Eigenschaften der Oberfläche (Farbe, Textur, Transparenzeigenschaften, etc.) bestimmen die Beleuchtungsintensität. Letzendlich wird versucht, die Strahlen bis zu den aussendenden Lichtquellen zurückzuverfolgen. Aus den berechneten Daten kann nun das Aussehen der sichtbaren Oberflächen, und somit das Bild ermittelt werden.

2.4.3 Verteiltes Rendern

Da beim Ray tracing die Berechnungsdauer sehr stark mit der Komplexität der 3D-Szene (Anzahl und Komplexität der Objekte) steigt, kann es sehr lange dauern, fotorealistische Abbildungen einer Szene zu berechnen. Bei sehr komplexen Szenen kann die Dauer des Renderns sogar mehrere Tage betragen (siehe auch Abbildung 7)

⁸Der Begriff 'backward ray tracing' kann zu Verwirrungen führen, da häufig davon gesprochen wird, dass man Strahlen vom Augpunkt aus aussendet. Lichtstrahlen werden jedoch zunächst nur von Lichtquellen ausgesendet. Für die Bildsynthese sind jedoch nur die Menge von Strahlen relevant, die den Augpunkt treffen. Daher betrachtet der Algorithmus nur diese Strahlen.



Abbildung 7: Dieses Bild wurde mit POV-ray in 4.5 Tagen, auf einem Athlon 5600+ gerendert. [18]

Ein Vorteil des Ray Tracing liegt jedoch darin, dass die Berechnung für einen Strahl (bzw. ein Pixel) unabhängig von der Berechnung der übrigen Strahlen geschehen kann. Somit ist Ray Tracing ein parallelisierbares Verfahren. In dieser Projektarbeit soll diese Eigenschaft genutzt werden, um mit Hilfe von Grid Computing ein paralleles Rendern eines Bildes zu ermöglichen und dadurch zu beschleunigen. Dazu wird das Bild, also die Menge der zu berechnenden Pixel, in eine Menge von Teilbildern zerlegt, und das Rendern dieser Teilbilder parallel ausgeführt, unter Zuhilfenahme des Grid. So ist es durchaus möglich, abhängig von der Anzahl und Rechenleistung der verwendeten Grid-Ressourcen, die Dauer des Renderns erheblich zu beschleunigen.

Das Konzept des parallelen Renderns ist vor allem bei Animationen sinnvoll. Eine Animation besteht in der Regel aus einer Sequenz vieler Einzelbilder, die alle gerendert werden müssen. Die Parallelität ist hier schon dadurch gegeben, dass man die Einzelbilder unabhängig voneinander rendern kann. Eine weitere Aufteilung der Einzelbilder selbst ist hier nicht unbedingt nötig. Paralleles Rendern mit Hilfe von Grid-Ressourcen oder ähnlichem ist hier teilweise sogar unabdingbar. So kann es je nach Länge der Animation, Anzahl der Frames pro Sekunde, Auflösung und Komplexität der Szene einen erheblichen Rechenaufwand bedeuten, die gesamte Animation zu rendern, sodass ein einzelner Rechner dies unter Umständen nicht bewältigen kann.

3 Entwicklung des Raytracing-Dienstes

In diesem Kapitel wird die Entwicklung des Raytracing-Dienstes beschrieben. Zunächst werden die verwendeten Werkzeuge und Software genannt. Es wird dann eine Anforderungsanalyse geben. In den folgenden Unterkapiteln wird auf Einzelheiten und Entscheidungen des Entwurfs eingegangen. Unter anderem werden bestimmte Themen wie Integration des Globus Toolkit 4 sowie Möglichkeiten der Authentifizierung an Grid-Ressourcen näher diskutiert. Im Zuge der Entwicklung des Raytracing-Dienstes wurde auch ein weiterer Dienst für das Globus Toolkit 4 entworfen. Dieser wird ebenfalls vorgestellt. Zuletzt wird noch beispielhaft eine Benutzeroberfläche demonstriert.

3.1 Verwendete Werkzeuge und Software

3.1.1 Venice

Der hier entwickelte Dienst läuft in Venice innerhalb eines Apache *Tomcat*-Servers und wird von diesem ausführbar gemacht. Der Dienst an sich ist in Java (Version 1.6.0) implementiert. Die entsprechenden Dateien und Klassen für Server und Client wurden mit Hilfe des Venice Service Compiler (siehe Kapitel 2.2.3) generiert.

Die graphischen Komponenten der Benutzeroberfläche wurden mit *Java Swing* und *Java GUI Programming Extensions*⁹ (JavaGPE) realisiert.

3.1.2 Globus Toolkit

Zur Anbindung des hier entwickelten Dienstes an das Globus Toolkit werden die in Globus Toolkit 4 (Version 4.0.8) vorhandenen Java-Bibliotheken, sowie das Modul *jglobus* (Version 1.7.0) aus dem *Commodity Grid Kit*¹⁰ (CoG-Kit) genutzt. Die Anbindung wird dabei als separate Applikation realisiert und durch *Java Remote Method Invocation* (RMI) verfügbar gemacht. Zur Generierung der Proxy-Zertifikate wird das GT4-Kommandozeilenwerkzeug *grid-proxy-init* verwendet.

3.1.3 POV-Ray

Der verwendete Raytracer ist der *Persistence of Vision Raytracer* (kurz: POV-Ray)¹¹. Das Projekt POV-Ray begann im Jahre 1991, aufbauend auf einem Raytracing Paket namens DKBTrace 2.12 von David Buck und Aaron Collins. Die Software ist Freeware und auf allen gängigen Plattformen verfügbar. Durch POV-Ray zu rendernde Szenen werden durch eine eigene Szenen-Beschreibungssprache beschrieben, deren Syntax den Programmiersprachen C und C++ ähnelt [19]. Die Beschreibung erfolgt dabei durch Definition verschiedener Objekte der Szene. Ausgehend von primitiven Objekten wie beispielsweise Kugeln, Flächen, Kuben oder Zylindern können komplexe Objekte mit vielfältigen Oberflächenstrukturen, Farben und Texturen beschrieben werden. Ausserdem wird die Position der Kamera festgelegt, die letztendlich den Blickwinkel auf die Szene bestimmt. Aus einer solchen Beschreibung heraus kann POV-Ray

⁹<http://www.markus-hillenbrand.de/javagpe>

¹⁰www.cogkit.org

¹¹<http://povray.org>

dann ein zweidimensionales Bild der Szene rendern. Verwendet wird hier die offizielle Version 3.6.1 für Linux. Da diese noch nicht multiprozessorfähig ist, werden die Vorteile parallelen Renderns selbst auf Rechnern mit mehreren CPUs nicht genutzt. Auch daher ist eine Aufteilung des Renderns mithilfe von Grid Computing sinnvoll.

3.1.4 Sonstige

Um die Teilbilder nach verteiltem Rendern wieder zusammzusetzen, wurde das Tool *composite* aus der *Image Magick*¹²-Bibliothek verwendet.

3.2 Anforderungen an den Dienst

Zunächst einmal stellt sich die Frage, welche Anforderungen an einen Ray Tracing Dienst gestellt werden. Der grundsätzliche Anwendungsfall, anhand dessen diese Anforderungen hergeleitet werden und der Dienst entwickelt wird, sieht wie folgt aus: Der Benutzer sollte eine Beschreibung seiner zu rendernden 3D-Szene in Form einer POV-Ray Beschreibungsdatei an den Dienst übergeben können. Nach einiger Zeit sollte er ein fertiges Ergebnis in Form eines gerenderten Bildes erhalten. Anhand dieses grundsätzlichen Szenarios lassen sich weitere Anforderungen identifizieren. Im Folgenden werden die identifizierten Anforderungen tabellarisch aufgelistet, mit einer Unterscheidung in funktionale (F) und nicht-funktionale (NF) Anforderungen. Bei der Entwicklung des Dienstes werden diese Anforderungen dann durch ihre Nummerierung referenziert:

Nr.	Anforderung
F0	Der Benutzer soll eine POV-Ray-Beschreibungsdatei an den Dienst übergeben können und nach einiger Zeit ein Ergebnis in Form eines gerenderten Bildes im PNG-Format erhalten.
F1	Oft besteht die Beschreibung eines Bildes aus mehreren Dateien (includes). Außerdem findet oft noch eine Konfigurationsdatei (.ini) Anwendung. Es soll die Möglichkeit bestehen, diese Dateien mit angeben zu können.
F2	Der Benutzer soll die Möglichkeit erhalten, das Ergebnis an beliebiger Stelle lokal abspeichern zu können.
F3	Ein wichtiger Parameter bei der Verwendung von POV-Ray ist die Auflösung des Bildes. Der Benutzer sollte also auch die Breite und Höhe des zu rendernden Bildes spezifizieren können. Davon wird dann im Wesentlichen auch die Dauer des Renderns abhängen.
F4	Da der Vorgang des Renderns einige Zeit beanspruchen kann, soll der Benutzer informiert werden, wenn sein Bild fertig gerendert wurde, oder wenn die Bearbeitung seines Auftrages aus irgendeinem Grund fehlgeschlagen ist. Als Möglichkeiten der Benachrichtigung werden das Versenden von E-Mail- und SMS-Nachrichten festgelegt. Der Benutzer soll, je nach Wunsch, festlegen können, ob er per E-Mail und/oder SMS benachrichtigt werden möchte, und kann in Zuge dessen auch entsprechende Kontaktdaten mitliefern.

¹²<http://www.imagemagick.org/>

F5	Je nach Größe des zu rendernden Bildes ist es oft sinnvoll, schon vorab einen Eindruck des Bildes zu bekommen. Hierzu soll der Dienst bei größeren Bildern die Möglichkeit bieten, Miniaturbilder (sog. Thumbnails) bereitzustellen.
F6	Der Benutzer soll auch die Möglichkeit haben, den aktuellen Status seiner Aufträge einsehen zu können. Es soll eine Liste der Aufträge mit dazugehörigem Status bereitgestellt werden. Ein solcher Status kann beispielsweise sein: "arbeitend", "beendet" oder "fehlgeschlagen".

Über die grundlegende Funktionalität hinaus soll ein wesentlicher Bestandteil des Dienstes sein, den Vorgang des Renderns mit Hilfe von Grid Computing parallelisieren zu können. Wie bereits in Kapitel 2.4.3 erläutert wurde, lässt sich das Rendern eines Bildes aufteilen. Bezüglich der Bearbeitung des Auftrages mit Hilfe von Grid Computing ergeben sich folgende Anforderungen:

F7	Der Benutzer soll wählen können, ob er das Bild auf konventionelle Art lokal im Venice Service Grid rendern lassen will, oder ob er parallelisiertes Rendern auf entsprechenden Grid-Ressourcen durchführen will.
F8	Wurde parallelisiertes Rendern ausgewählt, so soll er die Anzahl der zu rendernden Teile des Bildes bestimmen können. Dadurch kann er die Anzahl der parallel ablaufenden Rendervorgänge mitbeeinflussen ¹³ .
F9	Ein wichtiger Aspekt bei der Nutzung externer Grid-Ressourcen ist die Authentifizierung und Autorisierung der Nutzer. In der Regel geschieht dies durch Proxy-Zertifikate . Der Benutzer soll also die Möglichkeit haben, sich am Grid authentifizieren zu können. Für diesen Anwendungsfall sollen verschiedene Möglichkeiten erarbeitet werden.
NF0	Eine nichtfunktionale Anforderung, die bei der Entwicklung eines Dienstes stets berücksichtigt werden sollte, ist Wiederverwendung bereits bestehender Funktionalität. Ein wichtiges Ziel bei Service-orientierten Architekturen im Allgemeinen ist nämlich die Wiederverwendbarkeit vorhandener Dienste. Dies ist nur sinnvoll, wenn die entsprechenden Dienste dann auch wirklich wiederverwendet werden. Es soll also geprüft werden, inwiefern Funktionalität des Dienstes an andere Dienste delegiert werden kann.

¹³Wie sich später herausstellen wird, findet die Bearbeitung der Rendervorgänge nicht notwendigerweise parallel statt. Bei der Auftragsvergabe an das Grid wird davon abstrahiert, wann und wo die einzelnen Aufträge durchgeführt werden. Davon bekommt der Auftragsgeber nichts mit.

3.3 Entwurf des Dienstes

Im Folgenden soll der Entwurf des Dienstes vorgestellt werden. Im Sinne eines “Contract-first”-Ansatzes (siehe auch 2.3.5) werden zunächst die Operationen und Datentypen des Dienstes erläutert. Darauf aufbauend werden dann die einzelnen Entwurfsentscheidungen bezüglich des Dienstes detaillierter ausgeführt, stets mit Bezug auf die im vorigen Kapitel definierten Anforderungen. Auch wird die Benutzerschnittstelle des Dienstes kurz präsentiert.

3.3.1 Entwurf der Operationen

- **render** Die Operation `render` stellt die grundlegende Funktionalität des Dienstes dar. Der Benutzer kann die für das Rendern benötigten Daten übergeben. Das Bild wird dann lokal im Venice Service Grid auf einem Rechner gerendert.
- **renderInGrid** Diese Operation erweitert die Funktionalität der ersten Operation, indem sie den Vorgang des Renderns auf zusätzliche Grid-Ressourcen verteilt. Diese Grid-Ressourcen werden über die Globus Toolkit Middleware angesprochen. Bei großen Bildern kann der Rendervorgang durch die Verteilung beschleunigt werden. Zum Zugriff auf das Grid werden zusätzliche Informationen benötigt. Dies wird im folgenden Kapitel erläutert.
- **getStatus** Diese Operation dient der aktuellen Statusabfrage. Man erhält Informationen darüber, welche Bilder sich zurzeit in Bearbeitung befinden, und welche bereits fertig gerendert worden sind.
- **getResult** Ist ein Bild fertig gerendert, so kann es über diese Operation abgerufen werden und anschließend gespeichert werden. Nach Abruf des Bildes werden die Informationen über den Rendrauftrag gelöscht.
- **deleteJob** Ein Auftrag kann vor oder nach Beendigung gelöscht werden. Es werden dann alle Informationen über den Auftrag aus dem Dienst entfernt.

3.3.2 Datentypen

Die Übertragung von Daten zum und vom Dienst geschieht über XML-basierte Nachrichten. Die Datentypen, die dabei übertragen werden können, werden durch XML-Schemata spezifiziert. Die vom Venice Service Grid unterstützten Datentypen beinhalten dabei sowohl öffentlich zugängliche Datentypen wie `String` und `Integer`, als auch spezifische Datentypen, die für die entsprechenden Dienste innerhalb von Venice nötig sind.

Die Schnittstelle des POVRay-Dienstes muss so spezifiziert werden, dass sie den Anforderungen an die zu übertragene Information Rechnung trägt. Das heißt, es sollen möglichst alle notwendigen Informationen übertragen werden können, die Schnittstellenbreite jedoch soll möglichst gering bleiben. Die Informationen sollen also möglichst sinnvoll in Form neuer Datentypen zusammengefasst werden.

Die für den POV-Ray-Dienst neu eingeführten Datentypen beruhen auf der logischen Zusammenfassung von vier wesentlichen Informationskomponenten: Das sind (a) die Informationen, die zum Rendern des Bildes benötigt werden, (b) die Informationen, die zum Zugriff auf die externen Grid-Ressourcen benötigt werden, dann (c) die Statusinformationen, und zuletzt

noch (d) die Kontaktinformationen des Benutzers. Im Folgenden werden diese Informationstypen genauer ausgeführt.

(a) RenderingInformation. Die Beschreibung der 3D-Szene für das Rendern eines Bildes kann auch externe Abhängigkeiten haben (z.B. includes). Das heißt, es kann vorkommen, dass der Benutzer mehrere Dateien an den POV-Ray-Dienst übergeben muss, damit das Bild vollständig gerendert werden kann. Unter diesen Dateien ist stets die Hauptdatei (mit der Endung .pov). Oft ist auch eine Konfigurationsdatei (mit der Endung .ini) notwendig, mit der der Benutzer verschiedene Einstellungen hinsichtlich der Qualität und Art der Ausführung für das Rendern vornehmen kann. Zusätzlich kann die Hauptbeschreibung auch noch von verschiedenen anderen zusätzlichen Dateien abhängig sein, die dann entsprechend mitgeliefert werden müssen.

Die in Venice vorhandenen Datentypen wie zum Beispiel "File" eignen sich nicht dafür. Man kann zwar mit Hilfe dieses Datentyps auch mehrere Dateien als Array versenden, aber es ist nicht ohne Weiteres feststellbar, welches die Hauptdatei für den Rendervorgang ist. Die Einführung eines neuen Datentyps zur Spezifizierung des Renderns ist auch dadurch gerechtfertigt, dass man die Angabe von Höhe und Breite des Bildes in diesen Datentyp integrieren kann, und so die Schnittstellenbreite insgesamt verringert.

Der neue Datentyp muss also die Angabe einer Hauptdatei ermöglichen, sowie die optionale Angabe einer Konfigurationsdatei und/oder zusätzlich benötigter Dateien. Somit kann der POV-Ray-Dienst klar trennen, was die Haupteingabedatei ist, und welches die Konfigurationsdatei ist. Zusätzlich soll noch die optionale Angabe von Höhe und Breite (F3) möglich sein. In Listing 1 sieht man die Typ-Definition für die Informationen, die zum Rendern benötigt werden.

Listing 1: Auszug aus der XML-Schema Datei POV-Ray.xsd, Datentyp zur Übertragung von Render-Informationen

```
<complexType name="RenderingInformation">
  <sequence>
    <element name="mainFile" type="basic:File"/>
    <element name="iniFile"
      type="basic:File"
      minOccurs="0"/>
    <element name="additionalFiles"
      type="basic:File"
      minOccurs="0" maxOccurs="100"/>
    <element name="width"
      type="xsd:Integer"
      minOccurs="0"/>
    <element name="height"
      type="xsd:Integer"
      minOccurs="0"/>
  </sequence>
</complexType>
```

Man erkennt, dass sich der neue Datentyp aus Basisdatentypen zusammensetzt. Eine Nachricht, die sich an diesem Datentyp orientiert, enthält nun genau eine Hauptdatei, eine optionale Konfigurationsdatei, optional viele¹⁴ zusätzliche Dateien, sowie optionale Angabe von Höhe

¹⁴Die Anzahl wurde hier beschränkt auf maximal 100 Dateien. Dies ist nötig, da die Dienst-Schnittstelle sonst anfällig für eine *Denial of Service* Attacke wäre. Ein Benutzer könnte dann das System überlasten, indem er sehr viele Dateien überträgt.

und Breite. Werden Höhe oder Breite nicht festgelegt, werden Standardwerte genommen.

(b) GridRenderingInformation. Um die Verarbeitung durch Grid-Ressourcen zu ermöglichen, werden einige Informationen benötigt. Zum einen muss spezifiziert werden, in wie viele Teile das Bild zerlegt werden soll, die dann einzeln gerendert werden. Diese Anzahl ist gleichzusetzen mit der Anzahl von parallel abgesendeten Render-Aufträgen an das Grid. Damit der Nutzer sich am Grid authentifizieren kann, muss ein entsprechendes Proxy-Zertifikat übertragen werden können. Über die Möglichkeiten der Authentifizierung wird später noch diskutiert. Die Angabe eines Proxy-Zertifikats ist hier zunächst optional. Außerdem sollte der Benutzer auch die GRAM-Knoten spezifizieren können, zu denen die Aufträge gesendet werden sollen. Dies geschieht durch die Angabe von entsprechenden Informationen zur Verwendung der Grid-Ressourcen mittels GRAM und GridFTP. In Listing 2 sieht man den Datentyp für die Grid-Informationen. Die Informationen zur Spezifikation eines Grid-Knoten werden hier durch *GT4Contact* definiert. Die hier verwendeten Datentypen *gt4:JobFactoryInformation* und *gt4:GridFTPHost* wurden für den GT4-Dienst entworfen, der zusätzlich zum POV-Ray-Dienst entwickelt wurde und in Kapitel 3.3.8 behandelt wird.

Listing 2: Datentyp zur Übertragung von Informationen zur Nutzung von Grid-Ressourcen

```
<complexType name="GT4Contact">
  <sequence>
    <element name="factoryInformation"
      type="gt4:JobFactoryInformation"/>
    <element name="gridFTPInformation"
      type="gt4:GridFTPHost">
  </sequence>
</complexType>

<complexType name="GridRenderingInformation">
  <sequence>
    <element name="nrOfTiles" type="int"/>
    <element name="proxyFile"
      type="basic:File"
      minOccurs="0"/>
    <element name="gt4Contacts"
      type="tns:GT4Contact"
      minOccurs="0" maxOccurs="100"/>
  </sequence>
</complexType>
```

(c) POVRayJobInfo. Bei der Darstellung des Status eines Rendering-Vorgangs stellt sich ein ähnliches Problem wie bei der Übertragung der Render-Informationen. Es soll eine Liste von Aufträgen angezeigt werden, die jeweils einen anderen Status haben können. Um dies zu realisieren, reichen auch hier die in Venice vorhandenen Datentypen nicht aus. Um diese Information sinnvoll übertragen zu können, ist es auch hier notwendig, neue Datentypen zu definieren. Wie auch in den vorherigen Beispielen ist es möglich, den Datentyp aus Basisdatentypen zusammenzusetzen. So wurde ein neuer Datentyp "POVRayJobInfo" eingeführt, der sowohl die Referenz auf den Auftrag enthält (Job-ID), als auch den derzeitigen Status, sowie Startzeitpunkt und evtl. den Beendigungszeitpunkt. Außerdem kann das Thumbnail mitgeliefert werden, sofern dies schon erstellt worden ist. Listing 3 zeigt diesen Datentyp.

Listing 3: Datentyp zur Anzeige des Auftragsstatus

```

<simpleType name="POVRayJobState">
  <restriction base="string">
    <enumeration value="working"/>
    <enumeration value="finished"/>
  </restriction>
</simpleType>

<complexType name="POVRayJobInfo">
  <sequence>
    <element name="jobID" type="basic:UUID"/>
    <element name="state" type="tns:POVRayJobState"/>
    <element name="startTime" type="dateTime"/>
    <element name="endTime" type="dateTime" minOccurs="0"/>
    <element name="thumbnail" type="basic:File" minOccurs="0"/>
  </sequence>
</complexType>

```

Bei einfachen Datentypen (XML: SimpleType) kann man mit *restriction* die Menge der zulässigen Werte für ein Element beschränken, im vorliegenden Fall auf "working" und "finished" für das Element "state".

(d) ContactInformation. Zuletzt werden noch die Kontaktdaten für E-Mail und SMS zu einer logischen Einheit gebündelt. Womit der Benutzer letztendlich benachrichtigt wird, kann optional bestimmt werden. Auch begünstigt diese Bündelung die einfache Erweiterung um zusätzliche Kommunikationsmedien. Als Datentyp sieht dies wie folgt aus:

Listing 4: Datentyp zur Anzeige des Auftragsstatus

```

<complexType name="ContactInformation">
  <sequence>
    <element name="email"
      type="email:EmailAddress"
      minOccurs="0"/>
    <element name="sms"
      type="string"
      minOccurs="0"/>
  </sequence>
</complexType>

```

Hier wird auf einen Datentyp zurückgegriffen, der bereits in Venice vorhanden ist. Der Typ "EmailAddress" wird im Rahmen des E-Mail-Dienstes benutzt.

3.3.3 Asynchrone Auftragsbearbeitung

Da das Rendern des Bildes eine geraume Zeit in Anspruch nehmen kann, ist es für einen Dienstaufreifer wenig sinnvoll, zu warten bis der Vorgang abgeschlossen ist, um dann das Bild zu erhalten. Anders ausgedrückt sollte der Aufruf des Dienstes mit einer schnellen Antwort einhergehen. Dies, und die Tatsache, dass die Dienste im Venice Service Grid eine maximale Antwortzeit von drei Minuten haben (ansonsten wird der Aufruf abgebrochen), führt dazu, dass die Ausführung des Auftrags asynchron zum Aufruf des Dienstes verlaufen muss. Daher wurde der Dienst so konzipiert, dass der Benutzer den Auftrag erteilen kann, ein Bild rendern

zu lassen, und eine direkte Antwort in Form eines Universal Unique Identifier (UUID¹⁵) bekommt. Diese UUID kann später als eindeutige Referenz auf den Auftrag benutzt werden, um das Ergebnis des Renderns 'abzuholen'.

Um die Asynchronität zu ermöglichen, wird die eigentliche Funktionalität des Dienstes in Threads ausgelagert. Der Vorteil ist, dass der Dienst eine schnelle Antwort geben kann, und trotzdem parallel an seinem Auftrag weiterarbeiten kann. Der Nachteil ist, dass im Falle eines Fehlers bei der Ausführung keine direkte Fehlermeldung an den Dienstaufrufer zurückgegeben werden kann. Dies kann nur indirekt geschehen, indem man den Benutzer mittels alternativer Benachrichtigungsmethoden informiert. Letzendlich lässt sich dieses Problem durch das Versenden von E-Mail und SMS lösen. Abbildung 8 demonstriert den geschilderten Ablauf im Falle der Operation `renderInGrid` anhand eines Flussdiagramms.

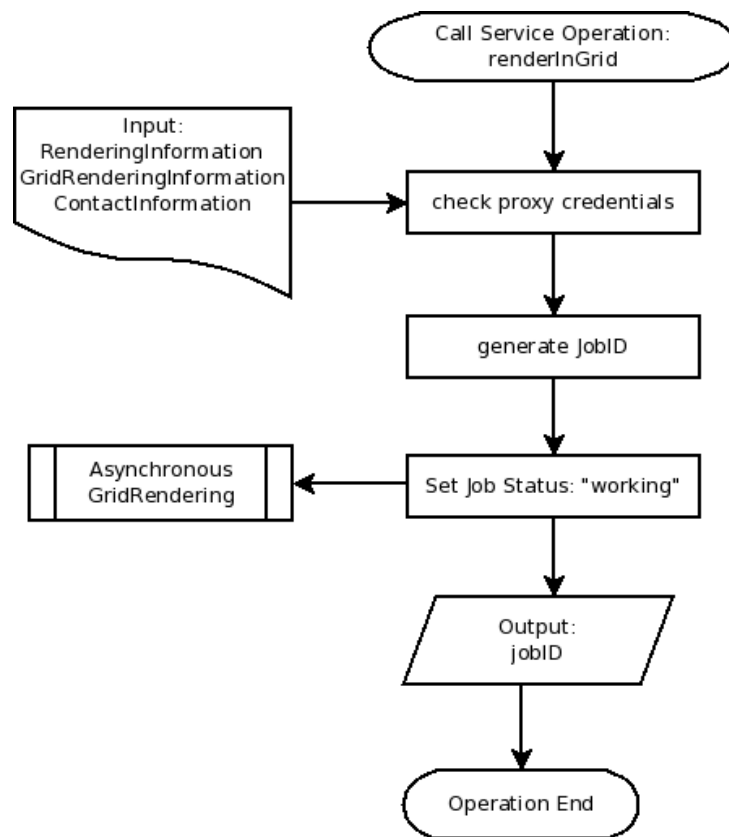


Abbildung 8: Flussdiagramm der Operation `renderInGrid`

3.3.4 Benachrichtigung des Benutzers

Nach Bearbeitung des Auftrags soll der Benutzer, der im Besitz der UUID ist, die den Auftrag identifiziert, benachrichtigt werden. Naheliegende Kommunikationsmethoden sind E-Mail und

¹⁵UUID ist ein Standard für Identifikatoren, spezifiziert durch die Open Software Foundation. Da die Wahrscheinlichkeit sehr gering ist, zwei gleiche UUIDs zu generieren, eignen sie sich zum Beispiel zur (quasi) eindeutigen Identifizierung von Ressourcen in verteilten Systemen und zur Sicherstellung von Threadsicherheit. Siehe auch <http://tools.ietf.org/html/rfc4122>.

SMS (F4). Die Schnittstelle des Dienstes wurde so entworfen, dass sie Möglichkeiten bereitstellt, entstprechende Kontaktdaten für E-Mail oder SMS bereitzustellen. Das Versenden von E-Mails und SMS geschieht dabei durch Nutzung von in Venice bereits vorhandenen Diensten. Dadurch wird die Wiederverwendung innerhalb Venice erhöht (NF0). Der Dienst wird dadurch robuster und es fällt keine zusätzliche Entwicklungsarbeit für zusätzliche E-Mail- und SMS-Funktionalität an.

3.3.5 Status und Vorschau

Eine zusätzliche Operation ermöglicht den Abruf des Status (F6) von Aufträgen. Der Benutzer erhält eine Liste aller seiner Aufträge mit den dazugehörigen Statusmeldungen. Entweder ist das "arbeitend" oder "beendet". Auf eine Statusmeldung "fehlgeschlagen" wurde bewusst verzichtet (Siehe dazu 3.3.10). Auch der Startzeitpunkt des Auftrags, sowie der Zeitpunkt der Beendigung wurde zum Status hinzugefügt.

Zusammen mit dem Status wird außerdem die Vorschau des Bildes mitgeliefert. Die Generierung eines Thumbnails (F5) für das zu rendernde Bild wird wie der eigentliche Rendervorgang asynchron ausgeführt. Da das Thumbnail in einer sehr geringen Auflösung (100x100) gerendert wird, ist es relativ schnell fertig und steht dem Benutzer als Vorschau auf das eigentliche Bild schnell zur Verfügung.

3.3.6 Grundsätzlicher Ablauf des Renderns

Abhängig davon, ob der Benutzer sein Bild lokal oder im externen Grid rendern lassen will, ändert sich der Ablauf entsprechend. Wählt der Benutzer lokales Rendern, so ist der Ablauf weniger komplex. Der Dienst nutzt die lokale POV-Ray-Installation und führt entsprechend der mitgelieferten Render-Informationen (d.h. Dateien, Breite und Höhe) das Rendern als lokalen Prozess durch. Nach Abschluss oder Fehlschlag des Rendern wird der Benutzer über die entsprechenden Kommunikationsmittel informiert.

Wählt der Benutzer jedoch das Rendern auf externen Grid-Ressourcen, so wird der Ablauf komplexer. Zunächst muss sich der Dienst mit entsprechendem Proxy-Zertifikat am Grid authentifizieren (mehr dazu in Kapitel 3.3.9). Anschließend muss POV-Ray auf die Grid-Knoten kopiert und dort kompiliert werden. Dies geschieht mittels GridFTP (2.3.4) Ist POV-Ray bereits von einem vorigen Durchgang kompiliert, kann dieser Schritt weggelassen werden. Sodann werden die benötigten Beschreibungs- und Konfigurationsdateien ins Grid gesendet.

Nun erfolgt die Aufteilung des Bildes und die Entsendung entsprechender Render-Aufträge an die Grid-Ressourcen. POV-Ray unterstützt die Angabe von Start- und Endspalte¹⁶ eines zu rendernden Bildes. So kann man pro POV-Ray-Aufruf nur einen Teil des Bildes von einer bestimmten Spalte beginnend bis zu einer Endspalte rendern lassen. Dieses Feature von POV-Ray wird nun genutzt, um eine Anzahl N verschiedener Aufträge zu erzeugen, die jeweils einen anderen Teil des Bildes berechnen.

Die Beschreibungen der Aufträge, die nun an das GRAM (2.3.3) gesendet werden, können durch die *Resource Description Language* (RSL) formuliert werden. Diese bietet die Möglichkeit, einige grundlegende Parameter bei der Beschreibung eines Auftrags auszudrücken¹⁷. So

¹⁶POV-Ray unterstützt auch die Angabe von Start- und Endzeile eines Bildes, aber für den Zweck des Teilens des Bildes in gleich große Stücke reicht die Angabe von Spalten.

¹⁷Über die hier beschriebenen Parameter hinaus können auch noch weitere Einstellungen vorgenommen werden, die hier aber nicht benötigt werden. Beispielsweise kann man benötigte Daten vorab innerhalb des Grid an

kann man die ausführbare Datei (executable) festlegen, den Ort der Ausführung bestimmen (directory), sowie zusätzliche Parameter der ausführbaren Datei benennen (arguments). Im XML-Format sieht dann eine RSL-Beschreibung eines POV-Ray-Auftrages zum Beispiel wie in Listing 5 aus.

Listing 5: RSL-Beschreibung eines POV-Ray-Auftrages

```
<job>
  <executable>/bin/bash</executable>
  <directory>${GLOBUS_USER_HOME}/pov</directory>
  <argument>povray.sh</argument>
  <argument>render</argument>
  <argument>+Idemo.pov</argument>
  <argument>-w400</argument>
  <argument>-h200</argument>
  <argument>+SC0</argument>
  <argument>+EC39</argument>
  <argument>+Odemo0000.png</argument>
</job>
```

Man erkennt, dass hier der erste Teil eines Bildes erstellt wird, da nur die Spalten 0 bis 39 gerendert werden (gekennzeichnet durch die Argumente +SC0 und +EC39). Die Breite des Gesamtbildes ist 400 Pixel, die Höhe 200 Pixel.

Der Status eines jeden Auftrags wird mittels einer sogenannten *Endpoint Reference* (siehe 3.3.8) überwacht. Es wird nun gewartet bis jeder Auftrag fertig ist. Die gerenderten Teilbilder werden dann aus dem externen Grid via GridFTP kopiert und lokal zusammengesetzt. Dies wird mit dem Image Magick Werkzeug *composite* bewerkstelligt. Nach der Zusammenfügung des Bildes wird der interne Zustand des Render-Auftrags als “beendet” gesetzt. Das Resultat steht nun zum Download über die Operation *getResult* zur Verfügung. Der gesamte Ablauf des Renderns ist zusammengefasst als Flussdiagramm in Abbildung 9 zu sehen.

3.3.7 Anbindung an Globus Toolkit 4

Bei der Anbindung des Dienstes an das Globus Toolkit und dessen Komponenten GRAM und GridFTP wurden entsprechende Java-Bibliotheken aus der Globus Toolkit-Software und die *jpglobus*-Bibliothek aus dem CoG-Kit verwendet. Zunächst wurde eine Nutzung der in der Globus-Software enthaltenen Kommandozeilen-Werkzeuge in Betracht gezogen. Zum einen bieten Kommandozeilen-Werkzeuge aber keine saubere Schnittstelle zur Funktionalität und sind damit fehleranfälliger, zum anderen bieten die vorhandenen Globus Tools wie beispielsweise *globusrun-ws*¹⁸ und *globus-url-copy*¹⁹ keine direkte Möglichkeit zur Angabe eines eigenen Proxy-Zertifikats. Der Dienst soll aber in dieser Hinsicht variabel sein (siehe Kapitel 3.3.9).

Eine direkte Integration der Globus-Bibliotheken mit den in Venice vorhandenen Bibliotheken war nicht möglich. So nutzt das Globus Toolkit, genauso wie Venice, die *Axis*-Bibliothek der Apache Foundation als SOAP-Engine für ihre Webservices, jedoch in einer älteren, inkompatiblen Version²⁰. Ein wesentlicher Unterschied der beiden Versionen liegt in der Art der Er-

entsprechende Stellen kopieren und auch wieder entfernen (*fileStageIn*, *fileStageOut*). Auch kann man Dateien festlegen, in welche die Ausgabe der Ausführung geschrieben wird (*stdout*, *stderr*).

¹⁸Ermöglicht das Versenden von Aufträgen an das GRAM

¹⁹Ermöglicht das Versenden von Dateien via GridFTP

²⁰Version 1.2RC2 in GT4 und Version 1.4 in Venice

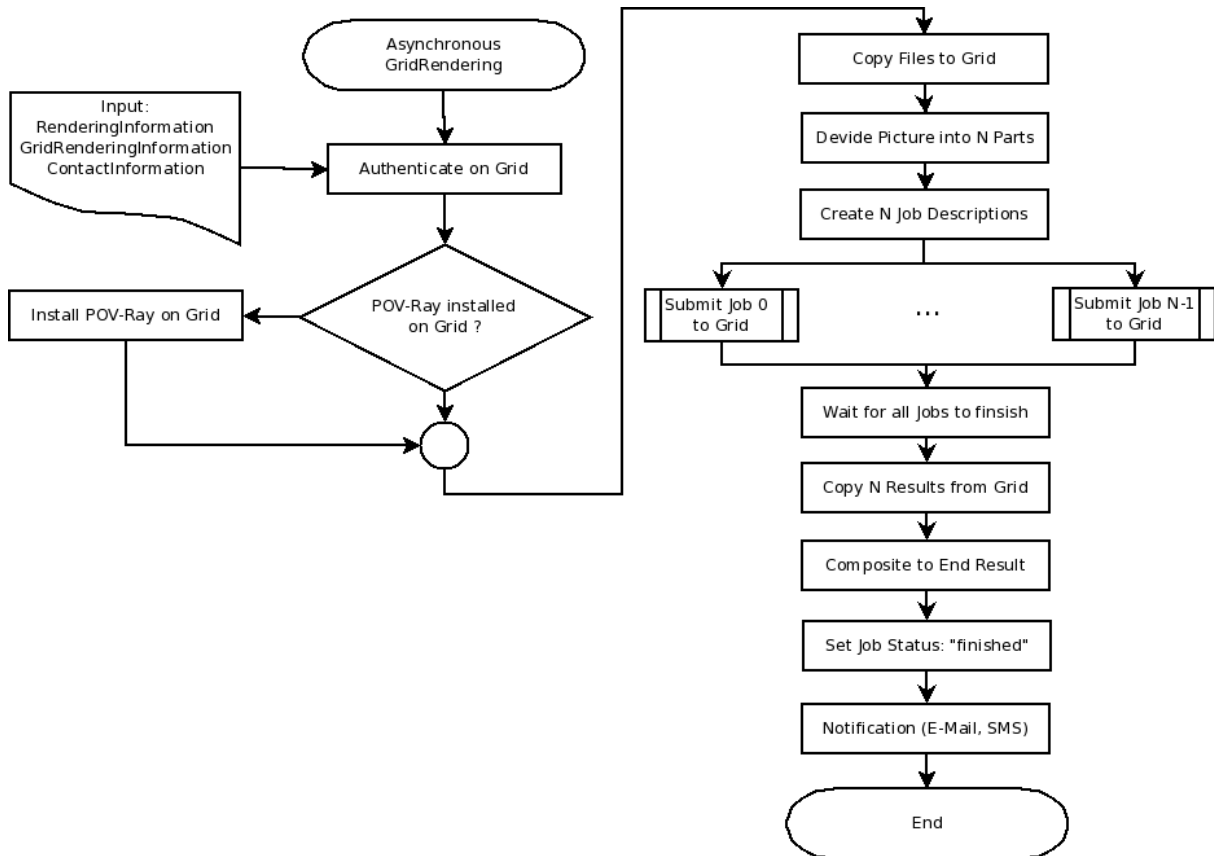


Abbildung 9: Flussdiagramm für das Rendern im Grid

stellung von Client- und Server-Stubs, die zur Kommunikation via SOAP benötigt werden. Aufgrund dieses Unterschieds erhält man eine Fehlermeldung beim Versuch, Globus-Bibliotheken zusammen mit der aktuellen, in Venice verwendeten Axis-Version zu benutzen²¹. Es ist nicht ohne weiteres möglich, beide Bibliotheken parallel innerhalb der gleichen Java Virtual Machine (JVM) zu verwenden. Der Classloader der JVM lädt beim Start von Venice zunächst die aktuelle Axis-Version. Sollen nun Globus-Bibliotheken genutzt werden, müsste Axis in der entsprechenden Version neu geladen werden. Leider konnte im Rahmen der zeitlichen Möglichkeiten keine Lösung für dieses Problem gefunden werden. Daher wurde entschieden, die Anbindung an das Globus Toolkit 4 als eine von Venice separate Applikation zu entwickeln, um so den Versionenkonflikt bezüglich Axis zu vermeiden.

Die besagte Applikation ist eine weitere Java-Anwendung, welche die Globus-Bibliotheken konfliktfrei bezüglich der in Venice benutzten Axis-Version verwenden kann, da sie in einer separaten JVM gestartet wird, und daher von vornherein die richtige Axis-Version benutzen kann. Um aus Venice heraus möglichst optimal auf die Applikation zugreifen zu können, wird *Java Remote Method Invocation* (Java RMI)[21] genutzt. Java RMI ermöglicht die Erstellung von verteilten Java-Applikationen, indem es Methoden entfernter Objekte, d.h. aus anderen JVMs, zugänglich macht. Auch wird durch diese Art des Zugriffs eine Fehlerbehandlung erleichtert, wodurch wiederum die Robustheit der Anwendung unterstützt wird.

²¹ siehe Punkt 4.5 in

<http://www.globus.org/toolkit/docs/development/4.1.1/common/javawscore/user/index.html>

Wird nun der POV-Ray-Dienst dazu verwendet, das Bild mit Hilfe externer Grid-Ressourcen zu rendern, so wird die hier beschriebene Applikation gestartet, und die entsprechenden Methoden werden über RMI angeboten und vom POV-Ray-Dienst genutzt.

Da die Funktionalität des Globus Toolkit 4, die der POV-Ray-Dienst verwendet, auch für andere Dienste von Nutzen sein kann, ist es sinnvoll, diese Funktionalität als eigenen Dienst bereitzustellen. Dieser GT4-Dienst wird im folgenden Kapitel beschrieben. Den Zusammenhang zwischen dem POV-Ray-Dienst, dem GT4-Dienst, der RMI-Applikation und den externen Grid-Ressourcen wird in Abbildung 10 skizziert.

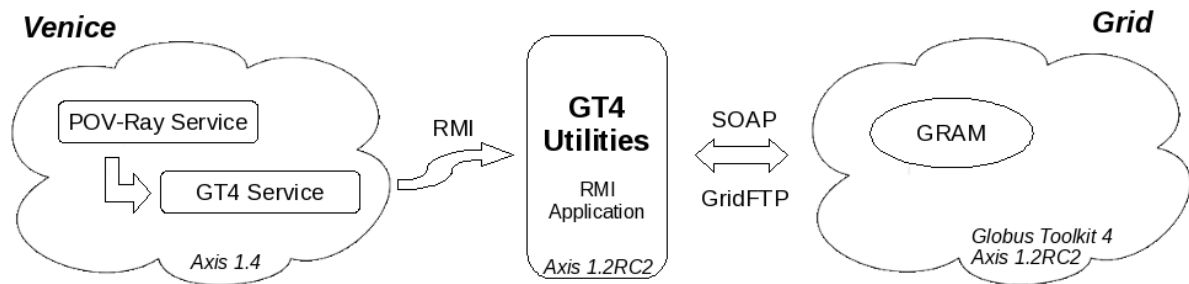


Abbildung 10: Anbindung von Venice an das Globus Toolkit 4

3.3.8 Dienst für Globus Toolkit 4

In Kapitel 2.2.2 wurde zwischen vertikalen und horizontalen Diensten unterschieden. Der POV-Ray-Dienst ist ein vertikaler Dienst, da er zunächst nur in der Domäne des Renderns von Nutzen ist. Andere Applikationen werden den Dienst nur verwenden, wenn sie ein Bild rendern möchten. Daher wird die Menge an aufrufenden Diensten eher gering bleiben, sodass nur wenig Wiederverwendung stattfinden wird. Das Zugreifen auf externe Grid-Ressourcen und das Versenden von Aufträgen an ein Grid jedoch ist Funktionalität, die domänenunabhängig von Nutzen sein kann. Verschiedenste Applikationen können diese Funktionalität gebrauchen, um rechenintensive Aufgaben an entsprechende Grid-Ressourcen auszulagern. Der Wiederverwendungsgrad eines Dienstes, der eine solche Funktionalität bereitstellt, ist daher höher, sodass man ihn in diesem Sinne auch als horizontalen Dienst einstufen könnte.

Aufgrund dieses höheren allgemeinen Nutzens ist es sinnvoll, die im Rahmen des POV-Ray-Dienst entwickelte Funktionalität zum Zugriff auf Globus Toolkit 4 in einen eigenen Dienst auszulagern, um so auch Wiederverwendung zu begünstigen (NF0). Dieser sogenannte GT4-Dienst stellt alle Funktionalitäten bereit, die vom POV-Ray-Dienst benötigt werden. Dazu gehören das Transferieren und Löschen von Dateien in Grid-Ressourcen mittels GridFTP, sowie das Versenden von Aufträgen mittels GRAM.

Zur Nutzung von GridFTP wurden die Operationen **copyFileToGrid**, **copyFileFromGrid** und **deleteFileInGrid** entworfen. Diese setzen jeweils ein gültiges Proxy-Zertifikat und eine Angabe über Host und Port des Servers, über den die GridFTP-Kommunikation stattfinden soll, voraus. Für diese Angabe wurde ein eigener Datentyp entwickelt, der auch in den Datentypen des POV-Ray-Dienstes referenziert wird (siehe Listing 2). Der Datentyp selbst findet sich in Appendix A. Der Transfer von großen Dateien stellt jedoch zurzeit noch ein Problem dar.

Venice selbst unterstützt bislang nur den Transfer kleinerer Dateien²² über die Webservice-Schnittstellen. Daher kann die GridFTP-Funktionalität des GT4-Dienstes nicht im vollen Umfang genutzt werden. Venice wird in dieser Hinsicht aber noch erweitert werden, sodass auch der Transfer größerer Dateien möglich sein wird.

Der Kern des Dienstes ist die Operation **submitJob**. Diese setzt ebenfalls ein gültiges Proxy-Zertifikat voraus. Anders als bei GridFTP wird in GRAM die zugegriffene Grid-Ressource durch einen Webservice repräsentiert. Dieser Webservice wird auch *Managed Job Factory Service* (MJFS) genannt. Im Wesentlichen generiert der MJFS aus einer Auftragsbeschreibung einen entsprechenden Auftrag in Form eines *Managed Job*. Dieser wird dann von einem lokalen Job Scheduler²³ im Grid ausgeführt. Jeder Auftrag bekommt eine eindeutige ID, mit derer der Benutzer Kontrolle über die Ausführung des Auftrages ausüben kann. So kann er den Auftrag zum Beispiel anhalten und neustarten, als auch den aktuellen Status der Abarbeitung abfragen.

Um dem MJFS eine Auftragsbeschreibung zu senden, wird dessen URL²⁴ benötigt. Des weiteren erfolgt die Angabe des zu verwendenden Job Schedulers. Dies ist der sogenannte *Factory Type*. Um diese Informationen zu repräsentieren, wurde für den Dienst der Datentyp *JobFactoryInformation* eingeführt. Dieser befindet sich in Appendix A.

Des weiteren muss dem MJFS die Auftragsbeschreibung selbst gesendet werden. Dies geschieht durch die Beschreibungssprache RSL (siehe auch 3.3.6). Hierzu wurde ein Datentyp eingeführt, der einige der Möglichkeiten von RSL beinhaltet, die im Rahmen des POV-Ray-Dienstes benötigt werden. So kann ein Ausführungsverzeichnis festgelegt werden, sowie die auszuführende Datei und Argumente der Ausführung. Der Datentyp findet sich ebenfalls in Appendix A. Es liegt nahe, hier den in Globus Toolkit definierten RSL-Datentyp *JobDescriptionType* zu verwenden und daraus die entsprechenden für den Dienst notwendigen Java-Klassen zu generieren, oder die für Globus generierten Java-Klassen direkt zu verwenden. Der zeitliche Aufwand für diese Integration ist jedoch sehr hoch. Um den vorläufigen Betrieb des Dienstes in einer initialen Version zu ermöglichen, ist es daher sinnvoll, einen neuen Datentyp einzuführen, der sich auf die für den POV-Ray-Dienst Eigenschaften von RSL beschränkt. Eine Erweiterung des Dienstes sollte also auch auf die Integration der in Globus verwendeten Datentypen mit Venice eingehen.

Dies gilt auch für den aktuellen Status des Auftrags. Hierzu hat Globus einen Datentyp namens *StateEnumeration*. Dieser beinhaltet verschiedene Zustände, die ein Auftrag einnehmen kann, beispielsweise *Unsubmitted*, *Pending*, *Active*, *Finished* oder *Failed*. Der GT4-Dienst bietet über die Operation **getStatus** hier eine Möglichkeit, diesen Status abzufragen. Intern werden dabei unter anderem sogenannte *Endpoint References* (EPR) gespeichert. Diese sind XML-Dateien, die die Wiederherstellung des Kontaktes zu einem Auftrag mit bestimmter ID ermöglichen. Mit Hilfe dieser EPRs kann man dann auch den aktuellen Status des Auftrags erfragen. Dies nutzt der Dienst, um den Status dann an den Dienst-Nutzer weiterzureichen. Außerdem kann die Nutzung des Notification Services in Betracht gezogen werden, um andere Dienste über die Beendigung eines Grid-Auftrages zu informieren.

²²Die Höchstgrenze bei 8 MB festgelegt.

²³Mögliche Job Scheduler sind u.a. *PBS*[23], *Condor*[24] oder *SGE*[25]. Dies ist abhängig von der Globus-Installation

²⁴URL: <http://tools.ietf.org/html/rfc1738>

3.3.9 Authentifizierung am Grid

Ein wichtiger Aspekt bei der Nutzung externer Grid-Ressourcen ist die Authentifizierung des Benutzers. Dies geschieht in Globus Toolkit 4 im Rahmen der GSI in der Regel durch Proxy-Zertifikate (siehe 2.3.5). GSI unterstützt auch die Authentifizierung mittels Benutzername/Passwort anhand der WS-Security Spezifikation [22]. Jedoch bietet GSI dann nur Authentifizierung, aber keine fortgeschrittenen Sicherheitsmerkmale wie beispielsweise Delegation von Aktivitäten[14]. Diese Art der Authentifizierung wird hier daher nicht betrachtet. Eine weitere Möglichkeit ist der Verzicht auf Authentifizierung. Dies wird auch “anonymous transport-level security” genannt. Gemeinhin nimmt man dann an, dass die Authentifizierung nicht auf Transport-Ebene, sondern auf einer anderen Ebene, beispielsweise der Nachrichtenebene, stattfindet. Man kann aber die Authentifizierung auch auf allen Ebenen weglassen.

Für den POV-Ray-Dienst stellt die Bereitstellung von und der Umgang mit Proxy-Zertifikaten einen wichtigen Teil der Funktionalität dar. In diesem Kapitel werden daher mehrere Arten der Bereitstellung von Proxy-Zertifikaten, sowie deren Vor- und Nachteile erläutert (F9). Dies geschieht mit Hinblick auf den POV-Ray-Dienst, ist aber aber auf beliebige Dienste oder Applikationen übertragbar.

Bereitstellung durch Benutzer. Die erste Möglichkeit besteht darin, den Benutzer ein Proxy-Zertifikat erstellen zu lassen und über die Schnittstelle der Dienstoperation dem POV-Ray-Dienst zur Verfügung zu stellen. Dafür ist es notwendig, dass der Benutzer ein entsprechendes Zertifikat besitzt. Normalerweise muss dieses Zertifikat von einer entsprechenden Certificate Authority ausgestellt werden. Auch muss der Zugriff auf die entsprechenden Grid-Ressourcen für dieses Zertifikat erlaubt sein. Das bedeutet, dass der Zertifikatsbesitzer seitens der Globus Toolkit Middleware innerhalb der Konfiguration eingetragen sein muss²⁵. Diese Voraussetzungen bedeuten einen erheblichen Mehraufwand, wenn der Benutzer nicht bereits im Besitz eines entsprechenden Zertifikats ist und damit Zugriff auf bestimmte Grid-Ressourcen hat. Außerdem muss das Zertifikat dem Besitzer lokal vorliegen, damit er auf Grundlage des Zertifikats ein Proxy-Zertifikat erstellen kann. Dies muss aber nicht zwangsläufig der Fall sein, wenn man als Benutzer zum Beispiel gerade nicht an seinem eigenen Rechner ist. Auch muss seitens des Venice Frameworks darauf geachtet werden, dass das Proxy-Zertifikat nicht in fremde Hände gerät, da man sich damit Zugriff auf die entsprechenden Grid-Ressourcen verschaffen kann, wenn auch nur zeitlich begrenzt. Ein Vorteil dieser Variante ist aber, dass der Benutzer, wenn er im Besitz eines Zertifikates ist und entsprechende Zugriffsrechte auf Grid-Ressourcen hat, diese Ressourcen dann auch dem Dienst angeben kann und für die Durchführung des Renderns verwenden kann. So ist seitens des Benutzers eine flexiblere Änderung der benutzten Grid-Ressourcen möglich.

Bereitstellung durch den Dienst. In der zweiten Variante hat der POV-Ray-Dienst selbst Zugriff auf externe Grid-Ressourcen in Form eines Zertifikats. Wie im ersten Fall muss ein Zertifikat für den Dienst beantragt bzw. ausgestellt werden und die entsprechenden Rechte seitens der Grid-Ressourcen konfiguriert werden. Dies geschieht aber nun vorher und vor allem hat der Benutzer diesen Aufwand nicht. Der Benutzer kann dadurch auch als Laie die Grid-Ressourcen über den POV-Ray-Dienst nutzen, ohne Wissen über die Verwendung von Proxy-Zertifikaten und Zertifikaten im Allgemeinen zu besitzen. Jedoch fehlt nun seitens der Grid-Ressourcen die Transparenz über die Nutzzeiten einzelner Benutzer. Es gibt nur den einen Venice-Benutzer,

²⁵Dies geschieht durch sogenannte *gridmap*-Dateien. Diese enthalten Listen, die eine Abbildung von Zertifikaten auf lokale Benutzernamen vornehmen.

der stets Aufträge an das Grid sendet, während es seitens Venice aber verschiedene Benutzer mit verschiedenen Nutzungszeiten sein können. Dies stellt vor allem ein Problem dar, wenn die im Grid genutzte Zeit als Berechnungsgrundlage für Nutzungsgebühren verwendet wird. Ein weiteres Problem stellt die Konfiguration von Venice bezüglich des zu speichernden Zertifikats da. So muss das Zertifikat zusammen mit dem privaten Schlüssel innerhalb Venice sicher abgelegt werden können, damit diese nicht in fremde Hände geraten. Auch muss das Passphrase zur Generierung des Proxy-Zertifikats eingegeben und entsprechend hinterlegt werden. Natürlich würde bei der erstmaligen Einrichtung von Venice diese Konfiguration vorgenommen werden müssen, um die externen Grid-Dienste nutzen zu können. Dabei kann dann eine Menge von Grid-Ressourcen angegeben werden, für die eine Berechtigung besteht. Der Benutzer kann dann jedoch nicht, wie in der ersten Variante, flexibel neue Ressourcen hinzufügen, sondern ist auf die Ressourcen beschränkt, für die Venice berechtigt ist.

Das stärkste Argument aber, das gegen eine Bereitstellung des Zertifikats durch Venice spricht, ist der Verstoß gegen Richtlinien der *EU Grid Policy Management Authority* [26]. Die EUGridPMA ist eine internationale Organisation, die europaweit die Vertrauensstruktur für Authentifizierung in e-Science Grid-Systemen koordiniert. Ihr Ziel dabei ist es, Anforderungen festzulegen und Best Practices vorzuschlagen, um damit eine allgemeine Vertrauensdomäne für die Authentifizierung von Entitäten beim Zugriff auf inter-organisationelle, verteilte Ressourcen zu schaffen. Dabei gibt sie auch Richtlinien aus, die bei der Vergabe von Zertifikaten eingehalten werden sollen. Möchte man also Grid-Ressourcen nutzen, die den Bedingungen der EUGridPMA unterliegen, dann sollte die eigene Zertifikatinfrastruktur ebenfalls diesen Richtlinien folgen. So schreibt die EUGridPMA vor, dass zu jedem Zertifikat genau eine Entität gehört, die durch dieses Zertifikat identifiziert wird, und dass über die gesamte Lebensdauer des Zertifikats keine andere Entität mit diesem Zertifikat in Verbindung gebracht werden darf [27]. Dies beinhaltet auch, dass der private Schlüssel, der zu dem Zertifikat gehört, Anderen nicht offenbart und nicht mit ihnen geteilt werden darf. Eine Bereitstellung des Zertifikats durch Venice jedoch würde diese Richtlinie unterlaufen, da viele verschiedene Nutzer sich auf diese Weise indirekt als die Entität "Venice" ausgeben würden.

Natürlich ist es eine streitbare Angelegenheit, welche Entität letztendlich auf die Grid-Ressourcen zugreift. Ist es der Benutzer des POV-Ray-Dienstes, oder ist es der Dienst selbst. Wenn man zu dem Ergebnis gelangt, dass der Dienst selbst die Entität ist, die auf die Grid-Ressourcen zugreift, so gelangt man zu anderen Hürden. Im Sinne der EUGridPMA gibt es höchstens zwei Möglichkeiten, den Dienst als Entität einzustufen: Als *Host and Service Entity* oder als *Automated Client or Robot Entity*²⁶. Als Host and Service Entity unterläge der POV-Ray-Dienst der Beschränkung, dass er das Zertifikat nur nutzen darf, um sichere Identifikation, und dadurch sichere Kommunikation zu ermöglichen. Er darf das Zertifikat nicht nutzen, um sich selbst als Client oder Dienstanwender für andere Dienste zu authentifizieren [28]. Damit ist eine Definition des POV-Ray-Dienstes als Host and Service Entity nicht möglich, da er das Zertifikat ja zum Zugriff auf die Grid-Ressourcen nutzt. Eine Einstufung des POV-Ray-Dienstes als Automated Client or Robot Entity ist schon aufgrund der Definition schwer möglich. So wird bei solchen Entitäten vorausgesetzt, dass sie sich wiederholende, automatisierte Prozesse durchführen, und dabei ein Zertifikat nutzen, um die notwendigen Rechte für die Durchführung ihrer Aufgaben zu erlangen [29]. Da der POV-Ray-Dienst nur auf Anfrage eines Benutzers auf Grid-Ressourcen zugreift, ist er in diesem Sinne kein automatisierter Client oder Roboter. Auch unterliegen solche Entitäten laut EUGridPMA immer der Verantwortung eines einzelnen menschlichen Indi-

²⁶Der dritte Typ von Entitäten behandelt natürliche Personen.

viduums, in dessen Namen die Entität agiert. Gelänge also eine Einstufung des Dienstes als Automated Client and Roboter Entity, so müsste eine einzelne Person die Verantwortung für die Aktivitäten dieses Dienst übernehmen, auch wenn es faktisch gesehen viele verschiedene Nutzer des Dienstes gibt. Insgesamt steht also eine Bereitstellung des Zertifikats durch den POV-Ray-Dienst im Widerspruch zu den Bedingungen der EUGridPMA.

Bereitstellung durch MyProxy. Eine gängige Methode zur Verwaltung von Proxy-Zertifikaten ist die Verwendung eines MyProxy-Servers²⁷. MyProxy bietet die Möglichkeit, eigene Proxy-Zertifikate online zu speichern und zu verwalten. Dies hat zum Beispiel den Vorteil, dass man das eigene Zertifikat nicht lokal auf jedem ausführenden Rechner halten muss, um ein Proxy-Zertifikat zu generieren, sondern stets ein Proxy-Zertifikat online erfragen kann. Dazu generiert ein Benutzer ein Proxy-Zertifikat mit entsprechender Laufzeit, welches im MyProxy Speicher hinterlegt wird. Mittels Benutzernamen und Passwort kann der Nutzer sich später dann aus dem hinterlegten Proxy-Zertifikat neue Proxy-Zertifikate online generieren lassen.

Hinsichtlich des POV-Ray-Dienstes ist eine Erweiterung um die MyProxy-Funktionalität von Vorteil. Sowohl Dienst- als auch Nutzerseitig kann die Generierung des Proxy-Zertifikats über einen MyProxy-Server geschehen. Dabei bleiben die Vor- und Nachteile der ersten Variante, der Bereitstellung des Proxy-Zertifikats durch den Benutzer, erhalten. Jedoch ergibt sich durch die Ortsunabhängigkeit, die die Online-Generierung des Proxy-Zertifikats mit sich bringt, der Vorteil, dass der Client ebenfalls unabhängig dahingehend ist, ob lokal ein Zertifikat des Benutzers vorliegt. Dadurch ergibt sich auch eine Ortsunabhängigkeit des Clients.

Keine Verwendung von Proxy-Zertifikaten. Es ist möglich, einen Globus Toolkit Container ohne GSI-Unterstützung zu starten, d.h. gänzlich ohne Sicherheitsunterstützung. Dadurch wird der Zugriff auf die entsprechenden Grid-Ressourcen auch ohne Authentifizierung, insbesondere ohne Proxy-Zertifikat möglich. Die Nutzung solcher Ressourcen bringt für den POV-Ray-Dienst den Vorteil, dass er keinen Aufwand mit der Bereitstellung von Proxy-Zertifikaten hat. Er kann die Ressourcen direkt verwenden, ohne sich zu authentifizieren. Jedoch haben dann auch andere Nutzer eventuell ungewollten Zugriff auf die Ressourcen, falls sich diese nicht in einer privaten, durch eine Firewall oder ähnliches geschützten Domäne befinden. Der Verzicht auf den Umgang mit Proxy-Zertifikaten macht also nur Sinn, wenn der POV-Ray-Dienst private Grid-Ressourcen (beispielsweise in Form eines Clusters) zur Verfügung hat, die entsprechend konfiguriert worden sind. Da es im Allgemeinen aber die gängige Methode ist, Globus Toolkit mit Sicherheitsunterstützung zu starten, würde ein Verzicht auf Authentifizierung auch eine Einschränkung auf ausschließlich privat zur Verfügung stehende Grid-Ressourcen bedeuten. Die Verantwortung für die Sicherheit dieser Ressourcen liegt dann beim POV-Ray-Dienst, bzw. bei Venice.

Im Rahmen des POV-Ray-Dienstes wird die erste Variante umgesetzt, da sie die flexibelste Nutzung von Grid-Ressourcen ermöglicht. Eine Erweiterung dieser Funktionalität wäre durch die dritte Variante gegeben. Die zweite Variante kommt aufgrund der genannten Nachteile nicht in Frage, während Variante drei das Vorhandensein entsprechend konfigurierter Grid-Ressourcen voraussetzt, und daher ebenfalls nicht umgesetzt wird. Die folgende Tabelle enthält zur besseren Übersicht eine Zusammenfassung der Vor- und Nachteile der verschiedenen Varianten:

²⁷<http://grid.ncsa.illinois.edu/myproxy>

Bereitstellung des Proxy-Zertifikats	
Vorteile	Nachteile
Variante 1: Benutzer	
<ul style="list-style-type: none"> - weniger Verwaltungsaufwand für den Dienst - Flexible Nutzung verschiedener Grid-Ressourcen 	<ul style="list-style-type: none"> - Benutzer muss Zertifikat besitzen - Benutzer muss Zugriff auf GT4-Ressourcen haben - Zertifikat muss lokal vorliegen
Variante 2: Dienst	
<ul style="list-style-type: none"> - kein Aufwand für Benutzer 	<ul style="list-style-type: none"> - Verstoß gegen Richtlinien der EU-GridPMA - fehlende Nutzungstransparenz - Vorkonfiguration des Dienstes notwendig - Beschränkung auf vorkonfigurierte Grid-Ressourcen
Variante 3: MyProxy	
<ul style="list-style-type: none"> - wie Variante 1 - Ortsunabhängigkeit des Clients bzgl. des Zertifikats 	<ul style="list-style-type: none"> - wie Variante 1
Variante 4: Keine Authentifizierung	
<ul style="list-style-type: none"> - kein Verwaltungsaufwand für den Dienst 	<ul style="list-style-type: none"> - GT4 läuft ohne GSI - Sicherheit muss anders gewährleistet sein (Firewall etc.) - nur Zugriff auf private Ressourcen

3.3.10 Fehlerbehandlung

Um den Dienst gegen Fehler und Ausnahmefälle robust zu machen, muss er intern eine angemessene Behandlung dieser Fehler beinhalten. Um dies zu bewerkstelligen, muss zunächst einmal erwägt werden, welche Fehler auftreten können. Man wird nie alle möglichen Fehler auflisten können. Jedoch sollte die Abdeckung möglichst hoch sein. Danach ist es notwendig, geeignete Strategien zu entwickeln, um den Fehlern zu begegnen und dem Benutzer eine geeignete Erklärung des aufgetretenen Fehlers anbieten zu können. In diesem Kapitel wird auf drei Aspekte der Fehlerbehandlung eingegangen, die den POV-Ray-Dienst betreffen.

Status “fehlgeschlagen”. In Kapitel 3.2 wurden die Anforderungen an den Dienst festgelegt. In Punkt F6 der Anforderungen wird festgehalten, dass der Benutzer den aktuellen Status des Renderns abfragen kann. Unter anderem wird der Status “fehlgeschlagen” genannt. In den Dienst selbst wurde dieser Status jedoch nicht mit übernommen. Schlägt ein Rendereauftrag aus irgendeinem Grund fehl, so wird der Benutzer über seine hinterlegten Kontaktinformationen darüber in Kenntnis gesetzt. Danach werden die Daten des Rendereauftrags gelöscht, um

nicht unnötigerweise Speicherplatz zu verbrauchen. Den Status weiterhin zu speichern, würde zusätzlichen Aufwand bedeuten, der unnötig ist, da der Benutzer ja per E-Mail oder SMS informiert wird. Aus demselben Grund macht es keinen Sinn, darauf zu warten, dass der Benutzer den Auftrag manuell löscht, nachdem er festgestellt hat, dass er fehlgeschlagen ist. Es nützt ihm nichts, dass die Daten weiterhin vom Dienst vorgehalten werden.

Stilllegung des Dienstes. In kritischen Fehlerfällen auf Dienstseite sollte eine Stilllegung der Dienste erfolgen. Kritisch bedeutet dabei, dass intern eine Bedingung gegeben ist, die einen weiteren Betrieb des Dienstes unmöglich macht. Ein Beispiel dafür wären nicht aufgelöste Abhängigkeiten zu anderen Softwarebibliotheken, wie beispielsweise POV-Ray oder Globus Toolkit. Dieser Fall könnte eintreten, wenn diese Bibliotheken z.B. nicht installiert sind. In einem solchen Fall ist es nicht sinnvoll, den Dienst nach außen weiterhin zur Verfügung zu stellen. Von Diensten, die man beispielsweise über einen Verzeichnisdienst wie den Domain Information Service erfragt und kontaktiert, und die sich als verfügbar präsentieren, erwartet man, dass diese bei korrekter Benutzung auch funktionieren. In einem kritischen Fehlerfall ist dies nicht gegeben.

Wiederaufnahme der Bearbeitung. Ein wichtiger Fehlerfall, der eintreten kann, ist eine vorzeitige Beendigung des Dienstes, zum Beispiel in Folge eines Ausfalls oder Neustarts von Venice. Ein Benutzer des POV-Ray-Dienstes sollte trotzdem erwarten können, dass bei einem Neustart des Dienstes seine Aufträge weiter bearbeitet werden, auch wenn sie vorher durch äußere Umstände abgebrochen worden sind. Bei lokalem Rendern ist dies nur dadurch möglich, dass man den Renderprozess neu startet. Beim Rendern im Grid ist dieser Weg auch möglich. Man generiert einfach neue Aufträge, sendet sie zum Grid und wartet wiederum auf deren Beendigung. Eine bessere Möglichkeit aber ist es, bereits erteilte Rendereaufträge im Grid auf deren Status hin zu überprüfen. Hat der Dienst nämlich vor Beendigung bereits parallele Rendereaufträge an das Grid verteilt, so ist es möglich, dass diese immer noch bearbeitet werden oder bereits beendet worden sind, da die Grid-Ressourcen ihre Aufträge ja unabhängig vom Zustand des Venice-Dienstes ausführen. Da der GT4-Dienst die Endpunktreferenzen dieser Aufträge sichert, kann deren Status auch bei einem Neustart des Dienstes erfragt werden. Dies wiederum kann der POV-Ray-Dienst nutzen, um seine Aufträge wieder aufzunehmen und dadurch eine schnellere Bearbeitung zu ermöglichen.

3.3.11 Benutzerschnittstelle

Die Benutzerschnittstelle, also der Client, wurde mittels Swing entworfen und ist an das Layout anderer in Venice verfügbarer Clients angelehnt, um eine einheitliche Darstellung zu gewährleisten. Dies begünstigt die Benutzbarkeit des Dienstes. Dabei muss gesagt werden, dass der Client vollkommen austauschbar ist und unabhängig vom eigentlichen Dienst existiert. Er nutzt lediglich die vom POV-Ray-Dienst angebotenen Operationen und stellt eine benutzerfreundliche Schnittstelle zu diesen dar.

Der Client bietet die Funktionen des Dienstes mit Hilfe zweier Panels an. Im ersten wird der Benutzer über den Status seiner POV-Ray-Aufträge informiert (F6). Es heißt daher Status-Panel. Es enthält eine Liste aller abgeschickten Aufträge mit der entsprechenden Job-ID, sowie deren aktueller Bearbeitungsstatus und ein vorberechnetes Thumbnail zur leichteren Orientierung. Auch wird der Zeitpunkt der Auftragserteilung angezeigt, sowie der Zeitpunkt der Beendigung des Auftrages, falls dies bereits geschehen ist. Die Liste kann jederzeit aufgefrischt werden, um eventuelle Änderungen im Status eines Auftrags sichtbar zu machen. Wurde ein Auftrag

abgeschlossen, so steht das gerenderte Bild seitens des POV-Ray-Dienstes zur Verfügung. Der Auftrag kann dann in der Liste selektiert und vom POV-Ray-Dienst abgerufen werden. Nach Abruf des Bildes wird es innerhalb des POV-Ray-Dienstes, und sodann auch aus der Liste entfernt. Man kann einen Auftrag aber auch schon vor oder nach Beendigung löschen. Er wird dann ebenfalls aus der Liste entfernt. Das Status-Panel wird in Abbildung 11 demonstriert.

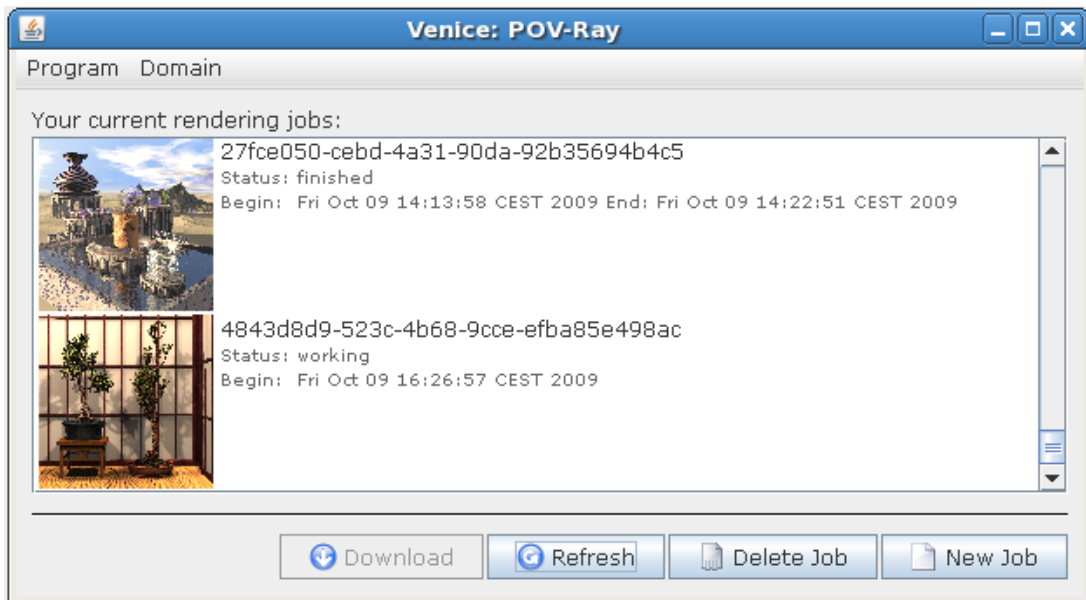


Abbildung 11: Status-Panel des POV-Ray-Clients

Ausgehend vom Status-Panel erreicht man über den Button "New Job" das Auftrags-Panel. Hier können sämtliche Einstellungen für einen Auftrag vorgenommen werden, und dieser dann an den POV-Ray-Dienst übermittelt werden. Dazu gehören die in Kapitel 3.3.2 beschriebenen Informationen, welche den Datentypen entsprechend gegliedert sind. So lassen sich die benötigten Dateien sowie Höhe und Breite des zu rendernden Bildes angeben. Über eine Checkbox kann man auswählen, ob man parallelisiertes Rendern durchführen möchte. Entsprechend werden dann die Informationsfelder für das Grid-Rendern aktiviert. Hier kann man dann die benötigten Informationen für GRAM und GridFTP eintragen. Außerdem kann man sich ein Proxy-Zertifikat aus einem vorhandenen Zertifikat generieren lassen. Zuletzt kann man auswählen, ob man per E-Mail oder SMS über Beendigung oder Fehlschlag des Prozesses informiert werden will, und entsprechende Kontaktdaten eintragen. Die Informationen werden dann über den Button "Render" an den POV-Ray-Dienst gesendet. Anschließend wird auch die Liste des Status-Panels aktualisiert, sodass man seinen neuen Auftrag dort direkt sehen kann.

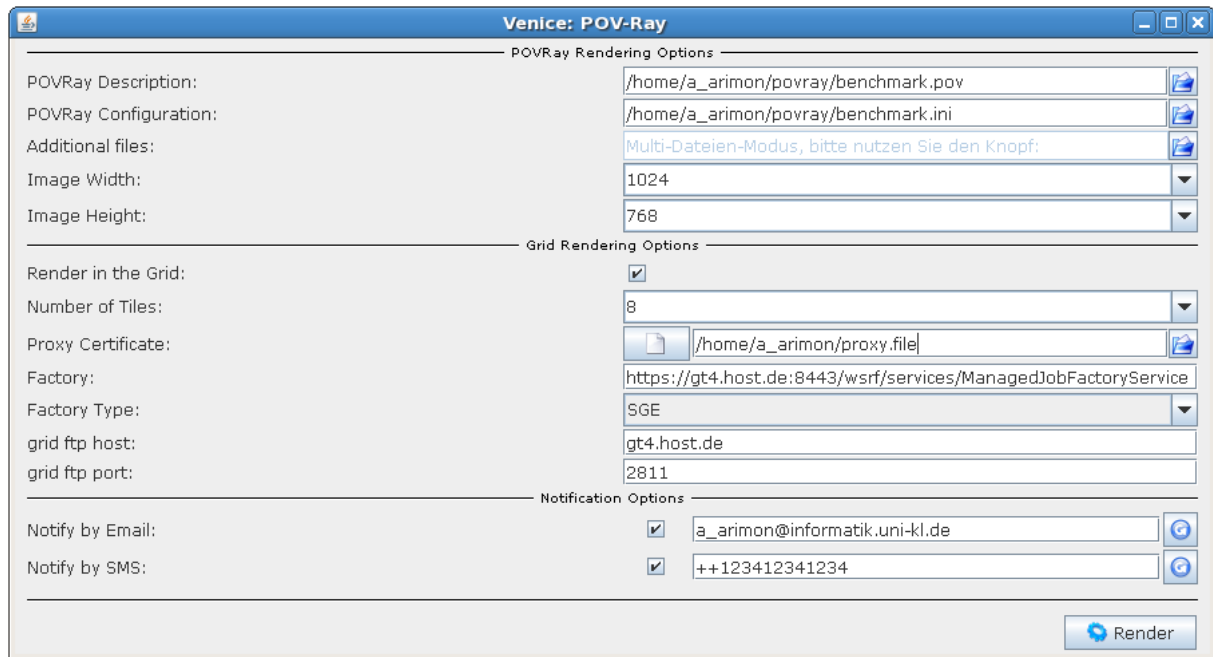


Abbildung 12: Auftrags-Panel des POV-Ray-Clients

4 Evaluation

Ein Fokus dieser Arbeit ist die Nutzung von Grid-Ressourcen, um die Ausführung des Renderns zu parallelisieren. Das Ziel dabei ist es, die Ausführung zu beschleunigen, den Dienst also performanter zu machen. Der POV-Ray-Dienst wird in diesem Kapitel hinsichtlich dieses Zieles evaluiert. Um eine solche Evaluation durchführen zu können, müssen geeignete Parameter ausgewählt werden. Außerdem muss eine geeignete Experimentierumgebung gegeben sein. Im Folgenden wird das Experiment beschrieben.

4.0.12 Parameter

Eingabebeschreibung. Zunächst muss bestimmt werden, welche Beschreibung als Eingabe für den Render-Prozess dienen soll. Verschiedene Beschreibungen von 3D-Szenen unterscheiden sich hinsichtlich ihrer Komplexität, was wiederum in unterschiedlichen Laufzeiten des Renderns resultiert. Da die Komplexität einer Beschreibung jedoch nur schwer quantitativ messbar ist, ist eine Gegenüberstellung dieser Laufzeiten wenig sinnvoll. Sinnvoller ist es, eine Beschreibung auszuwählen, die möglichst alle Eigenschaften und Funktionalitäten von POV-Ray nutzt, und somit repräsentativ für viele andere Beschreibungen ist. Hierzu findet sich auf der Webseite des POV-Ray Renderers ein Referenzbild, ein sogenannter Benchmark. Ein weiterer Vorteil dieses offiziellen Benchmarks ist die Möglichkeit des direkten Vergleichs mit anderen Systemen, die anhand dieses Benchmarks evaluiert worden sind, auch wenn dies nicht Ziel dieser Arbeit ist. Abbildung 13 zeigt das Referenzbild, welches für die Experimente verwendet wird.

Lokales Rendern vs. parallelisiertes Rendern im Grid. Vor allem soll mit den Experimenten gezeigt werden, dass die Parallelisierung des Renderns eine Beschleunigung mit sich bringen kann. Auch soll gezeigt werden, in welchem Maße die Beschleunigung vom Grad der

Abbildung 13: POV-Ray Benchmark²⁷

Parallelisierung abhängt. Der Grad der Parallelisierung bezieht sich dabei auf die Anzahl N der parallel ausgeführten Render-Prozesse. Dabei werden für N die Werte 1, 2, 4, 8, 16, 32 und 64 festgelegt. Eine Besonderheit ist dabei der Wert $N=1$, da er faktisch keine Parallelisierung darstellt. Jedoch besteht ein Unterschied darin, ob man das Rendern lokal und direkt im Venice Service Grid oder parallelisiert im externen Grid ausführt. Auch besteht Grund zur Annahme, dass eine Verdoppelung von N in etwa eine Halbierung der Rechenzeit bedeutet. Dass dies nicht notwendigerweise zutrifft, kann mehrere Gründe haben: Zum einen bringt der Vorgang der Parallelisierung einen zeitlichen Overhead mit sich, beispielsweise aufgrund des höheren Verwaltungsaufwands, der Installation der Software im Grid, und des Transferierens der Dateien zwischen Venice und den Grid-Ressourcen. Zum anderen kann man bei der Nutzung von Grid-Ressourcen aufgrund der Heterogenität der Hardware-Ressourcen nicht genau sagen, unter welchen Hardware-Bedingungen der Prozess des Renderns stattfindet. Auch kann man im Allgemeinen nicht sagen, wann ein Prozess ausgeführt wird. Dies hängt u.a. von der Auslastung der genutzten Grid-Ressourcen ab. Es kann durchaus vorkommen, dass Grid-Ressourcen von den Prozessen vieler anderer Benutzer ausgelastet sind, so dass ein Prozess lange warten muss, bis er vom Job Scheduler ausgeführt wird. Unter solchen Bedingungen ist eine quantitative Analyse nur schwer möglich. Für dieses Experiment wurden daher Grid-Ressourcen exklusiv verfügbar gemacht. Auch sind die Hardware-Voraussetzungen dieser Grid-Ressourcen sehr homogen. Details über die Ressourcen finden sich im nächsten Kapitel. Die exklusive Nutzung der Ressourcen und deren Homogenität begünstigt einen direkten Vergleich von lokalem Rendern und Rendern im Grid, ist jedoch nicht repräsentativ für die allgemeine Nutzung von Grid-Ressourcen unter den Bedingungen des Mehrbenutzerbetriebs.

Auflösung. Die Auflösung des Bildes spielt eine Rolle. Je höher die Auflösung, desto mehr

²⁷<http://www.povray.org/download/benchmark.php>

Pixel sind zu rendern, was in erhöhter Laufzeit resultiert. Für die Gegenüberstellung von lokalem Rendern gegenüber Rendern im Grid ist dies von Belang, da aufgrund des zeitlichen Overheads der Parallelisierung zu erwarten ist, dass für geringere Auflösungen ein lokales Rendern schneller ist als das parallelisierte Rendern. Dieser Sachverhalt soll mit den Experimenten ebenfalls untersucht werden. Dazu werden verschiedene Auflösungen gewählt, von 320×240 Pixel bis 1920×1200 Pixel.

4.0.13 Umgebung

Im Folgenden werden die dem Gesamtsystem zugrundeliegenden Hardware-Ressourcen beschrieben. Sie stellen im architektonischen Gesamtkontext des Systems den Fabric Layer dar (siehe Kapitel 2.1.4). Der POV-Ray-Dienst ist dabei die Applikation, die mit Hilfe des Venice Service Grid und Globus Toolkit diese Hardware-Ressourcen nutzt, um seine Aufgaben zu erledigen. Der POV-Ray-Dienst kann damit dem Application Layer zugeordnet werden.

Das Venice Service Grid und seine Dienste laufen auf Basis eines Tomcat Servers auf einem Linux-System. Die Hardware dieses Systems enthält zwei Intel Xeon E5345 Quad-Core-Prozessoren zu je 2,33 GHz. Die Software POV-Ray in der Version 3.6.1 ist noch nicht mehrkernfähig. Das bedeutet, dass für jeden Rendervorgang stets nur ein CPU-Kern gleichzeitig genutzt wird.

Die externen Grid-Ressourcen sind ein Cluster, bestehend aus 23 Blades, die auch jeweils zwei Intel Xeon E5345 Quad-Core-Prozessoren mit 2,33 GHz enthalten. Jeder der Blades hat einen Hauptspeicher von 32 GB RAM. Wie vorhin erwähnt, ist die Hardware in der Experimentierumgebung also sehr homogen.

4.0.14 Durchführung

Zur Durchführung des Experiments wurden die oben genannten Ressourcen exklusiv verfügbar gemacht, um so eine Verfälschung der Ergebnisse durch nebenläufige Prozesse anderer Benutzer zu vermeiden. Für jede mögliche Auflösung und jeden Wert N für die Anzahl parallel rendernder Grid-Prozesse wurde das Experiment mindestens fünf mal wiederholt, und daraus jeweils ein Mittelwert gebildet. Dies soll ebenfalls die Verfälschung der Ergebnisse durch externe Ereignisse und Gegebenheiten mindern. Größere zeitliche Ausreißer wurden dabei vor der Auswertung entfernt. Zur Messung der Dauer jedes Gesamtrendervorgangs wurde der in Kapitel 2.2.2 beschriebene TimeKeeper-Dienst benutzt. Die Dauer der Installation wird bei diesen Experimenten weggelassen. POV-Ray ist bereits im Grid installiert, sodass eine erneute Installation nicht nötig ist. Im realen Anwendungsfall muss dies nicht so sein. Daher wurde eine separate Messung der Installationsdauer von POV-Ray durchgeführt. Sie beträgt ca. drei Minuten. Diese Zeit muss beim parallelisierten Rendern im Grid hinzugerechnet werden, wenn man davon ausgeht, dass ein Rendern mit Hilfe des Dienstes zum ersten Mal durchgeführt wird und POV-Ray im Grid noch nicht installiert worden ist.

4.0.15 Ergebnisse

Die Ergebnisse der Experimente sind in Abbildung 14 graphisch dargestellt. Die Kurven zeigen die Gesamtausführungszeit in Abhängigkeit von der Anzahl der Render-Prozesse. Jede Kurve

repräsentiert dabei eine andere Auflösung²⁸. Die horizontalen, gepunkteten Linien deuten für jede Auflösung jeweils die durchschnittliche Ausführungszeit bei lokalem Rendern an. Hiermit kann direkt verglichen werden, ob und ab wievielen Render-Prozessen sich ein paralleles Rendern für eine bestimmte Auflösung lohnt. Es muss beachtet werden, dass die hier gezeigten Werte nur für den POV-Ray-Benchmark gelten. Das Rendern anderer 3D-Szenen führt höchstwahrscheinlich zu anderen konkreten Ergebnissen. Wann sich also ein paralleles Rendern lohnt, hängt auch von der zu rendernden 3D-Szene ab. Insgesamt lassen sich die hier gezeigten Ergebnisse durchaus verallgemeinern, da der POV-Ray-Benchmark für diesen Zweck ausgelegt ist.

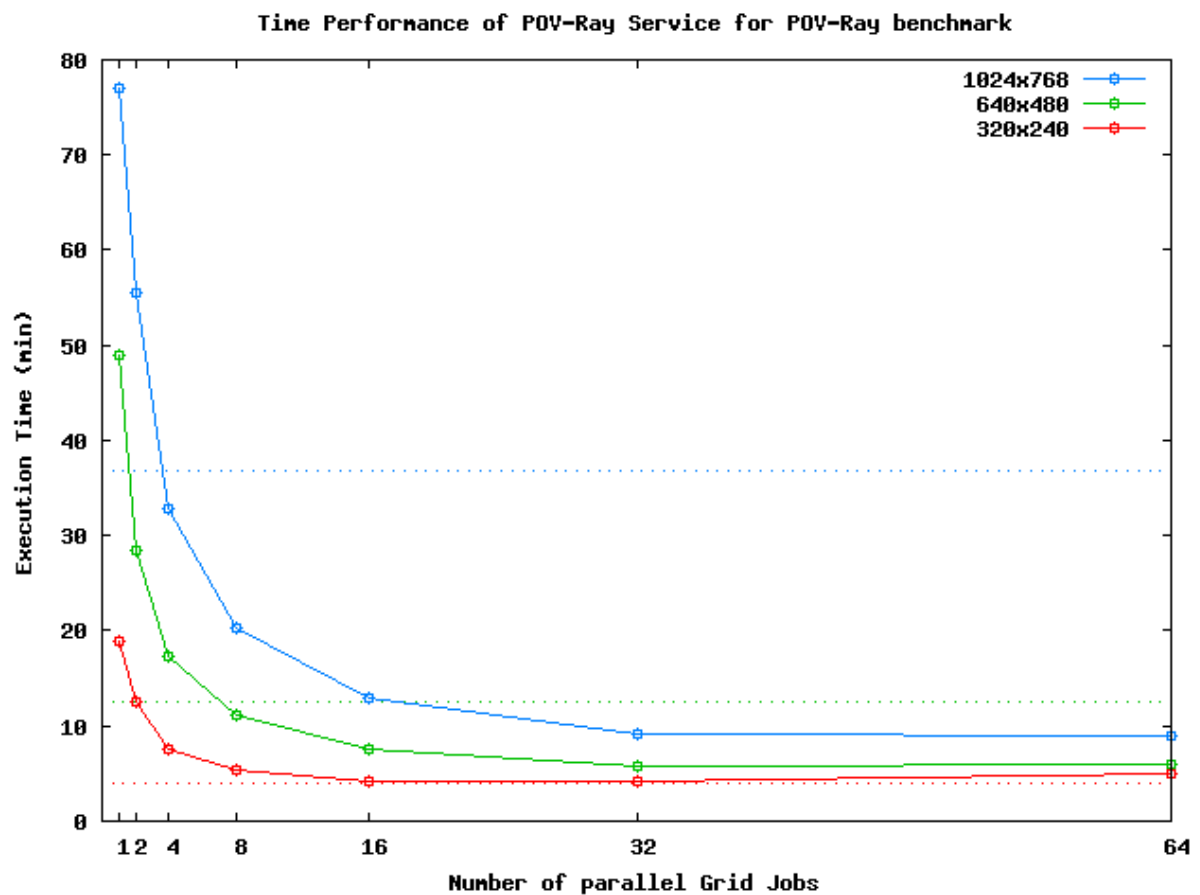


Abbildung 14: Ergebnisse der Evaluation: Absolute Dauer

Zunächst sieht man, dass bei höheren Auflösungen natürlich eine höhere durchschnittliche Ausführungszeit besteht. Man sieht auch, dass mit Zunahme der parallelen Render-Aufträge fast immer eine Verkürzung der Ausführungsdauer einhergeht. Insgesamt können vier wichtige Beobachtungen gemacht werden:

(1) Eine Verdopplung der Render-Prozesse führt nicht zu einer Halbierung der Gesamtausführungszeit. Man kann dies an allen Kurven und für jede Anzahl von Prozessen problemlos schon anhand der Abbildung feststellen. Die konkreten Werte bestätigen dies. Sie können in

²⁸Die Auflösungen 1280x1024 und 1920x1200 wurden ebenfalls untersucht, wurden aber der Übersicht wegen aus der Grafik weggelassen. Die Ergebnisse dazu sind in Appendix B zu finden.

Appendix B eingesehen werden. Eine Erklärung dafür ist der zeitliche Overhead, den das Parallelisieren mit sich bringt. Jeder Auftrag erzeugt zusätzlichen Aufwand bspw. für Job Scheduling, Monitoring oder Datentransfer, aber auch für das Initialisieren von POV-Ray selbst, sodass eine Halbierung der Zeit auf homogenen²⁹ Hardware-Ressourcen nicht erreicht werden kann.

(2) Paralleles Rendern lohnt sich erst ab einer gewissen Auflösung. Der direkte Vergleich mit der Ausführungszeit bei lokalem Rendern (gepunktete Linien) zeigt dies. So sieht man, dass sich paralleles Rendern für eine Auflösung von 320x240 gar nicht lohnt. Bei einer Auflösung von 640x480 benötigt man mindestens 8 Prozesse, während man bei einer Auflösung von 1024x768 bereits mit 4 Prozessen schneller ist als lokales Rendern. Auch dies kann durch den zeitlichen Overhead des Parallelisierens erklärt werden.

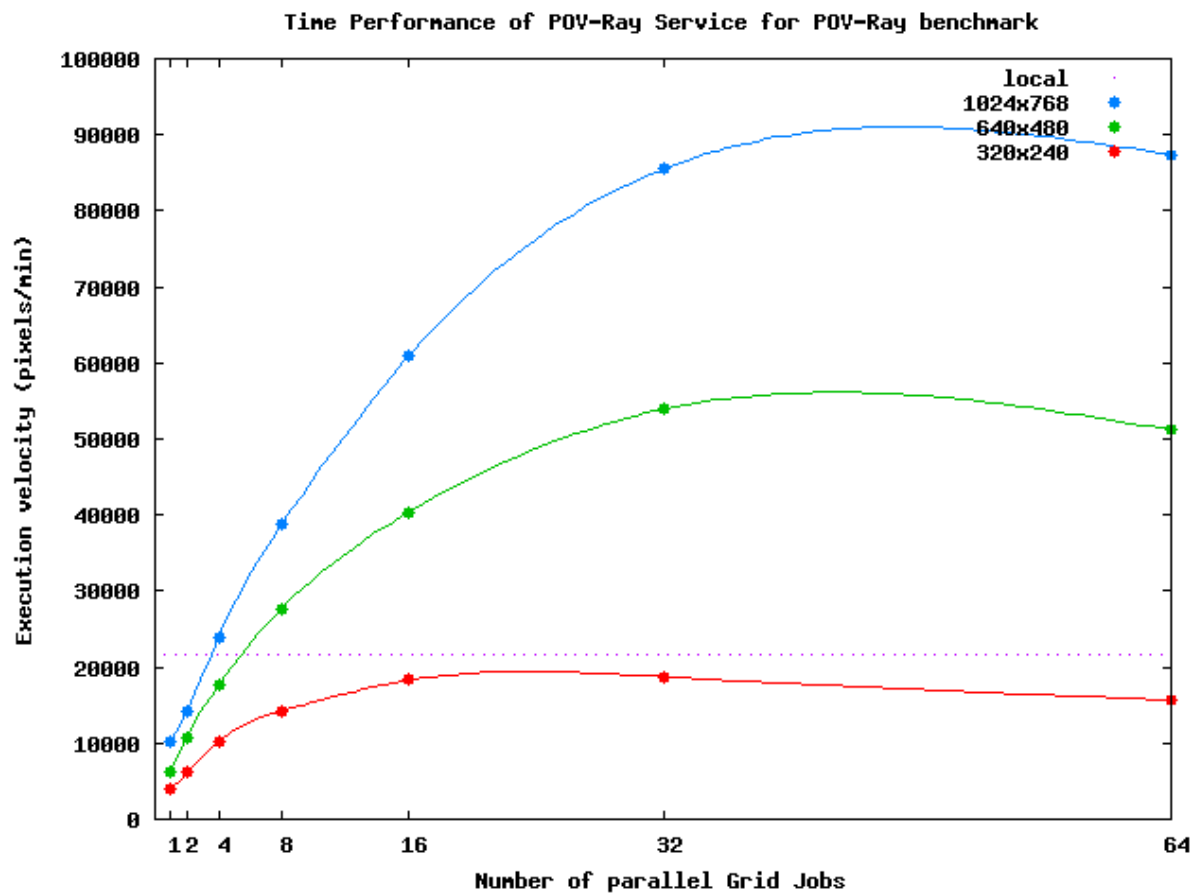


Abbildung 15: Ergebnisse der Evaluation: Geschwindigkeit

(3) Ab einer gewissen Anzahl von parallelen Prozessen stagniert die Ausführungszeit, bzw. sie steigt wieder an. Man sieht dies ganz leicht bei allen Kurven, wenn man den Übergang zwischen $N=32$ und $N=64$ betrachtet. Ein Grund dafür stellt wieder der zeitliche Overhead dar. Werden die parallelen Renderaufträge immer mehr, so wird die zu rendernde Fläche pro Teilauftrag immer kleiner. Analog zu Punkt (2) lohnt sich dann ein paralleles Rendern jedoch nicht mehr, da der zusätzliche Aufwand pro Auftrag den Ertrag des Parallelisierens übersteigt.

²⁹Auf heterogenen Hardware-Ressourcen ist dies im Allgemeinen möglich, da diese ja auch schnellere Rechner beinhalten können.

(4) Je höher die Auflösung des Bildes, desto mehr lohnt sich die Parallelisierung. Diese Beobachtung ist eng verwandt mit Beobachtung (2), ist jedoch anhand der ersten Abbildung weniger offensichtlich. Daher wurde die Geschwindigkeit des Renderns der Anzahl der parallelen Render-Prozesse gegenübergestellt. Die Geschwindigkeit des Renderns ist dabei die Anzahl der Pixel, die durchschnittlich pro Minute gerendert werden. Sie wird errechnet, indem die Gesamtanzahl der gerenderten Pixel durch die Gesamtlaufzeit dividiert wird. In Abbildung 15 sieht man dies graphisch dargestellt³⁰. Man sieht hier, dass die Geschwindigkeit bei hoher Auflösung deutlich größer ist als bei niedrigen Auflösungen. Auch dies lässt sich wiederum aus dem zeitlichen Overhead des Parallelisierens erklären. Der Anteil dieses zeitlichen Overheads an einem einzelnen Render-Auftrag ist bei höherer Auflösung geringer, so dass insgesamt eine höhere Ausführungsgeschwindigkeit erreicht werden kann. Man sieht aber auch, dass diese Geschwindigkeit im Sinne von Punkt (3) bei einer entsprechend hohen Anzahl von parallelen Prozessen ($N=64$) wieder abnimmt.

³⁰Die Kurven wurden mit Spline-Interpolation angenähert. Die horizontale Linie stellt die durchschnittliche Geschwindigkeit beim lokalen Rendern dar.

5 Related Work - Instant Grid

Das Instant Grid Projekt war ein vom Bundesministerium für Bildung und Forschung gefördertes Projekt, an dem unter anderem das Fraunhofer Institut für Rechnerarchitektur und Softwaretechnik beteiligt war³¹. Instant Grid ist eine Knoppix³²-basierte Live-CD mit einer vorkonfigurierten Grid-Umgebung auf Basis des Globus-Toolkit. Ein PC wird von der CD gestartet und fungiert als Frontend des Instant-Grid. Weitere PCs im gleichen lokalen Netzwerk können der Instant-Grid-Umgebung leicht hinzugefügt werden. Gemeinsam mit dem Frontend bilden diese Rechner ein funktionsfähiges Test-Grid. Mit dem Instant-Grid können Grid-Technologien ausprobiert und eigene Grid-Anwendungen entwickelt und getestet werden. Das vorkonfigurierte Globus-Toolkit erlaubt das Ausführen und Überwachen von beliebigen Programmen auf allen Rechnern im Instant-Grid, sowie die parallele Nutzung dieser Rechner mit entsprechend vorbereiteter Software.

Unter anderem bietet Instant Grid eine Webschnittstelle und ein Portlet für die Nutzung von POV-Ray innerhalb des Instant Grid und bietet damit ähnliche Funktionalität wie der in dieser Projektarbeit vorgestellte POV-Ray-Dienst. So kann mithilfe der Webschnittstelle parallelisiertes Rendern durchgeführt werden. Auf diese Weise lässt sich mit Instant Grid schnell eine "Render-Farm" zusammenstellen.

Instant Grid geht bei der Parallelisierung ähnlich vor wie der POV-Ray-Dienst: durch zeilenweises Aufteilen des Gesamtbildes. Die Webschnittstelle gibt dabei u.a. Auskunft, auf welchem Rechner gerade welche Teile gerendert werden und welche Teile bereits fertig sind (Abbildung 16) Zusätzlich zu Parametern wie Höhe und Breite des Bildes kann hier noch die Ausgabe in Form von JPG gewählt werden (Der POV-Ray-Dienst beschränkt sich hier auf PNG, wobei eine Umwandlung von PNG in JPG leicht möglich ist). Auch sind weitere Qualitätseinstellungen wie beispielsweise Antialiasing direkt möglich. Diese Einstellungen können aber auch in einer Konfigurationsdatei übergeben werden.

Neben dem Rendern von 3D-Szenen bietet das Instant Grid auch das Rendern von Animationen an. Dies geschieht im Prinzip durch das Rendern mehrerer Einzelbilder der Animation auf die gleiche Weise wie beim Rendern normaler Szenen. Aus der Animation kann nach der Fertigstellung ein Video generiert werden, welches auch direkt angeschaut werden kann.

Host	Start frame	End frame	Status
server	1	3	Done!
bombay	4	6	Done!
bombay	7	9	Done!
server	10	12	Done!
bombay	13	15	Done!
server	16	18	Done!
karachi	19	21	Rendering
bombay	22	24	Rendering
server	25	27	Rendering
	28	30	Queued
	31	33	Queued
	34	36	Queued
	37	39	Queued
	40	42	Queued
	43	45	Queued
	46	50	Queued

Abbildung 16: Detaillierter Status eines Auftrags im Instant Grid³¹

³¹ www.instant-grid.org

³² <http://www.knoppix.org/>

6 Zusammenfassung und Ausblick

In dieser Projektarbeit wurde die Entwicklung und Evaluation eines Raytracing-Dienstes für das Venice Service Grid vorgestellt. Ein besonderer Schwerpunkt bildete dabei die Einbeziehung von Grid Computing zur Beschleunigung der Funktionalität des Dienstes. Es wurden daher zunächst die Begriffe “Grid Computing” und “Raytracing” erklärt, sowie das Venice-Framework und die verwendete Globus Toolkit 4 Middleware vorgestellt.

Grid Computing hat zum Ziel, Netzwerke, Kommunikation, Datenverarbeitung und Information zu integrieren, und einen einfachen, überall verfügbaren Zugriff auf diese Ressourcen zu ermöglichen. Es eignet sich unter anderem, um aufwändige Prozesse, wie beispielsweise das Rendern von 3D-Szenen, auf entsprechende Rechenkapazitäten zu verteilen.

Ein Algorithmus zum Rendern von 3D-Szenen ist Raytracing. Er basiert auf der Rückverfolgung von Lichtstrahlen zur Lichtquelle hin. Da der Aufwand des Algorithmus bei komplexen Szenen sehr hoch ist, lohnt sich in vielen Fällen eine Parallelisierung des Vorgangs. Eine Möglichkeit, eine solche Parallelisierung zu bewerkstelligen, bietet Grid Computing.

Der Raytracing-Dienst wurde für das Venice Service Grid entworfen. Das Venice Service Grid ist ein offenes Webservice-basiertes Framework für verteilte Applikationen, welches ein leichtes Erstellen, Entwickeln, Integrieren und Benutzen von Diensten ermöglicht.

Zur Nutzung von externen Ressourcen im Sinne des Grid Computings wurde die Middleware Globus Toolkit 4 verwendet. Die in dieser Middleware enthaltenen Komponenten GRAM und GridFTP ermöglichen das Ausführen von Aufträgen und das Transferieren von Daten auf externe Grid-Ressourcen.

In Kapitel 3 wurde der entwickelte Raytracing-Dienst beschrieben. Der Dienst nutzt die Software POV-Ray, um ankommende Beschreibungen von 3D-Szenen zu rendern. Dabei können verschiedene Einstellungen wie Höhe und Breite des zu rendernden Bildes vorgenommen werden. Der Status sowie eine Vorschau aktueller Aufträge kann ebenfalls abgefragt werden. Der Dienst stellt eine Schnittstelle zur Verfügung, die eine Nutzung von externen Grid-Ressourcen für verteiltes Rendern möglich macht. Der Dienst nutzt dabei wiederum einen weiteren, eigens dafür entwickelten Dienst, der Funktionalität des Globus Toolkit 4 in Venice integriert. Wichtig bei der Nutzung externer Grid-Ressourcen ist die Authentifizierung des Benutzers. Hierzu wurden mehrere Möglichkeiten der Bereitstellung von Proxy-Zertifikaten diskutiert. Ist ein Bild fertig gerendert, kann der Benutzer mittels E-Mail oder SMS benachrichtigt werden und sein Bild anschließend vom Dienst erhalten.

Im vierten Kapitel wurde untersucht, inwiefern durch eine Parallelisierung des Renderns eine Beschleunigung erreicht werden kann. Dazu wurden konkrete Experimente gemacht und ausgewertet. Anhand der Ergebnisse lässt sich eine deutliche Beschleunigung feststellen, die sich vor allem für komplexe Bilder mit hoher Auflösung lohnt.

Es gibt verschiedene Ansatzpunkte, anhand derer man den Raytracing-Dienst als auch den Dienst für das Globus Toolkit erweitern könnte. Zunächst einmal könnte man den Raytracing-Dienst dahingehend erweitern, dass man nicht nur Einzelbilder, sondern ganze Animationen rendern kann. Dies kann auch mit Hilfe von Grid Computing geschehen. Dadurch könnten viele Einzelbilder parallel gerendert werden. Eine Funktionalität, welche die Benutzerfreundlichkeit des Raytracing-Dienstes verbessern würde, wäre eine Vorabschätzung der Dauer des Renderns. Es wäre beispielsweise denkbar, dass die Dauer des Renderns von der Dauer der Erstellung der Vorschau her geschätzt werden kann. Vorausgesetzt, man kann Grid-Ressourcen nutzen, die

weitestgehend homogen sind, lässt sich eine solche Schätzung auch nutzen, um zu entscheiden, ob ein Auftrag lokal oder im Grid gerendert werden sollte, und wie viele parallele Prozesse optimal sind. Dies könnte anhand von Statistiken geschehen. Weiterhin könnten detailliertere Statusaussagen gemacht werden, indem man beispielsweise die Anzahl und den Fortschritt der parallel verarbeiteten Aufträge im Grid anzeigt.

Der GT4-Dienst könnte um Funktionalität für Monitoring and Diagnostics Services (MDS) erweitert werden. So könnten verfügbare und weniger ausgelastete Ressourcen dynamisch ausfindig gemacht und ankommende Aufträge an diese verteilt werden³³. Generell kann der GT4-Dienst noch um weitere Funktionalität aus dem Globus Toolkit erweitert werden. In Hinblick auf die Authentifizierung von Benutzern stellt die Anbindung an MyProxy einen sinnvollen nächsten Schritt dar. Auch die in Kapitel 3.3.8 angesprochene Integration der Datentypen aus GT4 und Venice wäre ein Ansatzpunkt.

Aufgrund der hohen Anforderungen an verfügbare Rechenkapazitäten, die heutzutage und in Zukunft gestellt werden, wird der Einfluss von Grid Computing auf Verfahren der Datenverarbeitung immer größer. Die im Rahmen dieser Projektarbeit entwickelten Dienste sind sowohl ein nützliches Beispiel für die Integration von Applikationen und Grid-Ressourcen, als auch eine erfolgreiche Anwendung von Grid Computing im Zusammenhang aufwändiger Berechnungen, wie sie beispielsweise beim Rendern von 3D-Szenen vorkommen.

³³Dies stellt eine Erweiterung der Architektur im Sinne des Collective Layer aus Kapitel 2.1.4 dar.

A Code-Beispiele

Listing 6: Auszug aus der XML-Schema Datei GT4.xsd, Datentyp zur Angabe von Host und Port bei GridFTP-Kommunikation

```
<complexType name="GridFTPHost">
  <sequence>
    <element name="host" type="string"/>
    <element name="port" type="int"/>
  </sequence>
</complexType>
```

Listing 7: Auszug aus der XML-Schema Datei GT4.xsd, Datentyp zur Angabe von GRAM-Kontaktinformationen

```
<complexType name="JobFactoryInformation">
  <sequence>
    <element name="factory" type="string"/>
    <element name="factoryType" type="string"/>
  </sequence>
</complexType>
```

Listing 8: Auszug aus der XML-Schema Datei GT4.xsd, Datentyp zur Auftragsbeschreibung

```
<complexType name="JobDescription">
  <sequence>
    <element name="executable" type="string"/>
    <element name="directory"
      type="string"
      minOccurs="0"/>
    <element name="arguments"
      type="string"
      minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

B Experimentelle Ergebnisse

Resolution	local	Grid: Number of Render Processes						
		1	2	4	8	16	32	64
320x240	4.0	18.9	12.6	7.5	5.4	4.2	4.1	4.9
640x480	12.5	48.9	28.5	17.4	11.1	7.6	5.7	6.0
1024x768	36.8	77.1	55.5	32.8	20.3	12.9	9.2	9.0
1280x1024	61.2	157.0	83.0	50.1	27.6	18.3	12.8	12.0
1920x1200	107.2	254.6	161.6	91.0	52.8	31.8	20.6	17.0

Abbildung 17: Dauer der Rendervorgänge in Minuten

Literatur

- [1] I. Foster, C. Kesselman *The Grid: Blueprint for a New Computing Infrastructure* 1998, CA: Morgan Kaufmann
- [2] F. Berman, A. Hey, G. Fox *Grid Computing - Making the GLocal Infrastructure a Reality*, John Wiley & Sons Ltd, 2003
- [3] I. Foster, C. Kesselman, S. Tuecke *The Anatomy of the Grid*, 2000
- [4] I. Foster *What is the Grid? A Three Point Checklist*, 2002
- [5] WLCG: <http://lcg.web.cern.ch/lcg/>
- [6] M. Hillenbrand, J. Götze, G. Zhang, P. Müller *A Lightweight Service Grid based on Web Services and Peer-to-Peer*, 2007
- [7] Venice: <http://www.v-grid.info>
- [8] M. Hillenbrand, J. Götze, P. Müller *Contract-first Service Development Within the Venice Service Grid*, 2008
- [9] Globus Toolkit: <http://www.globus.org/>
- [10] I. Foster *Globus Toolkit 4: Software for Service-Oriented Systems*, 2006
- [11] B. Allcock, J. Bresnahan, R. Kettimuthu et. al. *The Globus Striped GridFTP Framework and Server*
- [12] T. Dierks, C. Allen *The TLS Protocol Version 1.0*, <http://www.ietf.org/rfc/rfc2246.txt>
- [13] W. Diffie, M. Hellman *New Directions in Cryptography*
- [14] The Globus Security Team *Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective*, 2005
- [15] <http://de.wikipedia.org/wiki/Bildsynthese>
- [16] Andrew S. Glassner *An Introduction to Ray Tracing*, USA: Morgan Kaufmann Publishers Inc., 1989
- [17] <http://de.wikipedia.org/wiki/Raytracing>
- [18] Jonothan Hunt "*Pebbles*", <http://hof.povray.org/pebbles.html> , 2008
- [19] <http://de.wikipedia.org/wiki/Povray>
- [20] RSL: http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html
- [21] Java RMI: <http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp>
- [22] WS-Security: <http://www.ibm.com/developerworks/library/specification/ws-secure/>

- [23] Portable Batch System (PBS): <http://www.openpbs.org>
- [24] Condor: <http://www.cs.wisc.edu/condor>
- [25] Sun Grid Engine (SGE): <http://gridengine.sunsource.net/>
- [26] EUGridPMA: <http://www.eugridpma.org/>
- [27] EUGridPMA *Authentication Profile for Classic X.509 Public Key Certification Authorities with secured infrastructure*, Version 4.1, Dec 2006
- [28] EUGridPMA *Policy on host and service entities*, May 2009
- [29] EUGridPMA *Policy on automated client or robot entities*, May 2009

Abkürzungsverzeichnis

CA	Certificate Authority
CPU	Central Processing Unit
DFS	Distributed File System
DRS	Data Replication Service
EPR	Endpoint Reference
EUGridPMA ..	EU Grid Policy Management Authority
FTP	File Transfer Protocol
GRAM	Grid Resource Allocation Management
GridFTP	Grid File Transfer Protocol
GSI	Grid Security Infrastructure
GT4	Globus Toolkit 4
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JVM	Java Virtual Machine
LHC	Large Hadron Collider
MDS	Monitoring and Diagnostics Services
MJFS	Managed Job Factory Service
OGSA-DAI ...	Open Grid Services Architecture - Data Access and Integration
PNG	Portable Network Graphics
POV-Ray	Persistence of Vision Raytracer
RFT	Reliable File Transfer
RLS	Replica Location Service
RMI	Remote Method Invocation
RSL	Resource Description Language
SMS	Short Message Service
SSO	Single Sign-On
TLS	Transport Layer Security
UUID	Universally Unique Identifier
VO	Virtuelle Organisation
WAN	Wide Area Network
WLCG	World LHC Computing Grid
WSDD	Webservice Deployment Descriptor
WSDL	Webservice Description Language
XML	Extensible Markup Language