Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

***Studienarbeit***

# Untersuchung des Globus Toolkit als Grid Computing Middleware

vorgelegt von
***Roland Beintner***
***Hannes Osius***
am 24. Februar 2005

Studiengang Softwaretechnik

Betreuender Prüfer: Prof. Dr. Kai von Luck

**Fachbereich Elektrotechnik und Informatik**
**Department of Electrical Engineering and Computer Science**

# 1 Abstract

## 1.1 Deutsch

Diese Arbeit soll einen Überblick zum aktuellen Stand der Technologie Grid Computing liefern. Zunächst wird auf die Grundlagen des Grid Computing und seine Geschichte eingegangen, dann folgt ein Bericht über unsere Erfahrungen bei der Installation des Globus Toolkits auf einem Testgrid und der Entwicklung eines kleinen Gridservice, schließlich unsere Beurteilung, inwieweit Grid Computing an seinen Zielen bereits gemessen werden kann.

## 1.2 English

This document is supposed to give an overview about the emerging technology Grid Computing and it's current status. It starts with the basic technologies of Grid Computing and it's history, after which we will tell about our experiences installing the Globus Toolkit on our test grid, including the developement of a small grid service. Finally we try to estimate how Grid Computing today matches with it's objectives.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 2 Einleitung

Grid Computing ist zweifelsohne eine der Technologien, die die Arbeit mit Computern im Firmen-, in fernerer Zukunft wahrscheinlich auch im Heimbereich entscheidend mitbeeinflussen wird. Auf dem jetzigen Stand erlaubt es Firmen, aufwändige Berechnungen um ein Vielfaches schneller auszuführen (wie, wird diese Studienarbeit unter anderem aufzeigen), die Vision jedoch reicht um einiges weiter bis hin zur „Rechenpower aus der Steckdose“.

Um diese Vision, und damit das Ziel des Grid Computing, in einem Satz zu erklären: Für die Benutzung von Ressourcen wie Rechenzeit und Speicherplatz ist es unwichtig, wo und durch wen diese bereitgestellt werden, sondern nur noch ihre Kosten - sie werden als universelle Ressourcen behandelt.

Wie ernst Grid Computing inzwischen genommen wird, zeigt deutlich die Anzahl der Institutionen und Firmen, die sich an den verschiedenen Projekten beteiligen - ob Sun, IBM oder Microsoft, die DARPA oder die EU.

Die weite Verbreitung des Begriffs „Grid“ macht es zunächst notwendig, eine Begriffsklärung vorzunehmen. In ihrem Buch „The Grid: Blueprint for a New Computing Infrastructure“, (Foster und Kesselman, 1998) das als Grundlage des Grid Computing gilt, schrieben Ian Foster und Carl Kesselman: „A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.“

Die Schlagwörter sind also verlässlich, konsistent, überall verfügbar und kostengünstig.

In weiteren Artikeln wurde die Definition verfeinert, so dass schließlich eine Checkliste entstand - was muss für ein Grid gegeben sein?

1. Das Grid koordiniert Ressourcen, die nicht unter einer zentralisierten Kontrolle stehen.
2. Es benutzt standardisierte, offene, universale Protokolle und Interfaces. Ein Grid ist also ausdrücklich nicht anwendungsspezifisch.
3. Es liefert nicht-triviale Dienste, womit gemeint ist, dass verschiedene Arten von Ressourcen (wie Speicherplatz, Rechenzeit) und Zielen (z.B. Verfügbarkeit, Antwortzeit, Sicherheit) in einer Weise kombiniert werden, dass das Endergebnis wertvoller als die Summe seiner einzelnen Teile ist.

Diese Einschränkungen sind wichtig, um bestimmte Architekturen vom Grid-Begriff abgrenzen zu können. So fällt z.B. Suns Grid Engine in den Cluster-Management Bereich und wird zentral verwaltet, aktuelle Distributed Computing Projekte wie SETI@HOME, auf das später in dieser Arbeit noch genauer eingegangen wird, sind fast immer auf eine bestimmte Anwendung zugeschnitten.

# 3 Geschichte des Verteilten Rechnens

(University of British Columbia, Dep. Computer Sciences, 2004)

Seit ungefähr 1970 entwickelte sich das „Distributed Computing“ als neue Disziplin des Hochleistungsrechnens. Mit der fortschreitenden Entwicklung von Netzwerktechnologien, wie z.B. Ethernet als erstes Local Area Network, war es möglich, zeitraubende, große Berechnungen relativ effizient auf mehrere Rechner zu verteilen. Die Alternative stellten damals wie heute die Supercomputer dar, Mainframes mit der vielfachen Leistung eines normalen Rechners.

In den achtziger Jahren setzte sich die Server-Client-Architektur als Paradigma der Anwendungsentwicklung durch. Je nach Anwendung wurde der Großteil der Berechnungen auf einem zentralen Server, oder aber auf den einzelnen Client durchgeführt. Die Verbreitung von TCP/IP und OSI als offene Standards sorgten für verläßliche Verbindungen, um Daten über die Netzwerke auszutauschen.

Heute versteht man unter verteilten Systemen eine Kollektion unabhängiger Rechner, die dem Benutzer als ein System erscheinen. Vorausgesetzt ist eine Vernetzung der Rechner untereinander, Ressourcen werden gemeinsam benutzt, und es herrscht eine einheitliche Sicht auf das Gesamtsystem vor.

Verteiltes Rechnen basiert nun auf nebenläufigen Prozessen und/oder Threads, d.h. jeder Teilprozess kann unabhängig von den anderen seinen Teilbereich berechnen. Die Methoden sind dabei vielfältig, die wichtigsten sollen hier kurz vorgestellt werden.

## 3.1 Distributed Computing

Distributed Computing war die erste Lösung, um verteilte Berechnungen anstellen zu können. Es beschreibt im allgemeinen die Architektur, Teile einer Anwendung nicht nur auf einem einzelnen, sondern auf mehreren Systemen parallel ablaufen lassen zu können. Dazu ist es notwendig, dass die Anwendung parallelisierbar ist, d.h. dass einzelne Arbeitsschritte des Prozesses von anderen entkoppelt, und zur Berechnung weitergegeben werden können.

Das bekannteste Beispiel für DC ist SETI@HOME, ein Programm zur Suche von außerirdischen Signalen im Weltraum. SETI@HOME ist ein Projekt der Universität Berkeley, und wird seit 1996 betrieben - in dieser Zeit wurden mit momentan 5,3 Millionen Benutzern insgesamt $6 * 10^{21}$ Fließkommaoperationen durchgeführt, entsprechend über zwei Millionen CPU-Jahren. (Stand: Januar 2005) (SETI@HOME-Website, 2005).

Die dahinterstehende Technik wird im folgenden Kapitel noch kurz erklärt.

## 3.2 RPC / RMI

RPC ist im allgemeinen die Technik, eine Prozedur auf einem fremden Rechner aufrufen zu können. Verbreitet ist sie in objektorientierten Programmiersprachen wie C++ oder Java, wobei es für beide Seiten unwichtig ist, in welcher Sprache die jeweils andere Seite implementiert wurde. Üblicherweise verwendet man einen sogenannten ORB (Object Request Broker) in Verbindung mit einem Namensdienst, um ebenfalls von dem genauen Standort der anderen Seite unabhängig zu sein. RPC ist eine Ausprägung der Server-Client-Architektur, wobei der Client den auf dem Server angebotenen RPC-Dienst aufruft, und die Ergebnisse für sich weiterverwertet.
RMI oder auch Java-RMI ist die von Sun propagierte RPC-Lösung für Java. Hierbei wird auf einem entfernten Java-Objekt eine Methode aufgerufen, wobei entfernt bedeutet, dass das Objekt auf einer anderen Virtuellen Maschine liegt, durch den Client aber wie ein lokales Objekt behandelt und angesprochen wird (Ausnahme: Besondere Exceptions für Verbindungsabbruch etc.). RMI benötigt einen Namensdienst für die initiale Bezeichnung des entfernten Objektes / Servers.
Beiden Methoden gemein ist der Nachteil, dass der aufrufenden Seite viele Details der entfernten Methode bekannt sein müssen. So ist z.B. der Austausch von Implementierungsdetails wie Methodensignaturen, die Festlegung von Übergabe- und Rückgabeparametern, teilweise auch programmiersprachenabhängiger Details, notwendig.
Des weiteren bringen sowohl RPC als auch RMI keine Methoden mit, um effizient Rechenlast in größeren Netzwerken verteilen zu können.

## 3.3 CORBA

Mit CORBA, der Common Object Request Broker Architecture, lassen sich einige der Nachteile von RPC und RMI vermeiden. Die von der Open Management Group entwickelte Software agiert als eine von Plattform, Betriebssystem und Programmiersprache unabhängige Middleware und definiert Protokolle und Dienste für den Zugriff auf entfernte Ressourcen.

Mit IDL steht eine einheitliche Sprache zur Verfügung, um Objekte und Schnittstellen zu beschreiben; eine weiter abstrahierende Lösung ergibt sich mit DII, bei dem auch unbekannte Objekte angesprochen werden können.

## 3.4 WebServices / SOAP / WSDL / UDDI

Mit WebServices ist es möglich, verteilte Anwendungen - bzw. in größerem Maße Server-Client-Architekturen - weiter zu abstrahieren und voneinander unabhängig zu machen. Ein WebService läuft typischerweise in einem Container (bekanntester Vertreter: Apache Tomcat), der die Laufzeitumgebung bereitstellt und sich um alle Kommunikation kümmert.
Vorteile ergeben sich durch das Konzept, die Services in WSDL zu beschreiben, so dass eine gewisse Unabhängigkeit von Details wie Datentypen erreicht werden kann. Genauere technische Beschreibungen finden sich im Kapitel "Grid Computing - Technologien", da Grid-Services auf dem Konzept der WebServices basieren. Konzepte wie SOAP und UDDI für die Entdeckung und Einbindung von Diensten lassen WebServices für viele Bereiche als beste Alternative dastehen. Hauptsächlich finden sie Einsatz für öffentlich zugängliche Dienste, so bieten z.B. Google oder Amazon bereits Dienste per WebService an.

# 4 Grid Computing

Grid Computing entstand als konsequente Weiterentwicklung der WebServices Architektur. So kann ein GridService auch als erweiterte Fassung eines WebServices verstanden werden; das zu Grunde liegende Prinzip ist das gleiche. Wie ein WebService besitzt der GridService einen eindeutigen URI, unter der der Dienst in Anspruch genommen werden kann. Die Definition erfolgt in gWSDL, einer temporäre Erweiterung des WSDL-Standards, um gridspezifische Elemente benutzen zu können.

Der Vorteil des Grids gegenüber klassischen WebServices liegt unter anderem in seiner Flexibilität. So können nicht nur Berechnungen über den Dienst ausgeführt werden, sondern auch Speicherplatz verteilt benutzt und komplexere Anwendungen ausgeführt werden. Auch sieht Grid bereits Möglichkeiten für Dienste wie Broking, Abrechnung von Leistungen u.a. vor - dazu mehr in den folgenden Kapiteln.

## 4.1 Grid Computing Technologien

Grid Computing ist in der Hauptsache eine Vermischung zweier aktueller Internet-Technologien: Verteiltes Rechnen (Distributed Computing) und WebServices.

### Distributed Computing

DC ist eine Technologie, die vor allem durch das Internet vorangetrieben wurde. Berechnungen, die auf einem einzelnen Rechner Jahrzehnte dauern würden, werden in kleine Einheiten „verpackt“, verteilt, auf Clientrechnern berechnet, um schließlich die Ergebnisse zentral wieder zusammenzufügen. Auf diese Weise wird die Berechnungsdauer - je nach Anzahl der Clientrechner - deutlich reduziert.

Die Methode für das bereits erwähnte SETI@HOME und ähnliche Projekte ist es, von einem zentralen Rechner aus die Berechnungspakete über das Internet zu verschicken. Auf dem Client ist eine Anwendung installiert, die diese Pakete in Empfang nimmt, zumeist in der Leerlaufzeit des Computers berechnet, und die Ergebnisse an den Server zurücksendet.

Meist laufen die Client-Programme als Bildschirmschoner, um dem Benutzer das Gefühl zu vermitteln, am Berechnungsvorgang teilzunehmen.

Die Benutzung einer auf der Zielplattform installierten Anwendung ist für Distributed Computing typisch. Über das Netz werden danach nur noch die Datenpakete transportiert, die Berechnung findet ausschließlich lokal statt. Dadurch können auch Modembenutzer problemlos an dem Projekt teilnehmen.

## WebServices

WebServices sind ebenfalls Server-Client-basierte Dienste, die über das Internet oder Intranet angeboten werden. Sie bauen auf XML-basierenden Standards auf und sind inzwischen weit verbreitet. Da GridServices im Grunde eine Erweiterung von WebServices sind, wird hier genauer darauf eingegangen.

WebServices tragen dem Bedarf einer verteilten Rechnerinfrastruktur Rechnung, indem sie eine Verbindung zwischen verschiedenen Anwendungen, Programmiersprachen und Plattformen bieten. Im Fahrtwasser des Begriffes tauchen Abkürzungen wie WSDL, SOAP und UDDI auf, die teilweise im Zuge dieser Studienarbeit noch näher erklärt werden.

WebServices steht für eine dienstebasierende Infrastruktur, in der die Partner (der Anbieter / Dienstleister und der Benutzer) einander bekannt sind bzw. gemacht werden, und auf diese Weise der Dienst benutzt werden kann. Vorraussetzung für den Dienst ist eine Beschreibbarkeit in der auf XML basierenden WebService Description Language WSDL, die den Dienst, seine Ein- und Ausgaben etc. definiert. Auch die Service Requests und Service Responses sind im XMLFormat gehalten.

Die Grundlage des Services ist ein eindeutiger URI, über den dieser Service erreichbar ist. Die WSDL definiert Zugangsmechanismen und den Ort (URI) der Ressource, so dass eine spezielle Middleware zunächst unnötig wird. Auch Übergabe- und Rückgabeparameter werden in WSDL definiert. Syntaxbeispiele sind im Abschnitt über gWSDL, der auf Grid Computing angepassten Variante von WSDL, enthalten.

Um eine größtmögliche Abstraktion der verwendeten Programmiersprachen zu erhalten, wird oft SOAP eingesetzt, das Simple Object Access Protocol. Der XML-basierte Standard definiert einfache und strukturierte Mechanismen, um über das HTTP-Protokoll auf Objekte zuzugreifen. Dabei kann SOAP in Systemen unterschiedlicher Komplexität, vom einfachen Nachrichtensystem bis zum RPC-System eingesetzt werden. Auf Details dieses Standards soll hier nicht weiter eingegangen werden.

## Grid Computing im Zusammenhang von Distributed Computing und Webservices

Mit Grid Computing werden die Stärken von Distributed Computing und WebServices vereinigt, wohingegen vor allem die Schwächen von DC entfallen. Im einzelnen:

### Universal

Im Gegensatz zu DC können nicht nur Berechnungen ausgeführt werden, sondern z.B. auch Daten dezentral gespeichert werden.

### Kein zentraler Server

In einem Grid kann jeder Rechner Client und Server in einem sein. Wenn Kapazitäten frei sind, stehen diese dem Grid zur Verfügung; werden dagegen welche benötigt, werden sie durch das Grid bereitgestellt.

### Weniger Beschränkungungen

In der typischen Distributed Computing - Umgebung steht exakt ein Programm auf allen Clients zur Verfügung, das genutzt werden kann. Für eine Programmänderung muss das neue Programm von den Clientbenutzern installiert werden. Im Grid sind universelle Schnittstellen vorhanden, durch die ein Programm zur Ausführung auf den Client geladen werden kann, ohne dass der Clientbenutzer etwas ausführen muss. Eine solche Offenheit benötigt natürlich weiterreichende Sicherheitsmechanismen, als sie im verteilten Rechnen notwendig (und vorhanden) sind.

### Definitionssyntax

„A Grid service instance is a (potentially transient) service that conforms to a set of conventions, expressed as Web Service Definition Language (WSDL) interfaces, extensions, and behaviors, for such purposes as lifetime management, discovery of characteristics, and notification. “(Global Grid Forum, 2003)

Gridservices basieren sowohl auf einer Einschränkung der WSDL (durch die vorgegeben Interfaces), als auch auf einer Erweiterung, die in die laufende Entwicklung der WSDL einfließt und in der nächsten Version 1.2 enthalten sein soll. Die OGSI spezifiziert daher die

sogenannte „gwsdl extension“, die WSDL u.a. um das Konzept der „stateful Webservices“ erweitert, und ein komplettes Interface für die Definition des kompletten Gridservice, inkl. Zeit-, Namens- und Fehlerdefinitionen, bietet.

## 4.2 Grid Computing im Konzept der „Virtuellen Organisation“

Unter einer virtuellen Organisation versteht man die Vereinigung von Unternehmensteilen zu einem Ganzen, die nicht an einen festen Ort gebunden sind. So sind in der klassischen VO die Mitarbeiter z.B. im Außendienst über die gesamte Welt verstreut, arbeiten aber im Unternehmensnetz über Technologien wie Email, Groupware, Videokonferenzen etc. wie in einem klassischen Unternehmen zusammen. In dieses Konzept passt Grid Computing durch Mechanismen, die die Integrierung verschiedener Standortkonzepte zu einem virtuellen Unternehmen vereinfachen. So werden z.B. Benutzer des lokalen Standortes über ein Mapping auf Grid-Benutzer abgebildet, so dass es für das Grid egal ist, wie die wirkliche lokale Benutzerstruktur aussieht. Ein Beispiel-Eintrag im grid.mapfile könnte so aussehen:

"/C=US/O=Globus/O=State University/CN=Joe User"juser

Dadurch wird der lokale Benutzer ’juser’ auf den Grid-Benutzer ’Joe User’ abgebildet, der über sein Zertifikat, die Zuordnung zu seiner Organisationseinheit etc. eindeutig identifiziert ist. Details zu dieser Form der Abbildung findet sich im Kapitel zur Sicherheitsstruktur des Grids.

Ein weiterer Grund, die Unterstützung für Virtuelle Organisationen auzubauen, findet sich in dem langsam steigenden Bedarf für unternehmensübergreifende Lösungen und Dienste. So kann überschüssige Kapazität nicht nur innerhalb des Unternehmens verkauft werden, sondern auch gewinnbringend nach aussen weitergegeben werden. Das Globus Toolkit bietet für diese Zwecke unter anderem das RessourceBroker-Konzept, mit dem Ressourcen gehandelt werden können.

## 4.3 Grid-Software

Gridnetzwerke benötigen eine Service-Software, die auf allen beteiligten Rechnern im Netzwerk laufen muss, die sogenannte Middleware.

Der bekannteste Vertreter ist das „Globus Toolkit“, ein OpenSource-Projekt, dass von einigen Regierungsorganisationen wie der amerikanischen DARPA und dem UK e-Science

Programm; sowie von Großfirmen wie IBM und Microsoft unterstützt wird. Dieses wird vor allem im US-Raum eingesetzt.

Im deutschen und europäischen Raum setzt man inzwischen auch auf die UNICORE-Software, die von hauptsächlich von deutschen Supercomputerzentren und der inzwischen teilweise von Intel übernommenen Pallas GmbH entwickelt wurde, und ebenfalls unter einer OpenSource-Lizenz steht. Diese Software wurde auch im Rahmen des EUROGRID-Projektes eingesetzt.

Weitere bestehende Middleware-Komponenten sind beispielsweise CONDOR (Condor Project, 2005), GridSphere (GridSphere, 2005), Sun N1 Engine (Sun Microsystems, 2005), Fraunhofer Resource Grid (Fraunhofer Institut, 2005) u.v.a.

# 5 Anwendungsfall

Gegeben sei eine Firma im technischen Umfeld, beispielsweise Maschinen- oder Fahrzeugbau, die über ein typisches Firmennetzwerk (Standard-PC-Arbeitsplätze wie CAD-, Office- und ähnliche, mehrere Server) verfügt, gleichzeitig aber für größere Berechnungen und Simulationen sowohl große Rechenleistung benötigt, als auch dadurch entstehende große Datenmengen effizient speichern muss.

**Konkrete Daten:**

**Infrastruktur:**

mehrere hundert PC - Arbeitsplätze
File-, Mail-, Webserver
100 MBit-Netzwerk

**Anwendungen:**

Arbeitsplätze: Office, CAD, etc.
Benötigt:
Berechnungen: Tagesrechenleistung mehrerer (z.B. 10) Prozessoren je Tag
Speicher: Durch die Berechnungen entsteht ein Datenaufkommen von mehreren Gigabyte je Tag

**Lösungsmöglichkeiten:**

1. **Großcomputer**

   Eine der Standardlösungen für diesen Anwendungsfall ist die Anschaffung einer oder mehrerer Großrechner, z.B. mit 16 oder 32 Prozessoren. Die Kosten für diese Lösung liegen im fünfstelligen Bereich. Sollte der Bedarf an Rechenleistung in Zukunft steigen, ist eine Skalierung kaum möglich bzw. wiederum sehr kostspielig. für den Bereich

Datenspeicherung wäre bei dieser Lösung die Anschaffung eines Raid-Systems mit der Möglichkeit, Daten auf externe Speichermedien, wie Tape oder DVD, zu sichern, notwendig. Auch hier entstehen hohe Kosten durch die notwendigen Speichermedien.

2. **Verteiltes Rechnen (Distributed Computing)**

   Ein Ansatz, um die Nachteile der ersten Lösung zu vermeiden, ist die Verteilung der Berechnungen auf mehrere Rechner.

   a) Dedizierte Server

      Bei der Anschaffung mehrere Server, die in einem Netzwerk zusammenarbeiten und die Jobs unter sich aufteilen, entstehen ebenfalls hohe Kosten (Hard-, evtl. Software). Gegeben wäre im Gegensatz zu Lösung 1) die leichtere Skalierbarkeit, da in einem DC-Netzwerk das Hinzufügen neuer Server relativ problemlos möglich ist.

   b) Nutzung der vorhandenen Arbeitsplatz-PC als DC-Clients

      Statt der Anschaffung mehrere Server besteht die Möglichkeit, die Rechenleistung auf die bereits vorhandenen Arbeitsplatz-PC zu verteilen. Nötig ist dann eine verteilte Anwendung, die auf diesen Clients laufen kann, zu berechnende Jobs vom Server anfordert, und die Ergebnisse wieder an diesen zurückliefert. Der entscheidende Vorteil gegenüber den bisherigen Lösungen ist der Wegfall der Notwendigkeit, große Summen in neue Hardware zu investieren. Möglich wird diese Lösung - wie auch die folgenden - durch die Tatsache, dass ein moderner PC mit Standardaufgaben (Office, CAD, Buchhaltung etc.) nicht annähernd ausgelastet ist, so dass viel freie Prozessorzeit verbleibt. Der DC-Prozess wird im Standardfall im Hintergrund laufen, und statt des Leerlaufprozesses einspringen, wenn der PC nicht ausgelastet ist.

   Beiden Varianten - 2a) und 2b) - ist gemein, dass sie die Probleme der Skalierbarkeit, sowie der Speicherung der Daten nicht zufriedenstellend lösen können. Bei dem Einsatz dedizierter Server ist eine Aufstockung der Rechenleistung mit der Anschaffung neuer Hardware verbunden, 2b) ist durch die Anzahl der in der Firma verfügbaren Clients begrenzt. Eine Auslagerung der Berechnungen auf Rechner außerhalb des Firmennetzwerkes ist im Fall von DC schwierig, da die benötigte Client-Software auf den externen PCs installiert werden müsste, und viele Rahmenbedingungen wie z.B. Abrechnung oder Sicherheit aufwändig geklärt werden müssen. Des weiteren ist es - in Abhängigkeit von der eingesetzten Software für die verteilte Berechnung - meist schwierig, die Art der zu verarbeitenden Jobs zu ändern, ohne Änderungen an der Software selbst vorzunehmen. Eine Veränderung in den Anforderungen macht demnach Veränderungen der Software auf den Clients notwendig, was großen Aufwand erzeugen kann.

Schließlich müsste die Speicherung der Daten, wie im ersten Lösungsansatz, an zentralisierten Punkten erfolgen.

3. **Grid Computing**

   Durch den Einsatz von Grid Technologie könnte man viele der in den ersten beiden Lösungen aufgeworfenen Probleme umgehen. Wie bei Distributed Computing ist die Anschaffung neuer Hardware nicht notwendig, da die bestehenden PCs für die Berechnungen und - im Gegensatz zu DC - auch für die Speicherung von Daten genutzt werden können. Durch die Variabilität der GridServices ist es möglich, diese Speicherung ebenfalls dezentral zu organisieren.
   Ein weiterer Vorteil der Grid Technologie ist an dieser Stelle die Unabhängigkeit im Bezug sowohl auf das Betriebssystem, als auch auf die eingesetzte Software der Clients. Notwendig ist es, einmalig die Middleware - wie z.B. das später noch genauer betrachtete Globus Toolkit - auf allen Clients zu installieren. Durch das Grid sind dann Möglichkeiten geboten, die für die Berechnung benötigte Software jeweils automatisch auf den Client zu laden. Eine Änderung in den Berechnungen erzeugt auf diese Weise kaum bis keinen Aufwand in der Administration der Clients. Die Skalierbarkeit ist in zwei Stufen gewährleistet. Innerhalb des firmeninternen Netzwerkes werden die Clients ausgelastet, soweit Rechenleistung und Speicherplatz zur Verfügung stehen. Sollten die Anforderungen über die eigenen Kapazitäten hinauswachsen, bieten die Spezifikationen und die Middleware Möglichkeiten, über Grid Broker Ressourcen einzukaufen, ohne dass wie beim Distributed Computing großer Aufwand durch Softwareinstallationen, Abrechnungen etc. entsteht. Viele der so benötigten Funktionen für eine Einbindung externer Ressourcen sind bereits vorgesehen.

   Weitere Vorteile bringen bestimmte Eigenschaften des Grid Computings mit sich. So ist es hier für einzelne Clients, die mehr Ressourcen benötigen, als lokal verfügbar, möglich, innerhalb des Grids Aufträge weiterzugeben, ohne die Dienste eines zentralen Servers in Anspruch nehmen zu müssen. Im Grid kann jeder Rechner sowohl als Client, als auch als Auftraggeber auftreten, natürlich nur in einem vom Adminstrator vorgesehenen Umfang.
   Eine weitere hervorgehobene Eigenschaft des Grid Computing ist das Konzept der virtuellen Organisation. So können verschiedene Unternehmen Verträge abschließen, die eine gegenseitige Nutzung von Ressourcen ermöglichen, um zum Beispiel Lastspitzen auszugleichen, oder dauerhaft Überkapazitäten zu handeln. Auch Lösungen wie ein „Marktplatz“ bzw. eine „Börse“ für Ressourcen sind bereits angedacht und in Umsetzung.

   Zu beachten bleibt, dass noch nicht alle Elemente, die Grid Computing ausmachen sollen, bereits fertig entwickelt sind. Gerade im Globus Toolkit bleibt an einigen Stel-

len - wie z.B. Ressource Broking - noch viel zu tun, bis praktikable und einsetzbare Lösungen vorliegen.

# 6 Grid am Beispiel Globus Toolkit

## 6.1 Aufbau des GT3

Das Globus Toolkit agiert als Middleware in einem Grid-Netzwerk. Es bietet die Infrastruktur, um GridServices zu installieren, den Zugriff auf Ressourcen zu koordinieren, eine Sicherheitsstruktur und spezielle Protokolle wie GridFTP.

Das GT3 hat in weiten Teilen den Status einer Referenzimplementierung für eine solche Grid-Middleware und orientiert sich streng an den Standards der OGSA und OGSI. Das Projekt wird unterstützt von vielen, vorrangig US-amerikanischen, Institutionen und Firmen.

Mit dem GT wird eine größtenteils plattformunabhängige OpenSource-Lösung geboten, mit der sich sowohl lokale Grids, als auch institutionsübergreifende Netzwerke (Virtual Organizations) steuern lassen.

Aufgrund seines Status und der Verbreitung haben wir uns entschieden, das Globus Toolkit für unseren Praxistest zu benutzen.

### Die Komponenten

- Core
- GSI (Grid Security Infrastructure)
- CAS (Community Authentication Service)
- GridFTP
- RFT (Reliable File Transfer)
- XIO (eXtensible Input/Output Library: TCP, UDP, HTTP, ...)
- etc.

## Core

Die Core-Komponente umfasst eine Implementierung der OGSI, die Sicherheitsstruktur, „System Level Services“ und den „Grid Service Container“, der die Laufzeitumgebung für die anwendungsspezifischen Services bietet. Das folgende Bild (6.1) verdeutlicht den Aufbau (Core-Komponenten sind weiß):

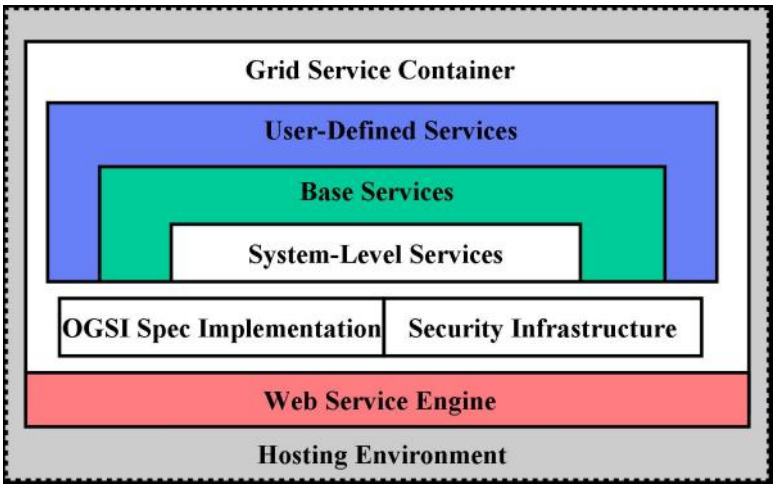


Abbildung 6.1: Beschreibung: GT3: Komponenten

Im folgenden wird auf einige der Komponenten genauer eingegangen.

### Sicheres Grid: Grid Security Infrastructure

Sicherheit ist eines der Hauptthemen des GridComputing. Zum einen ist es wichtig, das Grid gegen unberechtigte Beeinflussung von außen abzusichern, zum zweiten müssen die Parteien innerhalb des Grids eine Vertrauensbasis haben, um Dienste wie Ressource-Broker und Bezahlung für Ressourcen wie Rechenzeit sicher abwickeln zu können. Die verantwortlichen Teile sind zusammengefasst in der „Grid Security Infrastructure“ GSI.

Die Schlüsselkonzepte der GSI sind X.509 Zertifikate, TLS/SSL für Authentifikation und Nachrichtensicherheit, und X.509 Proxy-Zertifikate für Delegation und „single sign-on“. Auf die Vor- und Nachteile der einzelnen verwendeten Techniken soll in dieser Studienarbeit nicht weiter eingegangen werden.

Hervorgehobene Eigenschaften der aktuellen GSI-Version (GSI 3 im Globus Toolkit 3):

- Sicherheitsinfrastruktur für GridServices mit den GSI3-Bibliotheken, die u.a. Mechanismen für Authentifizierung, Authorisierung, Delegierung, Nachrichtenintegrität und Verschlüsselung bieten

- Keine priviligierten Services: kein einzelner Dienst im Globus Toolkit benötigt priviligierte Rechte („root“ -Rechte), der priviligierte Code ist in zwei sehr stark abgesicherten setuid-Programmen gekapselt
- Verwendeten Technologien basieren auf offenen Standards u.a. der IETF, W3C und des Global Grid Forum.
- Gridmap-Security: ein sogenanntes „Mapfile“ wird benutzt, um lokale Benutzer auf ihre Identität im Grid - das X.509 Zertifikat - abzubilden. Dies trägt sehr dazu bei, dass die Verbindung ansonsten vollkommen autarker Netzwerke zu einem Grid erleichtert wird.

Mehr Informationen zur Umsetzung der Sicherheitsinfrastruktur lassen sich in der Beschreibung des praktischen Teils dieser Studienarbeit finden.

## 6.2 Alternativen zum GT3

### Eurogrid / UNICORE

Das Eurogrid Projekt war ein Forschungsprojekt der Europäischen Kommission, um den Einsatz von Grid-Netzwerken in wissenschaftlichen und kommerziellen Gemeinschaften zu evaluieren. Als Grundlage wurde die UNICORE-Software verwendet, die von der Firma Pallas entwickelt, und inzwischen von Intel übernommen wurde. Leider sind weder zum Projekt, noch zu der UNICORE-Software im Augenblick weitergehende Informationen zu finden.

### Condor/Grid

Condor und Condor-G sind Systeme, deren Funktionsweise der des Globus-Toolkits ähnelt, und teilweise auch in der Lage sind, mit diesem zusammenzuarbeiten. Condor beschreibt sich selbst als „specialized workload management software“. Durch das zusammenschließen von Condor-Maschinen kann ein Grid aufgebaut werden, in dem Jobs verteilt berechnet werden; die Software spezialisiert sich dabei auf das Job Management. Während Condor die Middleware an sich beschreibt (die Software, die zum ausführen von verteilten Programmen benötigt wird), ist Condor-G ausgelagerte Job Management Komponente des Condor-Systems, die sowohl einen Condor-, als auch einen Globus-Pool von Rechnern mit Jobs versorgen kann, und dabei über mehr Möglichkeiten als der JobManager des Globus Toolkits bietet.

# 7 Praktische Umsetzung

## 7.1 Installation des Globus Toolkit

### Laborumgebung

Das benutzte Betriebssystem (Fedora Linux Core 2) wurde auf USB-Festplatten installiert. Eine Beschreibung der notwendigen Schritte für die Installation befindet sich im Anhang.

### Benötigte Programme

1. Java SDK >= 1.3.1; empfohlen: 1.4.x
2. Ant, empfohlen 1.6.1
3. JUnit, empfohlen 3.8.1
4. Ein Compiler-Compiler, z.B. Bison >= 1.875c
5. GNU Tar >= 1.12.25

Optional:

1. Jakarta Tomcat, empfohlen 4.1.24
2. Microsoft .NET Framework (nur Windows)
3. JDBC-kompatible Datenbank, z.B. postgres

### Installation

(The Globus Alliance, 2004)

Wir haben uns bei der Installation und Konfiguration eng an der Anleitung auf http://www-unix.globus.org/toolkit/docs/3.2/installation/ orientiert. An einigen Stellen weichen wir aber davon ab, da einige Probleme aufgetreten sind.

### Vorbereitungen

In unserem Beispielsystem gibt es eine Partition /grid, die für alle Programme rund um das Globus Toolkit benutzt wurde. Als Prerequisite wurden Java SDK 1.4.1_02, Ant 1.6.1 und JUnit 3.8.1 installiert, JACC und tar waren bereits auf dem System vorhanden.

Der Installationspfad für unsere Globus-Installation ist /grid/globus

Wir legen, wie empfohlen wird, zwei weitere Benutzer an:
globus - Administrativer Account - unter diesem Benutzer wird der Globus-Container laufen
grid - ein Standardbenutzer, der Grid-Dienste benutzen wird

### Installation des Toolkits

Für die Installation des Toolkits werden einige Umgebungsvariablen benötigt. Da diese auch im weiterem Verlauf der Konfiguration und Benutzung hin und wieder nötig sind, haben wir sie in der /etc/bashrc - Datei gesetzt, um sie dauerhaft im System zu haben:

```
export ANT_HOME="/grid/ant"
export JUNIT_HOME="/grid/junit"
export JAVA_HOME="/usr/local/java/jdk"
export CLASSPATH="$ANT_HOME/lib:$JUNIT_HOME/junit.jar:."
export PATH="$PATH:$ANT_HOME/bin:$JAVA_HOME/bin"
```

Auf der Toolkit-Seite (The Globus Alliance, 2005) gibt es mehrere Installationsmöglichkeiten, wie z.B. vorbereitete binary packages, und den Sourceinstaller. Wir entschieden uns für den Sourceinstaller, da es für unsere Distribution kein passendes fertiges Paket gab. Im Source-Paket enthalten sind zwei Shell-Scripte, die den Hauptteil der Arbeit übernehmen. Für die Grundinstallation genügt es daher, nach Download und Entpacken des Archives einzugeben:

./install-gt3 /grid/globus
./install-gt3-mmjfs /grid/globus

Die Grundinstallation ist damit bereits abgeschlossen.

## Konfiguration des Toolkits

Für Konfiguration und Benutzung werden zwei weitere Einträge in der /etc/bashrc benötigt:

```
export GLOBUS_LOCATION="/grid/globus"
source "$GLOBUS_LOCATION/etc/globus-user-env.sh"
```

Das globus-user-env.sh - Script setzt weitere Umgebungsvariablen. Diese werden hauptsächlich von dem Benutzer des Grids benötigt.

Als nächstes wird eine CA benötigt, um die Benutzerverwaltung einrichten zu können. Der empfohlene Weg ist es, auf eine bereits bestehende CA zurückzugreifen, was in den meisten Unternehmen sicher machbar ist. Als Alternative gibt es globus-simple-CA, ein Wrapper um das openssl-Package; was für unsere Zwecke genügt.

### Einrichten der CA

Vor den folgenden Eingaben steht jeweils der Benutzer, der sie ausführen muss. Manche Befehle müssen durch den Administrator des Grids auf diesem Rechner - der User globus - erfolgen, einige erfordern root-Rechte.

(globus) $GLOBUS_LOCATION/setup/globus/setup-simple-ca

Dieses Script führt durch das Erzeugen einer CA, inklusive Eingabe einer Passphrase für den CA-Schüssel, Generierung der Zertifikate etc.

In dem Verzeichnis /home/globus/.globus/simpleCA befinden sich dann unter anderem das private und das öffentliche Zertifikat der CA.

(root) $GLOBUS_LOCATION/setup/globus_simple_ca_CA_Hash_setup/setup-gsi-default

Dies schließt die Initialisierung der Globus-Security-Komponente ab.

Der nächste Schritt ist die Anforderung und Erzeugung des host key, also eines Schlüssels für den aktuellen Rechner.

### Host-Zertifikat erstellen

(root) grid-cert-request -host ’grid2’ -force

Der Parameter -force war bei uns notwendig, da der Befehl sonst in einer Fehlermeldung endete. Damit wird eine Anforderung auf einen Host key gestellt, der dann vom Besitzer der CA (in unserem Fall globus) signiert wird.

(globus) grid-ca-sign -in hostcert_request.pem -out hostsigned.pem

Das entstandene Zertifikat (hostsigned.pem) wird von root in das Verzeichnis /etc/grid-security gelegt.

### User-Zertifikat erstellen

Durch den Benutzer, der das Grid letzendlich benutzen soll - in unserem Fall grid - wird ein Benutzerzertifikat angefordert. Das Prinzip ist das gleiche wie bei dem Hostzertifikat:

Anforderung durch den User:

(grid) grid-cert-request

Signierung durch den CA-Besitzer:

(globus) grid-ca-sign -in usercert_request.pem -out signed.pem

Der User legt das Zertifikat in das Verzeichnis /.globus/usercert.pem.

### Test

(grid) grid-proxy-init -debug -verify

Wenn die letzte Zeile der Ausgabe der Form „Your proxy is valid until... “ entspricht, war die Generierung der Zertifikate erfolgreich, und alles ist an seinem Platz. Wichtig ist, dass jedes private Zertifikat im Dateisystem nur durch den Benutzer zugreifbar ist, dem es gehört, sonst wird die Initialisierung verweigert.

### Rechte einrichten

Zunächst müssen die Berechtigungen für die ausführbaren Dateien des Globus Toolkit geändert werden. Auch dies übernimmt ein Script:

(root) sh $GLOBUS_LOCATION/bin/setperms.sh

Danach wird eine Abbildung der lokalen Benutzer auf globale Benutzer vorgenommen. Dies ist notwendig, um die Rechte für Benutzer verwalten zu können. Die erforderliche Datei befindet sich in /etc/grid-security/grid-mapfile. Um unseren Benutzer grid dort hinzuzufügen, ist folgender Befehl nötig:

(root) echo „/O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/OU=localdomain /CN=grid“grid > /etc/grid-security/grid-mapfile

Erklärung: Das Subject des Benutzers bzw. seines Zertifikates wird auf den lokalen Benutzer abgebildet. Das Subject erhält man durch den Befehl:

(grid) grid-cert-info -subject

Ausgabe:
/O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/OU=localdomain/CN=grid

Schließlich müssen noch zwei Verzeichnisse zur globalen Library-Suche hinzugefügt werden. Dazu fügt man (als root) die Pfade $GLOBUS_LOCATION/lib und $GLOBUS_LOCATION/libexec zu der Datei /etc/ld.so.conf hinzu, und ruft anschließend ldconfig auf.

#### Abschließender Test

Erzeugen der mitgelieferten Beispiele:

(globus) cd $GLOBUS_LOCATION
(globus) ant samples
(grid) cd $GLOBUS_LOCATION
(grid) grid-proxy-init
(grid) globus-start-container
Hier sollten keine Fehlermeldungen mehr auftreten.

(grid) ant gui -Dservice.port=8080

Als Resultat des letzten Befehls öffnet sich eine grafische Oberfläche mit einigen Beispielen.

## Anmerkungen

Die Installation des Globus Toolkit ist relativ trivial, seine Konfiguration ist es nicht. Es sind einige Fallstricke vorhanden, die herauszufinden uns viel Zeit und Arbeit gekostet hat. An vielen Stellen sind bereits vorgefertigte Scripte vorhanden, die die Administration erleichtern, dennoch sind mehr als grundlegende Betriebssystemkenntnisse und Kenntnisse über die Arbeit mit Zertifikaten - alternativ ein großer Einarbeitungsaufwand - notwendig.

## 7.2 Wie man einen einfachen Grid Dienst schreibt

Für die Entwicklung eines Grid Service braucht es nicht viel. Um einen kleinen Grid Service zu schreiben, braucht man fünf Schritte.

1. Definieren der Interfaces des Services. Das wird mit GWSDL gemacht.
2. Implementieren des Service in Java.
3. Definiten der „deployment“ Parameter. Das wird mit WSDD gemacht.
4. Alles compilieren und in einer GAR Datei zusammenfassen. Das macht Ant.
5. Deploy the Service. Das macht auch Ant.

Wir haben versucht, im Rahmen der Studienarbeit, einen kleinen Grid Service zu implementieren. Als Service haben wir uns für eine Webserver Stress Test entschieden. Mit einem Webserver Stress Test wird ein Webserver oder eine Webanwendung auf Stabilität und Perfomance geprüft. Dabei stellen mehrere Client Anfragen beim Webserver und messen die Zeit bis zur Beantwortung. Dabei gibt es zwei Verfahren. Bei dem einen wird mit ein konstanten Anzahl von Clients zugegriffen. Beim anderen wird die Zahl der Clients kontinuierlich erhöht bis der Webserver nicht mehre mithalten kann. Solche Stress Test sind grade beim Aufbau von kommerziellen Webserver und Webanwendungen wichtig, um die Grenzen des System rechtzeitig zu erkennen und beseitigen. Es gibt ein Menge von kommerziellen und freie Stress Test Tools, ein paar haben wir uns genauer angeschaut:

- MS Web Application Stress Tool (Microsoft, 2005)
- Paessler Webserver Stress Tool (Paessler, 2005)
- Apache HTTP server benchmarking tool (Apache Software Foundation, 2005)

Alle Tools habe unterschiedliche Ausstattung was Optionen und Extras betrifft, aber eins können alle: komplette Webseiten vom einem Webserver abholen und die Zeit messen. Wir wollen im Rahmen der Studienarbeit ein kleinen einfachen Webserver Stress Test in Java implementieren. Dieser Stress Test sollte dann auf einem Grid System als Service installiert werden. Der Service wird über einen Client gesteuert, mit dem am die Parameter an das Grid System übermittelt.

## Schritt 1: Definieren der Interfaces

Der erste Schritt auf dem Weg zum Grid Service ist das Definieren des Interfaces. Mit diesem Interface legt man fest, wie der Grid Service von außen angesprochen wird. In der Grid Service und Web Service Sprache wird ein solches Interface normalerweise *portType* genannt. Ein portType beschreibt also die Schnittstelle eines Webservices, und wird in der Grid Web Server Description Language (GWSDL) definiert. GWSDL ist eine XML-basierte Notation, deren XML-Schema unter http://www.gridforum.org/namespaces/2003/03/gridWSDLExtensions bezogen werden kann. Diese Schema importiert das WSDL Schema und erweitert es. Das Globus Toolkit bietet zwei Möglichkeiten, die benötigte WSDL-Datei zu erzeugen.

1. Man schreibt die GWSDL-Datei per Hand, was den Vorteil hat, daß man die totale Kontrolle über die Beschreibung des Service portType hat. Andererseits ist es nicht sehr benutzerfreundlich.
2. Man lässt sich GWSDL-Datei erzeugen. Als Vorlage zum Erzeugen wird ein Java Interface benutzt.

Wenn man die sich die GWSDL-Datei über ein Java Interface erzeugen lässt, würde das Interface in etwa so aussehen:

```
public interface WebserverStressTool
{
        public void testServer(String url, int count, int threads);

        public long timeMin();
        public long timeMax();
        public long time();
}
```

Leider sind die Java zu GWSDL Umsetzer noch nicht so ausgereift wie die WSDL Umsetzer, so daß man dann doch Änderungen bzw Anpassungen von Hand machen mus. Gerade die Umsetzungen von nicht primitiven Datentypen gehen bei der GWSDL Erzeugung verloren oder scheitern.

### GWSDL

Eine GWSDL Datei ist auch nichts anderes als ein Java Interface Klasse. Im ersten Augenblick sieht diese XML Datei wesentlicher komplexer aus. Sicher braucht man im Vergleich zu einem Java Interface mehr Zeilen Code um das gleiche auszudrücken. Jede Methode des Java Interface besteht im GWSDL aus 3 Teilen:

- Ein Eintrag im <*portType*> Element, entspricht einer Methodensignatur.
- Zwei Einträge des Type <*message*>, der die Parameter für die ankommende und abgehende Nachricht beschreibt.
- Für jedes Parameter Element einer <*message*> gibt es ein Eintrag im <*types*> Element.

Das erste, was in der GWSDL Datei beschrieben wird, ist das <*definitions*> Element.

```
<definitions name="WebserverStressToolService"
  targetNamespace="http://www.globus.org/namespaces/2005/
      WebserverStressTool/WebserverStressToolService"
  xmlns:tns="http://www.globus.org/namespaces/2005/
      WebserverStressTool/WebserverStressToolService"
  xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
  xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/
      gridWSDLExtensions"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Das Element hat zwei Attributte:

- **name**: Der Name der GWDL Datei, dieser Name hat nichts mit dem Namen des PortType zutun.
- **targetNamespace**: Der targetNamespace ist der Namespace der GWSDL Datei. Das bedeutet, dass alles, was in dieser GWSDL Datei definiert oder beschrieben ist, zu diesem Namespace gehört.

In dem <*definitions*> Element werden auch die anderen Namespaces deklariert, die in der Datei benutzt werden. Zu beachten ist, daß *targetNamesace* hier auf das Kürzel *tns* abbildet.

Da das <*portType*> Element eine Erweiterung des OGSI portType ist, müssen wir die ogsi.gwsdl Datei mit folgendem Aufruf importieren.

```
<import location="../../ogsi/ogsi.gwsdl"
    namespace="http://www.gridforum.org/namespaces/2003/03/OGSI" />
```

Den <*portType*> definiert man mit dem <*gswdl:portType*> Tag, wobei des *gswdl* für den benutzen Namespace steht. Den *gwsdl* Namespace wurde am Anfang der Datei festgelegt. Das Element <*portType*> braucht 2 Parameter:

```
<gwsdl:portType name="WebserverStressToolPortType"
    extends="ogsi:GridService">
                .
                .
</gwsdl:portType>
```

- **name**: Der Name des portTypes
- **extends**: Das ist der Name des portTypes, der erweitert wird. Dieses ist eine der Haupterweiterungen von WSDL zu GWSDL. Damit wird es möglich, einen portType als Erweiterung eines anderen portTypes zu definieren. Oder anders gesagt: Man kann von einem anderen portType erben.

Innerhalb des *portType* -Elementes sind alle externen Methoden des Grid Service beschrieben. Jede Methode wir mit einem <*operation*> Element beschrieben.

```
<operation name="time">
        <input message="tns:timeInputMessage"/>
        <output message="tns:timeOutputMessage"/>
        <fault name="Fault" message="ogsi:FaultMessage"/>
</operation>
```

Jedes <*operation*> Elemente besteht aus einem <*input*>, <*output*> und <*fault*> Element. Alle diese Elemente haben ein <*message*> Attribut, welches auf ein Nachricht-Element verweist. So werden die Nachrichten beschrieben, die beim Aufruf (input), erfolgreicher Verarbeitung (output) oder beim einem Fehler (fault) ausgetauscht werden. Die Fault Nachricht wird im OGSI GWSDL definiert, weshalb der vorher importiert wurde. Das heisst, daß für jeder Methode zwei Nachrichtenelemente geschrieben werden müssen.

```
<message name="timeInputMessage">
    <part name="parameters" element="tns:time"/>
</message>

<message name="timeOutputMessage">
    <part name="parameters" element="tns:timeResponse"/>
</message>
```

Jede Nachricht besteht aus 1-n Elemente des Typs <*part*>. Das Element <*part*> beschreibt hier den oder die Übergabeparameter bzw. den oder die Rückgabeparameter einer Operation des Grid Service. Mit dem Attribut *element* wird beschrieben, was für ein Objekt übertragen wird. Beschrieben werden diese Element innerhalb des <*types*> Element.

```
<types>
    .
    .
    <xsd:element name="time">
      <xsd:complexType/>
    </xsd:element>

    <xsd:element name="timeResponse">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="value" type="xsd:long"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    .
    .
</types>
```

Im *<type>* Element werden die Objekte in die einzelnen Bestandteile zerlegt und auf die XML Standttypen abgebildet. Nachrichten, die keinen Übergabewert haben, werden hier mit einem leeren *<complexType>* Element aufgelistet.

#### Namespace Mapping

Der Vorteil von (G)WSDL ist die Sprachunabhängigkeit. Eigentlich ist es egal, in welcher Programmiersprache der Service oder der Client geschrieben wird. Aber das bedeutet nicht nur einen Vorteil, an machen Stellen müssen dann Umwege gegangen werden, um zum Ziel zu kommen, so auch hier. Es müssen die XML-Namespaces auf die Java Paketstruktur abgebildet werden. Mit Hilfe des GlobusToolkit lassen sich aus der GWSDL Datei die so genannten Stubs-Klassen erzeugen. Damit die Stubs-Klassen in dem richtigen Jave Paket landen, brauchen wir eine Mapping-Datei.

```
http\://www.globus.org/namespaces/2005/WebserverStressTool/
    WebserverStressToolService=webserverStressTool.stubs
http\://www.globus.org/namespaces/2005/WebserverStressTool/
    WebserverStressToolService/bindings=webserverStressTool.stubs.
    bindings
http\://www.globus.org/namespaces/2005/WebserverStressTool/
    WebserverStressToolService/service=webserverStressTool.stubs.
    service
```

Der erste Namespace ist der *targetNamespace* der GWSDL Datei. Dieser Namespace wird auf das Java Paket webserverStressTool.stubs abgebildet. Die anderen beiden Namespace werden vom GlobusToolkit automatische erzeut, wenn die GWSDL Datei um die benötigten *bindings* erweitert wird.

### Schritt 2: Implementieren des Service in Java

Nachdem man nun das Interface des Grid Servies definiert hat, ist es Zeit, den Grid Service zu implementieren. Das machen wir in Java. Unser Webservice Streetest besteht aus drei Java Klassen. Die Klasse *WebserverStressToolImpl* implementiert das Interface *WebserverStressToolPortType*, das ist die Stub Klasse, die in Schritt 1 aus der GWSDL Datei erzeugt wurde. Der Service startet 1 bis n Threads der Klasse *StressThread*. Diese Threads führen dann die Anfrage beim Webserver durch und messen die Zeit, die gebraucht wird, um eine komplette Webseite zu empfangen. Der Vorgang wird dann 1 bis n mal wiederholt. Das Ergebnis besteht aus dem min, max und dem Mittelwert der gemessenen Zeiten. Diesen Werte werden in einer Instanz der Klassen *ResultOB* gespeichert.

Die Klasse *WebserverStressToolImpl* erbt von der Klasse *GridServiceImpl*. Dies ist wichtig, denn alle Grid Services müssen von dieser Klasse erben. Die Klasse *GridServiceImpl* ist eine *skeleton* Klasse, denn sie enthält das Grundgerüst (die Basisfunktionalität), auf dem jeder Grid Service aufbaut. Diese Klasse implementiert all die Operationen, die im GridService PortTypt deklariert sind.

Im Vergleich zu einem Webservice behält ein Grid Service seinen internen Status, solange er nicht zerstört wird. Diese Eigenschaft wird *stateful* genannt. Um diese Eigenschaft zu zeigen, gibt es eine Klassenvariablen, die das Ergebnis enthalten.

```
private ResultOB mainROB = new ResultOB();
```

Dann müssen die in den Stubs definierten Methoden nur noch mit Leben gefüllt werden und schon ist der Grid Service fertig. Wenn beim Ausführen des Service ein Fehler auftritt, gibt es die Möglichkeit, eine RemoteException zu entwerfen. Mit dieser Exception kann man dem Client den Fehler und eine Nachricht übermitteln.

```
public void testServer(String hostname, int count, int threads)
   throws RemoteException {
    threadList = new StressThread[threads];
    resultOBList = new ResultOB[threads];

    try {
```

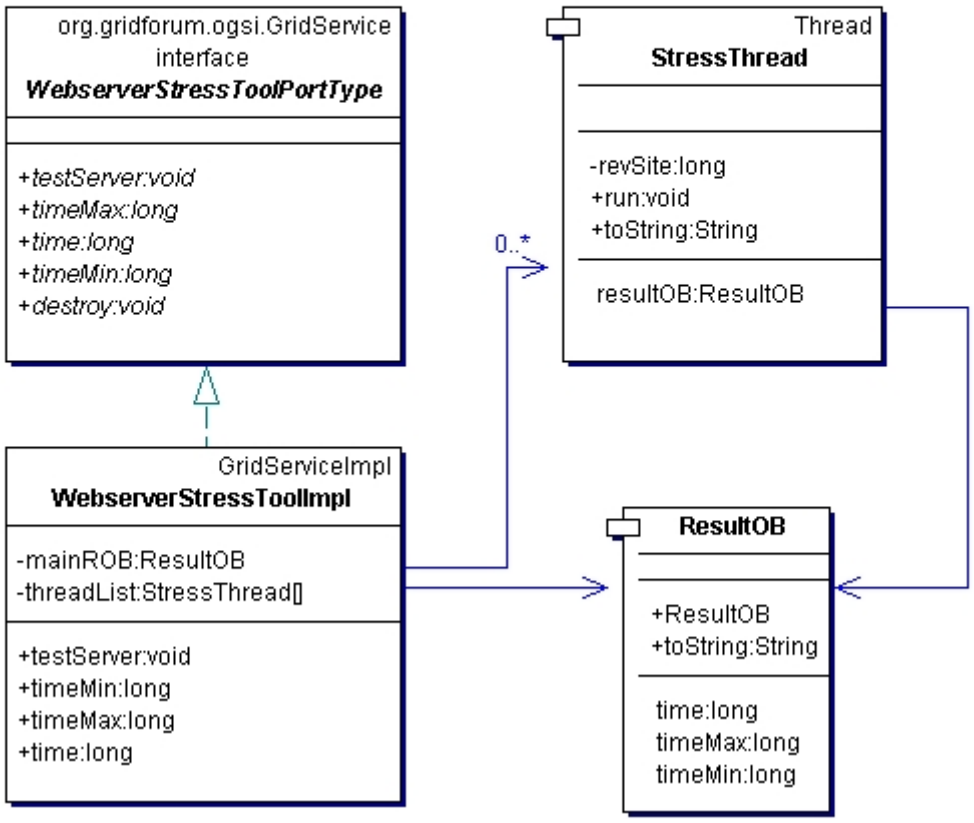


Abbildung 7.1: UML-Diagramm des Grid Service

```
        this.getURL(hostname);
    } catch (MalformedURLException e) {
        throw new RemoteException(e.getMessage());
    }

    this.startThreads(count);

    try {
        this.waitForEnd();
    } catch (InterruptedException e) {
        throw new RemoteException(e.getMessage());
```

```
        }

        System.out.println("Main Thread: All Thread have left the
            system");
        System.out.println("Main Thread: " + mainROB.toString());
    }

    public long timeMin() throws RemoteException {
        return this.mainROB.getTimeMin();
    }

    public long timeMax() throws RemoteException {
        return this.mainROB.getTimeMax();
    }

    public long time() throws RemoteException {
        return this.mainROB.getTime();
    }

    public ResultOB resultOB() throws RemoteException {
        return this.mainROB;
    }
```

Die Methode *testServer()* wird aufgerufen, um den Test zu starten. Die Methode wandelt den übergegeben String hostname in ein URL-Object um. Anschließend wird die eigene Methode *startThreads()* aufgerufen, in dieser Methode werden die einzelnen Threads erzeugt, die dann die Anfragen stellen. Am Ende werden dann mit der Methode *waitForEnd()* die Ergebnisse der einzelen Threads eingesammelt. Mit den Remotemethoden *timeMin()*,*timeMax()* *time()* läst sich dann das Ergebnis abfragen. Statt dieser 3 Methoden hätten wir auch eine Methode schreiben können, die ein Result-Objekt zurück gibt. Dafür hätte man die GWSDL Datei anpassen müssen. Wir haben davon abgesehen, weil es den Zeitrahmen gesprengt hätte. Den Test als einzelne Thread zu starten, ist nur eine Notlösung. Eigentlich war geplant, die Threads über das Grid auf mehrere Computer zu verteilen. Leider ist es uns nicht gelungen herauszufinden, wie man mehrere Computer zu einem Grid zusammensteckt, oder wie man die Software verteilt.

### Schritt 3: Definiten der ”deployment” Parameter

Nachdem wir die Funktionalität des Grid Service definiert (GWSDL) und den Grid implementiert haben, muss der Service noch auf einem Grid Webserver installiert werden. Diese Ver-

fahren wird *deployment* genannt. Für das Verteilen des Gird Service braucht man eine Datei, die alle wichtigen Eckdaten des Grid Service enthält. Aus diesem Deployment Discriptor liest der Grid Webserver, unter welcher Adresse er diesen Grid Service zu veröffentlichten hat. Der Deployment Descriptor ist in WSDD (Web Service Deployment Discriptor) geschrieben. Für unseren WebserverStressTool sieht der Deployment Descriptor so aus:

```
<?xml version="1.0"?>
<deployment name="defaultServerConfig" xmlns="http://xml.apache.org
    /axis/wsdd/"
        xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
            >

        <service name="studienarbeit/WebserverStressTool" provider=
            "Handler" style="wrapped">
                <parameter name="name" value="Webserver␣Stress␣Tool
                    "/>

                <parameter name="className" value="
                    webserverStressTool.stubs.
                    WebserverStressToolPortType"/>
                <parameter name="baseClassName" value="
                    webserverStressTool.impl.WebserverStressToolImpl
                    "/>

                <parameter name="schemaPath" value="schema/gt3ide/
                    WebserverStressToolService/
                    WebserverStressTool_service.wsdl"/>

                <parameter name="allowedMethods" value="*"/>
                <parameter name="persistent" value="true"/>
                <parameter name="handlerClass" value="org.globus.
                    ogsa.handlers.RPCURIProvider"/>
        </service>
</deployment>
```

**Das Element <service>**

```
<service name="studienarbeit/WebserverStressTool" provider="Handler
    " style="wrapped">
```

Das Element <*service*> legt die Adresse fest, unter der der Grid Service zu erreichen sein wird. Zusammen mit der Adresse des Grids erhält man dann den Grid Service Handel (GSH). Läuft der Grid Container zur Beispiel auf der lokalen Maschine, ist die Adresse des WebServiceStressTool: http://localhost:8080/ogsa/services/studienarbeit/WebserverStressTool.

Das Service Element hat unter anderem noch einen Parameter *name*, der die Kurzform des Namens enthält.

**Die Element <ClassName> und <baseClassName>**

```
<parameter name="className" value="webserverStressTool.stubs.
    WebserverStressToolPortType"/>
<parameter name="baseClassName" value="webserverStressTool.impl.
    WebserverStressToolImpl"/>
```

Der Parameter <*className*> zeigt auf den PortType Klasse des Grid Service, während der Parameter <*baseClassName*> auf die implementierte Klasse des GridService zeigt.

**Das Element <SchemaPath>**

```
<parameter name="schemaPath" value="schema/gt3ide/
    WebserverStressToolService/WebserverStressTool_service.wsdl"/>
```

Der <*schemaPath*> gibt an, an welcher Stelle auf dem Server die WSDL Datie liegt. Der Grid Service Container braucht dies WSDL Datei, um Anfragen entgegen zu nehmen und beantworten zu können. Wichtig ist hier, dass es sich nicht direkt um die GWSDL Datei handelt, die wir im ersten Schritt ( 7.2 auf Seite 29) beschrieben haben, sonder um eine WSDL Datei, die aus unser GWSDL Datei erzeugt wird. Der Grund ist, dass das GlobusToolkit an machen Stellen auf schon vorhanden Webservices aufsetzt, wie dem Apache Axis. Dieser versteht nunmal nur WSDL, also wird die Datei konvertiert. Die GWSDL Datei wird aber auch mit an den Grid Server übergeben.

**Die Gemeinsamen Parameter**

```
<parameter name="allowedMethods" value="*"/>
<parameter name="persistent" value="true"/>
<parameter name="handlerClass" value="org.globus.ogsa.handlers.
    RPCURIProvider"/>
```

Diese Parameter geben an, ob der Grid Service persistent ist, und welche Methoden von aussen aufgerufen werden können.

### Schritt 4: Erstellen einer GAR Datei

Nachdem das Interface definiert ist, die Dienste implementiert und die Deployment Paramenter festgelegt sind, muss aus diesen einzelnen Dateien ein Grid Archiv zusammen gefasst werden. Folgende Schritte werden beim Erzeugen eines GAR Archives gemacht:

- Konvertieren des GWSDL Datei nach WSDL
- Erzeugen der *Stubs Klassen* aus dem WSDL
- Kompilieren der *Stubs Klassen*
- Kompilieren der Java Klassen
- zu einen Grid Archive zusammenpacken

Glücklicherweise müssen diese Schritte nicht von Hand gemacht werden, denn dafür gibt es das Tool *Ant*, wie es im Bild 7.2 zu sehen ist. Bei Apache Ant (Apache Software Foundation, 2004) handelt es sich um ein Kommandozeilen-orientiertes Übersetzungstool für Java Projekte. Ant verarbeitet eine XML-basierte Skriptdateien, auch Buildfile genannt. In dieser Datei sind alle Parament für das kompilieren und die Quelledateien festgelegt, weiter können auch noch Kommandozeilen Befehle mit Ant ausgeührt werden.

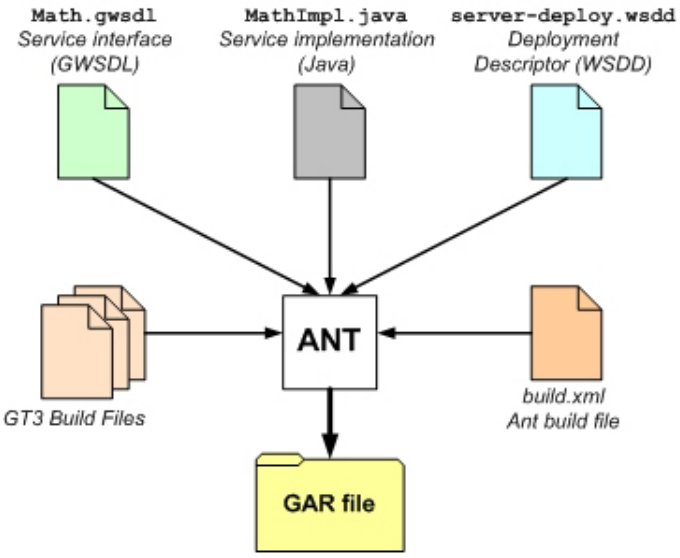


Abbildung 7.2: GAR Datei erstellung mit Ant

Den Buildfile haben wir nicht selber geschrieben, sonden dem Tutorial von (Sotomayor, 2004) entnommen. Um nun die Gar Datei zu erzeugen, muss man Ant noch ein Parameter mitgeben:

- package: Das Java Package, siehe Namespace Mapping ( 7.2 auf Seite 32).
- interface.name: Der Name des Java Interface.
- package.dir: Das Package Verzeichnis.
- schema.dir: Das Verzeichnis, in dem die GWSDL Datei liegt.
- service.name: Der Name des Grid Service
- gar.filename: Name des GAR Files

Um die GAR Datei zu erzeugen, wird Ant mit den folgenden Parameter aufgerufen:

```
ant -Djava.interface=true -Dpackage=webserverStressTool -Dinterface
    .name=WebserverStressTool -Dpackage.dir=webserverStressTool -
    Dservices.namespace=WebserverStressToolService -Dschema.dir=./
    schema/gt3ide/WebserverStressToolService -Dgar.filename=
    webserverStressTool
```

### Schritt 5: Deploy des Grid Service

Das Verteilen des Grid Services ist relativ einfach. Dazu wird das Grid Archiv auf den Computer mit dem Grid Service kopiert. Das Verteilen übernimmt Ant, es packt das Archiv aus und kopiert die Dateien an die richtig Stelle in der Globustoolkit Verzeichnisstruktur. Ant nimmt auch die nötigen Änderungen in den Konfiguationsdatein vor. Das alles wird mit dem folgenden Aufruf gemacht:

Um der Servcie zu deployen:

```
ant deploy -Dgar.name=<Path zum GAR File>
```

Um der Service zu deinstallien:

```
ant undeploy -Dgar.id=WebServerStressTest
```

Wenn man jetzt den Globus Container mit *globus-start-container -p <port>* startet, dann sollte man unter der Ausgabe auch folgendes finden:

```
http://<host>:<port>/ogsa/services/studienarbeit/
  WebserverStressTool
```

Wenn das alles geklappt hat, braucht man nur noch einen Client, der mit dem Grid Service zusammen arbeitet.

## Der Grid Server Client

Der Aufbau der Grid Service Clients ist sehr ähnlich zu einem normalen Webservice Client. Um die Persistenz eines Grid Service zu zeigen, haben wir zwei Clients geschrieben. Einen, der den StressTest ausführt, und einen Client, der das Ergebnis abruft.

### Client1

Der Komplette Client sieht dann so aus:

```
public class ClientSimple{
  public static void main(String[] args){
    try{
      // Uebergabeparameter auslessen
      // 1. Die Adresse des Grid Service
      // 2. Die Adresse des test Servers
      // 3. Anzahl der Threads
      // 4. Anzahl der Versuche
      URL GSH = new java.net.URL(args[0]);
      int thread = Integer.parseInt(args[2]);
      int count = Integer.parseInt(args[3]);

      // Aus der Adress eine Referenz, um die Remotemethoden
         aufzurufen
      WebserverStressToolPortType stress = (new
         WebserverStressToolServiceGridLocator()).
         getWebserverStressToolServicePort(GSH);

      // Ausfuehren der RemoteMethoden
      stress.testServer(args[1],count,thread);

    } catch(Exception e) {
      System.out.println("ERROR!");
```

```
      e.printStackTrace();
    }
  }
}
```

Als erstes werden die Übergabeparameter ausgelesen und in die entsprechend Datentypen umgewandelt. Die Übergabeparameter sind:

1. die Adresse des Grid Service
2. die Adresse des zu testen Webservers
3. Anzahl der Threads
4. Anzahl der Wiederholungen

Nachdem die Übergabeparameter umgewandelt sind, beginnt die eigentliche Arbeit.

```
WebserverStressToolPortType stress = (new
    WebserverStressToolServiceGridLocator()).
    getWebserverStressToolServicePort(GSH);
```

Mit diesem Aufruf wird ein Objekt von unserem GridService erzeugt. An dieses Objekt lassen sich dann die Nachrichten schicken, die wir im PortType festgelegt haben.

```
stress.testServer(args[1],count,thread)
```

Mit diesem Aufruf wird dann der StressTest angestoßen.

#### Client2

Der zweite Client ruft über die Methoden *timeMin*, *timeMax* und *time* das Ergebnis des Stresstests ab und gibt es aus.

```
public class ClientSimple2 {
  public static void main(String[] args) {
    try {
      // Uebergabeparameter auslessen
      // 1. Die Adresse es Grid Service
      URL GSH = new java.net.URL(args[0]);

      // Aus der Adress eine Referenz, um die Remotemethoden
         aufzurufen
```

```
        WebserverStressToolPortType stress = (new
            WebserverStressToolServiceGridLocator()).
            getWebserverStressToolServicePort(GSH);

        // Ausfuehren der RemoteMethoden
        long timeMin = stress.timeMin();
        long timeMax = stress.timeMax();
        long time    = stress.time();
        System.out.println("Durchschnitt:" + time +" min:"+timeMin+"
            max:"+timeMax);

      } catch (Exception e) {
        System.out.println("ERROR!");
        e.printStackTrace();
      }
    }
}
```

Wenn man also den ersten Client folgendermaßsen aufruft,

```
java ClientSimple http://<server>:<port>/ogsa/services/
    studienarbeit/WebserverStressTool http://snooze.zuhause.local 10
     10
```

wird ein StressTest auf den Webserver *snooze.zuhause.local* ausgeführt. Der Test wird mit 10 Wiederholungen und 10 Threads ausgeführt. Nach dem der Client fertig ist, kann man das Ergebnis mit dem ClientSimple2 abrufen.

```
java ClientSimple2 http://<server>:<port>/ogsa/services/
    studienarbeit/WebserverStressTool
Ausgabe:
Durchschnitt:117 min:2 max:3604
```

## Erstellen eines Grid Service mit Eclipse

Da das Entwicklen von Grid Diensten ohne ein IDE mühsam ist, gibt es auch so die ersten Bemühungen, die Grid Service Entwicklung in die IDE Eclipse (Sotomayor u. a., 2004) zu integrieren. Diese Eclipse Plugin ist mit der *Version 0.1 Alpha 2* noch in eimen frühen Entwicklungs Stadium, aber er lässt sich durchaus zur Entwicklung eines Grid Dienstes benutzen. Denn folgende Merkmale sind schon eingebaut:

1. Erzeugen des Gar Archivs.
2. Automatische Generation der WSDD Datei.
3. Automatische Generation des Namespace Mapping.
4. Teilweise Erzeugung der GWSDL Datei.
5. Teilweise Erzeugung von Java Klassen.
6. Teilweise Synchronisation der Java Klassen mit der GWSDL Datei.

In den nächsten Versionen sollen folgende Merkmale erweitert werden:

- Das Globus Sicherheits Framework soll über Dialoge konfigurierbar sein.
- Das Installieren des Grid Service auf dem Grid.
- Debuggen des Grid Service.
- Integration der Globus JavaDocs.
- Volle Synchronistion von den Java Klassen mit der GWSDL Datei.

#### Installtion des Plugins

Der Plugin wird in des Eclipse Plugin Verzeichnis entpackt. Nach dem Start von Eclipse muss der Plugin noch konfiguriert werden. Dazu öffnet man den Preference Dialog vom dem Plugin (Bild 7.3 auf der nächsten Seite), zu finden im Meün *Windows->Preference*. Hier muss der Path zum Globus Toolkit angepasst werden. Alle anderen Werte sind so OK.

#### Erstellen eines neuen Projektes

Beim Erstellen eines neuen Globus Toolkit Projektes erscheint der Dialog 7.4 auf Seite 45.

- **Project Name:** Ist der Projekt Name. Hier WebserverStressTool.
- **Base Package:** Der Name des Basis Paketes, in diesem Paket werden die Klassen, der WSDD und auch das Stub Paket abgelegt.
- **Base Namespace:** Der Base Namespace ist der *target namespace* ( 7.2 auf Seite 29).
- **Name of Interface/PortType:** Aus diesen Namen werden die Namen für den PortType, die Stubs und andere abgeleitet.

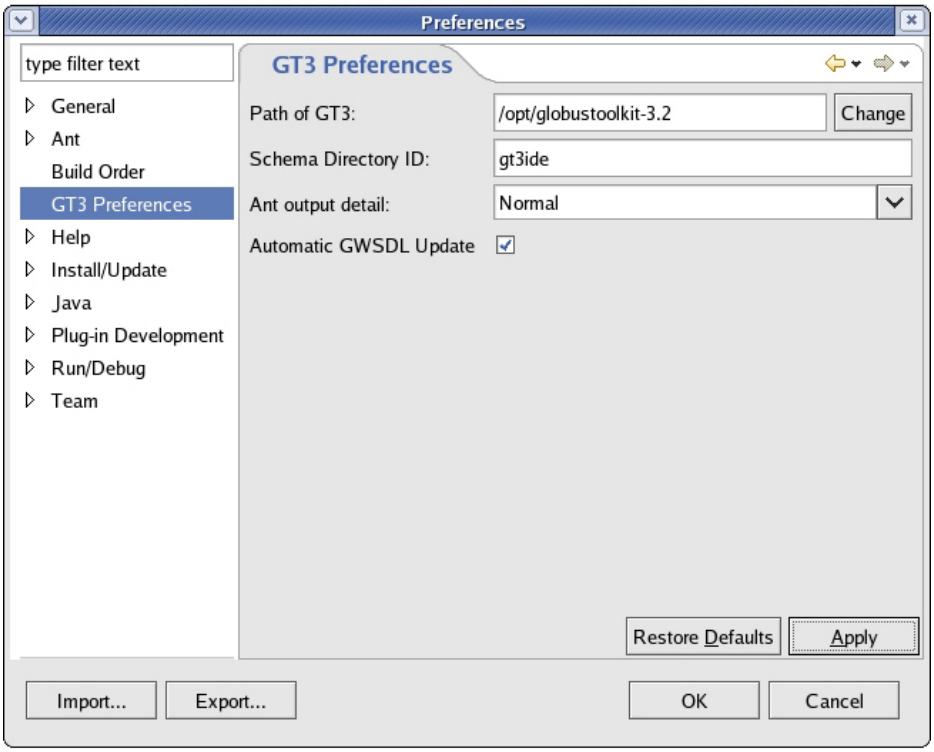


Abbildung 7.3: Die Preferences des Plugins

- **Service GSH:** Die Adresse, unter der der Grid Service veröffentlicht wird. Da alle GSH gleich anfangen (http://localhost:8080/ogsa/services), muss man nur den letzten Teil angeben.
- **Is this a factory service:** „Nein“ wenn man eine persisten Grid Service schreiben will. „Ja“ wenn man einen „factory“ Service schreiben will.
- **Provide implementation skeleton?** Der Plug-in kann die Basis Java generieren, in der dann nur noch die Methoden implementiert werden müssen. Man kann wählen zwischen Java Klassen, die von *GridServiceImpl* erben, oder einer Java Klasse, die sich wie ein Operations Provider verhält, oder keiner generierten Java Klasse.

Nachdem man ein neues Projekt erzeugt hat, erscheint im Package Explorer von Eclipce ein GT3 Projekt (siehe 7.5 auf der nächsten Seite). Wir haben die Klassen, die wir in 7.2 auf Seite 33 erstellt haben, in das Projekt kopiert. Danach mussten wir noch die GWSDL Datein synchronisieren (über das Kontextmeun). Danach wurde mit Hilfe des Plug-ins die Stubs und die GAR Datei erzeugt (siehe 7.6 auf der nächsten Seite und 7.7 auf der nächsten Seite). Danach kann man man die GAR Datei wie in „Schritt 4: Erstellen einer GAR Datei“ (auf 7.2 auf Seite 38) auf dem Grid verteilen.

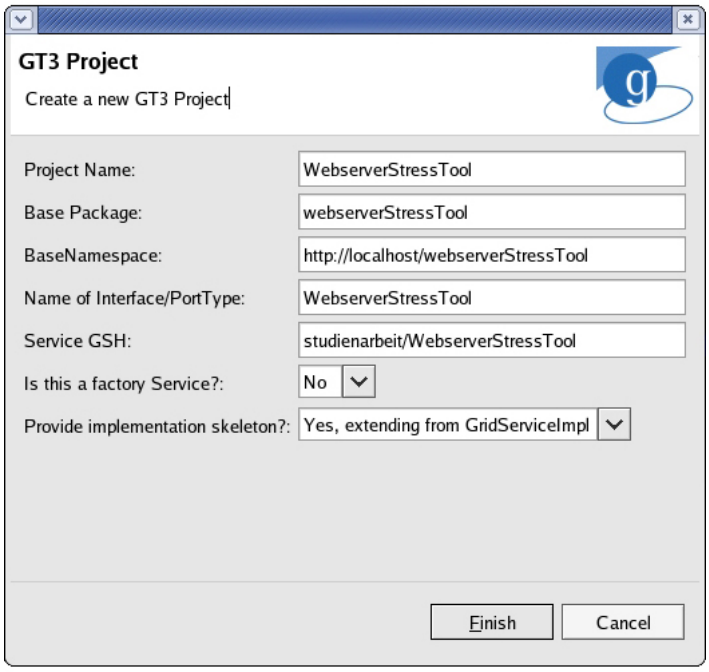


Abbildung 7.4: Erstellung eines neuen Projektes

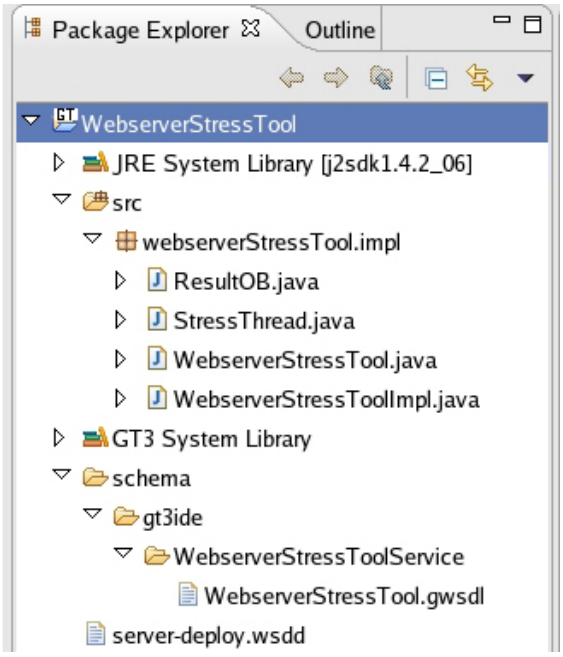


Abbildung 7.5: Das GT3 Projekt

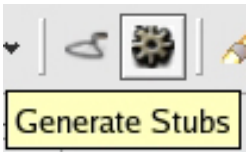


Abbildung 7.6: Erzeugen der Stubs Dateien

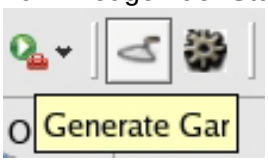


Abbildung 7.7: Erzeugen eine Gar Datei

# 8 Fazit

Wir haben in dieser Studienarbeit sowohl den allgemeinen Stand des Grid Computing, als auch den des Globus Toolkit als Referenzimplementierung im Speziellen untersucht. Unsere Bewertung folgt als Ergebnis dieser Untersuchungen.

Grid Computing befindet sich momentan in einer raschen Entwicklung. Sowohl die grid-eigenen Spezifikationen wie OGSA und OGSI, als auch die zugrundeliegenden Technologien wie WebServices ändern sich schnell und werden ständig weiterentwickelt. Für den professionellen Einsatz in Unternehmen ist diese Entwicklung im Moment eher ein Hinderungsgrund. Die Technologien sind weit davon entfernt, wirklich ausgereift zu sein, und es ist schwer möglich, ohne grossen Eigenaufwand ein vollständig auf die eigenen Bedürfnisse passendes Grid aufzubauen.
Allein in den letzten Wochen sind einige Meldungen veröffentlicht worden, wie z.B. die von Sun etablierte Börse für Rechenzeit (Heise Newsticker, 2005b), (Heise Newsticker, 2005c) oder die Gründung der Organisationen Globus Consortium durch IBM, Intel und andere (Heise Newsticker, 2005a) und MegaGrid u.a. durch Oracle und Intel (Heise Newsticker, 2004).

Ebenso wie Grid Computing selbst befindet sich auch das Globus Toolkit in steter Weiterentwicklung. Im Dezember 2004 war das Release der ersten Alpha-Version des GT 4, wir haben mit der Version 3.2 gearbeitet. Ende April 2005 soll GT 4 offiziell released werden. Viele Komponenten und Schnittstellen ändern sich bei dem Release, so dass eine einfache Migration nicht sicher erscheint.

Wie unsere praktischen Erfahrungen mit dem Globus Toolkit zeigen, ist es noch ein weiter Weg, bis dem Anwender eine einfach zu benutzende Möglichkeit zur Verfügung steht, ein Grid einzusetzen. Sehr viel vermeidbarer Administrations- und Einarbeitungsaufwand war nötig, bis die meisten grundlegenden Services zufriedenstellend liefen, auf viele Punkte wie z.B. die Verteilung eines Service mussten wir aus Zeitgründen dann verzichten. Für den professionellen Einsatz in einer nicht auf Grid Computing spezialisierten Firma gehen wir von einem sehr hohen Aufwand für die Einrichtung, Anpassung und Pflege eines Grid-Systems aus.

Abschließend bleibt zu sagen, dass Grid Computing eine Technologie ist, die man als Alternative zu Mehrprozessor-Systemen im Auge behalten sollte. Ein spezialisiertes Distributed Computing Netz kann in näherer Zukunft durch ein Grid abgelöst werden, Clusterlösungen

können durch Grid um verteilte Speicherung und bessere Auslastung von Clients erweitert werden.
In dem momentanen Zustand ist Grid allerdings nur für Lösungen geeignet, bei denen viel Zeit und Manpower investiert werden kann, bis ein zufriedenstellendes System vorhanden ist.
Von der Vision der Rechenpower aus der Steckdose ist die Realität noch sehr weit entfernt.

# 9 Anhang

## 9.1 Installation von Fedora Core 2 auf einer USB-Festplatte

Fedora Core ist der Nachfolger der Linux-Distribution RedHat Linux. Die aktuelle Version ist Fedora Core 3, dieses Howto bezieht sich auf den Vorgänger, Fedora Core 2.

Bei der Installation der Distribution stießen wir auf das Problem, dass es nicht vorgesehen ist, Fedora auf eine per USB angeschlossene Festplatte zu installieren. Über einige Umwege ist es dennoch möglich, die Installation vorzunehmen, daraus entstand diese Anleitung.

### 1. Lokale Installation

Für die Installation ist es zwingend erforderlich, die Festplatte zunächst in den Rechner einzubauen und anzuschließen - z.B. am IDE oder S-ATA-Bus. Über die Installations-CD wird dann eine Standard-Installation der Distribution vorgenommen. Unsere Partitionierung:

sda1: /
sda2: /grid
sda3: /usr
sda4: – [erweiterte Partition]
sda5: /home

Von der Festplatte lässt sich nach Abschluss der Installation booten. Für die weitere Konfiguration wird die Festplatte nun per USB angeschlossen.

## 2. Konfiguration für den USB-Boot

Schließt man nun die Festplatte per USB an, wird zunächst kein Boot-Sektor gefunden. Daher ist ein Start von der Fedora Rescue CD notwendig (die erste Installations-CD, beim Start „linux rescue “eingeben). Die Suche nach Fedora-Installationen kann abgebrochen werden, danach steht eine Shell zur Verfügung.

Eingabe (bezogen auf unsere Partitionierung)

```
mkdir /bla
mount /dev/sda1 /bla
chroot /bla
mount /dev/sda3 /usr
```

Nun muss das initrd - die Startkonfiguration des Kernels - neu erzeugt werden, um die USB-Treiber gleich zu Beginn des Startvorgangs in das System zu laden:

```
mkinitrd --preload=usb_storage --preload=uhci_hcd --preload=
    ehci_hcd --preload=ext3 --preload=jbd --preload=scsi_mod --
    preload=sd_mod /boot/initrd -`uname -r `.img.new `uname -r `
```

„uname -r“gibt die aktuelle Kernelversion in den Befehl aus, in unserem Fall 2.6.5-1.358.

Sichern des alten, und einsetzen des neuen initrd:

```
mv /boot/initrd -`uname -r `.img /boot/initrd -`uname -r `.img.old
mv /boot/initrd -`uname -r `.img.new /boot/initrd -`uname -r `.img
```

Als Bootloader für das System verwenden wir hier den LiLo Bootloader, da GRUB sich nicht auf der USB-Platte installieren ließ.

```
----------[ /etc/lilo.conf ]----------
prompt
timeout=50
default=grid
boot=/dev/sda
disk=/dev/sda
bios=0x80
lba32

image=/boot/vmlinuz-2.6.5-1.358
        root=/dev/sda1
        label=grid
        initrd=/boot/initrd-2.6.5-1.358.img
```

```
append="rhgb␣root=/dev/sda1"
```

Unsere Kernelversion muss wiederum bei Bedarf ersetzt werden.

Mit einem Neustart ist die Konfiguration abgeschlossen, und das System sollte sich von USB booten lassen.

Für ein Kernelupdate ist es nötig, die Prozedur ab Punkt 2 zu wiederholen.

# Abkürzungsverzeichnis

| | |
|---|---|
| CA | Certification Authority |
| CORBA | Common Object Request Broker Architecture |
| DII | Dynamic Invocation Interface |
| GSH | Grid Service Handle |
| GSR | Grid Service Reference |
| GT3 | The Globus Toolkit Version 3 |
| GWSDL | Grid WebService Description Languare |
| http | HyperText Transver Protocol |
| IDL | Interface Definition Language |
| IETF | Internet Engineering TaskForce |
| OGSA | The Open Grid Service Architecture |
| OGSI | The Open Grid Service Infrastructure |
| RMI | Remote Method Invocation |
| RPC | Remote Procedure Call |
| Soap | Simple Oobject Acces Protocol |
| uddi | Universal Description, Discovery and Integration |
| uri | Uniform Resource Identifier |
| url | Uniform Resource Locator |
| W3C | World Wide Web Consortium |
| wsdd | Webservice Deployment Discriptor |
| wsdl | WebService Description Languare |

# Literaturverzeichnis


[Apache Software Foundation 2004] APACHE SOFTWARE FOUNDATION: *Apache Ant*. 2004. – URL http://ant.apache.org/. – Zugriffsdatum: 2004-12-03 38

[Apache Software Foundation 2005] APACHE SOFTWARE FOUNDATION: *Manual Page: ab*. 2005. – URL http://httpd.apache.org/docs/programs/ab.html. – Zugriffsdatum: 2005-01-24 28

[Condor Project 2005] CONDOR PROJECT: *Condor Project Homepage*. 2005. – URL http://www.cs.wisc.edu/condor/. – Zugriffsdatum: 2005-02-23 15

[Foster und Kesselman 1998] FOSTER, Ian ; KESSELMAN, Carl: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998 6

[Fraunhofer Institut 2005] FRAUNHOFER INSTITUT: *Fraunhofer Resource Grid*. 2005. – URL http://www.fhrg.fhg.de/. – Zugriffsdatum: 2005-02-23 15

[Global Grid Forum 2003] GLOBAL GRID FORUM: *Open Grid Services Infrastructure (OGSI)*. 2003. – URL http://www.ggf.org/documents/GWD-R/GFD-R.015.pdf 13

[GridSphere 2005] GRIDSPHERE: *GridSphere Portal*. 2005. – URL http://www.gridsphere.org/gridsphere/gridsphere. – Zugriffsdatum: 2005-02-23 15

[Heise Newsticker 2004] HEISE NEWSTICKER: *Oracle, Dell, EMC und Intel starten Grid-Computing-Projekt*. 2004. – URL http://www.heise.de/newsticker/meldung/54065. – Zugriffsdatum: 2005-02-22 46

[Heise Newsticker 2005a] HEISE NEWSTICKER: *Neue Organisation zur Förderung von Grid-Computing*. 2005. – URL http://www.heise.de/newsticker/meldung/55466. – Zugriffsdatum: 2005-02-22 46

[Heise Newsticker 2005b] HEISE NEWSTICKER: *Sun macht Ernst mit Grid-Computing*. 2005. – URL http://www.heise.de/newsticker/meldung/55827. – Zugriffsdatum: 2005-02-22 46

[Heise Newsticker 2005c] HEISE NEWSTICKER: *Sun möchte Börse für Rechenzeit etablieren*. 2005. – URL http://www.heise.de/newsticker/meldung/55990. – Zugriffsdatum: 2005-02-22 46

[Microsoft 2005] MICROSOFT: *Web Application Stress Tool*. 2005. – URL http://www.microsoft.com/technet/archive/itsolutions/intranet/downloads/webstres.mspx. – Zugriffsdatum: 2005-01-24 28

[Paessler 2005] PAESSLER: *Web Server performance Software*. 2005. – URL http://www.paessler.com/webstress. – Zugriffsdatum: 2005-01-24 28

[SETI@HOME-Website 2005] SETI@HOME-WEBSITE: *SETI@HOME: Current Total Statistics*. 2005. – URL http://setiathome.ssl.berkeley.edu/totals.html. – Zugriffsdatum: 2005-01-23 9

[Sotomayor 2004] SOTOMAYOR, Borja: *The Globus Toolkit 3 Programmer's Tutorial*. 2004. – URL http://gdp.globus.org/gt3-tutorial/multiplehtml/. – Zugriffsdatum: 2004-03-12 39

[Sotomayor u. a. 2004] SOTOMAYOR, Borja ; LÓPEZ, Marcos ; SÁNCHEZ, Txus: *A Globus Toolkit Plug-in for Eclipse*. 2004. – URL http://gt3ide.sourceforge.net. – Zugriffsdatum: 2004-06-11 42

[Sun Microsystems 2005] SUN MICROSYSTEMS: *N1 Grid Engine 6*. 2005. – URL http://www.sun.com/software/gridware/index.xml. – Zugriffsdatum: 2005-02-23 15

[The Globus Alliance 2004] THE GLOBUS ALLIANCE: *Globus Toolkit 3.2: Installation Guide*. 2004. – URL http://www-unix.globus.org/toolkit/docs/3.2/installation/. – Zugriffsdatum: 2005-01-23 23

[The Globus Alliance 2005] THE GLOBUS ALLIANCE: *The Globus Toolkit*. 2005. – URL http://www-unix.globus.org/toolkit/. – Zugriffsdatum: 2004-12-18 24

[University of British Columbia, Dep. Computer Sciences 2004] UNIVERSITY OF BRITISH COLUMBIA, DEP. COMPUTER SCIENCES: *A Brief History of Distributed Computing*. 2004. – URL http://www.ugrad.cs.ubc.ca/~cs416/X/Notes/01ry/02.history.html. – Zugriffsdatum: 2005-01-23 8