Endbericht: Clouds — Peerbasiertes On-Demand Computing

Projektgruppe 538

Michael Adler, Nikolay Astahov, Florian Blümel,
Tim Dauer, Peter Henschel, Jens Krummnacker,
Marco Löhken, Sebastian Roy, Florian Schulz,
Christopher Sonneborn

31. März 2010

Inhaltsverzeichnis

1 Einleitung

Der vorliegende Bericht stellt die Ergebnisse des Projekts 538 „Clouds — Peerbasiertes On-Demand Computing“ vor, das am Institut für Roboterforschung der Technischen Universität Dortmund im Sommersemester 2009 und Wintersemester 2009/2010 stattfand.

Mit dem Beginn der Projektgruppenarbeiten im April 2009 wurden durch Vorträge während einer Seminarfahrt die grundlegenden Kenntnisse im Bereich der Softwareentwicklung und spezielle Kenntnisse für das Projektziel, hinsichtlich des aktuellen Stands der Technik im Bereich der Delokalisierung computerbasierter Datenhaltung und Verarbeitung und deren Hochverfügbarkeit geschaffen.

Neben der Homogenisierung des Wissenstandes der Projektgruppenteilnehmer durch diese Vorträge konnten auch erste Vorstellungen bezüglich der späteren Realisierung der Projektgruppenziels gewonnen werden. Die Gliederung der Kapitel des folgenden Endberichts folgt überwiegend dem klassischen Life-Cycle für Softwareentwicklung, der sich in der Reihenfolge in die Anforderungsanalyse, Entwurfs-, Modellierungs- und Implementierungsphase chronologisch subsumieren lässt.

Durch die kritische Diskussion und spätere Qualitätssicherung mit Hilfe von Softwaretests konnte die Chronologie der genannten Phasen nicht immer eingehalten werden. Beispielsweise erforderten Schwächen der ersten Entwurfsergebnisse während der folgenden Modellierungsphase die Rückkehr zur Entwurfsphase.

Um die Lesbarkeit dieser Abhandlung zu wahren wird die beschriebene Reihenfolge der Entwicklungsphasen strikt eingehalten. Innerhalb der Kapitel werden jedoch die unterschiedlichen Modelle und Technologieentscheidungen denen sich die Projektgruppe gegenüber sah ergebnisorientiert diskutiert.

1.1 Motivation

Im vergangenen Jahrzehnt ließen sich zwei Trends innerhalb der IT-Branche vor allem für Unternehmen ablesen. Auf der einen Seite zeigt sich, dass der kontinuierlich steigende Bedarf an Rechenleistung und Speicherplatz einen steigenden Kostenfaktor für den Betrieb, die Instandhaltung und die Erweiterung bedeutet. Auf der anderen Seite zeigt sich eine gegenläufige Kostenentwicklung als zweiter Trend: Der konsequente Ausbau der Netzinfrastruktur führt zu immer schnelleren und kostengünstigen Internetverbindungen, die es ermöglichen, auch größte Datenmengen in einem angemessenen Zeitrahmen über große Entfernungen hinweg zu versenden. Als Reaktion auf diese beiden Trends entwickelte sich das Konzept, komplexe Datenverarbeitung und -haltung nicht mehr lokal durchzuführen, sondern die entsprechenden Ressourcen von einem darauf spezialisierten

Dienstleister über schnelle Datentransfers zu beziehen. Unter diesem Gesichtspunkt ist die erst in den letzten Jahren aufgekommene Bezeichnung **Cloud Computing** lediglich eine neue Terminologie für ein bekanntes Konzept, das vielfach auch als der „Zusammenschluss aus SaaS (Software as a Service), PaaS (Platform as a Service) und IaaS (Infrastructure as a Service)“[1] bezeichnet wird.

Der Grundgedanke des Cloud Computings, Hardware zur Verfügung zu stellen, ohne dabei den Umfang oder die gesamte Infrastruktur offen zu legen, soll für den Benutzer folgenden Nutzen bringen:

- Es soll der Eindruck unbegrenzt verfügbarer Rechenleistung entstehen, die nur bei Bedarf abgerufen werden muss.
- Eine unverhältnismäßige Erweiterung der Rechen- bzw. Speicherleistung muss nicht im Voraus durchgeführt werden, wie dies beim Betreiben lokaler Ressourcen unumgänglich ist. Der Benutzer hat damit vielmehr die Möglichkeit, die verfügbare Rechenleistung jederzeit bedarfsgerecht zu erhöhen.
- Kosten für die Nutzung der Rechenleistung bzw. Speichernutzung fallen nur dann an, wenn diese tatsächlich genutzt und benötigt werden.

Eine exemplarische Realisierung der Idee des Cloud Computings findet sich in der kommerziellen Lösung des Unternehmens Amazon mit der Elastic Compute Cloud (EC2). Hierbei greift das Unternehmen auf bestehende, unternehmenseigene Rechenzentren zurück, die bedarfsgerecht zusammengeschlossen werden können. Dabei setzt das Unternehmen auf eigene, nicht öffentliche Standards, was eine Verwaltungsstruktur impliziert, die das dynamische Hinzufügen unternehmensfremder Ressourcen nicht leistet.

Eine vollständige Dezentralisierung und Vermeidung fester Strukturen wäre durch die Realisierung eines Cloud Computing-Systems mittels P2P-Kommunikation gegeben. Hierbei werden die einzelnen Rechenknoten als Peers betrachtet, die für die Kommunikation ein P2P-Netz aufspannen. Wenn jeder Peer in diesem Netz jeden für die Vitalität des Gesamtsystems benötigten Dienst zur Verfügung stellen kann, wäre das Hinzufügen und Entfernen von Ressourcen – Knoten – zum System dynamisch zur Laufzeit möglich.

Der hohe Grad an Dynamik eines P2P-Systems, der durch das Vermeiden zentraler Verwaltungsstrukturen in diesem System erzielt wird, macht es andererseits jedoch notwendig, umfassende Mechanismen zu planen, die die Organisation des Gesamtsystems automatisiert durchführen und abstimmen. Zu untersuchen ist auch, wie in einem durch P2P-Kommunikation realisierten System Aspekte wie Hochverfügbarkeit und Ausfallsicherheit gewährleistet werden können.

Das in dem vorliegenden Bericht beschriebene System stellt eine mögliche Realisierung für eine auf P2P-Kommunikation basierende Compute Cloud dar. Weiterhin liefert das erstellte System eine Grundlage für Machbarkeitsstudien oder Betrachtung anderer Anwendungsszenarien. So wäre vorstellbar, dass nicht mehr ein einzelner Rechner als Peer

[1] UC Berkeley Reliable Adaptive Distributed Systems Laboratory http://radlab.cs.berkeley.edu/ February 10, 2009

des P2P-Netzes betrachtet wird, sondern regional verteilte Rechenzentren mit Rechenclustern die Peers bilden, die bedarfsgerecht dynamisch zu einem gemeinsam arbeitenden Cloud-System verbunden werden können.

1.2 Zielsetzung

Für die grundlegende Zielsetzung der Projektgruppe ist die Ausschreibung der Projektgruppe vom 31. Januar 2009 maßgeblich:

> Ziel der Projektgruppe ist es, eine peerbasierte, verteilte Infrastruktur für die dynamische Bildung von Compute Clouds zu schaffen. In diesem System soll es Teilnehmern möglich sein, sich autonom zu einem Verbund (Cloud) zusammenschließen zu können, innerhalb dessen Benutzer von außen ihre Anwendungen ausführen.

Implementierung einer Middleware für Cloud-Systeme

Nach einer intensiven Analyse der allgemeinen Problemstellung ist als Grundlage für weitere Untersuchungen eine Middleware zu implementieren, die die Grundfunktionen des Systems zur Verfügung stellt. Zu den Grundfunktionen eines Knoten gehört die Kommunikation mit anderen Knoten. Ferner muss jeder Knoten Aufträge ausführen können und Zugriff auf die Anwender- und Konfigurationsdaten haben. Es muss ein Scheduling zur Ausführung von Aufträgen durchgeführt werden, so dass wartende Aufträge gesammelt werden können und auf freie bzw. frei werdende Ressourcen verteilt werden. Eine Benutzerschnittstelle soll dem Anwender eine einfache und intuitive Bedienung ermöglichen, um seine Aufträge in das System einzugeben, zu konfigurieren und anschließend im Cloud-System zur Ausführung bringen.

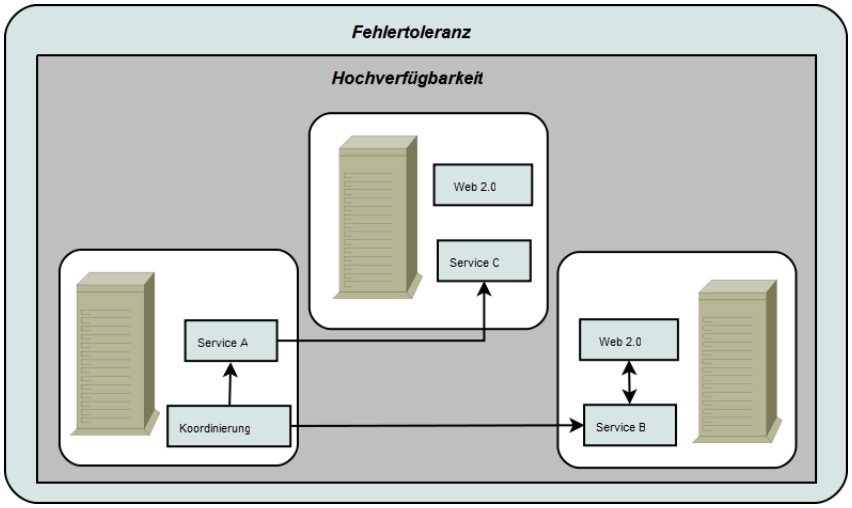


Abbildung 1.1: Zusammenarbeit von Recheneinheiten

Entwicklung von Ausfallsicherheits- und Fehlertoleranzkonzepten

Die wesentliche Aufgabe der Projektgruppe besteht darin, Konzepte zur Robustheit des Cloud-Systems zu erarbeiten. Dafür muss das System in der Lage sein, Knotenausfälle zu erkennen und auf den Wegfall zu reagieren. Es werden zwei Kategorieren von Knotenausfällen unterschieden: Zum Einen kann ein Knoten planmäßig abgeschaltet werden; in diesem Fall ist es möglich, einen laufenden Auftrag abzubrechen oder die Abschaltung bis zur Fertigstellung des Auftrags zu verzögern. Der zweite Fall ist der spontane Ausfall eines Knotens, der z.B. durch einen Stromausfall oder Netzwerkfehler entstehen kann. In dieser Situation müssen Datenverluste vermieden werden. Ferner muss berücksichtigt werden, dass eine oder mehrere System-Komponenten, die der Knoten ausführt, nicht mehr zu Verfügung stehen.

Bei Betrachtung der Ausfallsicherheit muss neben den laufenden Anwendungen des Cloud-Benutzers auch die Datenbank beachtet werden, in der Anwenderdaten und die Accounting-Daten gespeichert sind. Optional soll ein hochverfügbares verteiltes Dateisystem geschaffen werden, dass die Benutzerdaten und die Ergebnisse der Benutzeraufträge redundant speichert.

Untersuchung der Performance unter Last- und Problemsituationen

Bei der Untersuchung der Performance muss bewertet werden, wie sich das Gesamtsystem für reale Situationen im Normalbetrieb oder unter hoher Nutzungslast verhält. Dabei soll die Anzahl der Aufträge und die Anzahl der Knoten variieren, so dass eine Empfehlung gegeben werden kann, mit wie vielen Knoten das Cloud-System einen zuverlässigen Betrieb ermöglicht.

Hierbei ist ein Szenario zu entwickeln, das die Auslastung bei unterschiedlichen Cloud-Konfigurationen testet. Dazu gehören die Anzahl der Aufträge, die sich in der Cloud befinden, und die Auslastung der Datenbank.

2 Entwurf

Die erste Aufgabe, die sich der Projektgruppe stellte, war die Erarbeitung eines Modells des zukünfigen Cloud-Systems entsprechend der Projektgruppenbeschreibung. Dazu wurden die Anforderungen aus der Projektgruppenbeschreibung analysiert; sie werden nachfolgend vorgestellt und in Abbildung 2.1 dargestellt. Im Anschluss an dieser Analyse werden Anwendungsfälle erarbeitet, die als weitere Grundlage für die Modellierung dienen.

2.1 Anforderungen

Das zu erstellende Cloud-System soll einen modularen Aufbau aufweisen. Dadurch ergibt sich die Möglichkeit, das System in Teilsysteme aufzuteilen und so zu einem späteren Zeitpunkt Erweiterungen und Änderungen am System einfach vornehmen zu können.

Koordinierungsdienst

Im Cloud-System muss es einen Koordinierungsdienst geben, der einen Ausführungsplan für eingereichte Benutzeranwendungen (Jobs) erstellt. Dieser schreibt vor, welcher Job wann von welchem Knoten bearbeitet wird. Die Verteilung der Jobs gemäß dem Plan soll vom Koordinierungsdienst durchgeführt werden. Zur Erstellung des Ausführungsplans muss der Koordinierungsdienst die Verfügbarkeit von Knoten ermitteln. Es müssen außerdem eventuell Jobs neu verteilt werden, wenn ihre Ausführung wegen Fehlern in der Cloud-Infrastruktur fehlschlägt, zum Beispiel durch Verlust des ausführenden Knotens.

Ausführungsdienst

Ein Ausführungsdienstes auf den Knoten der Cloud soll die Zuweisungen vom Koordinierungsdienst entgegennehmen und die entsprechende Benutzeranwendung ausführen. Nach Beendigung eines Auftrags muss der Ausführungsdienst dessen Ergebnis zurückliefern und sich für weitere Ausführungen verfügbar melden.

Web 2.0-basierte Benutzerschnittstelle

Für den Zugriff auf das Cloud-System ist eine Web 2.0-basierte Schnittstelle zu erstellen. Über diese Schnittstelle soll aus Nutzersicht ein einfaches Lifecycle-Managementsystem bedient werden, in dem die Bearbeitungsfortschritte eines Kundenauftrages erkennbar sind. Aus Sicht des Cloud-Betreibers ist über die Web-Schnittstelle eine umfassende Administration der Cloud-Ressourcen und der Benutzerverwaltung möglich. Es sollen

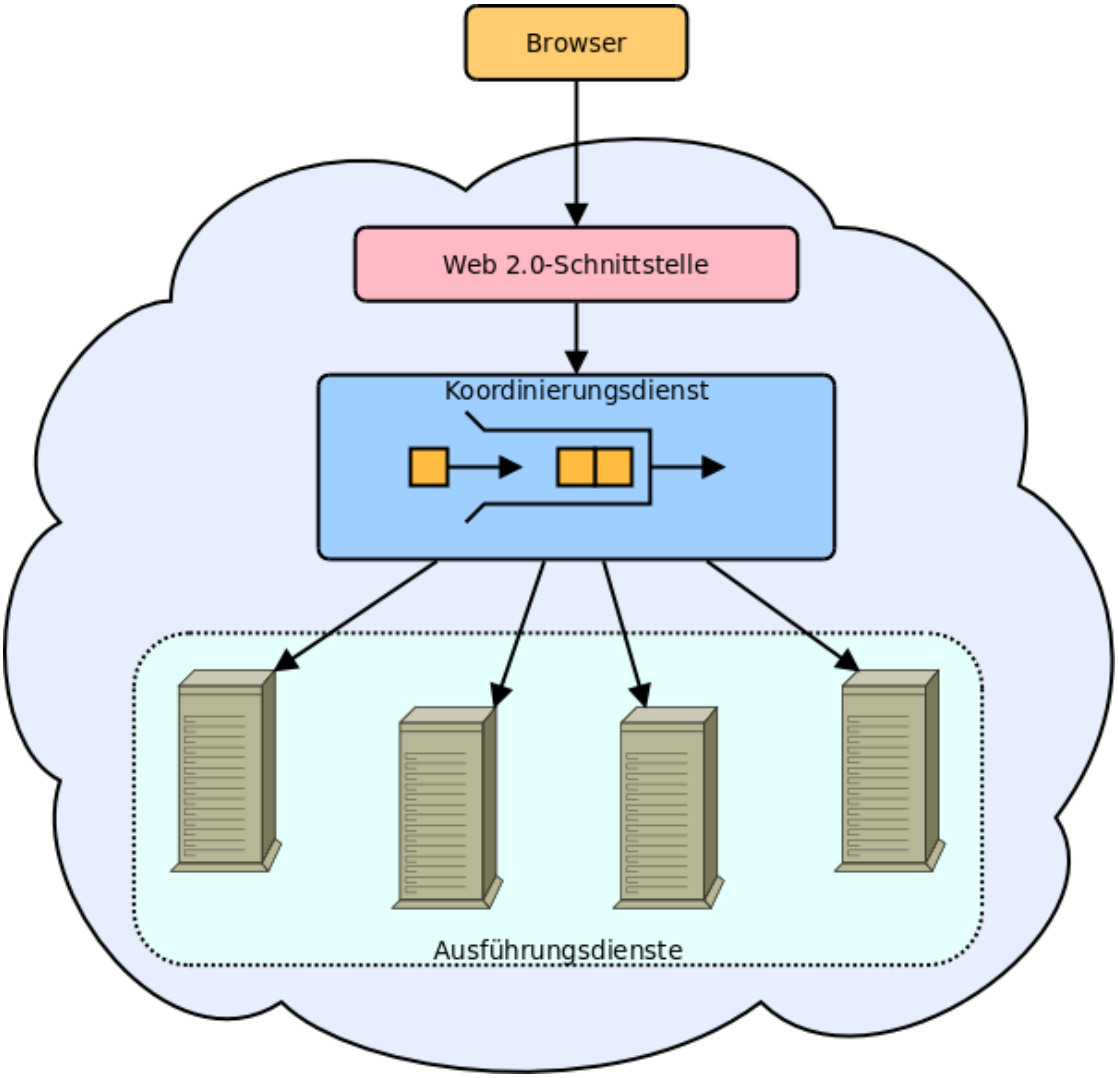


Abbildung 2.1: Grafische Darstellung der geforderten Teilsysteme

außerdem Accounting-Daten abrufbar sein, wie z.B. für einen Cloud-Benutzer entstandene Kosten.

Robustheit gegen den Verlust eines oder mehrerer Knoten

Als Schwerpunkt der Projektarbeit soll ein Konzept entwickelt werden, wie auf ausgefallene Teilsysteme des Cloud-Systems reagiert werden kann. Dabei muss die Cloud als Gesamtsystem funktionsfähig bleiben, Benutzer sollen einen Komponentenausfall nicht bemerken. Ein zentraler Aspekt ist die Hochverfügbarkeit von Diensten. Dabei sollen andere Knoten der Cloud die Funktionen eines ausgefallenen Knoten bzw. Dienstes übernehmen. Eine besondere Rolle spielt dabei die Datenhaltung. Für Robustheit gegen

den Verlust von Daten vorhaltenden Knoten müssen die Daten in der Cloud redundant vorliegen. Ferner muss berücksichtigt werden, wie das Cloud-System reagiert, wenn weggefallene Knoten wieder dem Gesamtsystem beitreten und bereits existierende Dienste zusätzlich anbieten.

2.2 Anwendungsfälle

Um ein Modell erstellen zu können, ist es notwendig, zunächst alle Anwendungsfälle zu identifizieren und zu klassifizieren.

Im Cloud-System werden drei Benutzerrollen betrachtet, die nach ausführbaren Aktivitäten unterteilt werden. Die allgemeinste Benutzerrolle ist die der „Person“. Eine „Person“ wird dadurch charakterisiert, dass sie noch nicht am Cloud-System angemeldet ist. Durch die Anmeldung am Cloud-System erhält eine „Person“ die Rechte der Benutzerrolle „User“. In dieser Rolle können die Funktionen der Cloud in Anspruch genommen werden. Für die Verwaltung der Cloud-Ressourcen bzw. der Benutzerdaten ist die Benutzerrolle „Admin“ vorgesehen. Der „Admin“ hat vollständigen Zugriff auf alle Daten, die im Cloud-System verfügbar sind. Diese Aufteilung ist aufeinander aufbauend. Ein Benutzer in der Rolle „Admin“ kann auch die Funktionen eines „Users“ ausführen und dieser die einer „Person“. Nachfolgend werden für die einzelnen Akteure die Benutzeraktionen erklärt und in Abbildung 2.2 dargestellt.

Die Rolle „Person“

Die Rolle „Person“ beschreibt Anwendungsfälle, bei denen ein Anwender sich dem Cloud-System gegenüber für die Sitzung noch nicht legitimiert hat. Anschließend können nachfolgende Aktionen ausgeführt werden:

Benutzer registrieren
: Durch die Eingabe von persönlichen Daten kann eine „Person“ sich am Cloud-System registrieren. Die Daten werden benötigt, um genutzte Rechenleistung in Rechnung stellen zu können. Dafür ist es ratsam, die eingegebenen Daten zu prüfen, bevor der Account genutzt werden kann.

Login ins Cloud-System
: Durch diese Aktion erhält eine „Person“ den Status „User“. Vorausssetzung dafür eine vorangegangene Registrierung an das Cloud-System.

Die Rolle „User“

Mit „User“ wird der Kunde identifiziert, der sich erfolgreich in das Cloud-System eingeloggt hat. Der „User“ kann die Funktionalität des Cloud-System nutzen und dabei folgende Aktivitäten ausführen:

Benutzerdaten anfordern
: Diese Aktion soll jedem identifizierten Cloud-Benutzer seine bei der Registrierung angegebenen persönlichen Daten zurückliefern.

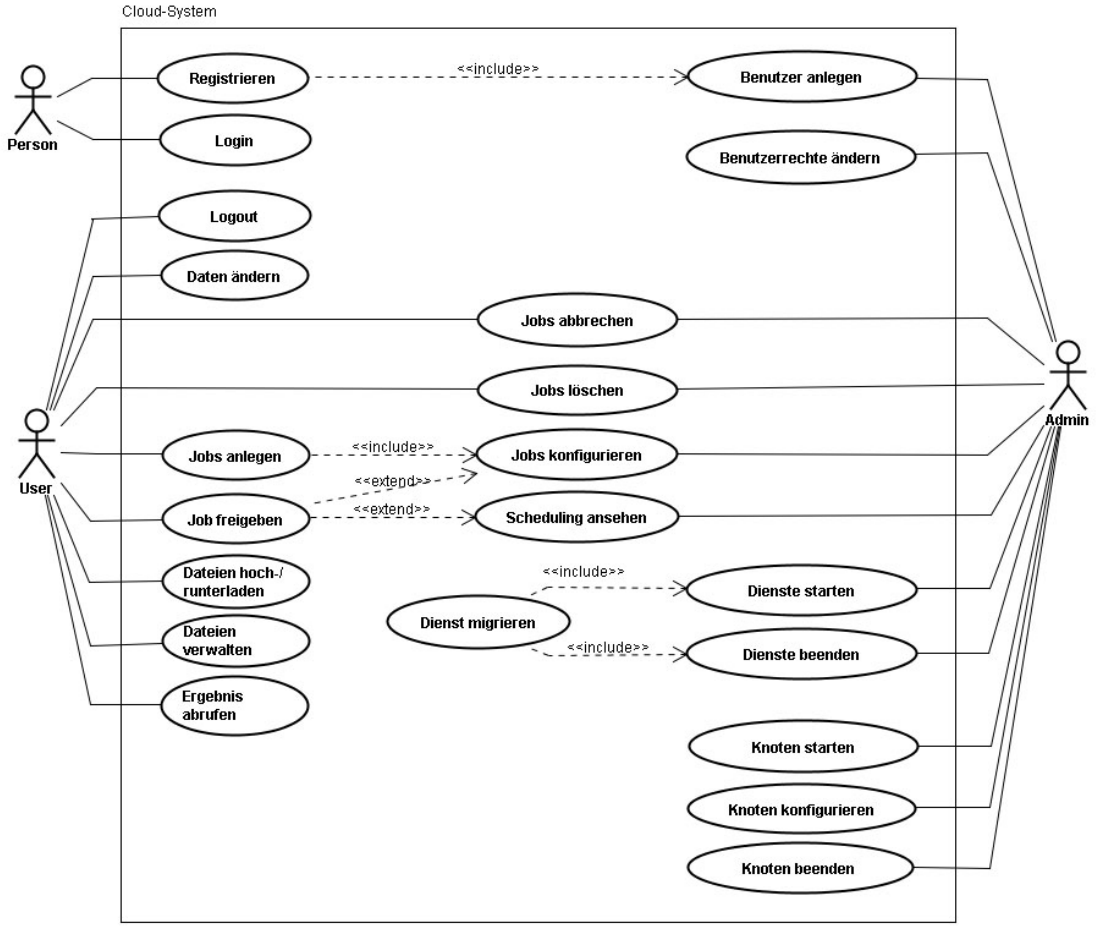


Abbildung 2.2: Die Anwendungsfälle für das Cloud-System

Benutzerdaten ändern

Der eingeloggte „User“ kann seine persönlichen Daten ändern.

Job anlegen

Der „User“ hat die Möglichkeit, einen neuen Job anzulegen und Parameter zur Ausführung des Jobs anzugeben.

Eigenen Job freigeben

Nachdem ein Job fertig konfiguriert ist, kann dieser freigegeben und dem Schedulingverfahren übergeben werden. Der Job wartet dann auf seine Ausführung.

Eigenen Job abbrechen

Der „User“ hat die Möglichkeit, einen eigenen Job abzubrechen, also einen durch Freigabe erteilten Ausführungsauftrag zurückzuziehen. Wird der Job bereits ausgeführt, wird die Ausführung abgebrochen.

Eigenen Job löschen
Ein „User“ kann einen seiner gestoppten Jobs löschen.

Status eines eigenen Jobs abfragen
Der „User“ erhält eine Rückmeldung, in welchem Status sich der Job befindet.

Liste eigener Jobs sehen
Der „User“ erhält eine Auflistung aller seiner im Cloud-System befindlichen Jobs.

Eigenen Job anzeigen lassen
Der „User“ kann sich einen eigenen Job und die dazu eingestellten Parameter anzeigen lassen.

Eigene Jobs konfigurieren
Der „User“ kann einen angezeigten Job konfigurieren, etwa um eingestellte Parameter zu verändern.

Eigene Datei ins Cloud-System einbringen
Der „User“ kann für die Bearbeitung eines Jobs Dateien zur Ausführung in die Cloud hochladen.

Eigene Datei(en) anzeigen lassen
Der „User“ erhält eine Übersicht über seine eigenen Dateien, die er zuvor in das Cloud-System eingebracht hat.

Eigene Datei(en) einem eigenen Job zuweisen
Der „User“ kann eine oder mehrere Dateien einem Job zuweisen.

Zugeordnete Datei(en) eines Jobs anzeigen lassen
Ein „User“ kann sich eine Übersicht über Dateien anzeigen zu lassen, die dem Job zugeordnet sind.

Eigene Dateien löschen
Benötigt ein „User“ eine oder mehrere bereits hochgeladenen Dateien nicht mehr für die Ausführung eines Jobs, kann er diese löschen.

Zugeordnung von Datei(en) aufheben
Ein „User“ kann die Zuordnung von Dateien zu Jobs aufheben.

Ergebnis abrufen
Der “User“ muss die Möglichkeit besitzen, die vom Cloud-System produzierten Ergebnisdateien einzusehen und gegebenenfalls herunterzuladen.

Statistiken ansehen
Der „User“ erhält eine Statistik über die Laufzeiten seiner Jobs.

Die Rolle „Admin“

Der Cloud-Benutzer in der Rolle „Admin“ ist für die System- und Benutzerverwaltung zuständig. Dazu erhält der „Admin“ Zugang zu den persönlichen Daten eines „Users“ und zu dessen eingestellten Jobs.

Neuen Benutzer anlegen
Der „Admin“ kann einen neuen Cloud-Benutzer anlegen. Diese Funktion dient nicht nur dem Admin zu Testzwecken, sondern kann auch erfolgen, wenn ein „User“ Probleme mit der Registrierung hat.

Benutzerdaten einsehen
Der „Admin“ kann die persönlichen Daten einer registrierten Person einsehen. Dies ist zur Verifikation notwendig, etwa von Rechnungsdaten.

Benutzerdaten ändern
Der „Admin“ als zentraler Ansprechpartner im System hat die Möglichkeit, persönliche Daten eines „Users“ zu ändern.

Benutzerdaten löschen
Der „Admin“ kann einen „User“-Account löschen, wenn der Kunde dies wünscht und alle betriebswirtschaftlichen Abrechnungen mit dem „User“ beglichen sind.

Alle Benutzer auflisten
Der „Admin“ kann sich durch diese Funktion einen Überblick verschaffen, welche Cloud-Benutzer im System insgesamt registriert sind.

Übersicht über vergebene Benutzerrollen
Der „Admin“ erhält eine Übersicht darüber, welche Rollen die Cloud-Benutzer erhalten haben. Dadurch wird es dem „Admin“ zusätzlich ermöglicht, Benutzerrechte zu vergeben und zu entziehen.

Alle Jobs auflisten
Der „Admin“ erhält eine Auflistung aller Jobs und deren Zustände.

Job löschen
Der „Admin“ muss die Möglichkeit haben, Kundenaufträge zu löschen. Diese Aktion könnte aus Wartungsgründen notwendig werden.

Scheduling anzeigen lassen
Der „Admin“ kann sich den Ausführungsplan anzeigen lassen. Durch diese Informationen kann er die Auslastung des Gesamtsystems erkennen und gegebenenfalls weitere Knoten in das Cloud-System hinzufügen oder entfernen.

Knoten herunterfahren
Der „Admin“ kann einen Knoten gezielt herunterfahren, um ihn anschliessend warten oder aus dem Gesamtsystem entfernen zu können.

Dienst auf einem Knoten erlauben oder verbieten
Durch unterschiedliche Hardwareausstattung und Erreichbarkeit von Knoten können sich Spezialisierungen für bestimmte Dienste ergeben. Beispielsweise ist ein Knoten mit einer geringen CPU-Leistung, aber mit großer Festplattenkapazität besonders dafür geeignet, in der Datenspeicherung eingesetzt zu werden. Daher muss dem Administrator die Möglichkeit eingeräumt werden, Dienste auf Knoten zu konfigurieren.

Dienst auf einem Knoten ausführen
Der „Admin“ kann einen Dienst gezielt auf einem Knoten starten.

Dienst auf einem Knoten stoppen
Der „Admin“ kann einen Dienst stoppen.

Knoteninformationen eintragen
Ein Knoten kann mit Informationen über seinen Standort und einer Beschreibung versehen werden. In dieser Beschreibung können zum Beispiel Leistungsmerkmale des Knoten eingetragen werden.

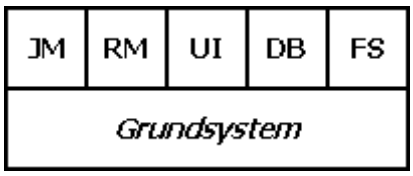
Knoteninformationen anzeigen
Neben der Knotenbeschreibung und dem Standort sollen durch den Knoten angebotene und für den Knoten erlaubte Dienste angezeigt werden.

Knoten auflisten
Mit dieser Funktion erhält der „Admin“ eine Liste der aktuell verfügbaren Knoten, in der die eingetragenen Knoteninformationen erkennbar sind.

Knoten-Statistiken einsehen
Der „Admin“ erhält eine Statistik des Gesamtsystems. Dazu gehört die Anzahl aller Knoten im System, die Anzahl der verfügbaren Knoten, die der rechnenden Knoten und die Anzahl der Knoten, die im Leerlauf sind.

2.3 Modellierung

Der Grundgedanke des Entwurfs ist, dass jeder Knoten in der Lage sein muss, jeden Dienst für das System anzubieten. Dies ermöglicht die Migration von durch Knotenverlust verlorenen Diensten auf andere Knoten. Dazu muss ein minimales Grundsystem existieren, das stets ausgeführt wird; daneben sollen optionale Dienste von Komponenten angeboten werden, die vom Grundsystem dynamisch gestartet und gestoppt werden können.



Grundlayout eines Knotens

Das Cloud-System setzt sich aus folgenden Komponenten zusammen:

Datenbank (kurz DB)
Diese Komponente verwaltet die verteilte, hochverfügbare Datenbank und stellt den Zugriff auf diese bereit (siehe 3.2).

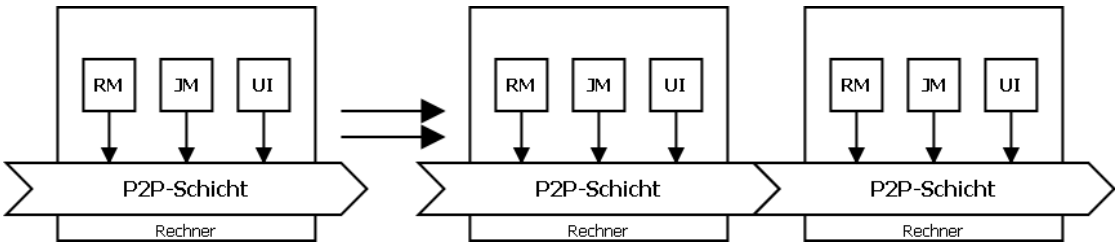
Resource-Manager (kurz RM)
Der Resource-Manager nimmt die Jobs vom Job-Manager entgegen und führt sie aus (siehe 3.3); er stellt also den Ausführungsdienst dar. Jeder zur Bearbeitung von Jobs geeignete Knoten sollte einen Resource-Manager anbieten.

Job-Manager (kurz JM)
Der Job-Manager verwaltet die eingereichten Jobs im System und verteilt sie auf freie Resource-Manager (siehe 3.4). Er realisiert den Koordinierungsdienst. Es sollte im Normalfall nur ein Job-Manager in der Cloud angeboten werden, Robustheit gegen temporären Mangel oder Überschuss muss aber gegeben sein.

User-Interface (kurz UI)
Die Komponente, die einen Webserver startet und die Web 2.0-Schnittstelle bereitstellt. Über diese greifen die Benutzer auf die Cloud zu (siehe 3.5). Die Benutzerschnittstelle muss von mindestens einem Knoten angeboten werden; es ist aber denkbar, zur Vermeidung von Ausfallzeiten oder zur Lastverteilung mehrere Instanzen verfügbar zu halten.

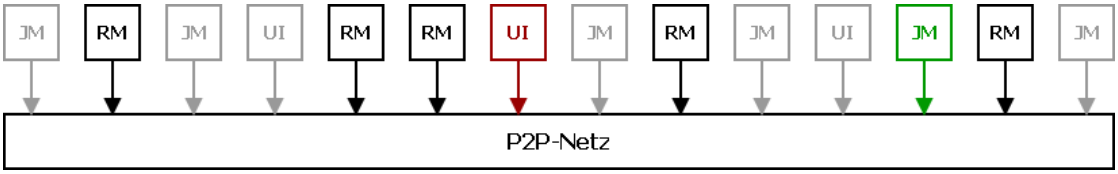
Dateisystem (kurz FS)
Diese Komponente verwaltet Benutzerdateien im Cloud-System und bietet anderen Komponenten Zugriff auf die Dateien (siehe 3.6).

Jeder Knoten kommuniziert mit anderen Knoten über eine P2P-Komponente im Grundsystem. Neue Knoten gliedern sich dabei in ein bestehendes P2P-Netz ein oder bauen ein neues P2P-Netz auf (siehe 3.1.5 Netzwerk).



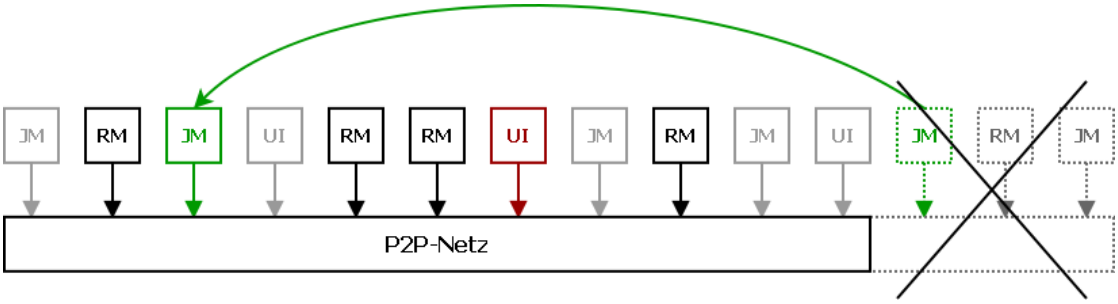
Knoten mit P2P-Modul

Aus der Sicht der Komponenten existiert eine Umgebung, in der die anderen Komponenten gefunden werden können. Die Komponenten kümmern sich dabei nicht um den Aufbau und Erhalt des P2P-Netzes, dies ist die Aufgabe der P2P-Schicht.



P2P-Netz mit laufenden und deaktivierten Diensten

Wenn nun ein Teil des P2P-Netz wegfällt, müssen ggf. an einer anderen Stelle neue Dienste gestartet werden. Dieser Aspekt der Hochverfügbarkeit von Komponenten soll als Teil des Grundsystems realisiert werden.



Ein anderer Dienst muss für einen ausgefallenden Dienst einspringen

2.3.1 Erster Entwurf

Im ersten Entwurf war eine zentrale Komponente CloudVerbund vorgesehen, die den Zugriff auf das P2P-Netz, auf die verteilte Datenbank und auf das verteilte Dateisystem verwaltet.

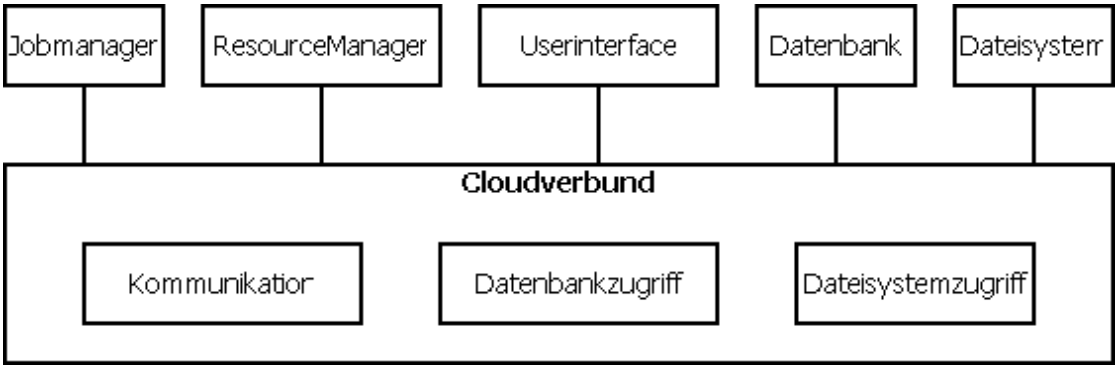


Abbildung 2.3: Erster Entwurf eines Knotens

Für jede der einzelnen Komponenten bietet der Cloud-Verbund eine für sie passende Schnittstelle an, damit die Komponente auf die Cloud zugreifen kann. So kann der Job-Manager z.B. über eine Methode `public RM[] getListOfRMs()` vom Cloud-Verbund eine Liste der Resource-Manager anfragen. Die Kommunikation zwischen den Komponenten muss immer indirekt über die zentrale Komponente laufen.

Dieser Entwurf wurde jedoch verworfen. Der Cloud-Verbund implementiert alle Schnittstellen für die Komponenten. Dadurch wird dieser zu einem monolithischen System, das alle Anfragen der Komponenten verarbeiten muss. Es würde also ein großer Teil der Programmlogik in dieser Komponente stecken. Außerdem macht dies auch eine Erweiterung des Systems schwierig, da immer diese Komponente angepasst und verändert werden muss. Durch die indirekte Kommunikation entsteht im Cloud-Verbund außerdem mehr Last und Overhead gegenüber direkten Aufrufen zwischen den Komponenten.

2.3.2 OSGi als Plattform-Lösung

Da als Konsequenz aus den Schwächen des ersten Entwurfs eine strenge modulare Trennung in Komponenten gefordert wurde, wurde genauer untersucht, ob und wie OSGi bei der Umsetzung helfen könnte. OSGi ist eine Software-Plattform, die das Modularisieren von Anwendungen erleichtern soll. Die im Entwurf identifizierten Komponenten sollten Dienste bereitstellen und miteinander über Dienst-Schnittstellen interagieren. Nach einer kurzen Einführung wird klar werden, dass dies einem der Kerngedanken hinter OSGi entspricht. Fraglich war außerdem, ob sich mit OSGi diese Dienst-Schnittstellen auch in einem verteilten System für andere Teilnehmer anbieten lassen. OSGi bietet zwar noch nicht die verteilte Funktionalität solcher Service-orientierten Architekturen (SOA), es stellte sich aber heraus, dass sich OSGi als Ausgangspunkt zur Erfüllung dieser Anforderung eignet.

Überblick

Die OSGi-Plattform ist ein offenes, modulares und skalierbares Komponentenmodell auf Basis von Java. Da eine umfassende Einführung in OSGi den Rahmen dieses Textes sprengen würde, sollen im Folgenden nur kurz die wesentlichen Begriffe und Eigenschaften angerissen werden.

Ein zentraler Begriff im Zusammenhang mit OSGi ist der des Bundles. Bundles in OSGi entsprechen den Komponenten der Anwendung. OSGi bietet für Bundles Lebenszyklus-Mechanismen, um Komponenten zu starten oder zu stoppen und um Abhängigkeiten zu kontrollieren und aufzulösen. Die Modularisierung wird durch Verschärfung des Sichtbarkeitsmechanismus von Java unterstützt; OSGi-Bundles stellen anderen Bundles nur die Java-Packages zur Verfügung, die explizit exportiert werden, und können nur die Packages nutzen, die sie explizit importieren. Die Erfüllung der sich daraus ergebenden „statischen“ Abhängigkeiten wird von OSGi-Umgebungen automatisch überwacht. OSGi-Bundles sind als JAR-Dateien realisiert und können auch als solche benutzt werden. Von „normalen“ Java-Bibliotheken unterscheiden sie sich nur durch zusätzliche Informationen in der `manifest`-Datei, wie zum Beispiel Abhängigkeiten und exportierte Pakete.

OSGi bietet außerdem die Möglichkeit, Funktionalität als Services zentral zu registrieren und so bereitzustellen. Solche Dienste werden mit Java-Interfaces beschrieben, die ein anbietendes Bundle implementiert (natürlich ohne diese Implementierung offen

legen zu müssen). Es ist weiterhin die Möglichkeit gegeben, das „Erscheinen“ und „Verschwinden“ von Diensten mit einem ServiceListener zu überwachen und somit einen Dienst nur dann anzubieten, wenn seine „dynamischen“ Abhängigkeiten von anderen Diensten erfüllt sind. Eine einfachere Möglichkeit hierzu ist mit einer standardisierten Erweiterung von OSGi um die sogenannten Declarative Services gegeben. Ein angebotener Dienst kann hierbei mit seinen Abhängigkeiten in einer XML[1]-Datei beschrieben werden. Die Erweiterung stellt dann sicher, dass der Dienst je nach Verfügbarkeit seiner Abhängigkeiten automatisch gestartet und gestoppt wird.

SOA auf Basis von OSGi

Das Registrieren eines Dienstes in OSGi hat den Zweck, diesen JVM[2]-weit zur Verfügung zu stellen; in der OSGi-Plattform gibt es keine Möglichkeit, einen Dienst systemweit oder netzwerkweit anzubieten. Glücklicherweise ist OSGi aber so konzipiert, dass die Plattform um solche Funktionalität erweitert werden kann.[3] Eine Erweiterung, mit der OSGi-Dienste über Netzwerkschnittstellen angeboten werden können, ist R-OSGi. Dazu wird die Dienstschnittstelle auf Client-Seite durch R-OSGi völlig transparent von einer Proxyklasse angeboten, so dass am Dienst keine Änderungen vorgenommen werden müssen. Der Dienst muss nur bei seiner Deklaration als zu exportieren gekennzeichnet sein.

Ergebnisse für den Entwurf

In der Einarbeitung mit OSGi stellte sich heraus, dass es die Modularisierung einer Anwendung in Komponenten und Dienste sehr vereinfachen kann. Nicht nur die Unterstützung eines Lebenszyklus für die Komponenten, sondern auch die Arbeitsersparnis durch Declarative Services und von R-OSGi generierten Proxy-Klassen sprechen sehr für OSGi; es wurde deswegen die Entscheidung getroffen, OSGi zu nutzen.

2.3.3 Zweiter Entwurf

Der zweite Entwurf sieht die Realisierung der Komponenten als OSGi-Bundles vor, welche vom OSGi-Framework dynamisch gestartet und gestoppt werden können. Außerdem kann ein Bundle direkt auf andere Bundles zugreifen, der Umweg über eine Cloud-Verbund-ähnliche Komponente ist deshalb nicht mehr nötig.

[1] Extensible Markup Language
[2] Java Virtual Machine
[3] Übrigens auf ähnliche Art wie die Declarative Services-Erweiterung, die in vielen OSGi-Implementierungen auch wirklich als Bundle realisiert ist.

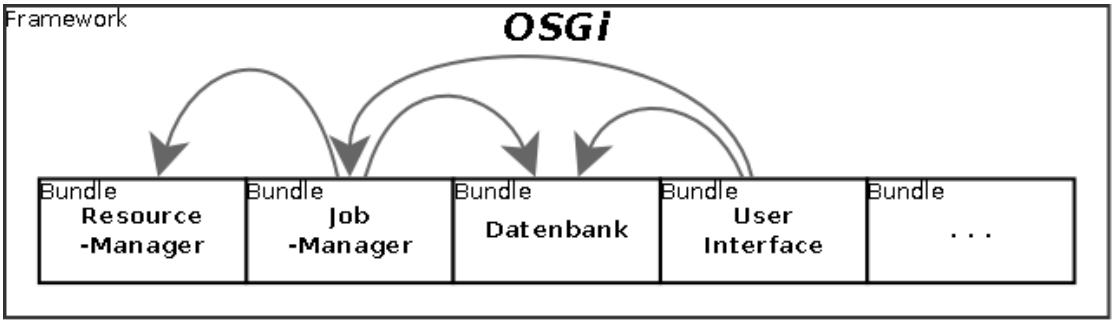


Abbildung 2.4: OSGi-Framework mit diversen Bundles

2.3.4 Der Job-Lebenszyklus

Neben der Modellierung der eigentlichen Systemarchitektur zeigte sich, dass auch die zu durchlaufenden Bearbeitungsphasen der Jobs im System einer detaillierten Modellierung bedürfen.

Die Jobs werden nicht zentralisiert abgearbeitet, sondern von einem Koordinierungsdienst (Job-Manager) an unterschiedliche Ausführungsdienste (Resource-Manager) verteilt, so dass mehrere Jobs parallel bearbeitet werden. Der Job-Manager dient hierbei als Verwaltungseinheit für die Verteilung der Jobs, aber auch für die Bearbeitungsfortschritte der Jobwarteschlange, also der Zustände der Jobs. Nicht zuletzt muss auch das Ziel, hinsichtlich der Ausführung eingebrachter Jobs Hochverfügbarkeit sicherzustellen, bei der Einführung differenzierter Zustände berücksichtigt werden.

Hierzu musste sich die Projektgruppe über die einzelnen Phasen eines Jobs vom Erstellen bis hin zur Freigabe und Verarbeitung im System klar werden. Die identifizierten Phasen eines Jobs innerhalb des Systems entsprechen dem Lebenszyklus eines Jobs. Die Phasen lehnten sich zunächst an die für die Spezifikation der Anforderungen benötigten Use-Case-Diagramme an. Als problematisch stellte sich jedoch heraus, dass die so definierten Lebenszyklus-Phasen zwar dem entworfenen System genügen würden, diese jedoch zu speziell bzw. feingranular waren, als dass diese der Forderung nach Erweiterbarkeit und Modularität des Systems Rechnung getragen hätten. Um den Anforderungen des Systems nach Erweiterbarkeit zu genügen, bot es sich an, einen Standard für die Beschreibung der zu durchlaufenden Phasen eines Jobs zu finden. Hier zeigte sich, dass der OGSA-BES-Standard[4] eine passende Grundlage darstellt. OGSA-BES beschreibt fünf Zustände `PENDING`, `RUNNING`, `CANCELLED`, `FINISHED` und `FAILED` und die Übergänge zwischen ihnen, erlaubt aber die problembezogene Erweiterung um Unterzustände. Die bei der Analyse identifizierten Zustände konnten daher differenziert modelliert werden, ohne dass der entworfene Job-Lebenszyklus dem Standard widerspricht.

[4] `http://www.ogf.org/documents/GFD.108.pdf`

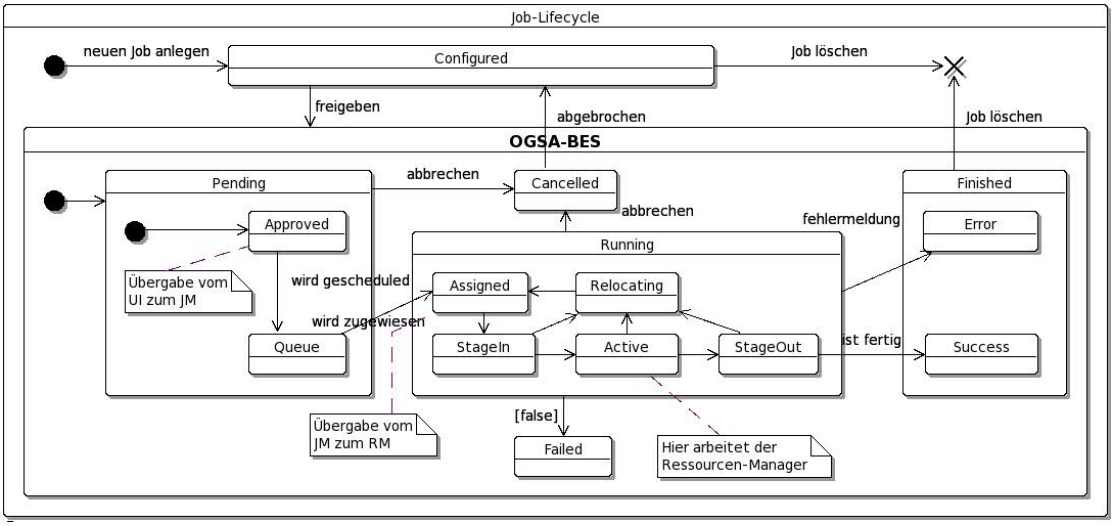


Abbildung 2.5: Der identifizierte Lebenszyklus als Erweiterung von OGSA-BES

Die einzelnen Phasen bzw. Zustände eines Jobs im System lassen sich wie folgt zusammenfassen:

`CONFIGURED`:
: Der Benutzer hat einen Job erstellt und eine vorläufige Konfiguration für den Job über die Benutzerschnittstelle festgelegt. Es ist also eine Jobbeschreibung in der Datenbank eingetragen, und möglicherweise benötigte Dateien zum Job wurden im Dateisystem abgelegt.

`PENDING`:
: Dieser Status beschreibt nach OGSA-BES Jobs, die auf eine Ausführung warten. In der vorliegenden Implementierung werden folgende Unterzustände angenommen:

  `APPROVED`:
  : Nachdem der Benutzer den Job erstellt und konfiguriert hat, kann der Benutzer den Job für die Bearbeitung freigeben. Sobald der Benutzer den Job über die Benutzerschnittstelle freigibt, erhält der Job den Status `APPROVED` und wird erst dann vom Job-Manager für das Scheduling beachtet.

  `QUEUE`:
  : Der Job wurde freigegeben und aufgrund des Status `APPROVED` in die Scheduling-Liste aufgenommen. Sobald der Job in die Scheduling-Liste eingetragen wurde, erhält er den Status `QUEUE`.

`RUNNING`:
: Dieser Status ist von OGSA-BES vorgeschrieben und ist Oberzustand folgender Zustände:

  `ASSIGNED`:
  : Der Job-Manager hat einen nicht arbeitenden Resource-Manager gefunden

und den Job an diesen übergeben, die Ausführung auf dem Resource-Manager wurde jedoch noch nicht gestartet.

`STAGEIN`:
Der Job wurde durch einen Resource-Manager angenommen, die Ausführung ist noch nicht gestartet. Der Job befindet sich im Status `STAGEIN`, solange der Resource-Manager mit dem Vorbereiten der Ausführung beschäftigt ist.

`ACTIVE`:
Nachdem der Resource-Manager die vorbereitenden Schritte abgeschlossen hat, beginnt dieser mit der Ausführung des Jobs. Der Job wechselt in den Status `ACTIVE`.

`STAGEOUT`:
Nachdem der Job durch einen Resource-Manager ausgeführt wurde, werden abschließend die Ergebnisse des Jobs in das verteilte Dateisystem geschrieben. In dieser Zeit der Nachbereitung der Ausführung durch den Resource-Manager erhält der Job den Status `STAGEOUT`.

`RELOCATING`:
Ein Job nimmt den Status `RELOCATING` an, wenn ein Resource-Manager, an den der Job übergeben wurde, diesen nicht ausführen kann. Die Meldung, dass der Job nicht ausgeführt werden kann und einem anderen Resource-Manager zugewiesen werden muss, kann durch den Resource-Manager selbst erfolgen. Ein Job-Manager kann die Neuzuweisung aber auch veranlassen, wenn er über den Ausfall eines Resource-Managers informiert wird. Es gibt zwei Argumente, die für die Einführung dieses Zustands und gegen einen Übergang zu `APPROVED` sprechen:

- OGSA-BES sieht einen streng linearen Ablauf zwischen den dort beschrieben Zuständen vor und erlaubt keine Rückführungen in zeitlich vorhergende Zustände (in diesem Fall von `RUNNING` nach `PENDING`).
- Es ist so möglich, Jobs mit Status `RELOCATING` bei der Erstellung des Ausführungsplan zu bevorzugen, so dass sie nicht erneut die gesamte Warteschlange durchlaufen.

`CANCELLED`:
Der Status `CANCELLED` ist der einzige Status im Lebenszyklus eines Jobs, der nur durch Benutzer-Interaktion erreicht werden kann. Er beschreibt einen über die Benutzerschnittstelle nach der Freigabe zurückgezogenen Job.

`FINISHED`:
Dieser Status beschreibt nach OGSA-BES einen vollständig ausgeführten Job.

`SUCCESS`:
Der Job hat eine fehlerfreie Ausführung gemeldet.

`ERROR`:
Der Job hat einen Fehlercode zurückgegeben, der als zusätzliche Information

gespeichert wurde.

`FAILED`:
Dieser Status ist von OGSA-BES für Jobs vorgesehen, die durch Ausnahmefälle im ausführenden System fehlschlagen. In der vorliegenden Implementierung des Systems nimmt ein Job diesen Status zu keinem Zeitpunkt an.

3 Realisierung

In der zweiten Hälfte der Projektarbeit wurden die in der Modellierung identifizierten Komponenten realisiert. In den folgenden Kapiteln wird die Umsetzung der einzelnen Teilsysteme betrachtet; dazu werden auch speziellere Anforderungen formuliert und eventuelle Framework-Entscheidungen begründet.

3.1 Grundsystem

Der Entwurf, der bei der Realisierung der Cloud verfolgt wird, sieht ein Grundsystem vor, das auf jedem Knoten zur Verfügung steht und auf Grundlage dessen die eigentliche Cloud-Funktionalität umgesetzt werden kann. Dieses Grundsystem muss die im Entwurf geforderte OSGi-Laufzeitumgebung und den Aufbau und die Nutzbarkeit eines P2P-Netzwerkes umfassen. Desweiteren soll auf jedem Knoten Funktionalität zur verteilten Datenhaltung zur Verfügung stehen, die von der Komponente Datenbank bereitgestellt wird; wegen der Austauschbarkeit dieser Komponente wird sie aber in Kapitel 3.2 unabhängig betrachtet.

3.1.1 Anforderungen an das Grundsystem

Das Grundsystem soll ein Diensteverzeichnis bieten, über das in der Cloud angebotene Dienste aufgefunden werden können. Es soll bei der Benutzung bestimmter Dienste unerheblich sein, ob der Dienst lokal oder entfernt angeboten wird. Dazu ist eine Erweiterung der OSGi-Diensteregistrierung um das Auffinden entfernter Dienste nötig. Das Gleiche gilt für das Starten und Stoppen von Diensten: Dabei ist für einen nutzenden Dienst nicht wesentlich, dass unter Umständen ein Wahlverfahren durchgeführt werden muss, um zu entscheiden, welcher Knoten eventuelle Abhängigkeiten von anderen Diensten erfüllen wird.

Ein wichtiges Ziel der Projektgruppe ist ein Konzept zur Sicherstellung der ständigen Verfügbarkeit von Diensten in ausreichender Anzahl. Um dies zu gewährleisten, muss es einen Dienst geben, der andere Dienste zur Laufzeit starten und beenden kann. Außerdem ist die Überwachung und das rechtzeitige Erkennen möglicher Engpässe im Dienste-Angebot der Cloud eine wichtige Voraussetzung einer rechtzeitigen Intervention. Durch einen überwachenden Dienst, der verteilt laufen kann, ist es dann möglich, die Dienste der Cloud dynamisch oder nach gewissen Regeln zu steuern.

Eine Netzwerk-Komponente soll für andere Komponenten die Netzwerkkommunikation bereitstellen. Damit soll erreicht werden, dass jede Komponente über eine einheitliche Schnittstelle mit Komponenten anderer Rechner kommunizieren kann, ohne dies

implementieren zu müssen. Durch die Kapselung der Kommunikation ist vor den Komponenten verborgen, wie die Kommunikation tatsächlich stattfindet. So lassen sich die Kommunikation und auch das Auffinden von Knoten problemlos auf eine andere Technik umstellen, solange die vorgegebene Schnittstelle implementiert wird.

3.1.2 Umsetzung der Komponenten mit OSGi

Die im Entwurf genannten Komponenten wurden jeweils als OSGi-Bundles realisiert, die einen der Komponente entsprechenden Dienst anbieten.

- Das Bundle `core` stellt eine verteilte Diensteregistrierung bereit.
- Die P2P-Funktionalität ist im Bundle `network` umgesetzt.
- In den Bundles `database`, `jobmanager`, `resourcemanager` und `userinterface` sind die gleichnamigen Komponenten umgesetzt.

Um die Austauschbarkeit der Cloud-spezifischen Komponenten zu gewährleisten, sind ihre Dienst-Schnittstellen im Bundle `cloud` gesammelt, welches selbst keine Dienste bereitstellt. Gleichzeitig beugt diese Trennung von Beschreibung und Bereitstellen eines Dienstes zyklischen Abhängigkeiten zwischen OSGi-Bundles vor. Aus diesem Grund finden sich dort auch die gemeinsam genutzten Klassen für die Geschäftslogik wie `Job`, `User` etc.

3.1.3 Die Laufzeitumgebung

Die Laufzeitumgebung wurde erstellt, um ein OSGi-Framework zur Ausführung der einzelnen Sinefa-Komponenten bzw. deren Bundles anzubieten. Nach dem Beschluss, OSGi zu verwenden, wurde es nötig, eine Umgebung zu schaffen, in der OSGi-Bundles ausgeführt werden konnten. Insbesondere musste das Testen von Zwischenergebnissen der Entwicklung möglich sein. Die Entscheidung fiel auf das Apache Felix-Framework, welches im folgenden Abschnitt näher beleuchtet wird. Zusätzlich wird dort die Begründung für diese Entscheidung dargelegt.

Apache Felix

Da die Erstellung einer eigenen OSGI-R4[1]-Implementierung keine Alternative darstellte, die Notwendigkeit einer Laufzeitumgebung allerdings gegeben war, wurden verschiedene OSGi-Framework-Implementierungen betrachtet. Als Anforderungen sind die Implementierung der OSGi-Basiskomponenten, sowie die Implementierung der Declarative Services zu nennen. Betrachtet wurden Equinox[2], Knopflerfish[3] sowie Apache Felix. Felix bietet eine vollständige Umsetzung der OSGi-R4-Spezifikation, insbesondere der benötigten Declarative Services. Felix stellt außerdem die problemfreie Zusammenarbeit mit dem

[1] `http://www.osgi.org/Specifications/HomePage`
[2] `http://www.eclipse.org/equinox/`
[3] `http://www.knopflerfish.org/`

Bundle Plugin for Maven sicher, das auch durch die Apache Software Foundation gepflegt wird und das Erstellen von OSGi-Bundles maßgeblich erleichtert. Auch Equinox und Knopflerfish erfüllen die gestellten Anforderungen. Beide bieten eine Implementierung der OSGi-R4-Spezifikation sowie der Declarative Services. Apache Felix wies jedoch von allen betrachteten Produkten die einfachste Nutzbarkeit auf. Die umfassende Dokumentation auf der Homepage des Produktes ermöglichte eine leichte Integration in das Projekt. Aus diesem Grund, sowie wegen der Nähe zu den weiteren verwendeten Produkten der Apache Software Foundation wie etwa Maven 2 (siehe 4.1.2) und Bundle Plugin for Maven, wurde Apache Felix als OSGi-Framework für die Umsetzung der Laufzeitumgebung ausgewählt.

Konfiguration der Laufzeitumgebung

Die Konfiguration des Apache Felix-Frameworks ermöglicht das Einstellen des Verhaltens der Laufzeitumgebung. Von besonderer Bedeutung ist hier das Gruppieren von Bundles in *startlevels*. Die Startlevel sind aufsteigend numeriert, beginnend mit 0. Auf Startlevel 0 befinden sich diejenigen Bundles, welche beim Start des Frameworks gestartet werden. In der Konfiguration ist weiterhin festgelegt, bis zu welchem Startlevel die Bundles gestartet werden sollen. Dies bedeutet, dass bei Festlegen von Startlevel 2 sämtliche Startlevel einschließlich Level 2 gestartet werden. Die Bundles der darüberliegenden Startlevel werden installiert, aber nicht gestartet. Die Festlegung der Startreihenfolge für Bundles ist von besonderer Bedeutung, da zwischen Bundles Abhängigkeiten bestehen. So muss ein Bundle, welches von einem Bundle auf Startlevel 1 abhängig ist, auf Startlevel 2 gesetzt werden, da es sonst durch die nicht erfüllte Abhängigkeit nicht gestartet werden kann. Die für das Cloud-System verwendeten *system bundles*(siehe 3.1.3) befinden sich auf Startlevel 1. Auf den darüberliegenden Leveln finden sich die Komponenten des Cloud-Systems.

Zusätzlich zur Konfiguration der OSGi-Laufzeitumgebung selbst wurden im Laufe der Entwicklung immer mehr Konfigurationsdateien für die verschiedensten Komponenten abgelegt. So befinden sich hier beispielsweise die Konfigurationsdateien für das `monitoring`-Bundle sowie für das `network`-Bundle. Dies bringt den Vorteil mit sich, dass Entwickler auch für Unterprojekte, an deren Entwicklung sie nicht beteiligt waren, Einstellungen vornehmen können.

Einbindung von Fremdbibliotheken

Außer den Bundles der Komponenten sind für den Betrieb der Cloud noch weitere Bundles notwendig. Zum Einen sind dies Bundles, die Teile der Ausführungsumgebung bereitstellen, wie die interaktive Konsole oder die Declarative Services. Zum Anderen sind dies aber auch Abhängigkeiten, die sich aus der Umsetzung der Komponenten ergeben – da OSGi recht verbreitet ist, lassen sich sehr viele Java-Bibliotheken schon als OSGi-Bundles benutzen. Die Zahl der verwendeten *system bundles* stieg im Verlauf der Projektgruppe stetig an. Beispiele hierfür sind der Einsatz von R-OSGi oder etwa eines HTTP-Servers für die Benutzerschnittstelle.

Da OSGi-Bundles auch wie „normale“ Java-Archive zu benutzen sind, ist es möglich, fremde Bundles nicht als eigenständige OSGi-Bundles zu behandeln, sondern direkt in die eigenen einzubetten. Um die Menge an gestarteten Bundles überschaubar zu halten, wurde entschieden, nur solche Abhängigkeiten als Bundles zu starten, die von mehreren Komponenten benutzt werden. Abhängigkeiten, die nur eine Komponente benötigt, werden soweit möglich direkt in diese eingebettet.

In der nachfolgenden Grafik 3.1 ist die Bundle-Struktur des Cloud-Systems dargestellt. Zu sehen ist die Unterteilung in Komponenten- und *system bundles*.

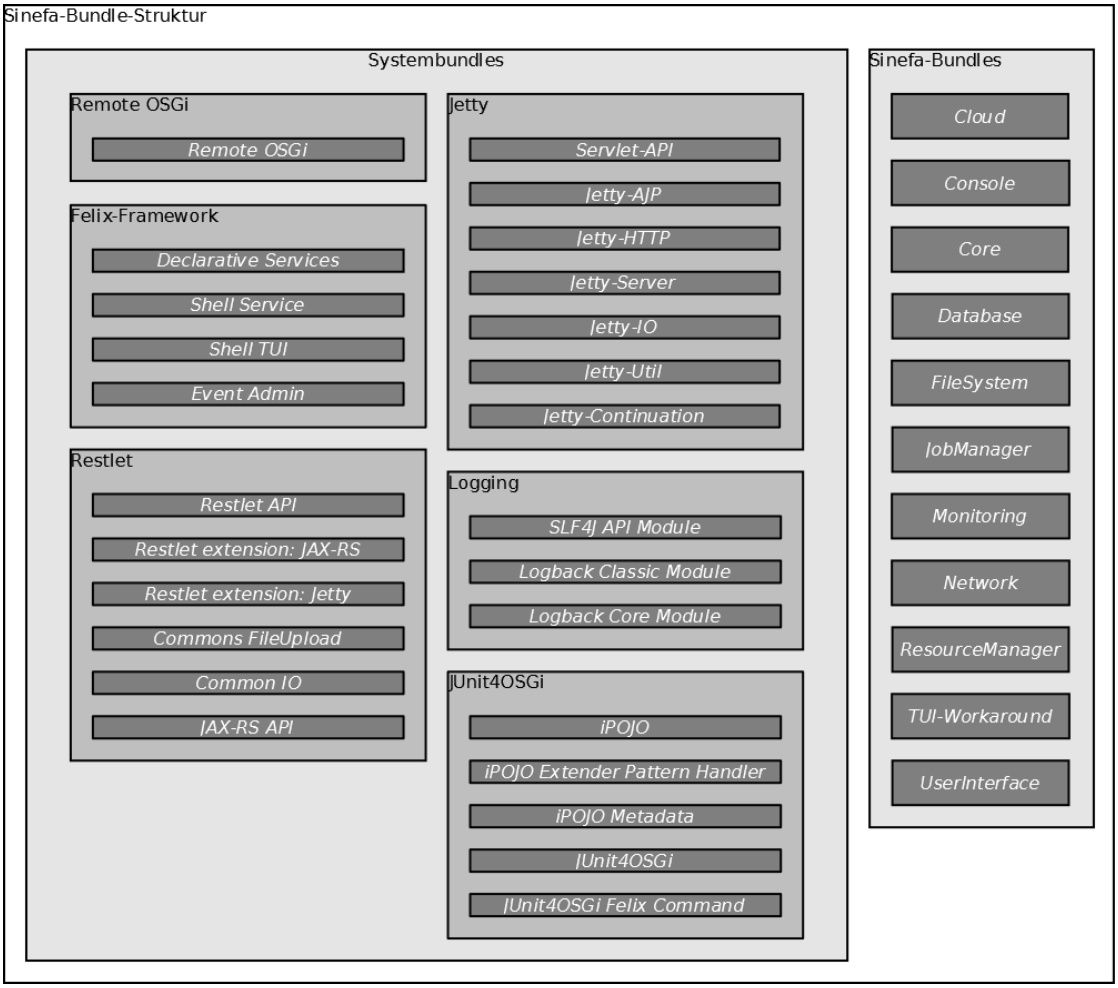


Abbildung 3.1: Bundlestruktur des Cloud-Systems

Da nicht alle benötigten system bundles in öffentlichen Maven-Repositories auffindbar waren, war die Notwendigkeit eines eigenen Maven-Repositories gegeben. Dieses sollte nicht nur die Ablage von system bundles ermöglichen, vielmehr sollten auch die Komponenten-Bundles dort gespeichert werden. Die Wahl fiel auf Apache Archiva[4]. Die

[4] `http://archiva.apache.org/`

Nähe zu Apache Maven sowie die unkomplizierte und zeitextensive Einrichtung legten diese Entscheidung nahe.

3.1.4 Verteilte Diensteregistrierung

In der OSGi-Plattform werden Dienste lokal in einer OSGi-Laufzeitumgebung registriert. Mit Erweiterungen wie R-OSGi[5] oder dem im OSGi-Kompendium[6] beschriebenen D-OSGi ist es möglich, Dienste anderer OSGi-Laufzeitumgebungen zu nutzen. Dazu muss der Dienst aber vom Nutzer explizit adressiert und angefordert werden. In einer P2P-Umgebung ergibt sich daraus zusätzlicher Aufwand bei der Nutzung eines Dienstes; es ist wünschenswert, die Informationen über Verfügbarkeit, Standort und Angebotszahl von Diensten an anderer Stelle zu verwalten, so dass Dienste weiterhin mit reinen OSGi-Mitteln angefordert werden können.

Der Entwurf sieht für diese Aufgabe eine Komponente ServiceRegistry vor, die das Diensteangebot in der P2P-Umgebung überwacht. Diese Komponente fordert transparent lokal benötigte Dienste von bereitstellenden Knoten an, stellt aber auch lokale Dienste für andere Knoten zur Verfügung.

Die OSGi-Plattform stellt für Erweiterungen dieser Art die benötigten Informationen bereit. Wartet ein Dienst auf die Verfügbarkeit eines anderen Dienstes, wird dazu ein Listener registriert. Eine Erweiterung kann sich über das An- und Abmelden solcher Listener benachrichtigen lassen und somit *on demand* Dienste von anderen Knoten lokal verfügbar machen.

Die Diensteregistrierungen der Knoten kommunizieren über die Netzwerk-Komponente. Zum Auffinden eines Dienstes wird ein `InterestAdvertisement` an die ganze Cloud verteilt, auf das anbietende Knoten antworten. Da Dienste unter Umständen erst später verfügbar werden, wird zusätzlich nach dem Starten von Diensten ein `ServiceAdvertisement` verschickt. Eine ServiceRegistry, die über einen „interessanten“ Dienst benachrichtigt wird, stellt mittels R-OSGi eine Verbindung zum anbietenden Knoten her. Der Dienst wird dann von R-OSGi lokal verfügbar gemacht. Auch Benachrichtigungen über beendete Dienste werden durch R-OSGi behandelt.

Die Kommunikation zwischen entfernten Diensten wird von R-OSGi durchgeführt. Es versteht sich, dass auch diese Kommunikation über die Netzwerk-Komponente laufen soll. Dazu stellt das `core`-Bundle eine Implementierung des von R-OSGi beschriebenen Dienstes `NetworkChannelFactory` bereit, durch den Transportkanäle angeboten werden, die Nachrichten über das P2P-Netz verschicken.

3.1.5 Netzwerk

Die Netzwerk-Komponente baut das eigentliche P2P-Netz auf. Sie muss dazu in der Lage sein, einen Knoten innerhalb des Netzes zu finden, sich einem vorhandenen Netz anzuschließen und zu bemerken, dass andere Knoten dem Netz beitreten oder es verlassen. Die Netzwerk-Komponente soll Nachrichten von Komponenten an andere Komponenten

[5] `http://r-osgi.sourceforge.net/`
[6] `http://www.osgi.org/Release4/Download`

oder an Netze zustellen. Es ist außerdem ein Versenden von Nachrichten an Untergruppen vorgesehen, denen Komponenten zum Empfang der Nachrichten vorher beitreten müssen.

Um all diese Aufgaben zu bewältigen, wurde beschlossen, JXTA zu verwenden. JXTA bietet die dafür nötigen grundlegenden Funktionalitäten.

Einführung in JXTA

JXTA ist ein Framework zur Entwicklung von P2P-Anwendungen. Es stellt dem Entwickler frei zugängliche Protokolle und Bibliotheken zur Verfügung, um ein eigenes P2P-Netz aufzubauen. Im weiteren Verlauf dieses Kapitels wird ein grober Überblick über Architektur und Komponenten von JXTA gegeben.

Architektur von JXTA

Die Architektur von JXTA ist in drei Schichten gegliedert: Die Plattformschicht, die Diensteschicht und die Anwendungsschicht.

Plattformschicht
: Die Plattformschicht ist die unterste Schicht. Sie bildet den Kern von JXTA. In ihr werden die grundlegendsten Funktionen wie Such- und Transportmechanismen für das Arbeiten in einem P2P-Netzwerk gekapselt. Diese Funktionalitäten sind für alle P2P-Anwendungen gleich und ermöglichen die Interaktion zwischen JXTA-Anwendungen.

Diensteschicht
: Über der Plattformschicht liegt die Diensteschicht. In dieser werden optionale Funktionalitäten und Dienste festgelegt. Solche Funktionalitäten werden nicht zwangsweise für den Aufbau eines P2P-Netzes benötigt, vereinfachen jedoch das weitere Arbeiten damit. Dort angebotenen Dienste sind beispielsweise der Discovery Service zum Auffinden anderer Knoten oder Authentifizierungsdienste.

Anwendungsschicht
: Die Anwendungsschicht ist die oberste Schicht. Auf dieser Schicht wird eine JXTA nutzende P2P-Anwendung realisiert.

JXTA-Komponenten

JXTA-P2P-Netzwerke organisieren sich in verschiedenen Komponenten, die im Folgenden erläutert werden.

Peers
: Ein P2P-Netzwerk besteht aus teilnehmenden Knoten. Solche Knoten werden Peers genannt und sind normalerweise PCs mit einer Netzwerk- bzw. Internetanbindung. Im Zuge des technischen Fortschritts eignen sich mittlerweise auch andere Geräte

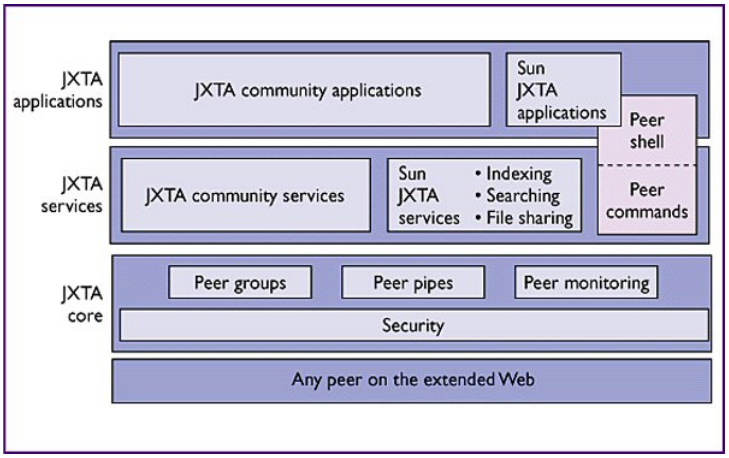


Abbildung 3.2: JXTA-Architektur

wie Handys oder PDAs als Peer. Von JXTA wird ein Peer durch eine eindeutige ID – die Peer-ID – identifiziert.

PeerGroup

Peers können in so genannten Peer Groups organisiert werden. Diese dienen in erster Linie rein organisatorischen Zwecken und bieten zunächst keine weiteren Vorteile. So kann man beispielsweise Gruppen erzeugen, in denen Peers gleiche Interessen, wie den Austausch von Dokumenten, verfolgen.

Als eine besondere Gruppe ist hier die „Net Peer Group“ zu nennen. Sie bildet die globale Gruppe, in der jeder Peer Mitglied ist. Es steht jedem Peer frei, daneben weiteren Gruppen beizutreten. Gruppen sind hierarchisch organisiert; jede neue Gruppe ist Untergruppe einer vorhandenen Gruppe.

Advertisements

Ein Advertisement ist eine Metadaten-Struktur, die von JXTA genutzt wird, um Informationen über Ressourcen (Peers, Gruppen, Pipes, ...) in einem Netzwerk zu repräsentieren und zu verbreiten. Zur Veröffentlichung des Advertisements schickt der Discovery Service dieses an jeden Peer innerhalb des aufgebauten JXTA-Netzes.

Advertisements haben eine bestimmte Lebenszeit, die dazu genutzt wird, veraltete Ressourcen zu entfernen, ohne dazu eine zentrale Kontrollinstanz zu nutzen. Um die Lebenszeit eines solchen Advertisements zu verlängern, muss es neu veröffentlicht werden.

Pipes

Die Kommunikation im JXTA-P2P-Netzwerk funktioniert über Pipes. Sie sind virtuelle Verbindungen zwische Endpoints und werden durch eine JXTA-ID eindeutig identifiziert. Endpoints wiederum kapseln die physischen Kontaktadressen eines Peers und fungieren als Start- und Endpunkt von Nachrichten. Pipes in JX-

TA sind unidirektional. Neben Unicast-Pipes zwischen zwei Knoten stehen auch Propagate-Pipes zur Verfügung, die Nachrichten an eine ganze Gruppe verteilen.

3.1.6 Überwachung der Systemkomponenten

Die Überwachung der Verfügbarkeit von Diensten in ausreichender Anzahl wird im umgesetzten Cloud-System von der Komponente Monitoring übernommen. Eine Erläuterung der Anforderungen, ein Überblick über vorhandene Systeme und ein kurzer Einblick in die Implementierung unserer Lösung werden in den folgenden Abschnitten gegeben.

Definition der Aufgaben des Monitorings

Die erste wichtige Anforderung an den Monitoring-Dienst ist Redundanz des Dienstes: Im Sinne der Ausfallsicherheit ist es nötig, dass nicht nur eine Instanz den Zustand der Cloud überwacht, da bei Wegfall des beherbergenden Knotens kein Monitoring mehr zur Verfügung stünde. Mögliche Ausfälle von Instanzen des Dienstes müssen ohne Komplikationen kompensiert werden, sodass stets ein einsatzfähiger Zustand der Cloud gewährleistet ist.

Zudem muss ein Prozess entwickelt werden, der nach dem Start eines Knotens die für einen nützlich und sinnvoll teilnehmenden Knoten nötigen Dienste startet. Diese Startprozedur (sog. Bootstrapping) sorgt dafür, dass sich der Knoten in die Cloud eingliedert (sich bei den anderen Knoten anmeldet), fehlende Dienste der Cloud selbst bereitstellt oder als Reserve-Knoten fungiert. Das Hauptaugenmerk lag bei der Entwicklung darauf, dass dieses Bootstrapping voll automatisch abläuft, sobald die OSGi-Laufzeitumgebung eines Knotens gestartet wird. Sobald der Knoten erfolgreich gestartet wurde, muss er in die bestehende Cloud eingebunden werden und unter Umständen Informationen über Dienste beziehen, die von anderen Knoten angeboten werden.

Einige Dienste sollen bzw. müssen auf allen Knoten laufen, andere wiederum sind optionale Komponenten, d.h. sie sollen nur auf einem Teil der Knoten oder sogar auf nur einem einzigen Knoten laufen. Die Verteilung der optionalen Dienste im Gesamtsystem lässt sich durch Regeln parametrisieren, die beschreiben, welcher Dienst wie oft unter welchen Voraussetzungen und auf welchem Knoten angeboten wird. Die Einhaltung dieser Regeln ist somit ein weiteres wichtiges Ziel der Monitoring-Komponente.

Um die Einhaltung solcher Regeln sicherzustellen, muss die Monitoring-Komponente Informationen über die Verfügbarkeit von Diensten vorhalten. So ist eine Kontrolle der Regeln in regelmäßigen Abständen ohne großen Kommunikationsaufwand anhand dieser zwischengespeicherten Daten möglich. Die Pflege dieser Daten geschieht dabei als Reaktion auf Benachrichtigungen von der Diensteregistrierung oder über P2P-Kommunikation.

Ein weiterer Vorteil des Vorhaltens von Daten über angebotene Dienste der jeweiligen Cloud-Knoten ist der Informationsgewinn für eine Benutzerschnittstelle. Ohne langwierige *service discovery* kann so eine entsprechende Übersicht angezeigt werden. Weiterhin ist so die Möglichkeit gegeben, zur Laufzeit Dienste zu starten, zu stoppen, zu verbieten oder zu erlauben.

Nagios als mögliches Überwachungs-Framework

Die Umsetzung eines Monitoring-Systems, welches diesen umfangreichen Anforderungen genügt, ist sehr komplex.

Die wohl bekannteste Freeware-Überwachungssoftware, welche in Betracht gezogen wurde, um diese Komplexität geeignet zu erfassen, ist Nagios. Dieses System ermöglicht auf verschiedenen Wegen (Skriptsprachen, Java, C, C++, etc.), den Zustand des Systems abzufragen und gemäß den auftretenden Problemen vorher definierte Lösungen auszuführen. Nagios ist ein sehr umfangreiches Produkt und hochgradig modular aufgebaut, was zunächst für die Verwendung als Überwachungssoftware spricht. Große Nachteile sind allerdings, dass Nagios vorher auf den zu überwachenden Maschinen installiert werden muss, und vor allem, dass Nagios nur für Linux-Systeme verfügbar ist. Da das Ziel der Projektgruppe jedoch eine plattformunabhängige Applikation ist, kommt Nagios für das Projekt nicht in Frage.

Stattdessen wurde – vornehmlich aufgrund der neuen und einzigartigen Struktur unserer Cloud – der Entschluss gefasst, ein eigenes Monitoring-System zu planen und zu implementieren, welches sich gut an unsere Bedürfnisse anpassen lässt.

Umsetzung

Der Regelsatz und die Liste der für den Knoten verbotenen Dienste wird vor Start des Knotens in einer `.properties`-Datei festgelegt. Beim Starten des Knotens werden diese Informationen eingelesen und für eine spätere Verwendung während der regelmäßigen Überprüfung vorgehalten.

Ein lokales Verzeichnis der lokal und entfernt laufenden Dienste wird als Reaktion auf OSGi-Ereignisse aufgebaut, deren Auslösung auch für entfernte Dienste von der verteilten Diensteregistrierung (siehe Kapitel 3.1.4) sichergestellt ist. Die Änderungen wegen neu aufgefundener oder nicht länger verfügbarer Dienste werden auch an alle anderen Monitoring-Dienste in der Cloud kommuniziert.

Damit ein Dienst in der Cloud überwacht werden kann, muss das Bundle den Dienst `Monitorable` als OSGi-Service exportieren und innerhalb des Bundles implementieren. Dieser Dienst bietet der Monitoring-Komponente Informationen, insbesondere die Art des zu überwachenden Dienstes. Dabei ist zwischen zwei Typen der Überwachung zu unterscheiden. Auf der einen Seite ist es möglich Bundles als solche (z.B. JobManager, ResourceManager, UserInterface) zu überwachen. Auf der anderen Seite können Objekte, welche von laufenden Diensten erzeugt und verwaltet werden, überwacht werden. Im Systementwurf sind dies bislang die Objekte „Physikalische Datenbank“ und „FTP Server“, welche nicht in eigene Bundles oder Dienste ausgelagert wurden.

Ein *heartbeat* in festgelegter Periodizität löst eine Überprüfung auf die Einhaltung der eingelesenen und parametrisierten Regeln aus. Wenn die Anzahl der zur Verfügung stehenden Instanzen eines Dienstes eine Regel verletzt, wird versucht, den entsprechenden Dienst auf dem lokalen Knoten zu starten bzw. zu stoppen. Wenn dies nicht möglich ist, wird eine Warnung ausgegeben; ein anderer Knoten wird innerhalb seines *heartbeats* das Fehlen von bzw. einen Überschuss an Dienstinstanzen bemerken und entsprechend

reagieren.

Insbesondere während des Bootstrappings, während dessen normalerweise noch keine anderen Knoten bekannt sind, kommt es vermehrt zu Regelverletzungen. Da z.B. einige Dienste mindestens zweimal laufen sollen, aber ein Knoten immer maximal eine Instanz eines Dienstes anbieten soll, werden alle diese Regeln verletzt. Erst wenn ein zweiter Knoten gefunden und gebunden wird, können diese Regeln erfüllt werden.

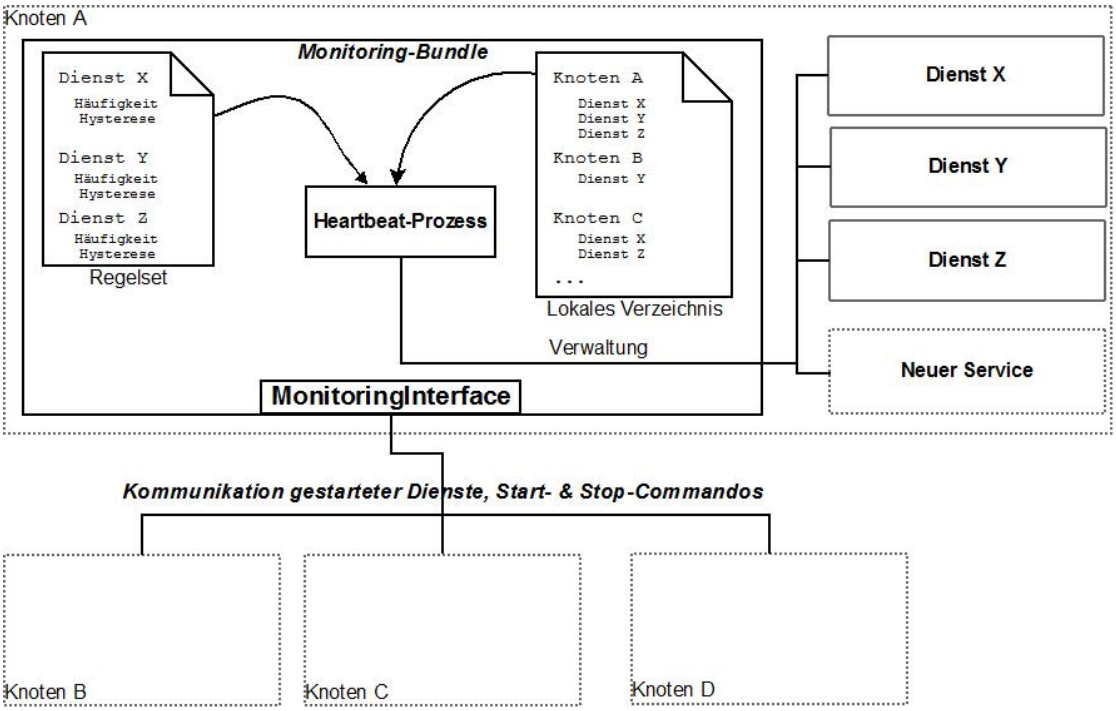


Abbildung 3.3: Auslegung der Monitoring-Komponente – ein Monitoring-Dienst auf jedem Knoten verarbeitet Informationen von entfernt liegenden Monitoring-Peers und hat Zugriff auf die lokale OSGi-Runtime.

Da die Definition solcher Regeln beliebig komplex werden kann, wurden einige vereinfachende Annahmen getroffen. Insbesondere werden die Cloudknoten als gleichwertig angenommen; es ist z.B. nicht möglich, einen Dienst abhängig von Hardwareausstattung oder Netzwerkzugang zu erlauben oder zu verbieten.

Die Regeln in der vorliegenden Umsetzung geben eine Vorschrift für die Anzahl von Dienstinstanzen in der Cloud, die verschiedene Formen annehmen kann:

log n:
: Dieser Richtwert bedeutet, dass $log_2 n$ Instanzen des Dienstes verfügbar sein sollen, wobei n die Anzahl der Knoten in der Cloud ist.

fix x:
: Es wird angestrebt, x Instanzen des Dienstes verfügbar zu halten.

percent x:
Der Dienst soll auf x Prozent der Knoten verfügbar sein.

Zusätzlich zu den Häufigkeitswerten kann für jeden Dienst ein Hysteresewert gesetzt werden, so dass die vorgegebenden Werte nicht als harten Grenzen interpretiert werden. Das Monitoring handelt erst dann, wenn die Anzahl der Dienstinstanzen auf den Richtwert +/- den Hysteresewert sinkt bzw. steigt. Dies verhindert ungewollte Dynamik beim Starten bzw. Stoppen von Diensten, macht jedoch das Reagieren auf kleinere Ausfälle weniger direkt.

3.2 Datenbank

Schon früh in der Planungsphase wurde deutlich, dass eine Möglichkeit der Datenhaltung benötigt wird, um den Zustand des Systems zu persistieren. Da die Cloud gegen den Ausfall einzelner Knoten robust sein soll, ist sogar eine redundante verteilte Datenhaltung erforderlich.

Demnach ist eine *verteilte* Datenbank zur persistenten Speicherung von Daten ein zentraler Bestandteil der Cloud. Im Folgenden wird näher auf die Anforderungen an eine solche Datenbank eingegangen. Außerdem werden mögliche Umsetzungen untersucht und kurz die Implementierung der gewählten Lösung erläutert.

3.2.1 Anforderungen an die Datenbank

Aus den entwickelten Entwürfen für das Gesamtsystem resultierte, dass die Datenbank eine zentrale Rolle in der Systemarchitektur einnehmen wird. Es müssen Daten, die den Zustand der Cloud repräsentieren, persistiert werden. Gespeichert werden dabei zum Einen die persönlichen Daten und Rechte der angemeldeten Nutzer. Neben einer Reihe von Standardattributen soll also auch ein gewisses Rechtemanagement auf die Datenbank abgebildet werden. Zum Anderen enthält die Datenbank Daten zu den Jobs, z.B. den Zustand und die Resultate sowie Beschreibungen. Desweiteren sollen Daten zu den einzelnen Knoten persistiert werden, z.B. welche Knoten aktiv in der Cloud vertreten sind bzw. waren.

Die Erweiterbarkeit des Attributumfangs in den einzelnen Bereichen (wie User-, Job- und Cloudverwaltung) war eine weitere wichtige Anforderung an die Datenbank. Dazu musste ein gut erweiterbares Datenbank-Layout geschaffen werden, so dass eventuelle zukünftige Anforderungen an die Cloud umsetzbar sind. Die Austauschbarkeit der Datenbank-Software sollte gewahrt sein, es sollte aber auch die Möglichkeit offen gehalten werden, künftig ein Objekt-relationales Mapping (ORM) zu verwenden, welches einen komfortableren Zugang zu Daten in der Datenbank bereitstellt. Gerade aus Gründen der Austauschbarkeit der Backends, aber auch wegen eventuell gewünschter Verwendung des ORM-Frameworks *Java Hibernate* fiel die Wahl auf JDBC. Ein JDBC-Treiber existiert zudem für so gut wie jedes Datenbank-Backend und kann einfach eingebunden werden.

Bei hochfrequenten Anfragen an die Datenbank muss zudem eine schnelle Behandlung von Lese-Anfragen gewährleistet sein. Ein absehbares Problem in diesem Zusammenhang

ist die zu erwartende schlechte Skalierbarkeit von Schreibanfragen bei wachsender Größe des Datenbank-Clusters, da Schreib-Locks auf den verwendeten Datenbanken gesetzt werden müssen.

Durch die Zielsetzung, eine dynamische Compute Cloud zu programmieren, ist Replikation eine zentrale Anforderung an die Datenbank. Es sollte sowohl möglich sein, Datenbankknoten dynamisch zu initialisieren und zur Verwendung einzugliedern, als auch überflüssig gewordene Knoten wieder aus der Cloud zu entfernen.

Eine der wichtigsten Anforderungen an die Datenbank ist jedoch die Hochverfügbarkeit. Spontan ausfallende Datenbank-Teile sollen so kompensiert werden können, dass weder Datenverlust noch merkliche Performanceeinbrüche zu verzeichnen sind. Außerdem ist es wünschenswert, die Daten so zu verteilen, dass Datenverluste ausgeschlossen werden können und die Integrität der Daten jederzeit gewährleistet ist.

Eine Möglichkeit hierzu wäre ein Striping-Verfahren, bei dem nicht alle Daten immer auf jeder Datenbank vorhanden sind, sondern in ausreichender Anzahl auf verschiedene Datenbanken aufgeteilt gelagert werden. Diese Möglichkeit wurde jedoch aufgrund der nicht abzuschätzenden Komplexität der Implementierung und mangels vorhandener Frameworks verworfen.

Eine andere Möglichkeit ist eine volle Spiegelung einzelner Datenbankknoten. Leseanfragen werden dabei nach dem Round-Robin-Prinzip auf die verwendeten Datenbanken verteilt, Schreibbefehle werden immer auf allen Datenbanken ausgeführt, um eine synchrone Datenhaltung zu gewährleisten. Dieses Vorgehen wurde in den weiteren Betrachtungen verfolgt.

3.2.2 HA-JDBC-Framework

In der Seminarphase wurden bereits erste Nachforschungen zu möglichen verteilten Datenbanken durchgeführt. Ein vielversprechendes Framework für eine verteilte Datenbank unter Java ist das Framework High-Availability Java Database Connection (HA-JDBC). HA-JDBC ist dabei eine Art Metaschicht (siehe Abbildung 3.4) zwischen den aufrufenden Java-Applikationen und physikalisch im Netz vorhandenen Datenbanken. Die Konfiguration und damit insbesondere, welche Datenbanken verfügbar sind, lässt sich initial durch eine XML-Datei beschreiben und zur Laufzeit verändern. Generell kann HA-JDBC mit jeder Datenbank kommunizieren, für die es einen entsprechenden JDBC-Datenbanktreiber gibt. Da dies für nahezu jede auf dem Markt verfügbare Datenbank der Fall ist, blieb das Hauptaugenmerk auch auf diesem Framework.

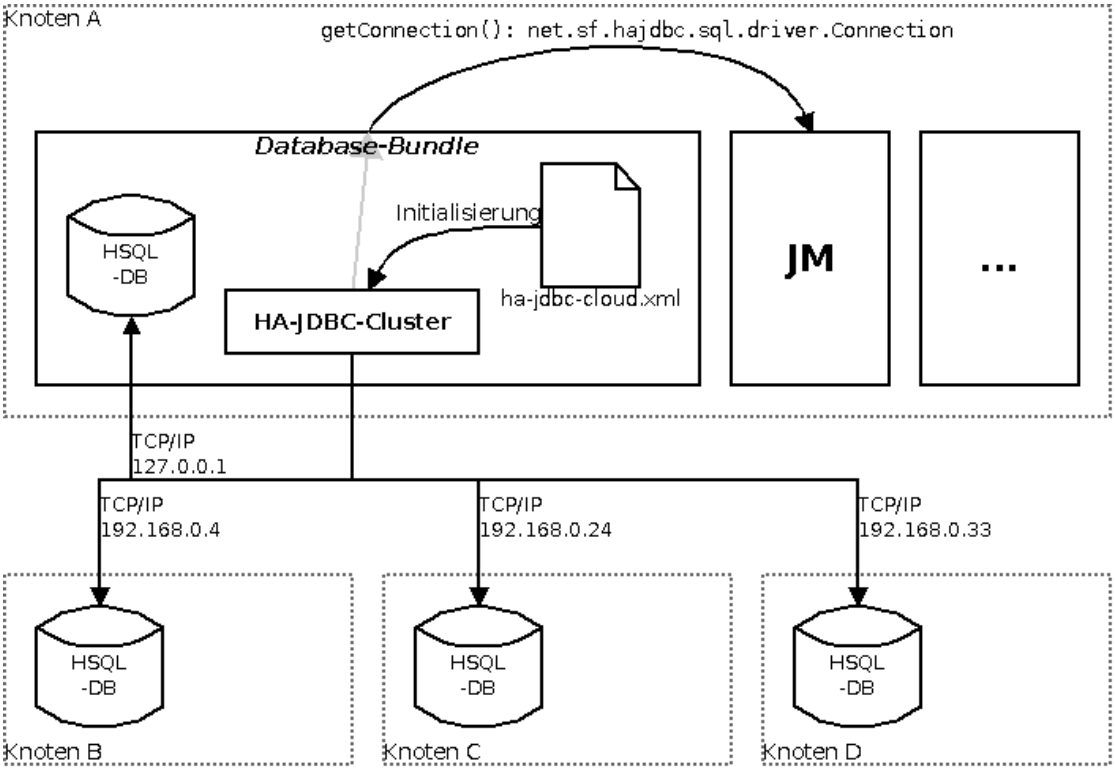


Abbildung 3.4: Die Datenbank-Komponente bietet anderen Komponenten auf dem Knoten eine JDBC-Connection zur verteilten Datenbank an.

Desweiteren bietet dieses Framework die Möglichkeit, eigene Synchronisations-Strategien zu definieren und geeignet zu verwenden. Dieses Feature wurde untersucht, erweitert und anschließend genutzt. So entstand beispielsweise eine neue Synchronisationsstrategie, die *FastSynchronizationStrategy*, welche nur die relevanten Daten synchronisiert und nicht diejenigen, welche ohnehin schon bei der Initialisierung in der Datenbank eingetragen sind.

Wird eine neue Datenbank in das HA-JDBC-Cluster eingepflegt, nutzen alle Knoten der Cloud zusätzlich diese neue Datenbank. Um das eben beschriebene Verhalten umzusetzen, ist es nötig, effiziente Abstimmungsalgorithmen zu implementieren, um Race-Conditions und potentiell asynchrone Datenbanken zu vermeiden. Zudem müssen Informationen über die neue Datenbank wie Name und Ort zwischen den einzelnen Knoten kommuniziert werden. Die von HA-JDBC zu diesem Zweck benutzte Bibliothek JGroups stellt eine Schwachstelle dar, da die Verwendung in einem P2P-Netz nur mit sehr viel Arbeitsaufwand implementiert werden kann. Daher wurde ein eigenes Verfahren zur Verteilung der relevanten Daten in der Cloud implementiert, welche die Netzwerk-Komponente verwendet (u.A. Peergroup-Kommunikation und R-OSGi-Service-Aufrufe).

Die oben genannten Argumente zeigen, dass HA-JDBC für dieses Projekt sehr vielversprechend ist. Nach einer Machbarkeitsstudie, in der eine Testimplementation zeigte, dass sich das Framework für das Projekt eignet, wurde eine Implementierung der An-

forderungen durchgeführt.

Der Entwurf der Systemarchitektur ist nur bis zu einem gewissen Grad an HA-JDBC gebunden, sodass mit vergleichsweise wenig Aufwand auch andere Lösungen erarbeitet werden können. Das System ist nicht zuletzt durch die Verwendung von OSGi so modular gestaltet, dass der Austausch von Technologien mit vertretbarem Aufwand umgesetzt werden kann. Sollte in Zukunft also festgestellt werden, dass Sequoia, Terracotta oder eine bislang nicht betrachete Technologie trotz der oben angeführten Gegenargumente besser geeignet ist, wird so ein Austauschen der Datenbank-Lösung ohne größere Änderungen am restlichen System möglich.

HA-JDBC skaliert bei Schreibzugriffen linear mit der Anzahl der verwendeten Datenbank-Backends, während Lesezugriffe mit wachsender Zahl von Backends durch die Verteilung der Anfragen performanter werden. Initial werden in der Cloud mindestens zwei Backends verwendet, um Datenverlust bei Ausfall eines Knotens zu vermeiden. Wieviele physikalische Backends jedoch tatsächlich Verwendung finden werden, ist von vielen Faktoren abhängig und sollte von Fall zu Fall entschieden werden. Voreingestellt wurden in den Tests $log_2 n$ physikalische Datenbanken in der Anzahl der Cloud-Knoten.

3.2.3 Alternativen: Tungsten's Sequoia und Terracotta

Obwohl HA-JDBC viele der Anforderungen zu erfüllen verspricht, wurden zwei Exkurse in weitere Technologien durchgeführt, um diese zu evaluieren und auf Tauglichkeit für das Projekt hin zu untersuchen.

Zunächst betrachtet wurde die ebenfalls hoch verfügbare Datenbank-Middleware *Tungsten's Sequoia*[7], die selbst im Vergleich zu HA-JDBC einen überwältigenden Funktionsumfang bereitstellt. Dem Anspruch, einen möglichst leichtgewichtige Cloud-Knoten zu erzeugen, konnte diese Lösung allerdings nicht standhalten. Vor allem, dass die Datenbank-Middleware auf einem Rechner fest installiert werden muss, sprach gegen deren Verwendung. Es wären also weitere Installationschritte zum Hinzufügen eines Knotens zur Cloud notwendig.

Als weitere Alternative, welche aber vollständig auf Datenbanken im herkömmlichen Sinne verzichtet, wurde das Framework Terracotta[8] in Betracht gezogen. Dieses Framework bietet eine Art verteilte Objektverwaltung, in der keine Tabellen und Verknüpfung zur Speicherung von Daten verwendet werden. Stattdessen werden Java-Objekte auf sogenannten Terracotta-Servern direkt gespeichert, und auch der Zugriff auf die Objekte wird über die Terracotta-Server gesteuert. Das ausschlaggebende Argument gegen den Terracotta-Ansatz war, dass erst die kostenpflichtige Version das zentrale Feature – die Replikation von Servern zur Laufzeit – bereitstellt.

3.2.4 hsqldb als Datenbank-Backend

An Abbildung 3.4 ist zu erkennen, dass HA-JDBC nur als Meta-Schicht zu physikalischen Datenbanken fungiert. Diese im Folgenden als Datenbank-Backends bezeichneten

[7] http://www.continuent.com/community/lab-projects/sequoia
[8] http://www.terracotta.org/

Komponenten des Systems sollen folgende Anforderungen erfüllen:

- vorhandener und frei verfügbare JDBC-Datenbanktreiber
- leichtgewichtig
- Möglichkeit einer persistenten Datenspeicherung (z.B. als Datei)
- zur Laufzeit start- und abschaltbar

Nach einigen Nachforschungen wurde die Open-Source-Datenbank hsqldb ausgewählt, welche unter der zum Zeitpunkt der Projektarbeit aktuellsten Version 1.8.10 Verwendung fand. Die Datenbank hsqldb hat verschiedene Ausführungsmodi, darunter auch die Möglichkeit, die Daten zur Laufzeit ausschließlich im Arbeitsspeicher des Rechners vorzuhalten und nur beim Abschalten auf die Festplatte zu schreiben. Dieses Verhalten wurde getestet und übernommen. Die schnelle Instantiierung bringt die für ein dynamisches System nötige Performanz und dieser Modus kommt zudem völlig ohne Installation einer Datenbanksoftware auf dem jeweiligen Rechner aus.

Die Daten, welche das Datenbank-Backend speichert, werden beim Abschalten der Datenbank in eine Datei geschrieben und dort persistiert. So ist die Möglichkeit eines späteren Backups gegeben, wobei dieser Aspekt im Rahmen der Projektgruppe nicht weiter verfolgt wurde.

Bevor die gestarteten Datenbanken dem Cluster hinzugefügt werden, müssen sie initialisiert werden, d.h. es müssen sowohl die nötigen SQL-Befehle zur Erzeugung und Verknüpfung der Tabellen ausgeführt werden, als auch benötigte Datensätze eingefügt werden. Außerdem muss eine Synchronisation mit einer in Verwendung befindlichen Datenbank durchgeführt werden, bevor den anderen Knoten des Clusters kommuniziert wird, dass eine neue Datenbank im Cluster aufgenommen wurde. Diese müssen die Datenbank in das Cluster der selbst verwendeten Datenbanken aufnehmen und nutzen. Erst nach der Eingliederung in das Cluster ist es möglich, diese Datenbank zu nutzen.

3.2.5 Modularisierung und Einbindung in die OSGi-Runtime

Die Datenbank-Komponente ist eine in sich geschlossene Einheit und kann jederzeit gegen andere Implementierungen ausgetauscht werden. Aus diesem Grund beinhaltet das Bundle unter anderem die zwei folgenden Bestandteile:

Package `edu.udo.irf.pg538.database`
: stellt die Dienstschnittstellen bereit, über die andere lokale Dienste die Datenbank-Funktionalität nutzen können.

Package `edu.udo.irf.pg538.database.internal`
: beinhaltet die tatsächliche Implementierung der im übergeordneten Package zur Verfügung gestellten Schnittstellen. Sie kann ausgetauscht werden, z.B. zur Verwendung anderer Technologien.

Der für andere (lokale) Dienste wichtigste Bestandteil ist die Dienstschnittstelle `Database`. Dieses definiert den vom Bundle zur Verfügung gestellten Dienst. Darunter befinden sich z.B. Methoden zum Bereitstellen einer JDBC-Verbindung zum HA-JDBC-Cluster oder die Funktionen für Replikation, Initiierung und Verwaltung physikalischer Datenbankprozesse. Zu beachten ist dabei, dass dieser Datenbank-Dienst lediglich innerhalb der lokalen OSGi-Runtime zur Verfügung steht und der Zugriff auf eine entfernt laufende Datenbank-Komponente nicht vorgesehen ist.

Eine wichtige Funktion ist das Vorhalten einer Verbindung zum verteilten Datenbank-Cluster. Je Knoten wird stets die gleiche JDBC-Datenbankverbindung genutzt (sog. Singleton-Verbindung). Diese Verbindung wird immer dann genutzt, wenn Daten aus der Datenbank für den lokalen Knoten gebraucht werden oder Datenänderungen geschrieben und allen anderen Cloudknoten zur Verfügung gestellt werden müssen. Ein absehbares Problem ist auch hierbei die schlechte Skalierbarkeit bei vielen physikalischen Datenbanken.

3.2.6 Datenbanklayout

Das Datenbanklayout ist sowohl auf gute Erweiterbarkeit im Verlauf der Entwicklung als auch auf effiziente Datenbankoperationen ausgelegt. Die schematische Auslegung ist in Abbildung 3.5 zu sehen. Die Erweiterbarkeit ist durch die dritte Normalform des Datenbank-Schemas gegeben.

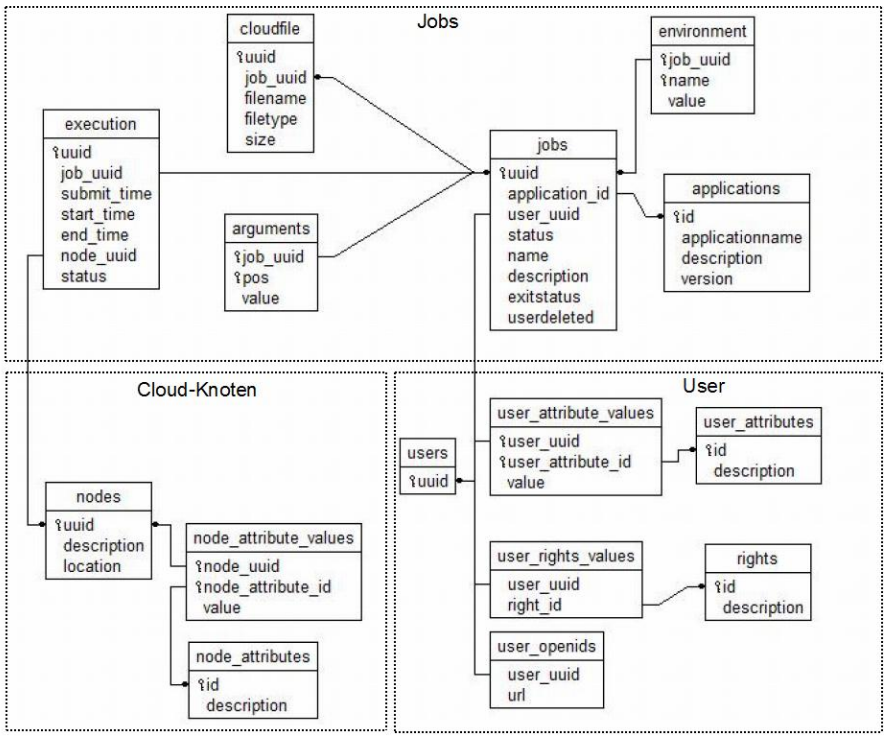


Abbildung 3.5: Datenbank-Setup in der grafischen Übersicht.

Generell kann man die Datenbanktabellen und deren Einträge in verschiedene Gruppen einteilen:

Jobs
: persistiert Daten der verschiedenen Jobs im System. Die Tabelle `execution` beinhaltet dabei Informationen darüber, welcher Job wann und mit welchem Ergebnis ausgeführt wurde. Sollte ein Job mehrmals ausgeführt werden, gibt es hier mehrere Einträge zu dem Job.

Users
: Daten der Nutzer sowie deren Zugangsdaten und Rechte im System.

Cloud-Knoten
: Beinhaltet Informationen über die momentan in der Cloud vorhandenen Knoten. Die Tabelle `nodes` wird dabei als Grundlage für eine Übersicht der Cloud in der Administrationsoberfläche genutzt.

Unter Verwendung der eigens entwickelten *FastSynchronizationStrategy* werden nur einige wenige der oben dargestellten Tabellen synchronisiert, da sich die Inhalte vieler Tabellen gleichen. Dies spart Zeit beim Synchronisationsvorgang und erhöht die Dynamik des Systems.

3.2.7 Setup des HA-JDBC-Clusters

Das Standardverhalten beim Anfordern der JDBC-Verbindung zum HA-JDBC-Clusters ist, dass zunächst die lokale System-Registrierung (auf Windows-Systemen) bzw. eine Datei im `home`-Verzeichnis des Benutzers (auf Linux-Systemen) auf Einträge bezüglich des letzten gültigen HA-JDBC-Zustands durchsucht wird. Ein solcher Eintrag wird erstellt, sobald in einem Programmablauf einmal eine Verbindung zum Cluster hergestellt und verwendet wurde. Wird ein entsprechender Eintrag gefunden, wird er auch in der aktuellen HA-JDBC-Instanz verwendet. Dies hat laut Dokumentation von HA-JDBC den Sinn, einen vergangenen Zustand des Clusters schnell wieder zu verwenden, anstatt die in der XML-Datei angegebenen Datenbanken zu suchen und zu überprüfen, ob diese antworten.

Dieses Verhalten ist allerdings eher unerwünscht, da einerseits ein sehr dynamisches Cluster mit häufig wechselnden beteiligten DB-Backends verwaltet wird und andererseits bei der Initialisierung wegen eingetragener, aber nicht antwortender Datenbanken SQL-Exceptions erzeugt werden. Um dieses Problem zu umgehen, wird der entsprechende Schlüssel der Registrierung bzw. der Eintrag im `home`-Verzeichnis vor dem Start des Clusters entfernt.

Im Folgenden wird die initiale Auslegung der hochverfügbaren Datenbank durch die zugehörige XML-Datei (siehe Quelltext 3.1) näher beleuchtet. Die angegebenen Daten in dieser Datei weichen stark von der Standardkonfiguration ab, welche für HA-JDBC vielfach vorgeschlagen wird. Es sind keinerlei Datenbanken in das Cluster fest eingetragen. Dies ist dem Umstand geschuldet, dass Informationen über weitere Datenbank-Backends

sowie Verbindungen dahin zum Startzeitpunkt der Datenbank-Komponente nicht vorliegen. Diese Information muss erst im Laufe des Startup-Prozesses gesammelt werden. Dazu wird bei der Aktivierung der Komponente eine Nachricht an alle Knoten der Cloud gesendet. Alle Knoten, auf denen ein Datenbank-Backend läuft, antworten mit Daten, die zur Eingliederung in das Cluster notwendig sind. Dem reibungslosen Ablauf zuträglich ist dabei der Umstand, dass standardmäßig die *passive* Synchronisation verwendet wird, bevor die jeweiligen Datenbank-Backends im lokal verwalteten Cluster aktiviert werden. Dies bedeutet, dass in der Startup-Phase des Knotens lediglich die Datenbanken eingetragen werden ohne eine Synchronisation durchzuführen. Die Authentifikation an der Datenbank erfolgt der Einfachheit halber über die Standardwerte der verwendeten Datenbank-Software (bei hsqldb mit dem Login „SA“ und einem leeren Passwort).

Umschlossen werden die einzelnen Datenbanken vom `<cluster>`-TAG. Die wichtigsten Attribute, welche bewusst gesetzt werden, sind das Attribut `balancer` und `Failure-Detect-Schedule`. Während der Balancer die Vorgehensweise bei Leseanfragen steuert (in diesem Fall werden Leseanfragen im Round-Robin-Prinzip an jeweils eine Datenbank im Cluster gestellt), beschreibt der Ausdruck im Failure-Detect-Schedule-Attribut eine minütliche Prüfung auf ausgefallene Datenbanken. Das HA-JDBC-Cluster ist so konfiguriert, dass es bei einem erkannten Ausfall einer der Datenbanken diese lediglich ignoriert, d.h. von Lesebefehlen ausspart und Schreibbefehle nicht mehr an diese schickt.

```
<ha-jdbc>
        <sync id="passive" class="net.sf.hajdbc.sync.
           PassiveSynchronizationStrategy">

        </sync>
        <sync id="fast" class="edu.udo.irf.pg538.database.internal
           .synchronization.FastSynchronization">
                <property name="fetchSize">1000</property>
                <property name="maxBatchSize">1</property>
        </sync>
        <cluster balancer="round-robin" default-sync="passive"
           dialect="hsqldb" meta-data-cache="eager"
                        transaction-mode="parallel" failure-
                           detect-schedule="0 * * ? * *" >
        </cluster>
</ha-jdbc>
```

Listing 3.1: Konfigurationdatei: `ha-jdbc-cloud.xml`

3.2.8 JDBC über JXTA

Obwohl die Kommunikation der Komponenten in der Cloud über R-OSGi und JXTA erfolgt, werden Datenbankanfragen je nach verwendeten Treiber weiterhin über den normalen TCP/IP-Stack durchgeführt. So verwendet der HSQLDB-JDBC-Treiber normale Java-Socket-Verbindungen über TCP/IP, um auf eine Datenbank auf einem entfernten

Rechner zuzugreifen. Es wäre jedoch wünschenswert, dass die JDBC-Kommunikation auch Verbindungen des aufgebauten P2P-Netzes benutzt.

Für die Realisierung einer Kommunikation von JDBC über R-OSGi bzw. JXTA wurden verschiedene Ansätze betrachtet, dargestellt in der folgenden Abbildung 3.6.

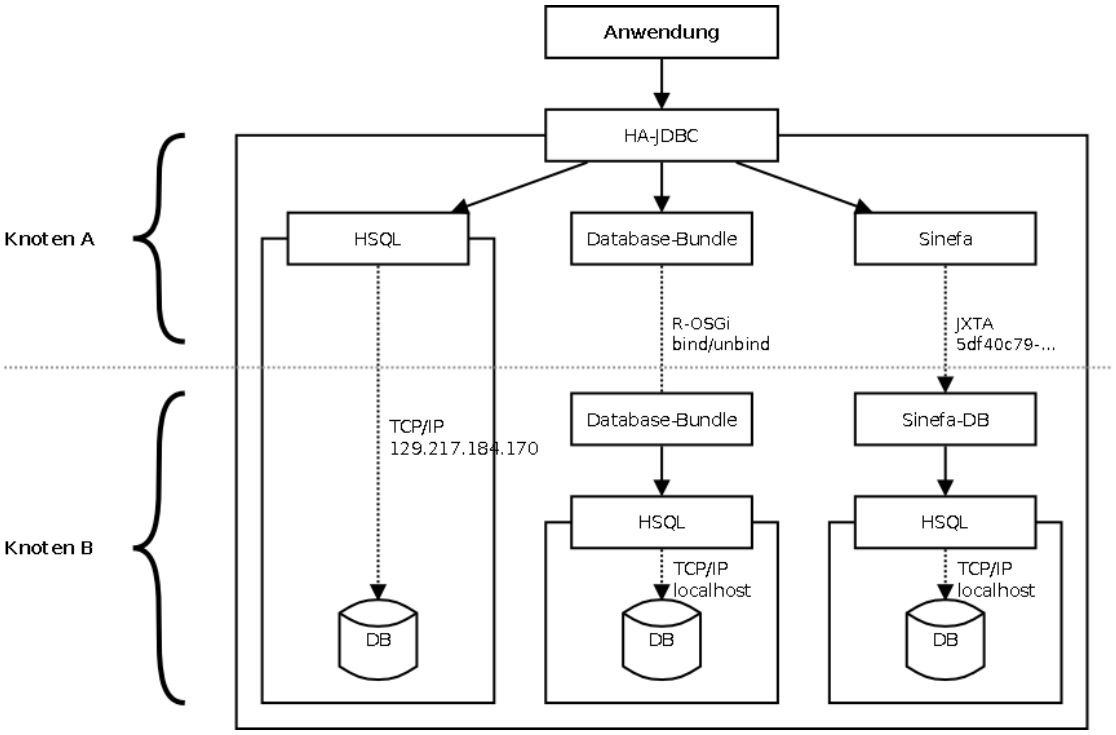


Abbildung 3.6: verschiedene Ansätze für eine JDBC-Kommunikation über R-OSGi oder JXTA

Auf der linken Seite ist der Aufbau der JDBC-Kommunikation dargestellt, wenn diese nicht über R-OSGi bzw. JXTA getunnelt wird. Der HSQL-JDBC-Treiber baut eine direkte TCP/IP-Verbindung mit dem Zielrechner auf, ohne das JXTA-Netz zu verwenden. Im mittleren Teil ist zu sehen, wie R-OSGi dazu verwendet wird, die Datenbank-Komponenten der anderen Knoten zu verwenden und so über R-OSGi-Mittel auf eine externe Datenbank zuzugreifen. Die rechte Version zeigt, wie ein eigener JDBC-Treiber die Datenbank-Anfragen über JXTA tunnelt.

Direkte Kommunikation über TCP/IP

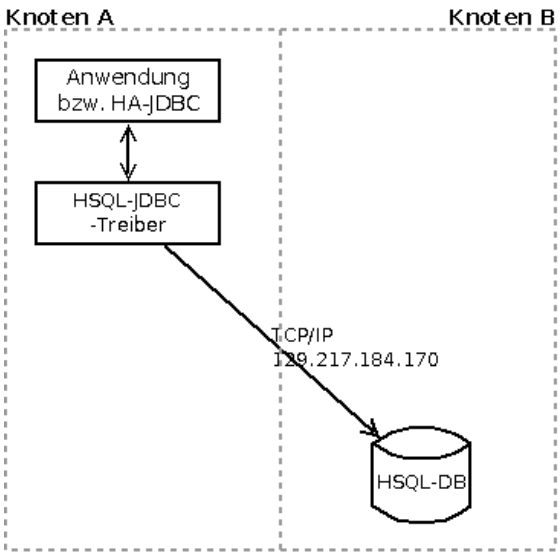


Abbildung 3.7: Zugriff auf eine externe Datenbank mit dem HSQL-JDBC-Treiber

Abbildung 3.7 zeigt die JDBC-Kommunikation ohne R-OSGi oder JXTA. Auf Knoten A wird eine JDBC-Verbindung zu der URL `jdbc:hsql://129.217.184.170` aufgebaut. Der HSQL-JDBC-Treiber liest die Komponenten der URL aus und stellt anhand der IP eine Verbindung zur externen Datenbank her. Die erste Idee für eine JDBC-Kommunikation über JXTA war es, dem HSQL-JDBC-Treiber eine neue Transportschicht „unterzuschieben“, so dass dieser nicht den TCP/IP-Stack mit der angegebene IP-Adresse verwendet, sondern den Verbindungsaufbau über JXTA durchführt. Jedoch bietet der HSQL-JDBC-Treiber nicht die Möglichkeit, die Transportschicht auszuwechseln; dieser Ansatz ist somit nicht durchführbar.

Automatische Erstellung von Proxy-Objekten via R-OSGi

Der zweite Versuch war es, die JDBC-Objekte auf den entfernten Knoten anlegen zu lassen und den Zugriff auf diese via R-OSGi an die anderen Knoten weiterzureichen. Abbildung 3.8 zeigt den genauen Aufbau.

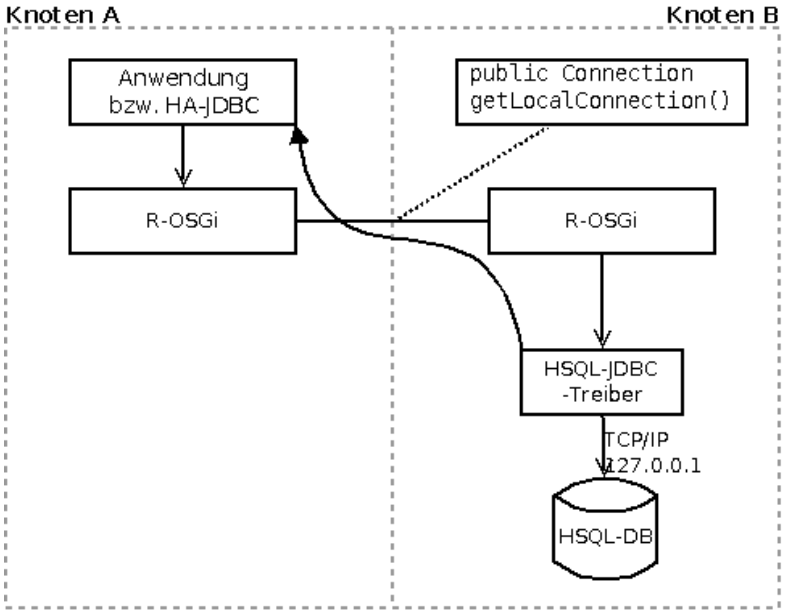


Abbildung 3.8: JDBC-Objekte werden via R-OSGi übertragen

Auf dem Knoten mit der Datenbank wird eine normale JDBC-Verbindung zu localhost bzw. 127.0.0.1 aufgebaut. Diese JDBC-Verbindung wird durch die Methode `getLocalConnection()` der Datenbank-Komponente aufgebaut und dann an den aufrufenden Knoten A zurückgeliefert (angedeutet durch den geschwungenen Pfeil in der Abbildung). Mit diesem JDBC-Objekt könnte nun Knoten A wie gewohnt auf eine Datenbank zugreifen; dass es sich um eine entfernte Datenbank handelt, bleibt bei der weiteren Benutzung verborgen.

Dieser Ansatz ist jedoch aus technischen Gründen nicht durchführbar. R-OSGi erstellt automatisch Proxy-Objekte für externe Dienste, von denen Methodenaufrufe an den entfernten Dienst weitergereicht werden. Parameter und Rückgabewerte dieser Methoden werden automatisch serialisiert und zwischen den R-OSGi-Instanzen ausgetauscht. Verständlicherweise sind JDBC-Connection-Objekte nicht serialisierbar, da sie z.B. Socket-Objekte zur Verbindung mit der Datenbank besitzen. Dieser Ansatz konnte daher nicht verfolgt werden.

Externe Java-Bibliotheken zum Tunneln von JDBC-Anfragen

Da die HSQL-JDBC-Objekte lokal erstellt werden müssen, aber nicht übertragen werden können, müssen diese Objekte weiterhin lokal gehalten werden. Der Knoten mit der Datenbank muss nun die Methodenaufrufe der anderen Knoten entgegen nehmen und lokal ausführen, vergleichbar mit Remote Method Invocation. Für dieses Problem gibt es schon implementierte Lösungen wie z.B. Virtual JDBC.

Leider konnte die Virtual JDBC-Bibliothek nicht verwendet werden, da auch hier das verwendete Transportprotokoll nicht ausgewechselt werden kann.

Eigener JDBC-Treiber als Proxy für JDBC-Objekte

Obwohl Virtual JDBC nicht verwendet werden konnte, hat es Ideen für ein eigenes System geliefert. Abbildung 3.9 zeigt den Aufbau des JDBC-Treibers, wie er letztendlich in der Datenbank-Komponente implementiert wurde.

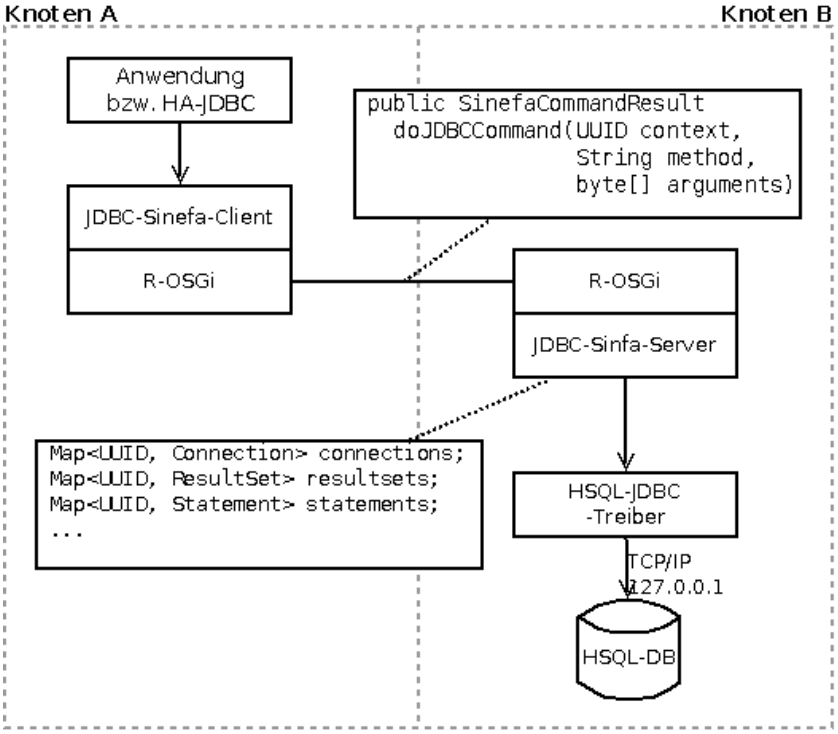


Abbildung 3.9: JDBC über R-OSGi tunneln

Wie bei Virtual JDBC besteht der JDBC-Treiber aus 2 Teilen. Der JDBC-Sinefa-Server-Teil läuft auf den Knoten mit der Datenbank und besitzt eine Liste aller lokal erstellten JDBC-Objekte. Jedes dieser Objekte wird einer UUID zugewiesen um die verschiedenen JDBC-Objekte auseinander zu halten. Die Datenbank-Komponente stellt eine neue Methode `doJDBCCommand(UUID,String,byte[])` bereit, um eine Methode auf so einem JDBC-Objekt ausführen zu lassen. Dabei wird als erster Parameter die Objekt-UUID übermittelt. Der String-Parameter gibt die Methode an, die aufgerufen werden soll, und das Byte-Array enthält die serialisierten Parameter der Methode. Je nach Methode wird ein Byte-Array mit der serialisierten Antwort zurückgeliefert (z.B. ein String bei `ResultSet.getString`) oder eine UUID eines erstellten JDBC-Objekts (z.b. ein Statement-Objekt von `Connection.prepareStatement`).

Im Client-Teil wird für jedes externe JDBC-Objekt ein lokales JDBC-Objekt mit der dazugehörigen UUID erstellt. Diese JDBC-Objekte können von der Anwendung wie gewohnt verwendet werden, da sie die entsprechenden `java.sql.*`-Schnittstellen implementieren. Diese Methoden führen jedoch selbst keine JDBC-spezifischen Aktionen

durch. Stattdessen werden bei einem Methodenaufruf die Parameter serialisiert und mit der UUID an die `doJDBCCommand(UUID,String,byte[])`-Methode übergeben. Die Antwort wird ausgewertet und an den Aufrufer zurückgeliefert (z.B. ein String-Objekt bei der `ResultSet.getString`-Methode).

Es hat sich gezeigt, dass es nicht nötig ist, die Weiterleitung aller 630 Methoden der `java.sql.*`-Schnittstellen zu implementieren. Die Komponenten der Cloud verwenden nur einen kleinen Teil dieser Methoden (ca. 120). Daher konnte während der Entwicklung des eigenen JDBC-Treibers die JDBC-Kommunikation in der Cloud zeitnah auf Nutzung des P2P-Netzes umgestellt werden.

3.3 Der Ausführungsdienst: Resource-Manager

Die Komponente Resource-Manager ist für die Ausführung von Jobs auf den Knoten des Cloud-Systems zuständig. Dabei nimmt der Resource-Manager einen Job vom Job-Manager entgegen, führt ihn aus und meldet nach vollständiger Ausführung das Ergebnis der Bearbeitung an den Job-Manager (siehe Abbildung 3.10).

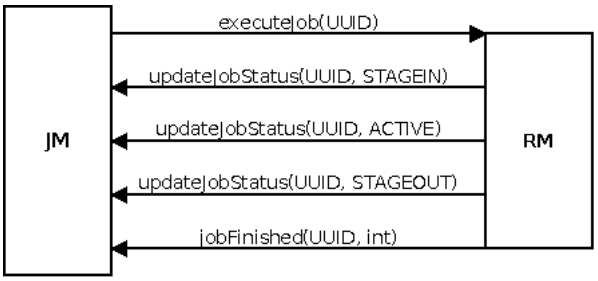


Abbildung 3.10: Methodenaufrufe zwischen Job-Manager und Resource-Manager

Die Jobs werden über die Methode `ResourceManager.executeJob(UUID)` eingereicht. Der Resource-Manager findet die zugehörige Jobbeschreibung anhand der UUID in der Datenbank. Danach wird der Job in einem neuen Thread ausgeführt. Dieser Thread erzeugt für die Ausführung diverse Verzeichnisse und Dateien mit der folgenden Struktur:

`JOB_TMPDIR/`
: Für jeden Job wird ein Verzeichnis, in dem weitere Unterverzeichnisse und Dateien für die Ausführung angelegt werden, im systemspezifischen Temp-Verzeichnis angelegt. Nach der Ausführung wird dieses Verzeichnis wieder gelöscht.

`JOB_TMPDIR/workdir/`
: Dies ist das Arbeitsverzeichnis für das auszuführende Programm. Zu Beginn der Ausführung enthält es eventuelle Eingabedateien zum Job; das ausgeführte Programm kann während seiner Berechnung zusätzliche Dateien ablegen. Diese Dateien werden nach der Ausführung automatisch in das verteilte Dateisystem übertragen.

`JOB_TMPDIR/pipes/`

In diesem Verzeichnis werden die 3 Standardströme STDIN, STDOUT und STDERR abgelegt bzw. ausgelesen. Da die Programme nicht über eine Shell ausgeführt werden, sind Umleitungen der Form `1>datei` in der auszuführende Programmzeile nicht möglich. Daher werden die Dateien durch weitere Threads kontinuierlich durch die Ausgabe des Programms (STDOUT und STDERR) befüllt. Die `STDIN`-Datei wird durch den Ausführungs-Thread angelegt und als Eingabestrom für das Programm verwendet.

Der Resource-Manager fordert vor der Ausführung alle benötigten Dateien zu einem Job aus dem verteilten Dateisystem an. Diese legt er in den Verzeichnissen `JOB_TMPDIR/workdir/` bzw. `JOB_TMPDIR/pipes/` ab.

Der Thread für die Ausführung des Programms ist nötig, da die benutzte Methode `Process.waitFor()` bis zur Terminierung des Programms blockiert. Für den Aufruf der Methode `ResourceManager.executeJob(UUID)` ist dies jedoch ungünstig, da die Methodenaufrufe der einzelnen Komponenten für eine asynchrone Ausführung ausgelegt sind. Der Thread meldet daher das Ende der Ausführung des Programms beim JobManager über die Methode `JobManager.jobFinished(UUID,int)`. Abbildung 3.11 veranschaulicht den Nachrichtenaustausch zwischen Job-Manager und Resource-Manager detailliert.

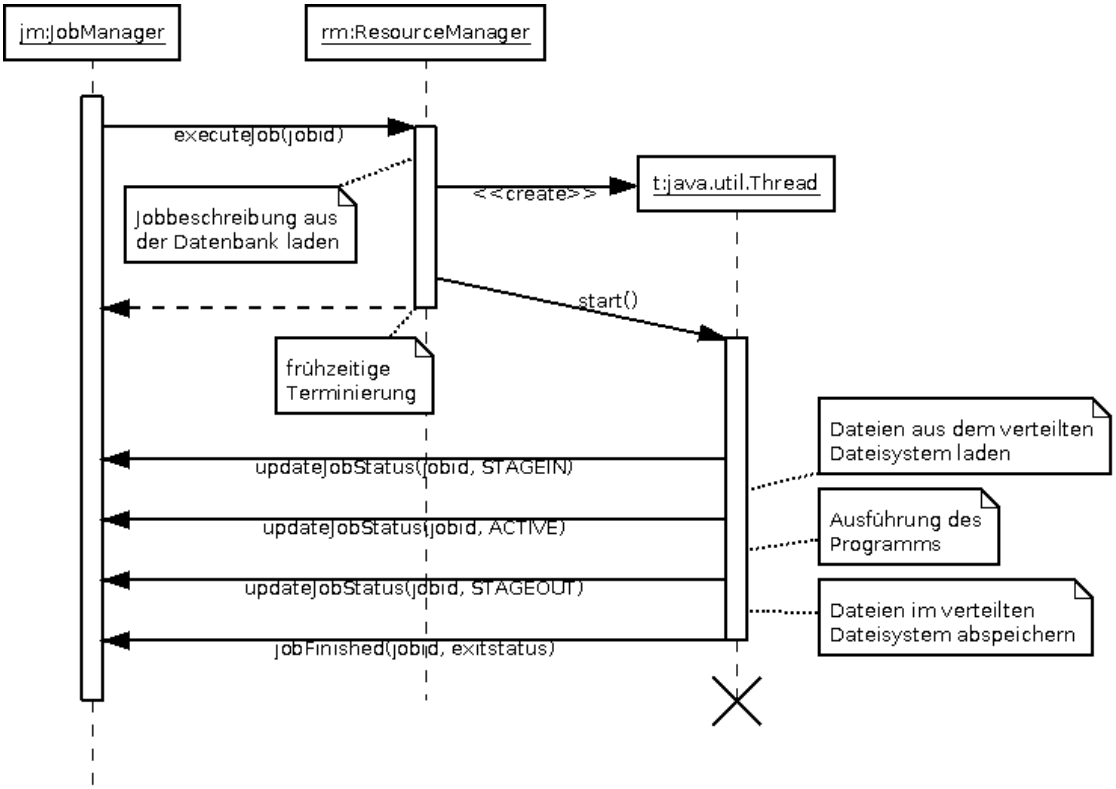


Abbildung 3.11: Methodenaufrufe zwischen JobManager und ResourceManager

Für die Ausführung des Programms wird das Verzeichnis `JOB_TMPDIR/workdir/` als Startverzeichnis angegeben. Die Ausführung ist jedoch nicht auf dieses Verzeichnis beschränkt. Das Programm befindet sich in keiner CHROOT-Umgebung und kann somit jede Aktion durchführen, die auch der Benutzer durchführen kann, der das Cloud-Programm gestartet hat. Außerdem gibt es keinen Disk-Quota für die Dateien, die vom Programm angelegt werden. Diese beiden Punkte müssen beachtet werden, wenn ein Knoten einen Resource-Manager für die Ausführung von Jobs bereitstellt.

3.4 Der Koordinierungsdienst: Job-Manager

Die Komponente Job-Manager hat im System die Aufgabe, die eingebrachten Jobs an die Resourcen-Manager zu vergeben. Weiterhin verwaltet der Job-Manager die Lebenszyklen der Jobs und implementiert in dem vorliegenden System eine einfache Scheduling-Strategie (First-Come-First-Serve, FCFS).

3.4.1 Scheduling

Jobs werden von einem Benutzer über die Web 2.0-Schnittstelle in das System eingebracht. Dabei erstellt ein Benutzer einen Job und kann diesen nach seinen Wünschen konfigurieren. Zu einem späteren Zeitpunkt kann der Benutzers den konfigurierten Job über die Benutzerschnittstelle für die Bearbeitung freigeben. Der Job-Manager wird dann über den Statuswechsel informiert.

Dabei wird der Job nicht direkt in die vom Job-Manager gehaltene Scheduling-Liste eingetragen, sondern lediglich der Status des Jobs von `CONFIGURED` auf `APPROVED` geändert. Das Einreihen von Jobs mit Status `APPROVED` in die Scheduling-Liste erfolgt dann durch eine periodisch gestartete Methode. Für das Scheduling ist eine einfache FCFS-Startegie vorgesehen. Die Jobs werden somit in der Reihenfolge in der diese in die Datenbak eingetragen und freigegeben worden sind in die Schedulingliste eingereiht und nacheinander versucht auf einem Resourcen-Manager auszuführen.

3.4.2 Zuweisen eines Jobs

Sobald ein Job an die erste Position der Warteschlange vorgerückt ist, wird versucht, den Job an einen nicht arbeitenden Resource-Manager zu übergeben. Zu diesem Zweck hält der Job-Manager eine Liste vorhandener Resource-Manager vor. Hierbei muss der JobManager zwischen solchen Resource-Managern unterscheiden, die „frei“ sind, also einen Job entgegen nehmen können, und solchen, die bereits mit der Ausführung eines anderen Jobs beschäftigt sind. Ist ein unbeschäftiger Resource-Manager verfügbar, wird dieser mit der Bearbeitung des Jobs beauftragt; der Resource-Manager wird dann versuchen, den Job zu starten und auszuführen.

Der Job-Manager wird von der OSGi-Laufzeitumgebung über hierfür implementierte `bind`- und `unbind`-Methoden über neue oder nicht mehr verfügbare Resource-Manager informiert. Bei fehlerfreiem Zusammenspiel der Komponenten (siehe 3.3 Der Ausführungsdienst: Resource-Manager) ist der Zustand der Resource-Manager jederzeit

bekannt. Zur Erhöhung der Robustheit wird jedoch zusätzlich periodisch der Zustand der bekannten Resource-Manager überprüft.

Behandlung von Fehlverhalten der Resource-Manager

Um die Ausnahmefälle eines Fehlverhaltens eines Resource-Managers abzufangen, kann der mit dem Job beauftrage Resource-Manager an den Job-Manager melden, dass das Starten bzw. Ausführen des Jobs nicht möglich ist. Es wird dann versucht, diesen Job einem anderen Resource-Manager zuzuweisen.

Auch wenn ein ausführender Resource-Manager unplanmäßig die Cloud verlässt, also insbesondere ohne abgemeldet zu werden oder die Information weiterzugeben, dass ein Job nicht weiter ausgeführt wird, muss der bearbeitete Job erneut vergeben werden. Zu diesem Zweck muss der Job-Manager die Information vorhalten, welchem Resource-Manager welcher Job zugewiesen wurde.

Um sicherzustellen, dass der Wiederholung auf eine dieser Arten fehlgeschlagener Ausführungen Vorrang eingeräumt wird, wird der Job in den Zustand `RELOCATING` gesetzt. Jobs mit diesem Status werden beim Scheduling an der Spitze der Warteschlange eingereiht, wobei sie wieder den Status `QUEUE` erhalten.

Das `RELOCATING`-„Dilemma“

Das Problem, das bei dieser bevorzugten Behandlung der Jobs mit Status `RELOCATING` auftreten kann, kann durch folgendes Szenario verdeutlicht werden:

Im Gesamtsystem ist für die Bearbeitung der Jobs nur ein Resource-Manager verfügbar. Der Job an der Spitze der Warteschlange wird an diesen Resource-Manager übergeben. Der Resource-Manager meldet dann dem Job-Manager, dass dieser Job nicht gestartet werden kann. Der Job wechselt in den Zustand `RELOCATING` und wird bei der Aktualisierung der Scheduling-Liste erneut an die erste Position gesetzt. Auch im zweiten Zuweisungsversuch soll nur der Resource-Manager im Gesamtsystem vorhanden sein, der nicht in der Lage war, diesen Job zu starten und auszuführen. Folglich blockiert dieser Job alle anderen Jobs der Scheduling-Liste, die auf dem vorhandenen Resource-Manager ausgeführt werden könnten.

Um dieses Problem zu vermeiden, führt der Job-Manger eine *blacklist* für jeden Job. Meldet ein Resource-Manager, dass der übergebene Job nicht ausgeführt werden kann, wird dieser ResourceManager in die *blacklist* dieses Jobs eingetragen. Beim Zuweisen eines Jobs wird zunächst geprüft, ob mindestens einer der freien Resource-Manager nicht in der *blacklist* des Jobs enthalten ist. Sollten alle freien Resource-Manager bereits in der *blacklist* des Jobs eingetragen sein, wird der in der Warteschlange folgende Job zugewiesen. Der nicht zuweisbare Job verbleibt allerdings an der Spitze der Warteschlange, um weiterhin bevorzugt vergeben zu werden, wenn weitere Resource-Manager verfügbar werden.

Job-Instanz und Ausführungsinstanz eines Jobs

Die Übergänge zwischen den in „Der Job-Lebenszyklus“ (2.3.4) beschriebenen Zuständen eines Jobs wird zentral durch den Job-Manager durchgeführt.

Da ein einzelner Job wiederholt zur Ausführung in der Cloud gebracht werden kann, unterscheidet der Job-Manager für die Verwaltung des Lebenszyklus zwischen der Jobbeschreibung und einer Ausführungsinstanz gemäß der Beschreibung. Für jede Freigabe des Jobs durch den Benutzer wird eine neue Ausführungsinstanz betrachtet. Jede dieser Ausführungsinstanzen beinhaltet als Information die UUID des Jobs, den aktuellen Status, Übermittlungszeitpunkt, Startzeitpunkt und Stoppzeitpunkt, erhält aber zur eindeutigen Unterscheidbarkeit der Ausführungen eine eigene UUID. Dass diese Daten für jede Ausführung bekannt sind, ist für die Abrechnung der genutzten Rechenleistung wesentlich: Durch Aufsummierung der Laufzeiten aller Ausführungsinstanzen des Jobs kann eine korrekte Abrechnung der genutzten Rechenleistung erfolgen.

3.5 Web 2.0-Benutzerschnittstelle

In diesem Kapitel wird das UserInterface(UI) vorgestellt. Dazu wird die auf der Server-Seite eingesetzte Architektur erläutert, sowie die Auswahl der genutzten Frameworks für die Erstellung der Server- und Client-Seite dargelegt. Ebenfalls wird die Umsetzung der Backend-Lösung auf der Server-Seite und die Web-Schnittstelle auf der Client-Seite im Einzelnen erläutert.

3.5.1 Anforderungen

Das UserInterface soll eine graphische Benutzerschnittstelle zur Darstellung der Konfigurationsmöglichkeiten der Cloud bieten, also insbesondere die Steuerung der Knoten und Dienste auf den jeweiligen Knoten und den darauf zu berechnenden Jobs. Ebenfalls ist es nötig zur Vergabe von Rechten einen Zugriff auf das Benutzermanagement einzurichten, damit ein autorisierter Zugriff auf die jeweiligen Ressourcen und Jobs möglich ist. Zu diesem Zweck ist eine Web2.0-Anwendung auf der Client-Seite vorgesehen, welche auf eine standardisierte API zurückgreift. Eine genaue Aufstellung von Aktionen und Rollen, die in dieser Anwendung auftreten, findet sich im vorangegangenen Kapitel Entwurf.

Im Betrieb des Cloud-Systems wird eine Webseite dynamisch im Browser generiert und aktualisiert. Damit ist es dem Benutzer möglich eine einfache und schnelle Interaktion mit der Web2.0-Anwendung und den dahinterstehenden Funktionen der Cloud einzugehen. Diese Idee findet sich unter dem Begriff Rich Internet Application (RIA) wieder. Typischerweise werden solche Anwendungen nicht installiert und ähneln einer Desktop-Anwendung. Die Berechnungen zur Darstellung der Web-Anwendung werden auf der Client-Seite ablaufen und entlasten damit den Webserver. Wichtig hierbei ist es, die Kompatibilität zu den verschiedenen Browsern und Plattformen zu wahren. Zum Einsatz bei einer RIA kommen dabei unter anderem Web-Technologien wie HTML, CSS und JavaScript aber auch Flash. Zur Realisierung einer RIA kann grundlegend AJAX (Asynchronous JavaScript and XML) benutzt werden, da hiermit eine asynchrone Da-

tenübertragung per HTTP zwischen einem Server und einem Client ermöglicht wird. Zur Datenübertragung der einzelnen Informationen muss ein bewährtes Datenaustauschformat herangezogen werden. Hierzu stehen z.B. JSON (JavaScript Object Notation) oder XML (Extensible Markup Language) zur Auswahl.

Durch eine standardisierte API ist gewährleistet, dass sowohl Frontend als auch Backend der Benutzerschnittstelle klar getrennte Module sind, so dass diese bei Bedarf unabhängig von einander ausgetauscht werden können. Um diese Anforderung umzusetzen ist eine REST-Architektur auf der Server-Seite vorgesehen. Diese Architektur soll nachfolgend genauer betrachtet werden.

3.5.2 REST

Bei REST (Representational State Transfer) handelt es sich um einen Softwarearchitekturstil zur Realisierung von Web-Services. Diese Architektur verwendung etablierte Standards. Zentral für REST sind das diesem zugrunde liegende Hypertext Transfer Protocol (HTTP) für den Zugriff auf Methoden und der Uniform Resource Identifier-Standard (URI) für die Adressierung.

Web-Seiten, Servlets oder Bilder stellen Ressourcen dar, die in der REST-Architektur über URLs adressiert und angesprochen werden. Eine Web-Anwendung besteht aus mehreren solcher Ressourcen, die über das HTTP Nachrichten austauschen. Auf eine Ressource kann jedoch nicht direkt zugegriffen werden, sondern nur über die der Ressource zugeteilten URI. Für die Manipulation und Erzeugung der Ressourcen reichen die vier HTTP-Methoden `GET`, `PUT`, `POST` und `DELETE`:

GET
: GET fragt die Repräsentation einer Ressource ab.

POST
: Mit der POST-Methode werden neue Ressourcen angelegt oder bestehenden Ressourcen Informationen hinzugefügt.

PUT
: Der Inhalt bestehender Ressourcen kann durch die PUT-Methode ersetzt werden.

DELETE
: Ressourcen werden mit DELETE gelöscht.

Es ist möglich, dass Ressourcen in einem REST-Webservice unterschiedliche Repräsentationen haben. So werden Daten in unterschiedliche Datenformate übertragen. Je nach Anfrage einer Repräsentation könnten die Daten z.B. als JSON, XML, PDF, HTML, usw. verschickt werden. Zudem gibt es auch Ressourcen, die auf andere Ressourcen verweisen. “Folgt ein Client einem Link in einer Repräsentation, so gelangt er von einem Zustand in einen anderen“[9], wodurch sich der Name Representational State Transfer erklärt. Im Gegensatz zu anderen Architekturen interessiert sich der

[9] `http://www.oio.de/public/xml/rest-webservices.htm`

REST-Server nicht für den aktuellen Clientstatus oder dessen Session, da der Server selbst zustandslos ist. Der Client muss seinen Status somit selbst verwalten. Aus diesem Grund besteht in der Regel keine spezielle Login-Funktionalität. Zugriffe auf Ressourcen lassen sich z.B. mit HTTPS (Hypertext Transfer Protocol Secure) authentifizieren und autorisieren.

Zusammengefasst gilt REST als eine Richtlinie für die Nutzung von Web-Standards und eine Rückbesinnung auf die grundlegenden Web-Technologien. So kann mit den wenigen Methoden von HTTP die vollständige Kommunikation abgewickelt werden, die bei der Realisierung auch komplexer Webprojekte anfällt.

Für die Umsetzung der REST-Architektur stehen mehrere Frameworks zur Verfügung, die jedoch alle identische Funktionalitäten aufweisen. Das Restlet-Framework greift auf das standardisierte JAX-RS zurück, der Java API für RESTful Web-Services. Zudem erlaubt das Restlet-Framework die Erweiterung durch das Projekt geforderte Komponenten, wie z.B. einen Jetty-Webserver, das JSON-Datenformat oder die standardisierte Autorisierungs-API OAuth.

3.5.3 Framework-Auswahl

Nachstehend werden verschiedene Frameworks betrachtet, die die Umsetzung der Anforderungen ermöglichen sollen. Dabei wird untersucht welches Framework für die weitere Arbeit am besten geeignet ist.

JavaServer Faces (JSF)

JavaServer Faces ist ein Web-Framework zur Entwicklung von komponentenbasierten Benutzeroberflächen in Web-Anwendungen. Als Grundlage wird ein Java-Servlet-Container auf der Server-Seite benötigt. Die gesamte Verarbeitung der eingehenden Anfragen wird auf dem Server durchgeführt. Das Ergebnis eines Aufrufs im Browser ist, nach der Verarbeitung auf dem Server, eine HTML-Seite welche komplett an den Client übertragen werden muss.

Um eine Anwendung, die auf JavaServer Faces basiert, um AJAX-Funktionalitäten zu erweitern, benötigt es weitere Frameworks, wie z.B. ICEfaces oder RichFaces. Daher ist JSF nativ nicht dazu ausgelegt Rich Internet Applications zu generieren. Nachteilig erweist sich auch, dass der Entwickler bei der Erstellung den Fokus nicht auf eine Programmierhochsprache wie Java richten kann sondern zusätzlich auf die Verwendung von HTML, CSS, XML und JavaScript angewiesen ist.

Wicket

Apache Wicket ist ein Open-Source-Projekt, welches die Entwicklung von Web-Anwendungen unterstützt. Wicket verwendet dabei die Programmiersprache Java und zusätzlich HTML und CSS.

In Wicket liegt eine strikte Trennung von Logik und Design vor. Serverseitig werden mithilfe von Template-Dateien die anzuzeigenden Webseiten für den Client erstellt. Zur Gestaltung der Webseiten enthalten die Template-Dateien HTML und CSS. Bei

Wicket-Anwendungen werden die HTML-Tags mit Platzhaltern, so genannten Wicket-ID's, erweitert. Diese erweiterten Namensräume erlauben dann eine Adressierung im Java-Quellcode.

Über eine Wicket-ID kann der Inhalt in einer Template-Datei ausgetauscht werden. Der so aufbereitete Inhalt kann eine komplette Webseite sein. Es ist aber auch möglich nur, einen Teil einer Webseite zu erzeugen und mithilfe von AJAX im Client zu ersetzen. Eine Seite besteht dabei in der Regel aus einer Java-Klasse und einer HTML-Datei. Das separierte Vorgehen bietet Entwicklern einen komfortableren Zugang zum Layout.

Echo

Ein weiteres Framework zur Erstellung von Rich Internet Applications ist die Echo-Familie des Unternehmens NextApp. Von Anfang an ist Echo ein Open-Source-Projekt und unterstützt den Web-Entwickler bei der Programmierung von Web-Benutzeroberflächen. Die Bedienung des Echo-Frameworks ist dabei mit Swing oder SWT vergleichbar.

Derzeit existieren mit Echo2 und Echo3 zwei verschiedene Versionen des Frameworks. Mit Echo2 wird der Ansatz verfolgt, eine Webseite komplett auf einem Server zu erstellen. Der Austausch von Teilen einer Webseite kann dabei per AJAX vorgenommen werden. Erfolgen Interaktionen mit der Web-Benutzeroberfläche im Browser werden diese auf dem Server behandelt. Zur Gestaltung des Layouts der Webseite sowie zur Erstellung der erforderlichen Logik wird ausschließlich Java eingesetzt, so dass bei Echo2 auf die Verwendung von AJAX, JavaScript, HTML oder CSS verzichtet werden kann.

Während Echo2 lediglich den Betrieb serverseitiger Webseiten in einem Servlet-Container vorsieht, erweitert Echo3 diese Funktionalität um die Möglichkeit eine Web-Anwendung zu erstellen, welche auf einen Server verzichten kann. Für die Programmierung wird dabei auf JavaScrpit zurückgegriffen, so dass die Fokussierung auf einen reinen Java-Quelltext verloren geht. Falls Kommunikation bei der client-seitigen Variante entsteht, kann diese über HTTP erfolgen, wobei wahlweise XML- oder JSON-Formate ausgetauscht werden.

Google Web Toolkit (GWT)

Das Google Web Toolkit unterstützt als Framework die Erstellung von Rich Internet Applications und bietet eine breite Unterstützung von AJAX-Funktionalitäten. Die Programmierung der Web-Anwendung erfolgt vollständig in Java. Eine zusätzliche Erstellung von JavaScript und HTML wird so unnötig.

Der Java-Code wird intern mit einem Java-to-JavaScript Compiler in JavaScript sowie HTML und CSS übersetzt und kann somit, auch ohne Plugin, in allen gängigen Browsern angezeigt werden. Auf diese Weise kann auf Java-Ebene mit Hilfe von Klassen und Paketen eine Modularisierung der Komponenten vorgenommen werden. Der erzeugte JavaScript-Code wird vom Compiler optimiert um möglichst wenig Bandbreite bei der Übertragung zum Client nutzen zu müssen. Bei Bedarf kann zusätzlicher JavaScript-Code integriert werden, wenn z.B. JavaScript-Bibliotheken von Drittanbietern verwendet

werden sollen.

Mit der Version 2.0 von GWT wird ein In-Browser-Development-Mode angeboten, der mit Hilfe eines Browser-Plugins das einfache Debuggen in allen gängigen Browsern ermöglicht.

Die Client-Server-Kommunikation erfolgt im GWT standardmäßig per Remote Procedure Call (RPC). Die verwendete Umsetzung von RPC ist dabei von Google für das GWT entwickelt worden. Daher ist die Client-Server-Kommunikation, bei der Verwendung von RPC, auf die Google-Lösung festgelegt. Dies entspricht nicht den Anforderungen einer flexiblen und unabhängigen Technologie für die Server-Seite. Ebenfalls kann so keine REST-Schnittstelle verwendet werden. Mithilfe eines Restlet-Moduls kann dieser Nachteil aber umgangen werden. Diese Lösung bietet zudem den Vorteil, dass auf der Server-Seite eine beliebige Technologie zur Bearbeitung der REST-Anfragen zum Einsatz kommen kann.

Restlet Framework

Das für die Umsetzung der REST-Architektur gewählte Restlet[10] versteht sich als ein leichtgewichtiges in Java geschriebenes Open Source Framework und eignet sich für Client und Server Web-Anwendungen. Andere REST-Frameworks hingegen, wie Jersey[11] und RestEasy[12], sind mehr auf die Verwendung von Servlets spezialisiert. Zudem basieren diese beiden Frameworks ausschließlich auf der Java Spezifikation JSR 311 - The Java API for RESTful Web Services[13]. Die Spezifikation hat zwar den Vorteil, dass der Code im Vergleich zur Standard Restlet API kürzer und kompakter ist, limitiert damit allerdings auch gleichzeitig die Möglichkeit komplexe Anwendungen zu implementieren. So erlaubt Restlet z.B. den Einsatz von Filtern und Verzeichnissen mit statischen Inhalten, wie es bei WAR-Dateien[14] benötigt wird. Zudem ist es möglich JSR 311 als Extension in Restlet zu integrieren und somit beide Vorteile zu nutzen. Zu den Nachteilen von Restlet gehören gelegentlich auftretende Bugs und fehlende Features, wie z.B. OpenID.

Wie in den Anforderungen beschrieben wird eine Web 2.0-Benutzerschnittstelle benötigt, für die sich die Verwendung von AJAX auf Client-Seite anbietet. Somit wird Restlet lediglich serverseitig eingesetzt und es muss untersucht werden, wie gut eine Unterstützung durch Restlet mit AJAX-basierten Clients gegeben ist. Web 2.0-Anwendungen tauschen XML- oder JSON-Dokumente über HTTP aus, was mit SOAP orientierten Webservices, wie Apache Axis nur schwer zu realisieren ist, auch aus diesem Grund ist die Verwendung eines REST-Frameworks wie Restlet, bei dem eine vollständige Unterstützung solch eines Austausches gegeben ist, dem einer SOAP-Architektur vorzuziehen.[15]

[10] http://www.noelios.com/
[11] https://jersey.dev.java.net/
[12] http://www.jboss.org/resteasy/
[13] https://jsr311.dev.java.net/
[14] http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/WebComponents3.html
[15] http://www.eweek.com/c/a/Application-Development/Java-Project-Founder-Outlines-Benefits-of-Restlet/

Eine Restlet-Anwendung besteht aus mehreren Komponenten, wie dem Webserver, der Router-Klasse, den Application- und Resource-Klassen. Die Router-Klasse wird für das interne Routing benötigt, da diese bei jedem Serveraufruf als erstes kontaktiert wird. Dabei überprüft die Router-Klasse, welche URL aufgerufen wurde, sucht in den Application-Klassen die passende Ressource und leitet den Aufruf dann an die entsprechende Klasse weiter.[16] Als Beispiel dient der Aufruf der Projekt-Domäne durch den Client. Hierbei verweist die Router-Klasse auf die zuvor als Standard (default) definierte Klasse, die ebenso aufgerufen wird, wenn eine nichtdefinierte URL aufgerufen wird. In der Regel wird die Implementierung des Webservers in der Router-Klasse erstellt. Beim Aufruf einer Resource wird dieser Aufruf mit der dazugehörigen HTTP-Methode an die jeweilige Klasse weitergegeben. Neben dem HTTP-Methodenaufruf können Parameter als zusätzliche Information gesendet werden, die von der Router-Klasse gelesen werden. Diese Informationen können das Mitsenden eines Dokuments, z.B. im XML- oder JSON-Format sein oder eine URL. Beim Mitsenden einer URL kann so die empfangende Resource-Klasse den Informationsteil auslesen und als String-Objekt in der aufgerufenen Methode verwenden.

Rich Internet Application mit GWT

Das UserInterface für das Cloud-System verteilt sich grob auf zwei Komponenten. Die erste Komponente umfasst das Backend und bildet einen Zugang über einen Server mittels einer standardisierten REST-API zum Cloud-System. Die zweite Komponente des UserInterfaces ist das Frontend. Das Frontend soll für einen Benutzer oder Administrator eine Web-Anwendung als Schnittstelle zur Überwachung und Steuerung der Cloud und Jobs bereitstellen, dabei sollen Berechnungen auf der Client-Seite im Browser ohne Rückfragen zum Server auskommen.

Die Leistungsfähigkeit von GWT zeigt sich in Anwendungen wie GoogleWave. Zusätzlich ermöglicht der offene Standard die Entwicklung eigener Erweiterungen. Der Einstieg zur GWT-Entwicklung wird dem Entwickler durch dessen Anlehnung an Java sehr erleichtert, so dass das Erstellen der Benutzerschnittstelle der Implementierung durch „SWT“ oder „Swing“ gleicht. Anders als bei JSF, Wicket und Echo2 ist die Wahl der Backend-Technologie nicht vorgegeben und Berechnungen von Ereignissen und das Rendern der Webseite wird auf den Client ausgelagert und entlastet so den Server. Echo3 bietet ähnliche Argumente wie das GWT. Jedoch ist nur das Framework als Open-Source erhältlich, hingegen ist ein Plug-In für die Entwicklungsumgebung Eclipse EchoStudios proprietär. Ebenso nachteilig ist bei der Verwendung von Echo3, dass eine ausschließlich auf Client-Seite ausgeführte Anwendung als RIA nicht vollständig in Java realsiert werden kann, sondern zusätzlich JavaScrpt verwendet.

Aus diesen Betrachtungen geht hervor, dass das Google Web Toolkit für die Umsetzung der Web 2.0-Schnittstelle geeignet ist und mit Hilfe von Java, eine leistungsstarke Web 2.0-Anwendung erzeugen kann. Insbesonders mit der Kombination aus GWT und Restlet kann die Anforderung einer REST-Schnittstelle erfüllt werden. Daher wurde mit Hilfe des Google Web Toolkits für das Frontend eine Rich-Internet-Application erstellt.

[16] `http://www.restlet.org/documentation/snapshot/gwt/api/`

Die erstellte Anwendung setzt lose auf das Backend auf, wobei diese lose Kopplung durch die verwendete REST-API erreicht wird. Eine genaue Betrachtung der Themen REST und GWT wurde bereits in den vorherigen Abschnitten geführt.

Eine direkte Kommunikation vom Frontend mit einer der anderen Komponenten der Cloud (Job-Manager, Resource-Manager, Datenbank oder Filesystem) ist nicht vorgesehen. Somit ist ein Austausch oder eine Erweiterung der Frontend-Anwendung bzw. des Cloud-Systems jederzeit und unabhängig möglich.

Bei der Entwicklung der Web2.0-Anwendung in Eclipse muss die vom Google Web Toolkit mitgelieferte Laufzeitumgebung verwendet werden. Jedoch stellt diese Laufzeitumgebung nicht alle Java-Bibliotheken bereit. Somit stehen einige gebräuchliche Methoden nicht zur Verfügung, da diese JRE-Bibliotheken nicht in JavaScript umgesetzt werden können.

Die benötigte Projekt-Struktur für die GWT-Anwendung ähnelt den Strukturen der anderen Teilprojekte für die Cloud. Eine Ausnahme bildet das Verzeichnis `war`. Dieses zusätzliche Verzeichnis beinhaltet, nach dem Kompilieren des Java-Codes, alle notwendigen JavaScript-, HTML- und CSS-Daten zur Bildung eines Web-Archivs.

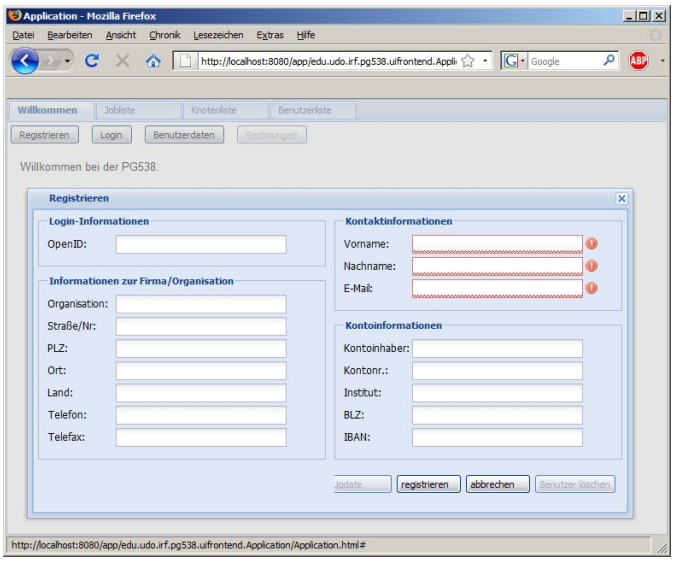


Abbildung 3.12: Prototyp der Maske zur Benutzerregistrierung

Das kompilierte Projekt wird außerhalb von GWT auf einem Server bereitgestellt und steht zur Anzeige in einem Browser zur Verfügung. Ein Screenshot der Web-Anwendung ist in Abbildung 3.12 zu sehen. Es wird die Maske zur Registrierung eines neuen Benutzers gezeigt.

Die Kommunikation der Daten zwischen Frontend und Backend wird unter Verwendung des Dateiaustauschformats JSON erreicht. Eine genauere Erläuterung findet sich in dem nachstehenden Kapitel „Kommunikation“.

GWT-Erweiterungen

Bei der Entwicklung einer Web-Anwendung mit GWT bieten die zahlreichen Erweiterungen von Drittanbietern einen großen Vorteil. In diesem Projekt wird auf die REST-Erweiterung zur Ersetzung der RPC-Aufrufe zurückgegriffen. Außerdem kommt GWT-ext für die Darstellung der Oberfläche zum Einsatz. Mit GWT-log ist es zudem möglich, Debug-Ausgaben aufbereitet im Browser anzuzeigen.

GWT und REST Das Google Web Toolkit bietet zur Sicherstellung der Kommunikation mit einem Server standardmäßig Remote-Procedure-Call (RPC) an. Dabei erfolgt die RPC Kommunikation über spezielle für das GWT bereitgestellte Methoden. Eine schematische Darstellung ist in der Abbildung 3.13 zu sehen. Auf der Server-Seite existiert für die RPC-Kommunikation ein Servlet-Container. Dieser Servlet-Container empfängt die eingehenden RPC-Anfragen und bearbeitet diese. Die Kommunikation mit der Anwendung im Browser erfolgt dabei über das Datenaustauschformat JSON. Bei diesem Verfahren wird eine zugrunde liegende J2EE-ServerPlattform benötigt. Dies verhindert jedoch die Kopplung mit anderen Backend-Lösungen.

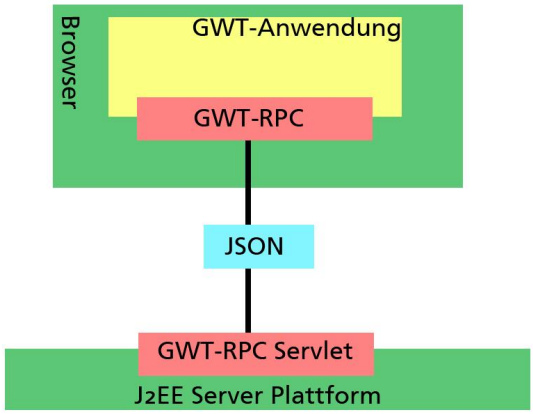


Abbildung 3.13: Client-Server Kommunikation mit RPC

REST ist in dieser Hinsicht sehr viel flexibler um Interoperabilität zwischen der Client- und Server-Lösung zu gewährleisten. Hierzu bietet Restlet ein Modul für GWT an. Abbildung 3.14 zeigt das client-seitige Restlet-GWT-Modul welches in den Google Web Toolkit-Code integriert ist. Ab diesem Zeitpunkt kann die Restlet-API zur Kommunikation mit dem Server genutzt werden. Es wird sowohl XML als auch das JSON Repräsentationsformat unterstützt. Auf Server-Seite kann nun eine beliebige Server-Plattform mit REST-Unterstützung installiert werden.

GWT erweitert durch GWText Zur Ergänzung der im GWT enthaltenen Widgets wurde die Open-Source-Lösung GWText als Erweiterung hinzugefügt. Alternativ hätten auch die in GWT enthaltenen Widgets erweitert und angepasst werden können, jedoch

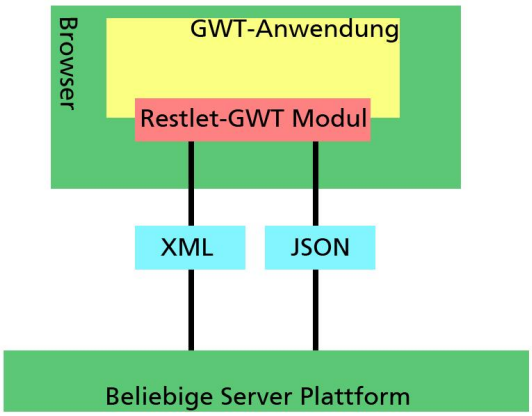


Abbildung 3.14: Client-Server Kommunikation mit RESTlet

erfüllt GWText alle notwendigen Anforderungen hinreichend, so dass auf die eigene Widget-Implementierungen verzichtet werden konnte.

GWText verwendet dabei die JavaScript-Bibliothek ExtJS des Unternehmens NextApp. Mit GWText wird das Layout der Widgets bzw. der gesamten Web-Anwendung erzeugt und stellt hierzu eine CSS-Konfigurationsdatei und die zugehörigen Grafiken zu Verfügung. Für die Darstellung der Statistiken wird von GWText Flash verwendet. Die Erzeugung der jeweiligen Diagramme erfolgt dabei, wie alle anderen Berechnungen auch, im Browser.

GWT erweitert durch GWTlog Bei der Erstellung, Wartung und im Betrieb muss die Web-Anwendung leicht diagnostizierbar sein. Es ist dabei wichtig, versteckte Fehler aufdecken zu können. Hierzu wird der GWT-Logger benutzt. Es werden Systeminformationen, Fehler und Debug-Meldungen über den GWT-Logger im Browser angezeigt. Anforderungen die sich an den Logger stellen sind vor allem:

- Kompatibilität zu GWT
- verfügbar ohne Lizenzgebühren
- frei modifizierbar
- leichtgewichtig
- flexible Konfiguration

Die Erweiterung GWT-log (Logging Library for Google Web Toolkit (GWT) with Deferred Binding) erfüllt die zuvor genannten Anforderungen. Der Quellcode ist frei verfügbar und steht unter der Apache-Lizenz (GPLv3 kompatibel). Die Log-Ausgaben sind leicht zu Konfigurieren. Im einzelnen lässt sich der Logger Ein- und Ausschalten und ermöglicht die Einstellung von verschiedenen Log-Stufen. Die Einbindung des Loggers

ist auf zwei Weisen möglich: Entweder direkt in der `web.xml` (also in der Konfigurations-Datei der Web-Anwendung) oder in einem GWT-Widget. So ist es möglich den Logger entsprechend an das Design der Web-Anwendung anzupassen.

3.5.4 Wahl des Webservers

Da die Benutzerschnittstelle über eine Webschnittstelle angeboten wird, musste auch ein HTTP-Server für diese Schnittstelle ausgewählt werden. Im Folgenden sollen die betrachteten Server vorgestellt werden, bevor dann die getroffene Entscheidung begründet wird.

Apache Tomcat

Zunächst wurde Apache Tomcat in der Version 6.0.20 getestet, der durch die Apache Software Foundation veröffentlicht wurde. Der Server wird als Open-Source-Projekt vertrieben.

Neben dem grundlegenden Funktionsumfang eines HTTP-Servers bietet Apache Tomcat einen Java-Servlet-Container sowie einen Java Server Page-Compiler, um JSPs in Servlets zu übersetzen. Auch SSL-Verbindungen werden unterstützt. Die Möglichkeit einer Einbettung in Java-Projekte ist gegeben, erfordert jedoch eine umfangreiche Konfiguration.

Jetty

Auch Jetty bietet einen kompletten HTTP-Server mit JSP- und Servlet-Container und SSL. Für den Test wurde die Version 7.0.0.RC5 verwendet.

Im Vergleich zu Apache Tomcat fällt vor allem die „Schlankheit“ des Jetty-Servers auf. Durch Anpassungen auf den im Projekt benötigten Funktionsumfang benötigt dieser nur noch ein Fünftel des vom Apache Tomcat benötigten Speicherplatzes. Der entscheidende Vorteil, der sich durch den geringen Speicherplatzbedarf ergibt, ist die einfache Intergrationsfähigkeit des Servers in andere Softwareprojekte, wie dies im Zuge der Projektgruppe verlangt ist.

Das Starten und Konfigurieren des Jetty-Servers erfordert nur wenige Zeilen Programmcode. Hierbei wird zunächst Port und IP-Adresse des Webservers festgelegt und danach die entsprechende WAR-Datei ausgeliefert.

Internal Restlet HTTP-Server

Das Restlet-Framework[17] stellt einen eigenen, internen HTTP-Server bereit, der zu Testzwecken genutzt werden kann, jedoch nur geringe Einstellungsmöglichkeiten bietet. Dieser Server ist im Funktionsumfang von Restlet bereits enthalten und stellt daher keine zusätzliche Abhängigkeit dar. Ähnlich wie der Jetty-Webserver benötigt auch der Restlet-Server nur wenige Codezeilen, in denen die sogenannten Server-Connectors initialisiert werden. Diese stellen eingehenden Serveranfragen (Requests) und ausgehenden

[17] `http://www.restlet.org/`

Serverantworten (Responses) zu, dabei ist es nötig, den Server-Connectors einen Port und das verwendete Protokoll zuzuweisen, in diesem Fall das HTTP.

Server-Entscheidung

Tomcat ist ein umfangreicher Server mit vielen Hilfen und Einstellungsmöglichkeiten, dieser Umstand erhöht den Speicherverbrauch des Webservers im Vergleich zu anderen Servern erheblich. Neben der Größe des Tomcat-Servers als Nachteil erwies sich auch das Einbetten als Systemkomponente schnell als sehr umständlich und zeitaufwändig und wurde daher zugunsten des Restlet-internen Servers aufgegeben.

Nach anfänglichen Versuchen, den Restlet-Webserver nicht nur für Tests während der Entwicklung, sondern auch für die ersten funktionsfähigen Versionen des Cloud-Systems zu nutzen, wurde klar, dass der beschränkte Funktionsumfang nicht ausreicht. Es traten verschiedene Server-Fehler auf und die Antwortzeiten bei Serveranfragen waren nicht tragbar. Die Umstellung auf einen eingebetteten Jetty-Server konnte jedoch unkompliziert erfolgen, wonach dann auch verschlüsselte SSL-Verbindungen mit der Benutzerschnittstelle realisiert werden konnten.

3.5.5 Umsetzung

In diesem Kapitel wird gezeigt wie die genaue Umsetzung der REST-Architektur durch Restlet im Backend aussieht. Es wird gezeigt, welche Klassen und Ressourcen genutzt werden und wie auf welche HTTP-Methoden reagiert wird. Zudem wird gezeigt, wie mit Hilfe vom GWT die Webanwendung erstellt wird.

REST-API mit Restlet

Als erstes wird die Implementierung auf Server-Seite vorgestellt. Die REST-Architektur setzt, wie schon in der Framework-Auswahl beschrieben, auf die Unterteilung der Aufgaben in unterschiedliche Resource-Klassen. Zudem kommt auf Server-Seite JAX-RS zum Einsatz-The Java API for RESTful Web Services.

JAX-RS

JAX-RS ermöglicht und vereinheitlicht die Nutzung der REST-Architektur im Rahmen von Webservices. Um die Implementierung von Webservices zu vereinfachen benutzt JAX-RS Annotation, wie es auch von anderen API's der Java Platform Enterprise Edition (JEE) bekannt ist. Die Referenzimplementierung stellt das Open-Source-Projekt Jersey da[18]. Allerdings kann über eine Extension die Java-Spezifikation JAX-RS auch in Restlet eingebaut und genutzt werden. Der Einsatz von JAX-RS ändert und vereinfacht den Aufbau der Klassen gegenüber der Standard-Implementierung grundlegend und dies nicht nur wegen der Annotationen. Das komplette URL-Management wird vereinfacht, da eine Resource-Klasse mit JAX-RS nicht mehr den kompletten

[18] `https://jersey.dev.java.net/`

Adresspfad benötigt, worunter die Resource erreichbar ist, sondern nur noch den für die Resource spezifizierten Teil, also den relativen Pfad. Als Beispiel dient diese URL `http://www.example.org/app1/wert/{zahl}`. Einer Resource wird in diesem Fall nur noch der Bereich `wert/{zahl}` zugewiesen, wobei `{zahl}` als Variable dient. Es ist auch möglich mehrere Resourcen in einer Restlet-Application zusammenzufassen, dies ist nötig, um zusammenhängenden Resourcen einen gleichen Teilpfad zuzuweisen.

Resource-Klassen

Der Webserver hat drei Application-Klassen und 31 Resource-Klassen, von denen nun ein paar kurz erklärt werden.

Users-Resource

Die `UsersResource` steht für die Liste aller Benutzer im System und kann diese ausgeben. Zugleich wird die `UsersResource` benötigt um neue Benutzer anzulegen. Zur Verfügung stehen hier die Befehle GET und POST.

Benutzerliste anfordern:

```
@Path("users")
public class UsersResource extends ServerResource {
    @GET
    @Produces("application/json")
    public Response list() {
        List<User> user = User.getAll(RestInterface.getDatabase().
            getConnection());
        return Response.ok(new Gson().toJson(user)).build();
    }
```

Mit `@Path` wird die URL für die Methoden auf `http://www.example.org/api/users` festgelegt. Als nächstes findet sich die HTTP-Methode GET als Annotation, damit wird der folgende Code nur ausgeführt, wenn es einen GET-Aufruf auf die `/api/users`-URL gibt. Mit `@Produces` wird angegeben, welches Datenformat der Response hat. Es kann auch mehrere `@Produces`-Annotationen geben, dann wird geschaut, welches Format der Client angefordert hat. Anschließend wird eine Liste alle Benutzer erzeugt und diese mit GSON in JSON umgewandelt und als Response mit Status-Code 200 verschickt.

Um einen neuen Benutzer einzutragen wird die HTTP-POST-Methode auf die Ressource ausgeführt und das JSON-Objekt mit den Daten des Benutzers übertragen. Mit `@Consumes` wird festgelegt, welches Datenformat akzeptiert wird oder wie mit bestimmten Datenformaten umgegangen wird. Somit kann `@Consumes` genauso wie `@Produces` mehrmals mit einer HTTP-Methode verknüpft werden. Mit GSON wird der eingehenden Benutzer von JSON in ein Java-Objekt umgewandelt. Anschließend wird der neue Benutzer mit dem Aufruf `User.addUser(con,user);` in der Datenbank hinzugefügt. Anschließend wird die URL zurückgegeben, unter der der neue Benutzer zu erreichen ist.

User-Resource

Um die Daten eines einzelnen Benutzers aufzurufen, Daten zu ändern oder den Benutzer zu löschen wird die `UserResource`-Klasse benötigt. Dafür muss hinter der users-URL noch die dem Benutzer gehörige UUID übergeben werden. Mit der Annotation `@PathParam` wird die UUID einem String zugewiesen. Anschließend wird der Benutzer aus der Datenbank geholt und zurückgegeben.

Beim Ändern von Benutzerdaten wird zuerst die entsprechende Person aus der Datenbank geholt. Dann wird aus den übergebenden JSON-Daten mit GSON ein Benutzer erzeugt. Alle neu gesetzten Attribut-Werte werden übergeben und danach wird der geänderte Benutzer gespeichert.

Um einen User zu löschen, wird dieselbe URL aufgerufen, wie beim Holen und Ändern eines Users, allerdings wird diese URL dann mit der HTTP-Methode DELETE aufgerufen. Ein Benutzer kann sich nicht selber löschen, sondern nur seinen Account deaktivieren. Ein Anmelden ist dann nicht mehr möglich. Damit soll verhindert werden, dass ein Benutzer seine ausstehenden Rechnungen einfach mitlöscht. Ein Admin hingegen kann einen Benutzer komplett löschen, d.h. es werden nicht nur die Benutzerdaten, sondern auch jeglicher Job und alle Dateien des Benutzers unwiederruflich gelöscht.

Jobs-Resource und Job-Resource

Die `JobsResource`-Klasse arbeitet analog zur `UsersResource`-Klasse. Die GET-Anfrage liefert eine Liste (`List<Job>`) der Jobs im System zurück und POST speichert den übergebenen Job in der Datenbank ab. Der Response vom Server enthält die URL zu dem neu erstellten Job im HTTP-header `Location`. Mit einem GET auf die `JobResource` wird der Job mit der angegebenen UUID zurückgegeben und mit einem POST wird der Job zur Ausführung freigegeben. Auch das Löschen eines Jobs läuft analog zum Löschen eines Users.

Node-Resourcen

Mit zehn Klassen beinhaltet das Knoten-Package die meisten Resource-Klassen und bietet überwiegend Funktionen für den Administrator. Diesem ist es möglich eine Liste aller im System befindlichen Knoten aufzurufen oder sich mehr Informationen über einen Knoten anzeigen zu lassen. Zudem wird angezeigt, welche Services auf den Knoten laufen und ob der Knoten gerade dabei ist, einen Job zu bearbeiten. Der Admin kann Services auf einem Knoten erlauben, verbieten, starten und stoppen und sich eine Statistik über die Cloud anzeigen lassen.

Kommunikation

Im Folgenden wird die Kommunikation zwischen Client und Server dargelegt. Hierzu gehört die Authentifikation und Verifikation der Benutzer, der Datentransport sowie das Versenden von E-Mails.

Authentifikation, Autorisierung und Verschlüsselung

Damit ein Benutzer nur die Funktionen seiner Rolle entsprechend nutzen kann, muss jede einzelne Anfrage beim Server authentifiziert werden. Dabei wird jedes mal kontrolliert, ob die Accountdaten des Benutzers gültig sind und ob der Benutzer die Rechte hat, die Funktion zu nutzen.

HTTP BASIC Authentifikation

Die HTTP-Authentifizierung als Bestandteil des Hypertext Transfer Protocols, wird genutzt, um Nutzer beim Webserver oder Webapplication zu authentifizieren. In der Regel muss sich der Nutzer nur einmal anmelden und ist fortan für alle weiteren Zugriffe autorisiert. Auf dem Webserver werden so Bereiche festgelegt für die Benutzername und Passwort benötigt werden. Versucht ein Benutzer auf gesicherte Informationen ohne Authentifikation zuzugreifen, schickt der Webserver den Statuscode 401 (Unauthorized) zurück. Liegen die Anmeldedaten beim Webserver im Cache vor, so werden diese genutzt. Liegen die Anmeldedaten hingegen nicht vor wird der Benutzer nach diesen angefragt. Die entsprechenden Daten werden dann an den Webserver im HTTP-Header gesendet und gespeichert. Der Webserver verifiziert die Daten und übermittelt bei erfolgreicher Authentifizierung die gewünschten Informationen. Für die Anpassung des Authetifikationsfensters wurde mit Hilfe von JavaScript die Authentifizierung in die Clientsoftware integriert, so dass das browsereigene Anmeldefenster nicht erscheint.

Bei der HTTP-Authentifizierung gibt es mehrere Möglichkeiten die Anmeldedaten zu codieren. Die für das Projekt gewählte und am häufigsten genutzte Art ist die Basic Authentication[19]. Dabei werden Benutzername und Passwort mit Base64[20] kodiert. Die Base64-Kodierung bietet jedoch nur eine sehr eingeschränkte Sicherheit. Alternativ zu Basic Authentication kann die Digest Access Authentication verwendet werden. Diese Authentifizierung arbeitet mit einem Hashcode, der in der Regel mit MD5[21] generiert wird. Es ist jedoch auch möglich eine eigene Verschlüsselung einzusetzen. Der Vorteil der Digest Access Authentication gegenüber der Basic Authentication ist, dass sich die Benutzerdaten schwerer rekonstruieren lassen. Bei beiden Authentifizierungsarten bergen jedoch das Problem, dass die restlichen Datenübertragungen unverschlüsselt bleiben. Um den gesamten Datenaustausch innerhalb des Projektes zu verschüsseln wird SSL eingesetzt. SSL macht eine aufwendige Kodierung der Anmeldedaten überflüssig, da SSL auch diese verschlüsselt. So ist mit der Kombination aus SSL und Basic Authentication eine Sicherung der Datenübertragung hinreichend erfüllt.

Verschlüsselung der Datenübertragung mit SSL

SSL ist die Abkürzung für Secure Sockets Layer und ist ein hybrides Verschlüsselungsprotokoll zur sicheren Datenübertragung im Internet. SSL ist im TCP/IP-Protokollstapel

[19] `http://tools.ietf.org/html/rfc2617`
[20] `http://tools.ietf.org/html/rfc4648`
[21] `http://tools.ietf.org/html/rfc1321`

zwischen TCP und der Anwendung angesiedelt. Damit eine gesicherte Verbindung zwischen einem Server und einem Client aufgebaut werden kann, übermittelt der Server dem Client ein Zertifikat. Ein solches Zertifikat enthält einen öffentlichen Schlüssel, mit dem sich der Server gegenüber dem Client authentifiziert, sowie die folgenden Informationen, um die Echtheit des Zertifikats sichherzustellen:

1. Name des Ausstellers
2. Informationen zu den Regeln und Verfahren
3. Gültigkeitsdauer
4. Name des Eigentümers
5. Digitale Signatur des Ausstellers

Das Zertifikat versichert dem User auf Clientseite, dass der Server vertrauenswürdig ist. Solche Zertifikate sind bei einer Zertifizierungsstelle (Certificate Authority, kurz CA) zu beziehen. Von dem käuflichen Erwerb wurde jedoch zu Gunsten eines selbst erstellten Zertifikats abgesehen.

Authentifizierung über OpenID

Als alternatives Authentifizierungs-System stand OpenID zur Auswahl. OpenID ist eine Umsetzung der Single-Sign-On(SSO)-Idee. Dabei benötigt ein Benutzer nur einen Accountnamen und ein Passwort und kann sich damit in allen unterstützenden Webanwendungen einloggen. Bei der Realisierung ergaben sich jedoch einige Probleme, so gibt es noch keine eigene Restlet-Erweiterung für OpenID. Diese wird erst mit der Restlet-Version 2.1 ausgeliefert[22]. Der Versuch OpenID in Eigenentwicklung in Restlet zu implementieren mündete in Kommunikationsproblemen zwischen dem verwendeten Webserver und den bestehenden OpenID-Providern, auch das Vorhalten der Session-ID birgt bei Ersatz eines ausgefallenen Webservers, das Problem, dass der Benutzer keine gültige Session-ID mehr hält und so gegenüber dem System nicht mehr authentifitiert ist. Somit wurde auf den Einsatz von OpenID verzichtet.

Versenden von Informationsmails

Das System kann Informationsmails an Benutzer und Administratoren verschicken. Wenn ein Benutzer nicht mehr im Besitz seiner Zugangsdaten ist, kann sich dieser die entsprechenden Daten zuschicken lassen. Der Webserver setzt in diesem Fall ein neues Passwort in der Datenbank und verschickt eine Mail mit den Zugangsinformationen an die vom Benutzer hinterlegte E-Mail-Adresse. Realisiert wird dies mit der Java-Mail-Biliothek, die der Webserver nutzt um per Simple Mail Transfer Protocol über SSL (SMTPS) die Mail an eine hierzu eingerichtete Googlemail-Adresse zu senden über die die E-Mail an die angegebene Adresse weitergeleitet wird.

[22] `http://restlet.tigris.org/issues/show_bug.cgi?id=446`

Format für den Datenaustausch

Um Probleme beim Datenaustausch zwischen Client- und Server-Seite zu vermeiden, werden die Daten in ein strukturiertes Format gebracht. Da die REST-Architektur mehrere Repräsentationen für Daten unterstützt, ist es möglich unterschiedliche Datenformate gleichzeitig anzubieten und zu empfangen. Der Client kann dann entscheiden, welches Datenformat angefordert werden soll bzw. welches Format die Daten haben, die der Client dem Server schickt. Dies bedeutet, dass auf der Client-Seite unterschiedliche Software eingesetzt werden kann, die mit unterschiedlichen Datenformaten arbeiten. Für das Projekt wurde ein bewährtes Datenformat gesucht. Zur Auswahl standen XML (Extensible Markup Language) und JSON (JavaScript Object Notation).

XML ist eine Auszeichnungssprache wie HTML oder LaTeX und strukturiert Informationen hierarchisch in Form von Textdaten. Im Vergleich zu JSON wirkt das Tag-System von XML für die Strukturierung künstlich aufgebläht und unübersichtlich. Mittlerweile werden XML-Daten für den Datentransfer komprimiert, um gerade bei größeren Daten Bandbreite zu sparen. Dafür muss allerdings auf beiden Seiten eine Möglichkeit zur Komprimierung und Dekomprimierung gegeben sein.

JSON hingegen gilt als ein schlankeres Datenaustauschformat und ist für Mensch und Maschine gleichfalls gut zu lesen bzw. zu parsen. Obwohl der Name den Verdacht nahelegt, dass JSON nur von JavaScript genutzt werden kann, ist JSON "ein Textformat, das komplett unabhängig von der Programmiersprachen ist".[23] JSON versucht Daten auf einfache Weise zu strukturieren, dabei stellt es sofort gültigen JavaScript-Code dar, der ausgeführt werden und somit auch leicht in ein JavaScript-Objekt überführt werden kann. Da die Client-Software aus AJAX-Code besteht, ist es möglich die Daten dort sofort und ohne Konvertierung zu nutzen. Somit fiel die Entscheidung das Projekt ausschließlich über JSON-Daten kommunizieren zu lassen.

Ein Beispiel einer JSON-Datei für einen Job zeigt, einen komplexeren Aufbau mit JSON-Arrays und -Objekten:

```
{   id:                  7a173cf0-abdb-4fe9-9e38-52305f7f3d4d,
    jobStatus:           SUCCESS,
    owner:               ad662d33-6934-459c-a128-bdf0393e0f44,
    description:         Dies ist ein JOB,
    name:                Ping auf 127.0.0.1,
    applicationName:     ping,
    applicationid:       2,
    applicationVersion: 1.0,
    exitstatus:          0,
    arguments:           [-n, 10, 127.0.0.1],
    environment:         {},
    executions:          [{
        uuid:        0b96f098-075a-48ec-a187-6ff7e65b1679,
        jobid:       7a173cf0-abdb-4fe9-9e38-52305f7f3d4d,
        submittime: 2010-02-21 18:38:07,
        starttime:  2010-02-21 18:38:07,
```

[23] http://json.org/json-de.html

```
        endtime:    2010-02-21 18:38:17,
        nodeuuid:   2ee0d715-203c-4335-9fe4-c113799d0f10,
        status:     SUCCESS
    }]
}
```

Listing 3.2: JSON Job-Darstellung

Beim Anlegen eines Jobs ist es notwendig, dass die Felder „Owner“ und „Name“ gesetzt werden. Als „Owner“ wird die UUID des Besitzers angegeben. Werden nur einige wenige Attribute geändert , so ist es möglich, nur die entsprechenden Änderungen zu übermitteln. Somit wird der Aufbau der JSON Datei auf Client-Seite dynamisch gehalten. Es besteht damit kein Zwang, den ganzen Job zu übertragen.

GSON auf der Server-Seite

Das von Google entwickelte GSON konvertiert auf Server-Seite Java-Objekte in JSON-Objekte, die dann an den Client verschickt werden. Mit GSON ist es auch möglich, JSON-Objekte zurück in Java-Objekte zu konvertieren.

```
User changedUser = new Gson().fromJson(json, User.class);
```

Hier wird aus dem JSON-Objekt `json` das Java-Objekt `changedUser` erstellt. Dafür benötigt GSON einmal das JSON-Objekt und die Klasse `User.class`.

In einigen Fällen, wie z.B. bei der Knotenliste, besteht ein JSON-Objekt aus zwei unterschiedlichen Java-Objekten. Da GSON jedoch lediglich ein Objekt konvertieren kann, standen zwei Möglichkeiten zur Auswahl. Zum einen durch das Schreiben einer Klasse, die genau das Objekt erzeugt, was in JSON umgewandelt wird. Zum anderen durch den Import der Klassen `org.json.JSONObject`, `org.json.JSONArray` und `org.json.JSONException` und der direkten Erstellung des gewünschten JSON-Objekt aus den Java-Objekten.

JSON auf der Client-Seite

Die für das System auf der Client-Seite implementierte JSON-Unterstützung wird mit dem Paket `com.google.gwt.json.client` realisiert. Dieses bietet Klassen zum Parsen und Erstellen von in JSON kodierten Werten an. Aus einem JSON-Objekt, das zuvor als Antwort auf eine Anfrage an eine Ressource auf der Server-Seite stammt, kann mit wenig Java-Code ein Wert zu einem Schlüssel ausgelesen werden.

3.6 Verteiltes Dateisystem

Jobs im Cloud Computing arbeiten üblicherweise auf Eingabedateien und erzeugen Ausgabedateien. Zur Vorhaltung dieser Dateien wurde die Dateisystem-Komponente entworfen. Diese Komponente stellt einen Speicherort für Dateien in der Cloud bereit, der von anderen Komponenten genutzt werden kann.

3.6.1 Anforderungen und genereller Aufbau

Die wesentlichen Anforderungen an das Dateisystem sind die der Ausfallsicherheit und Hochverfügbarkeit von Daten. Es muss davon ausgegangen werden, dass beliebige Knoten, die Daten vorhalten, jederzeit unplanmäßig die Cloud verlassen können. Die Dateisystem-Komponente muss also für eine redundante Verteilung der Dateien sorgen.

Ähnlich wie der Datenbank-Zugriff soll auch diese Komponente auf jedem Knoten bereitstehen. Dateien, die von anderen Komponenten der lokalen Dateisystem-Komponente übergeben werden, sollen von dieser im Hintergrund in der Cloud verteilt werden. Wenn eine Komponente eine Datei anfragt, muss diese zunächst von der Dateisystem-Komponente aus der Cloud gesammelt und dann an den Aufrufer weitergereicht werden.

3.6.2 Entwurf

Für hohe Ausfallsicherheit und Hochverfügbarkeit der Dateien war geplant, dass die Dateien, die in der Cloud abgelegt werden sollen, in stets gleich große Teile geteilt werden (splitting). Jeder dieser Teile sollte mehrfach in der Cloud auf verschiedenen Knoten abgelegt werden. So sollte sichergestellt werden, dass die Dateien bei einem Knotenausfall nicht verloren gehen. Durch das Aufteilen sollte außerdem für das Beziehen einer Datei ein Geschwindigkeitsvorteil aus parallelen Übertragungen ermöglicht werden. Für das Auffinden von (Teil-)Dateien sollte die Datenbank genutzt werden.

Dieser Entwurf wurde jedoch Aufgrund von Zeitmangel verworfen. Es wurde ein einfacher umzusetzender Lösungsansatz gewählt, bei dem in der Cloud ein einzelner FTP-Server gestartet wird und die Dateisystem-Komponenten alle Dateien auf diesen FTP-Server hochladen. Die Schnittstelle der Dateisystem-Komponente wird jedoch von der Verwendung eines FTP-Servers abstrahiert. Somit wird eine leichte Austauschbarkeit der Implementierung der Dateisystem-Komponente gewährleistet. Die Dateien im verteilten Dateisystem werden durch die Klasse `CloudFile` repräsentiert. Diese Klasse wird in den Methoden der Schnittstellen wie `FileSystem.uploadFile(CloudFile, File, ...)` und `FileSystem.downloadFile(CloudFile, File, ...)` zur Identifizierung der Dateien im verteilten Dateisystem verwendet.

Der Ansatz durch ein FTP-Server ist als Platzhalter zu sehen. Die gestellten Anforderungen von Hochverfügbarkeit und Ausfallsicherheit werden offensichtlich nicht erfüllt, da der FTP-Server einen *single point of failure* darstellt. Dies wurde jedoch für eine schnelle und einfache Implementierung eines verteilten Dateisystems in der Cloud in Kauf genommen. Für alle anderen Aufgaben der Cloud stand so ein Dateisystem zur Verfügung, von dem angenommen werden konnte, es *sei* hochverfügbar und ausfallsicher.

3.6.3 Implementierung

Für die Bereitstellung des FTP-Servers wurde die in komplett Java geschriebene Implementierung Apache FtpServer verwendet. Innerhalb der Cloud startet eine Dateisystem-Komponente den FTP-Server und teilt den anderen Dateisystem-Komponenten die Zugangsdaten zu diesem FTP-Server mit. Das Starten des FTP-Servers wird von der

Monitoring-Komponente ausgelöst. Dazu wird der FTP-Server als Service in dessen Konfiguration eingetragen, so dass das Monitoring selbständig dafür sorgt, dass in der Cloud unter allen Dateisystem-Komponenten stehts ein FTP-Server läuft (siehe 3.1.6 Überwachung der Systemkomponenten).

Die Dateisystem-Komponente stellt zwei Arten des Zugriffs auf die Dateien bereit.

- Bei der ersten Art werden die Dateien an die lokale Dateisystem-Komponente übergeben bzw. von ihr bezogen. Vor anderen Komponenten bleibt die Realisierung des Dateisystems durch einen FTP-Server verborgen. Diese Variante wird z.B. vom ResourceManager verwendet.

- Die zweite Art ist der direkte Zugriff auf den FTP-Server, der von der Dateisystem-Komponente nur vermittelt wird. Durch entsprechende Methoden werden dazu FTP-typische URLs der Form `ftp://user:pass@host/pfad` generiert, mit der ein FTP-Client direkt auf den Server zugreifen kann. Um Probleme mit der internen Verzeichnisstruktur des FTP-Servers zu vermeiden, erlauben die generierten URLs nur einen beschränkten Zugriff auf bestimmte Unterverzeichnisse des FTP-Servers. Diese Variante wird verwendet, um in der Benutzerschnittstelle eine einfache Möglichkeit zum Einreichen und Beziehen von Job-Dateien zu bieten. Diese Art des Zugriffs erforderte eine Erweiterung der Schnittstelle der Dateisystem-Komponente.

3.7 Statistiken

Das Cloud-System bietet die Möglichkeit der Anzeige verschiedener Statistiken. Die Notwendigkeit geht aus der Liste der Anforderungen hervor. Die Umsetzung ist als rudimentär zu bewerten, allerdings werden die relevanten Knoten- und Jobdaten abgedeckt. So könnten die umgesetzten Statistiken für das Accounting verwendet werden, da die hierfür notwendigen Werte bereits erfasst werden. Die Umsetzung eines umfangreichen OSGi-Bundles für Statistiken ist jedoch weiterhin möglich und bietet sich für eine komplexere Umsetzung der Statistikanforderung an.

Die ermittelten Werte der ausgegebenen Statistiken werden der Datenbank (siehe 3.2) entnommen. Dies geschieht mittels dafür erstellter SQL-Anfragen. Die Ausgabe und Berechnung der Statistikwerte erfolgt *on demand*; es werden keine Statistikdaten mittel- oder gar langfristig gespeichert.

Grundsätzlich können für die Statistiken Datengrundlagen verschiedenen Umfangs betrachtet werden. Einem Benutzer in der Rolle User werden die User-Statistiken angeboten, die sich auf seine Jobs beziehen. Es werden aber auch Admin-Statistiken erfasst, die sich auf sämtliche Jobs sowie die Knoten der Cloud beziehen und Benutzern in der Rolle Admin zugänglich sind.

Die gewählten Statistikwerte entsprechen den für einen Admin bzw. User relevanten Daten. So möchte ein Admin über die Anzahl der Knoten in der Cloud und deren Status informiert werden. Die ausgegebenen Statistiken verschaffen einen Überblick über Auslastung der einzelnen Cloud-Knoten sowie über die Performanz der Cloud. Ein User

kann beispielsweise die durchschnittliche Ausführungsdauer von ihm eingestellter Jobs betrachten.

3.7.1 Leistungswerte

Bei der Auswahl der statistischen Werte wurde berücksichtigt, welche grundlegenden Leistungswerte zur Bewertung des Cloud-Systems relevant sind. So sollten Performanz und Leistungsfähigkeit der Cloud bewertbar sein. Die berechneten Leistungswerte sind die durchschnittliche Wartezeit, die durchschnittliche Ausführungszeit sowie die durchschnittliche Verweildauer von Jobs. Die Wahl dieser Leistungswerte lässt sich auf die Betrachtung dieser Werte in wissenschaftlichen Veröffentlichungen über Hochleistungsrechnen zurückführen.

Für durchschnittliche Ausführungszeit von Jobs auf Resource-Managern werden nur die auch tatsächlich erfolgreichen Ausführungen betrachtet. Die durchschnittliche Wartezeit entspricht der Zeit, welche durchschnittlich zwischen der Freigabe und dem Beginn der Ausführung eines Jobs vergeht. Abschließend beschreibt die durchschnittliche Verweildauer die Zeitspanne zwischen Freigabe von Jobs und Abschluss der Berechnung.

3.7.2 User-Statistiken

Grundlage für die angebotenen Statistiken stellen alle Jobs des Benutzers dar. Implementiert sind im Einzelnen

- Anzahl der Jobs
- Durchschnittliche Wartezeit
- Durchschnittliche Ausführungszeit
- Durchschnittliche Verweildauer
- Summe aller Ausführungszeiten

3.7.3 Admin-Statistiken

Für die Rolle Admin werden detailliertere Statistikwerte berechnet. So kann sich dieser für einen von ihm frei wählbaren Bereich in die Vergangenheit anzeigen lassen, wie sich die Leistungswerte verändert haben. Es kann frei gewählt werden, welche Schrittweite (Millisekunde, Sekunde, ..., Monat, Jahr) betrachtet werden soll. Zudem kann die Anzahl der auszugebenden Ergebnisse gewählt werden. Es kann z.B. abgefragt werden, wie sich die durchschnittliche Wartezeit in den letzten 60 Minuten entwickelt hat. Ebenso können aber auch die Werte für den letzten Monat ausgegeben werden. Im Einzelnen sind implementiert:

- Anzahl der Knoten, die jemals in der Cloud registriert waren
- Anzahl der Knoten, welche aktuell in der Cloud registriert sind

- Anzahl der Knoten, welche aktuell keinen Job berechnen
- Anzahl der Knoten, welche aktuell einen Job berechnen
- Anzahl der in der Cloud eingetragenen Jobs
- durchschnittliche Wartezeit über alle Jobs
- durchschnittliche Ausführungszeit über alle Jobs
- durchschnittliche Verweildauer über alle Jobs
- durchschnittliche Wartezeit für einen bestimmten Zeitraum und mit einer bestimmten Ergebnisanzahl
- durchschnittliche Ausführungszeit für einen bestimmten Zeitraum und mit einer bestimmten Ergebnisanzahl
- durchschnittliche Verweildauer für einen bestimmten Zeitraum und mit einer bestimmten Ergebnisanzahl

4 Qualitätsmanagement

Im folgenden Kapitel wird die Organisation der Projektarbeit betrachtet. Neben verwendeten Werkzeugen zur Softwareerstellung und -qualitätssicherung wird auch das Projektmanagementsystem Scrum vorgestellt.

4.1 Werkzeuge

Als Software-Werkzeuge werden Programme oder Anwendungen bezeichnet, die den Softwareentwickler dabei unterstützen verschiedenste Aufgaben, die bei der Entwicklung von Softwareprojekten, entstehen zu bewältigen. Die Werkzeuge bieten dabei Möglichkeiten der Erstellung, der Fehlerbehebung (Debugging), der Versionsverwaltung, der Wartung und über verschiedenste Zusatzanwendungen jede denkbare benötigte Hilfe. Im Folgenden werden die eingesetzten Versionskontroll- und Build-Systeme beschrieben.

4.1.1 Versionsverwaltung: Subversion

Für die gemeinsame Arbeit an Dokumenten und an Quellcode wird das Versionsverwaltungssystem Subversion (kurz SVN) benutzt.

Bei Subversion werden die Dateien in einem zentralen Projektarchiv (*repository*) abgelegt. Die Versionierung erfolgt in Form einer einfachen Revisionszählung. Wenn Inhalte geändert werden, werden zwischen dem Projektarchiv und dem Bearbeiter jeweils nur die Unterschiede zu bereits vorhandenen Versionen übertragen.

Einer der Gründe für die Verwendung von SVN ist die Notwendigkeit der verteilten, gleichzeitigen Bearbeitung von Projekten. Es ist für verschiedene Personen möglich, Änderungen an ein und derselben Datei zu machen, ohne dass die Änderungen verloren gehen oder die Datei umständlich ausgetauscht werden muss. Zweier Versionen der gleichen Datei können im Zweifelsfall (bei sich überschneidenden Änderungen) verglichen und die eigenen oder andere Änderungen übernommen werden. Durch dieses *mergen* bleiben Änderungen zweier oder mehr Bearbeiter erhalten.

Eine weitere wichtige Funktion ist die Arbeit mit älteren Versionen von Dateien. Auf diese Art kann zu alten, als fehlerfrei bekannten Versionen zurückgekehrt werden oder neu auftretende Fehlerquellen eingegrenzt werden. Auch die Markierung von Versionen, z.B. zur Markierung von Veröffentlichungen, ist möglich.

4.1.2 Build-System: Maven 2

Maven 2 ist ein Build- und Projektmanagement-Tool der Apache Software Foundation. Die Nutzung eines Build-Tools wurde als unverzichtbar erachtet, da der Arbeitsaufwand

für die Erstellung eines Builds minimal gehalten werden sollte. Die wichtigste Aufgabe eines Build-Tools ist die schnelle und einfache Erstellung von ausführbaren Programmen. Dies umfasst unter anderem das Kompilieren des Quellcodes des Projekts oder auch die Erstellung von Manifest-Dateien. Im Idealfall ist dazu nur eine Nutzeraktion notwendig, welche den Build-Vorgang anstößt. Daraufhin werden alle erforderlichen Vorgänge automatisch ausgeführt. Die manuelle Erstellung eines Builds stellt einem enormen Aufwand dar. Dieser Aufwand wird durch den Einsatz eines Build-Tools minimiert. Darüber hinaus bietet Maven 2 noch weitere Funktionalitäten wie das Maven Dependency Management oder den Maven Lifecycle, welcher die bei der Arbeit an einem Projekt zu durchlaufenden Phasen abbildet.

Der Einsatz von Maven 2 wurde ohne die detaillierte Betrachtung von Alternativen beschlossen. Eine dieser Alternativen wäre beispielsweise Apache Ant gewesen. Vor dem Einsatz von Maven 2 erfolgte eine Betrachtung der Grundzüge und des Konzeptes. Nach dieser Betrachtung von Maven 2 fiel die Entscheidung für die Verwendung von Maven 2. Die Vorteile von Maven 2 gegenüber Apache Ant wurden unter Anderem im Dependency Management von Maven gesehen. Ein weiteres wichtiges Argument für Maven 2 ist die stark ansteigende Zahl von Projekten, welche Maven als Build-Tool nutzen. Maven avanciert zu einem Standardwerkzeug, daher liegt großer Nutzen im Sammeln von Erfahrung mit diesem Werkzeug. Nicht zuletzt bedeutet die große Maven-Community einen nicht zu verachtenden Vorteil. Diese sorgt durch die Erstellung von immer neuen Maven-Plugins für eine ständige Vergrößerung des Einsatzspektrums von Maven 2.

Der Einsatz von Maven 2 bedeutete zunächst großen Zeitaufwand für die Einarbeitung in die Grundzüge der Bedienung. Dies lässt sich daran ablesen, dass die Abstellung eines Gruppenmitglieds zur Einführung und Wartung des Build-Systems nötig war. Diese hohe Investition stellte sich jedoch als gerechtfertigt heraus. So ist nach einer Periode mit hohem Zeitaufwand nur noch geringer Aufwand für die Anpassung und Wartung des Build-Systems notwendig.

Eine detaillierte Beschreibung von Maven 2 lässt sich auf den Projektseiten des Apache Maven Projects[1] finden.

Bedeutung und Integration von Maven 2

Maven 2 ist essenzieller Bestandteil der Entwicklung des Cloud-Systems. Sämtliche Einzelprojekte setzen auf Maven auf. Durch die Konfiguration der verwendeten Maven-Plugins und durch das Einfügen der benötigten Maven-Dependencies wurde jedes Projekt individuell angepasst. Das Ziel war es, eine ausführbare Laufzeitumgebung zu erhalten, welche den aktuellsten in den Einzelprojekten vorhandenen Quellcode als Grundlage nutzt. Dieses Ziel sollte mit möglichst wenigen Benutzeraktionen erreichbar sein. So wird bei Aufruf der Maven-Phase *package* auf das Reaktorprojekt jedes einzelne Modul dieses Projektes erstellt. Die Module dieses Projektes entsprechen den Einzelprojekten der Projektgruppe.

Begonnen wird mit Aufruf der Maven-Phase `package` auf das Projekt der Laufzeitumgebung (siehe 3.1.3). Das Ergebnis ist eine ausführbare Laufzeitumgebung, welche

[1] `http://maven.apache.org/`

innerhalb von Eclipse oder durch Ausführen der JAR-Datei im Projektverzeichnis gestartet werden kann. Die durchgeführeten Schritte sind im Einzelnen:

1. Kopieren sämtlicher Fremdbibliotheken, welche als Bundle genutzt werden, an den für diese Bibliotheken spezifizierten Ort.
2. Kopieren sämtlicher Fremdbibliotheken, welche nicht als Bundle genutzt werden, aber für die Ausführung der Laufzeitumgebung ausserhalb von Eclipse benötigt werden, an den für sie spezifizierten Ort.
3. Kopieren von stabilen, in einem zentralen Archiv vorgehaltenen Versionen der Komponenten an den für sie spezifizierten Ort.
4. Erstellung einer JAR-Datei für die Laufzeitumgebung
5. Kopieren der erzeugten JAR-Datei in das Verzeichnis `/target/classes/`.

Nach Auführung dieser Schritte ist die Laufzeitumgebung ausführbar.

Nach Erstellung der Laufzeitumgebung werden sowohl die einzelnen Komponenten als auch die webbasierte Nutzerschnittstelle erstellt. Letztere wird als WAR-Datei ausgeliefert und in das für sie spezifizierte Verzeichnis der Laufzeitumgebung kopiert.

Bei sämtlichen Komponenten verläuft die Erstellung im Wesentlichen gleich. Grundlegende Punkte sind hier

- dass eine Bundle-Manifest Datei erzeugt und in das entstehende Bundle eingefügt wird,
- dass Abhängigkeiten, welche zur Laufzeit benötigt werden, im entstehendenBundle enthalten sind und
- dass das entstehende Bundle an den für es spezifizierten Ort im Projekt der Laufzeitumgebung kopiert wird; dabei wird die vorher kopierte, stabile Version ersetzt.

Nach Abschluss der Maven-Phase *package* wurden also sämtliche Bibliotheken an die für sie spezifizierten Orte kopiert. Es wurden außerdem für alle geänderten Komponenten aktuelle Versionen in die Laufzeitumgebung eingefügt.

Damit die genannten Vorgänge in der beschriebenen Form zusammenarbeiten, ist eine Anzahl von Maven-Plugins und deren Konfiguration notwendig. Diese werden im folgenden Abschnitt über die verwendeten Maven-Plugins beleuchtet.

Im abschließenden Abschnitt dieses Kapitels wird das Bundle Template beleuchtet. Dieses wurde erstellt, um als Vorlage für Komponenten-Projekte zu dienen.

Verwendete Plugins

Plugins spielen bei der Verwendung von Maven 2 eine gewichtige Rolle, da einzelne Ziele des Build-Prozesses durch diese gesteuert bzw. ausgeführt werden. Das Bundle Plugin for Maven sorgt beispielsweise für die Erstellung von OSGi-Bundles. Die große Bedeutung

dieses Vorgangs für die Verwendung von Sinefa lässt sich auch im Abschnitt „Die Laufzeitumgebung“ (3.1.3) nachlesen. Eine Erstellung von OSGi-Bundles ohne dieses Plugin würde einen immensen Zeitaufwand bedeuten. Durch die Verwendung dieses Plugins mit einer nahezu identischen Konfiguration in jedem Bundle wird dies vermieden. Das Plugin sorgt sowohl für die Erstellung des Bundle-Manifests für das jeweilige Bundle, als auch für die Auslieferung der notwendigen Abhängigkeiten im resultierenden Bundle.

Das Maven Antrun Plugin sorgt in jedem Einzelprojekt des Cloud-Systems für das Kopieren der resultierenden Ausgabedatei an die vorgesehene Stelle. Dieses Plugin ist für die Umsetzung von Apache Ant Befehlen in Maven 2 verantwortlich.

Zur Erstellung der webbasierten Nutzerschnittstelle wird das Google Web Toolkit Maven Plugin verwendet. Die webbasierte Nutzerschnittstelle ist ein Projekt, welches das Google Web Toolkit nutzt.

Bei der Entwicklung von Sinefa wurde noch eine Reihe weiterer, in ihrer Bedeuting nicht hoch anzusehender Plugins verwendet, auf die an dieser Stelle nicht weiter eingegangen werden soll, da dies den Rahmen dieses Berichtes überschreiten würde.

Es bleibt festzuhalten, dass die Verwendung von Maven-Plugins eine erhebliche Einsparung von Zeit bedeutet, da diese nach ihrer Konfiguration selbsttätig die ihnen gestellten Aufgaben abarbeiten.

Das Bundle-Template im Verlauf des Projekts

Das Bundle-Template bildet den Grundrahmen für alle Maven OSGi-Bundles und wurde zum Ende der Entwurfsphase hin erstellt. Es bietet eine voreingestellte Konfiguration für ein OSGi-Bundle und muss von Entwicklern eines Bundles für das Cloud-System nur geringfügig angepasst werden. So müssen im Project Object Model etwa der Name des Bundles sowie die Einstellungen für das Bundle Plugin for Maven angepasst werden. Desweiteren müssen die Abhänngigkeiten angepasst werden, da in diesem Bereich jedes Bundle individuell ist. Nach den nötigen Anpassungen ist das Projekt für den Einsatz bereit.

Das Bundle-Template wurde im Verlauf der Entwicklung aufgrund neuer Erkenntnisse oder Anforderungen mehrfach angepasst. So wurde beispielsweise durch die Nutzung der Declarative Services von OSGi das Anlegen einer `component.xml` in jedem Bundle nötig. Aus diesem Grund musste auch das Bundle-Template dahingehend angepasst werden.

In der aktuellen Version des Bundle-Templates bietet das Template alle Konfigurationen, um ein Bundle zu erstellen, das mit dem Cloud-System verwendet werden kann. So umfasst die aktuelle Version sowohl eine vollständige Konfiguration des Bundle Plugin for Maven, als auch eine Konfiguration für das Maven Antrun Plugin.

4.2 Softwarequalitätsmanagement

4.2.1 Testen der Komponenten

Das Testen von Komponenten während und nach der Implementierung ist ein wichtiger Aspekt der Sicherung von Softwarequalität. Durch Tests soll verifiziert werden, dass die

Komponenten genau die Aktionen durchführen, die im Entwurf spezifiziert wurden. In der Projektarbeit wurden zwei Typen von Tests verwendet: Unit Tests und Integration Tests.

Unit Tests

Bei einem Unit Test werden die einzelnen Klassen unabhängig voneinander getestet. Dabei werden die Methoden einer Klasse aufgerufen und geprüft, ob die Aktionen der Methoden fehlerfrei durchgeführt wurden. Dabei geht es nicht nur um Eingaben, die später in der Anwendung erwartet werden, sondern auch um bewusst fehlerhaft konstruierte oder für problematisch gehaltene Eingaben. Es muss desweiteren geprüft werden, ob die Methoden auf Fehler mit den spezifizierten Exceptions reagieren. Das Verhalten der Methoden muss daher dokumentiert werden. Dies kann mit Javadoc geschehen, in dem die Parameter beschrieben werden und angegeben wird, welche Exceptions die Methoden werfen können.

Ein Unit Test besteht aus sog. Test Cases. Sie enthalten eine Menge von Methoden die vom Test-Framework aufgerufen werden. Jede dieser Methoden wird entweder erfolgreich durchlaufen oder es wird eine Fehlermeldung erzeugt. Wenn nun alle Tests eines Test Cases erfolgreich durchgeführt worden sind gilt der gesamte Test Case als erfolgreich.

Um Unit Tests durchzuführen, existieren für Java einige Test-Frameworks wie JUnit3, JUnit4 und TestNG. Die Frameworks unterscheiden sich dabei nur in den Features und in den Schnittstellen für die Spezifikation von Tests.

In unserem Projekt können nur Unit Tests für Entity-Klassen durchgeführt werden, da die anderen Klassen und Komponenten eine OSGi-Laufzeitumgebung benötigen (siehe Kapitel 4.2.1). Somit existieren Unit Tests nur für Klassen wie Job und – für einige triviale Fälle wie Null-Pointer-Zugriffe – Unit Tests für die Implementierungen vom Job-Manager und Resource-Manager. Dabei handelt es sich um JUnit4-Tests, welche auch beim Build-Prozess von Maven (siehe Kapitel 4.1.2) durchlaufen werden. Wenn ein Test fehlschlägt, wird so der ganze Build-Prozess von Maven abgebrochen. Dadurch wird sichergestellt, dass keine fehlerhaften Projekte, Komponenten oder OSGi-Bundles erstellt werden.

Integration Tests

Bei einem Integration Test wird das Zusammenspiel von verschiedenen Klassen und Komponenten untereinander getestet. Die Komponenten laufen als Bundles in einer OSGi-Umgebung. Dies macht das Testen der Komponenten deutlich schwieriger, da die Tests auf die OSGi-Umgebung zugreifen müssen.

Zum Testen von OSGi-Applikationen existieren einige Frameworks wie JUnit4OSGi. Es gibt dabei zwei Konzepte, wie die Integration Tests durchgeführt werden.

1. Die erste Möglichkeit für Integration Tests besteht darin, die komplette OSGi-Applikation als Blackbox zu sehen. Somit führt man keinen Integration Test als solchen, sondern einen Unit Test durch und greift von außen auf die OSGi-Applikation zu (siehe Abbildung 4.1).

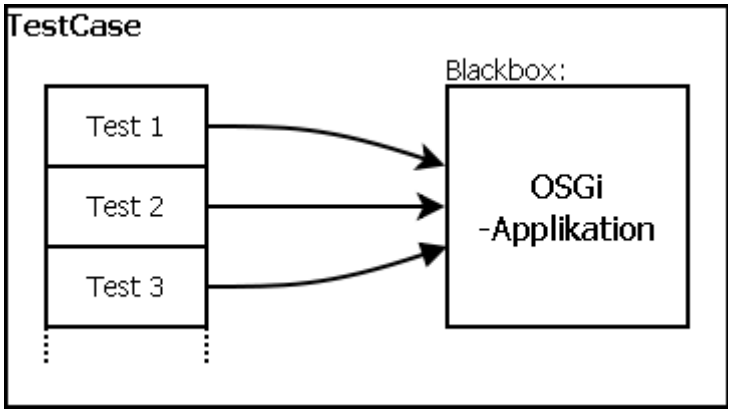


Abbildung 4.1: Testen einer OSGi-Applikation mit einem Unit Test

Auf diese Weise kann mit jedem Unit-Test-Frameworks wie JUnit4 ein Unit Test geschrieben werden. Dabei wird im Test Case das OSGi-Framework geladen und über dessen API auf den BundleContext zugegriffen. Über den BundleContext können dann die Bundles erreicht und die Tests entsprechend durchgeführt werden. Der Pseudocode im Listing 4.1 zeigt, wie zuerst die OSGi-Applikation gestartet wird und wie die Test-Methoden darauf zugreifen.

```
public class TestCaseFoobar {
    private Felix f;
    private BundleContext c;

    @Before
    public void setUp() {
        // starten der OSGi-Applikation
        this.f = new Felix();
        this.f.configureAndStart();
        // Bundle-Context laden
        this.c = this.f.getBundleContext();
    }

    @Test
    public void checkValueOfFoobar() {
        // arbeite mit "this.c" in den Testmethoden
    }
}
```

Listing 4.1: Pseudocode für den Zugriff auf ein OSGi-Framework (z.B. Felix)

2. Die zweite Möglichkeit besteht darin, die Test Cases in die OSGi-Bundles einzufügen und die Tests aus den OSGi-Bundles heraus zu starten. Das JUnit4OSGi-Framework verwendet dieses Konzept (siehe Abbildung 4.2).

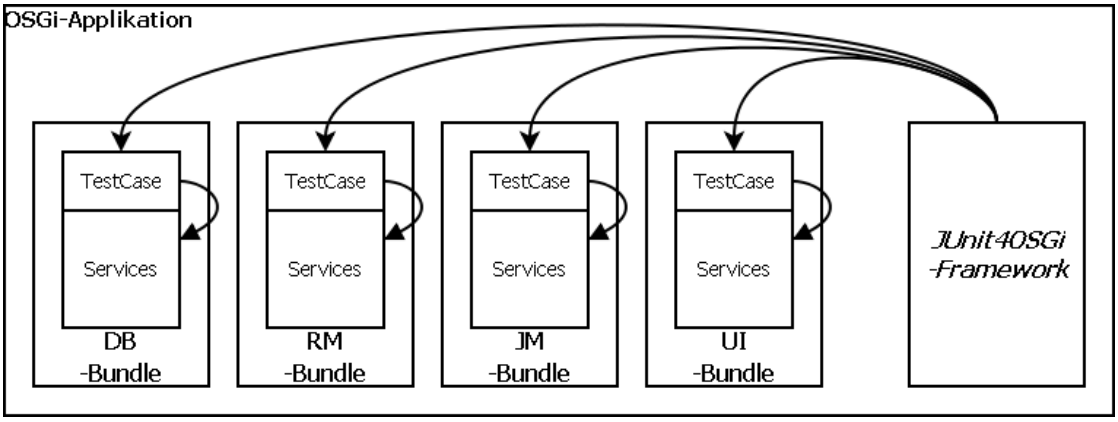


Abbildung 4.2: Testen der Bundles aus der OSGi-Applikation

> Die JUnit4OSGi-Bundles in der OSGi-Applikation erkennen beim Starten die Test Cases in den anderen Bundles. Durch den Aufruf `junit id` in der Felix-Konsole wird der TestCase vom Bundle mit der angegeben ID gestartet. Um die TestCases aller Bundles zu starten kann der Aufruf `junit all` verwendet werden.

Für die Integration Tests des Cloud-Systems wurde entschieden das JUnit4OSGi-Framework zu verwenden. In die bereits vorhanden OSGi-Umgebung mussten nur die JUnit4OSGi-Bundles eingefügt werden, um das Test-Frameworks zu installieren. Desweiteren muss in den Test Cases der Bundles weder die OSGi-Applikation gestartet noch muss die API des verwendeten OSGi-Frameworks verwendet werden. Dadurch, dass die Tests innerhalb der OSGi-Applikation laufen, erhalten diese automatisch Zugriff auf den BundleContext und somit den Zugriff auf alle Bundles in der Laufzeitumgebung. Zusätzlich können auch Bundles getestet werden die keine API außerhalb der OSGi-Applikation bereitstellen. Das Testen solcher Bundles wäre mit Unit-Test-Frameworks nicht möglich.

4.2.2 Trac

Trac ist ein freies, webbasiertes Projektmanagement-Werkzeug der Firma Edgewall Software. Die Entscheidung, Trac zu nutzen, fiel nach vergleichender Betrachtung von Trac und Jira.

Jira ist ein kostenpflichtiges Produkt des Unternehmens Atlassian. Jira bietet im Vergleich zu Trac eine umfangreichere, ausgereiftere Ticketverwaltung. So ist es mit Jira beispielsweise möglich, Untertickets zu einem bestehenden Ticket anzulegen. Dies kann wünschenswert sein, um z.B. Abhängigkeiten zwischen Aufgaben abzubilden, und ist durch die Ticketverwaltung von Trac nicht nachbildbar. Trac bietet im Gegensatz zu Jira ein integriertes Wiki. Da die Notwendigkeit eines solchen Wikis zu Dokumentations- und Organisationszwecken jedoch gegeben war, hätte bei einer Entscheidung für Jira ein externes Wiki erstellt oder genutzt werden müssen. Nach Abwägung der Vor- und Nach-

teile der jeweiligen Produkte fiel die Wahl auf das kostenfreie Trac, da dessen Nachteile im Vergleich zu Jira durch die entstehenden Kosten für dieses aufgewogen wurden.

Trac wird zur Verwaltung von Tickets sowie zu Dokumentations- und Organisationszwecken genutzt. Zudem findet sich hier eine Timeline, es ist möglich den Quellcode des SVN-Repositories anzuzeigen und die Roadmap zu betrachten, welche die Milestones des Projekts und deren Erfüllung und Beschreibung auflistet. Zu erledigende Aufgaben werden in Tickets festgehalten. In diesen findet sich eine Beschreibung der Aufgabe, zudem lassen sich dort Fortschritte dokumentieren. Tickets werden einer Prioritätsstufe zugeordnet und es ist möglich, ein Ticket einer bestimmten Person zuzuordnen.

Von großem Nutzen ist auch die Timeline, welche chronologisch sämtliche Änderungen am Wiki und an Quelltexten aufführt. Anhand dieser Liste lassen sich die letzten Änderungen am Projekt rekapitulieren.

Während der Projektarbeit diente das integrierte Wiki unter anderem zur Dokumentation. Es wurde benutzt, um Arbeitsabläufe, Fehlerbeschreibungen, die Rest-API sowie verschiedene Anleitungen verfügbar zu machen. Weiterhin lässt sich die Umsetzung von Scrum (siehe 4.3) wie dort beschrieben durch Wiki und Ticketsystem unterstützen.

4.3 Projektmanagement

Zu Beginn der Projektarbeit kam die Frage auf, nach welchem Projektmanagementsystem sich die Gruppe organisieren sollte. Durch die Einführung eines Projektmanagementsystems werden Führungsaufgaben festgelegt, die die Arbeit koordinieren und steuern. Es dient der erfolgreichen Planung, Organisation und Steuerung von Aufgaben mit dem Ziel, das Projekt erfolgreich durchzuführen.

Zur Auswahl stand neben dem klassischen Projektmanagement auch modernere agile Projektmanagementsysteme wie das Scrum-System. Das klassische Projektmanagement folgt dem Wasserfallmodell, d.h. die einzelnen Phasen folgen aufeinander aufbauend, ohne das ein Rücksprung zu einer früheren Planungs-Phase erfolgt.

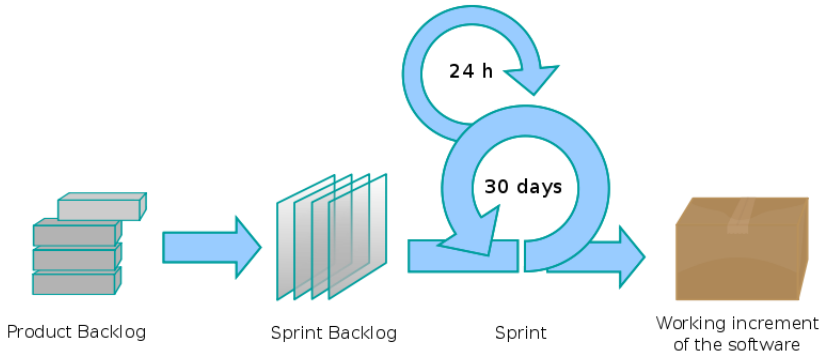


Abbildung 4.3: Agiles Projektmanagementsystem Scrum

In Scrum erfolgt die Projektplanung nach einem anderen Prinzip, das in Abbildung 4.3 dargestellt wird. Bei Scrum werden zunächst alle Ziele gesammelt, die während des Projektes umgesetzt werden sollen. Die Ziele werden priorisiert und bilden ein sogenanntes Product Backlog. Diese Aufgabe wird bei Scrum vom sogenannten Product Owner durchgeführt. Ein Sprint, ein Zeitraum von bis zu 30 Tagen, wird für die Umsetzung eines Teils des Product Backlogs eingesetzt. Die Ziele eines Sprints werden in einem Sprint Backlog festgehalten. Dieses setzt sich aus den ausgewählten, hoch priorisierten Zielen des Product Backlogs zusammen. Die Ziele eines Sprints werden vom Developer-Team umgesetzt, das sich selbst organisiert. Bei Scrum erfüllt ein Gruppenmitglied außerdem die Rolle des Scrum Masters. Seine Aufgabe ist, dafür zu sorgen, dass das Arbeitsumfeld für das Developer-Team optimal ist und eventuelle Kommunikationsprobleme gelöst werden. Außerdem leitet der Scrum Master die regelmäßigen Treffen des Teams, in denen jeder über Fortschritte und Probleme bei der Erledigung seiner Aufgaben berichtet.

Die Projektgruppe hat sich für die Benutzung von Scrum ausgesprochen. Der Vorteil von Scrum ist die agile und dynamische Art, die bei der klassischen Projektplanung nicht gegeben ist. Ein weiteres positives Merkmal von Scrum ist das in Sprints bereits fertig bearbeitete Aufgaben erneut aufgegriffen werden können.

Das Projektmanagement wurde durch das webbasierte Software-Werkzeug Trac (siehe 4.2.2) unterstützt; die Backlogs konnten in einem Wiki und die verteilten Aufgaben im Ticketsystem verwaltet werden.

4.3.1 Gruppeneinteilung und Arbeitsweise

Die Arbeiten in der Projektgruppe wurde anfangs von allen Teilnehmern zusammen durchgeführt. Dies diente hauptsächlich dazu, ein einheitlichen Wissenstand zu erreichen. Später wurden Kleingruppen gebildet, um mehrere Aufgaben zeitgleich zu bearbeiten. Zu diesen Aufgaben gehört unter anderem:

- Untersuchung von hochverfügbaren Datenbanken
- Konzept zum Starten und Stoppen von Diensten
- Auswahl eines Web 2.0-Frameworks
- Auswahl von Werkyeugen für die Softwareentwicklung
- Funktionsweise vom OpenID-Projekt
- Erstellung von Code-Konventionen

Mit Beginn der Implementierungsphase wurden die Konzepte von Scrum umgesetzt. Dazu wurde die Gruppe in die Rollen des Product Owners, Scrum Masters und des Developteams eingeteilt.

Die Projektgruppe hat sich anfangs zweimal pro Woche mit den Betreuern getroffen, um Fortschritte und Zwischenergebnisse vorzustellen und zu diskutieren. Zu jeder Sitzung wurde eine Protokollführer bestimmt, der die Ergebnisse und Aufgaben im Wiki des Trac einpflegte. Die Treffen der Kleingruppen wurde selbständig geplant und durchgeführt.

4.3.2 Fazit

Die Rolle der Product Owner war für die Teamarbeit sehr bedeutend, da diese im Vorfeld eine Priorisierung der Aufgaben und Tätigkeiten vollzogen haben. Dies reduzierte den Aufwand in jeder Entscheidungsphase und steuerte die Entwicklungsarbeit. Das Entwicklungsteam konnte zum Sprintanfang auf eine priorisierte Liste von Aufgaben zurückgreifen, die innerhalb des Sprintes umgesetzt werden sollten. Die Anzahl der angenommen Sprintziele war anfangs allerdings so groß, dass die Sprintziele nicht eingehalten werden konnten. Mit zunehmender Erfahrung konnte der Umfang der angenommenen Anforderungen besser eingeschätzt werden, so dass die ausgewählten Ziele im Allgemeinen erreicht wurden.

Die Rolle des Scrum Masters wurde innerhalb der Projektgruppe stärker in den Vordergrund gestellt, als es das Scrum-System ursprünglich vorsieht. Der Scrum Master diente nicht nur als Schnittstelle zwischen Product-Owner und Entwicklungsteam, sondern sorgte auch für einen strukturierten Ablauf bei den Sitzungen.

In die Entwicklungsarbeit integriert wurden die Backlogs der Product-Owner und des Entwicklungsteams. So entstand eine Dokumentation über die geleistete Arbeit.

Ein Kernstück des Scrum-Systems, die Daily Scrums, konnten allerdings nicht umgesetzt werden. Dies lag vor allem daran, dass das Team nicht täglich zusammentraf; in diesem Punkt ist Scrum nur schwerlich auf die Arbeit in einer Projektgruppe anwendbar.

Abschließend sei bemerkt, dass der anwendbare Teil von Scrum eine gute Vorbereitung auf die berufliche Tätigkeit darstellt. Dabei ist insbesondere die Präsentation der umgesetzen Anforderungen gegenüber dem Entwicklungsteam und dem Product Owner zu nennen.

5 Ergebnisse

Die sich aus den Anforderungen ergebenden Aufgaben der Projektgruppe waren vielfältig und erforderten die Einarbeitung in unterschiedlichste Entwicklungsumgebungen und Technologie-Standards. Die Projektgruppe konnte in vielen Bereichen auf bereits vorhandene Frameworks oder Programmbibliotheken zurückgreifen. Das ambitionierte Ziel, eine peerbasierte Cloud zu entwickeln, bedeutete aber natürlich, dass viel Zeit für die Implementierung eigener Entwicklungen aufgewendet werden musste. Im Zuge dessen mussten bei einigen Komponenten des Systems Einschränkungen und Kompromisse beim Grad der umgesetzten Funktionalität hingenommen werden. Diese Einschränkungen betreffen jedoch bei keiner der entwickelten Komponenten die als Projektziele definierten Anforderungen an das Gesamtsystem, sondern sind in erster Linie Einschränkungen von Merkmalen gegenüber einem produktiven Cloud-System, deren praxistaugliche Bearbeitung im Rahmen dieser Projektgruppenarbeit nicht vorgesehen war.

Ein besonderes Augenmerk, auf das bei allen Komponenten und damit auch beim Gesamtsystem gelegt wurde, ist der strikte modulare Aufbau. Diese Modularität wird es künftigen Projektgruppen oder Diplomanden erleichtern, einzelne Komponenten ohne größere Schwierigkeiten um gewünschte Funktionalität zu erweitern oder gar vollständig zu ersetzen.

Im Folgenden soll ein Überblick über die erzielten Ergebnisse des erstellten Cloud-Systems und offene Aufgaben zur Umsetzung eines produktiven Cloud-Systems gegeben werden. Dabei wird zunächst der Fokus auf das Grundsystem als Cloud-Middleware gelegt und im Folgenden die darauf arbeitenden Komponenten betrachtet.

5.1 Grundsystem

Das gegebene Ziel der Projektgruppe war es, eine Middleware für ein peerbasiertes Cloud-System zu entwickeln und zu implementieren, auf dem die für die Funktionalität benötigten Komponenten arbeiten und überwacht werden können. Wie im Kapitel Grundsystem detailliert beschrieben, ist es gelungen, dieses auf Grundlage einer OSGi-Laufzeitumgebung zu erstellen. OSGi erwies sich dabei als geeigneter Ausgangspunkt für die Implementierung. Die kleineren Schwierigkeiten, die sich bei der Verwendung ergaben, sind ausschließlich auf die spezifischen Eigenarten von OSGi zurückzuführen und konnten nach einer Einarbeitungsphase überwunden werden.

5.1.1 Netzwerk

Die Kommunikation zwischen den verteilten Diensten folgt dabei dem P2P-Konzept, das mit Hilfe von JXTA realisiert wurde. Mit JXTA wurde die geforderte P2P-

Basiskommunikation erfolgreich geschaffen. Im Zuge dieser Projektarbeit konnten jedoch nicht alle Aspekte dieses Frameworks ausschöpfend untersucht werden, so dass hinsichtlich einer Weiterentwicklung in diesem Bereich eine Überarbeitung nicht auszuschließen ist.

5.1.2 Überwachung von Komponenten

Die Überwachung der auf der Middleware aufsetzenden Dienste erfolgt über das integrierte Monitoring-System. Das Monitoring ist eine vollständige Eigenentwicklung und greift, im Gegensatz zur Laufzeit- und Kommunikationsumgebung, auf kein bestehendes Framework zurück. Die Überwachung erfolgt anhand von definierten Regeln, die Strategien für das Starten ausfallender Dienste und Beenden überflüssiger Dienste bereitstellen. Das Monitoring erfüllt dabei alle geforderten Eigenschaften.

Für eine Weiterentwicklung kommt eine Erweiterung der Verwaltung der Regeln in Betracht. Derzeit muss darauf vertraut werden, dass die bestehenden Regeln auf allen beteiligten Knoten gleichermaßen gelten. Hier sollte zukünftig eine Lösung berücksichtigt werden, die es ermöglicht, die Regeln aller Knoten zu jeder Zeit synchron und damit konsistent zu halten. Auch das Ändern der bestehenden Regeln zur Laufzeit ist bisher nicht möglich, aber als weiterer Entwicklungsschritt wünschenswert.

Zusammenfassend lässt sich sagen, dass die erstellte Middleware einerseits alle Forderungen der Projektziele erfüllt, andererseits aber auch geeignet ist, als Basis für ein produktives Cloud-System zu dienen.

5.2 Dienst-Komponenten

Im Folgenden wird betrachtet, welche Funktionalität von den aufsetzenden Komponenten bereitgestellt wird. Insbesondere wird auch ein Ausblick über mögliche Erweiterungen im Hinblick auf ein produktives Cloud-System gegeben.

5.2.1 Verteiltes Dateisystem

Um den Anforderungen nach Hochverfügbarkeit und Ausfallsicherheit auch für das verwendete Dateisystem Rechnung zu tragen, wurden durch die Projektgruppe einige bestehende verteilte Systeme für diesen Zweck betrachtet. Jedoch stellte sich heraus, das die verfügbaren Lösungen weder der Forderung nach einfacher Einbettung in das entwickelte System, noch hinsichtlich der durch die verwendete Entwicklungsumgebung vorgegebenen Systemvoraussetzungen entsprachen.

Die Anforderungen für eine eigene Entwicklung eines hochverfügbaren, verteilten Dateisystems wurden durch die Projektgruppe identifiziert und mit der Projektgruppenleitung diskutiert. Hierbei wurde auch der enorme zeitliche Aufwand für eine solche Entwicklung deutlich. Da der modulare Aufbau des Systems einen späteren Austausch der Dateisystem-Komponente problemlos ermöglicht, wurde beschlossen, die Forderung

nach Ausfallsicherheit für diese Komponente zu vernachlässigen. Aus diesem Grund wurde eine einfache Ausprägung eines Dateisystems in Form eines FTP-Servers gewählt.

Die grundlegenden Funktionalitäten werden bereitgestellt, zukünftig könnte aber ein verteiltes hochverfügbares Dateisystem eingesetzt werden. Es könnten etwa Dateien in Teilstücke – „Slices“ – aufgeteilt werden, die redundant über das Dateisystem verteilt vorgehalten werden. Als Weiterführung dieses Gedanken könnten dann die einzelnen Teile einer angeforderten Datei parallel von unterschiedlichen, naheliegenden Knoten abgerufen werden. Hierfür seien Distributed Hash Tables (DHT) als Ansatz oder die Projekte „Freenet“[1] und „GNUnet“[2] als Vorbilder genannt.

5.2.2 Resource-Manager

Als Ausführungsdienst der Cloud in den Projektgruppenzielen beschrieben, erfüllt der Resource-Manager alle gestellten Anforderungen. Der Resource-Manager kann die vom Job-Manager übermittelten Jobs ausführen und alle den Job betreffenden Status-Änderungen an den Job-Manager übermitteln. Ebenso können laufende Jobs abgebrochen werden. Desweiteren erfüllt der Resource-Manager die Forderung, die Job-Dateien in ein Dateisystem zu speichern oder aus diesem zu beziehen. Jegliches Fehlverhalten eines Resource-Managers wird durch eine entsprechende Ausnahmebehandlung aufgefangen.

Einschränkungen in der Realisierung des Resource-Managers betreffen funktionelle Aspekte. Grundsätzlich ist jeder Resource-Manager des Systems in der Lage, jede Art von Job auszuführen. Der Zeit ist die Art der zu verarbeitenden Jobs jedoch auf die vom Administrator freigegebene Programmliste beschränkt. So können eigene Anwendungen, beispielsweise unter Zuhilfenahme von VM-Images, nur dann durch den Benutzer eingestellt werden, wenn der Administrator eine entsprechende Software der Programmliste hinzufügt.

Bei der Entwicklung lag der Fokus zunächst auf der Fertigstellung der Grundfunktionalitäten wie Annehmen und Ausführen von Jobs. Hier kamen vor allem solche Jobs zum Einsatz, die nur eine geringe Laufzeit aufweisen, um Fehler der Implementierung effizient beheben zu können. Für solche Jobs war es nicht notwendig, Funktionen zum Pausieren eines Jobs anzubieten. Für Jobs, deren Laufzeit um ein vielfaches höher sind als die bisher verwendeten, könnte es jedoch interessant sein, einen Job anzuhalten und den Bearbeitungsfortschritt und Status zu sichern, um den Job auf einem anderen Resource-Manager weiterarbeiten zu lassen. Diese sicher nicht triviale Funktionalität ließe sich unter Umständen bei der Verwendung von virtuellen Maschinen zur Ausführung mit vertretbarem Aufwand umsetzen.

5.2.3 Job-Manager

Auch der Job-Manager erfüllt alle grundlegenden gewünschten und benötigten Funktionen. Daten werden in eine Scheduling-Liste eingetragen und an „unbeschäftigte“

[1] http://freenetproject.org/
[2] http://www.gnu.org/software/gnunet/gnunet.de.html

Resource-Manager übergeben. Dabei wird der Lebenszyklus der verarbeiteten Jobs zuverlässig umgesetzt.

Nach der Zielsetzung für die Projektgruppe sollte der Job-Manager lediglich die Scheduling-Strategie First-Come-First-Serve (FCFS) realisieren. Ein zukünftiges Ziel für eine Weiterentwicklung könnte hier die Implementierung nicht-trivialer Scheduling-Strategien sein, die dann im Gegensatz zur jetzigen Implementierung ihre Realisierung in einer eigenen Komponente finden könnten. So könnte dann der Administrator des Systems die Scheduling-Strategie gemäß den Verwendungsbedürfnissen wählen.

Auch hinsichtlich der Job-Zuweisung könnte der Job-Manager dahingehend erweitert werden, dass bei der Auswahl von ausführenden Knoten zusätzliche Benutzeranforderungen berücksichtigt werden, beispielsweise die Anzahl der benötigten CPUs, Arbeitsspeicher oder ähnliches.

5.2.4 Verteilte Datenbank

Die Datenbank ist keine explizit durch die Projektgruppenbeschreibung geforderte Komponente des Systems, sondern eine durch die Ausprägung der Architektur gewachsene Notwendigkeit für die Datenhaltung (siehe 3.2). Durch die Verwendung von HA-JDBC und HSQLDB konnte eine saubere Kapselung zwischen Datenbankzugriff und Datenbanksoftware erfolgen. Vorteilhaft durch die Verwendung von HA-JDBC und HSQLDB war zu dem, dass alle benötigten Grundlagen für Hochverfügbarkeit und Verteilung durch dieses Framework bereits vorhanden sind. Als nachteilig erwies sich HA-JDBC jedoch hinsichtlich der unterschiedlichen Kommunikationsprotokolle, mit TCP/IP auf Seiten von HSQLDB und auf der anderen Seite das in der Cloud zum Einsatz kommende P2P-Netz. Um die geforderte einheitliche P2P-Kommunikation zwischen allen Komponenten zu gewährleisten, musste ein JDBC-Treiber entwickelt werden, der die Datenbankabfragen von TCP/IP auf P2P-Nachrichten wandelt. Die Umstellung des Kommunikationsprotokolls für die JDBC-Anfragen geht jedoch mit einem nicht zu vernachlässigen Leistungsverlust einher, der besonders an dieser Stelle Raum für weitere Arbeiten an der Komponente Datenbank erlaubt.

Weiterhin kann untersucht werden, ob die verwendeten relationalen Datenbanken in der Verwendung hinreichende Effizienz bieten oder ob diese mit Hilfe einer alternativen Umsetzung wie Distributed Hash Tables (DHT) verbessert werden kann. Auch die Erweiterung durch ein Object Relational Mapping (ORM), z.B. unter Verwendung des Hibernate Frameworks, ist als nächster Entwicklungsschritt möglich und wurde bereits während der Entwicklung des vorliegenden Systems diskutiert, jedoch aufgrund des nicht abschätzbaren Zeitaufwandes verworfen.

5.2.5 Benutzerschnittstelle

Die Benutzerschnittstelle wurde durch ein Web 2.0-Frontend benutzerfreundlich realisiert. Mit den bereits beschriebenen Technologien (siehe 3.5) ist es gelungen, eine zustandslose Webanwendung zu implementieren, der vollständig in das bestehende System integriert wurde und der die Forderung nach Web 2.0-Funktionalität erfüllt.

Als direkte Erweiterungen für die Benutzerschnittstelle kann die Authentifizierung mit Hilfe des OpenID-Verfahrens genannt werden. Die Realisierung im bestehenden System war geplant, musste jedoch aus Zeit- und Priorisierungsgründen zurückgestellt werden (siehe hierzu auch 3.5 Authentifizierung über OpenID).

Durch das in der Middleware implementierte Monitoring-System genügt der Webserver den Forderungen nach Ausfallsicherheit und Hochverfügbarkeit. Für ein Produktivsystem ist jedoch die Lastverteilung (Load Balancing) des Webservers durch den Einsatz entsprechender Redundanzsysteme zu realisieren. So kann erreicht werden, dass bei steigender Zahl von Benutzeranfragen an das Web 2.0-Frontend durch bedarfsgerechte Skalierung der Anzahl der zur Verfügung stehenden Web-Server, die Antwortzeit der Benutzerschnittstelle akzeptabel bleibt.

5.2.6 Accounting

Einige zur monetären Verrechnung von Cloud-Diensten benötigte Daten wie Ausführungszeit oder genutzte Bandbreite liegen zwar im System vor, es wurde aber kein Geschäfts- oder Tarifmodell und keine Benutzerschnittstelle für diese Aufgaben entwickelt.

5.2.7 Statistiken

Eine eigene Statistik-Komponente stellt eine sinnvolle Erweiterung der derzeitigen Implementierung dar. Bestimmte statistische Werte könnten so als Reaktion auf relevante Ereignisse effizient und nur bei Bedarf aktualisiert werden.

Ein offensichtlicher Ansatzpunkt für Erweiterungen zur jetzigen Implementierung ist die Erfassung weiterer statistischer Werte. Für das verteilte Dateisystem der Cloud ließe sich z.B. dessen Auslastung sowie die durchschnittliche Dateigröße je Benutzer oder Job anzeigen. Nach einem Wechsel des Scheduling-Verfahrens könnten andere Kennwerte zur Beurteilung interessant werden, z.B. durch Fragmentierung verursachte Wartezeit, die durch den Einsatz von Backfilling reduziert würde.

Die Möglichkeiten der Erweiterung der auszugebenden Statistiken sind mannigfaltig und vielschichtig.