IPP-Report

Max-Planck-Institut für Plasmaphysik

Alexander Krüger

Entwicklung einer Lösung zur Verteilung und Überwachung von Analysemodulen am Wendelstein 7-X

IPP 13/17
Dezember, 2010

fachhochschule stralsund
university of applied sciences

fachbereich elektrotechnik + informatik
school of electrical engineering + computer science

Entwicklung einer Lösung zur Verteilung und Überwachung von Analysemodulen am Wendelstein 7-X

Masterarbeit
zur Erlangung des akademischen Grades
„Master of Science“
Im Studiengang Informatik

Erstgutachter: Prof. Dr. rer. nat. Gero Wedemann
Zweitgutachter: Dr. rer. nat. Georg Kühner

vorgelegt von
Alexander Krüger
Matrikelnummer: 6344
Kranichgrund 3
18437 Stralsund

vorgelegt am
05.05.2009

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1. Max-Planck-Institut für Plasmaphysik

Das Max-Planck-Institut für Plasmaphysik (IPP) mit Hauptsitz in Garching und dem dazugehörigen Teilinstitut Greifswald ist eines der großen Zentren für Fusionsforschung in Europa und beschäftigt sich mit den physikalischen Grundlagen der Kernverschmelzung. Im Teilinstitut Greifswald arbeiten mehr als 350 Mitarbeiter an der Realisierung des Projektes „Wendelstein 7-X“. Dieses Projekt hat das Ziel die Hauptkomponenten eines möglichen zukünftigen Fusionsreaktors zu testen. Im Gegensatz zu allen vorherigen Fusionsexperimenten ist am Wendelstein 7-X geplant, im Dauerbetrieb zu arbeiten. Das Projekt soll voraussichtlich bis 2014 aufgebaut sein und ist dann die weltweit größte Fusionsanlage vom Typ Stellarator (vgl. [Nie07]).

1.2. Situation am IPP

Bei allen Experimentdurchläufen die am Wendelstein 7-X in Zukunft durchgeführt werden, übernehmen Diagnostiken die Aufgabe Daten zu ermitteln. Dabei ist die jeweilige Konfiguration jeder Diagnostik[1] genau definiert und auch die Archivierung der ermittelten Daten ist genau beschrieben und nachvollziehbar. Somit wurde am Institut eine Struktur geschaffen, mit der es möglich ist ein Experiment zu jedem Zeitpunkt nachvollziehen und acquirierte Daten analysieren zu können (siehe Abschnitt 2.1.1).

Des Weiteren wurde am Institut eine Konfigurationsdatenbank entwickelt, in der alle Analyseschritte und Analyseabhängigkeiten für die jeweiligen Experimente spezifiziert sind

[1] Mit Diagnostiken sind im Sprachgebrauch des Instituts im Wesentlichen alle Geräte am Experimentaufbau gemeint, die über Sensorik Daten aufnehmen.

(siehe Abschnitt 2.1.2. Außerdem existiert nun ein einheitlicher Web-Service für den Zugriff auf Experimentdaten.

Mithilfe von sogenannten Analysemodulen (siehe Abschnitt 2.2.4), diese kapseln die eigentliche Analysefunktion, ist es für den Physiker möglich Analysen zu implementieren, die einem definierten Standard folgen.

1.3. Motivation der Arbeit

Am Institut ist man im hohen Maße darauf angewiesen computergestützt Daten auszuwerten. Die Entwicklung von plattformabhängigen, kaum wiederverwendbaren bzw. erweiterbaren, nichtstandardisierten und lokalen, meist unversionierten, Implementierungen birgt jedoch viele Nachteile in sich. Außerdem ist das Institut bestrebt, möglichst standardisierte Strukturen zuschaffen. Beispielsweise werden durch die Steigerung der Wiederverwendbarkeit Kosten, Zeitaufwände und Risiken minimiert.

Deshalb liegt der Schritt für eine weitere Abstraktion der bisherig existenten Strukturen zu computergestützten Datenanalyse nahe. Diese Abstraktion wird durch die am Institut entwickelten Analysemodule2.2.4 erreicht. Auf Basis dieser Analysemodule entstand die Anforderung Analysemodule in ganzen Netzwerken wie Services verwenden zu können. Diese Services sollen hochdynamisch auf beliebigen Computerknoten am Institut bei Bedarf installiert werden können (siehe Abschnitt 3.1). Die Grundlage um Analysemodule in dieser Form verwenden zu können, soll ein System sicherstellen, das im Rahmen dieser Arbeit entwickelt werden soll.

1.4. Ziel der Arbeit

Im Rahmen dieser Arbeit soll ein System entwickelt werden, das die Administration, die Steuerung und die Überwachung von Analysemodulen bzw. Analyse-Services gewährleistet. Dabei stellt das System unter Anderem die Infrastruktur für die Analyse-Services bereit.

Die Schwerpunkte der Arbeit sind somit die Verfeinerung der funktionalen und nichtfunktionalen Anforderungen, der Entwurf einer Architektur die den Anforderungen (siehe Abschnitt 3) genügt, die Implementierung eines Kernsystems sowie das Testen des entwickelten Systems im Rahmen von Modul- und Integrationstests. Gegebenenfalls soll auch überprüft werden, in wieweit das System nicht nur zur Offline-Analye (siehe Abschnitt 2.2.1) sondern auch für Onlineanalysen (siehe Abschnitt 2.2.2) eingesetzt werden könnte.

1.5. Vorgehen

Im Folgenden wird zunächst die Umgebung und die Situation am IPP genauer betrachtet, um anschließend die funktionalen und nichtfunktionalen Anforderungen und weitere Einflußfaktoren zu ermitteln und zu verfeinern. Darauf aufbauend werden vergleichbare Projekte betrachtet, um weitere Informationen über derartige Systeme zu erlangen. Anschließend werden Lösungsstrategien erarbeitet und eine Architektur entworfen. Im Rahmen der Arbeit wird das entworfene System prototypisch implementiert und Testszenarien unterworfen. Abschließend findet eine Diskussion statt, um gesammelte Erfahrungen, den Status und einen Ausblick zu beschreiben.

2. Grundlagen

Bevor im Weiteren die Anforderungen und die Systemanalyse besprochen werden können, müssen zum besseren Verständnis auf der einen Seite die vorhandenen Strukturen zur Datenanalyse am Institut genauer beschrieben werden. Andererseits existieren am Institut viele Fachbegriffe, die im Laufe dieser Arbeit, primär zur Beschreibung der Anforderungen (siehe Kapitel 3), häufig verwendet werden. Auf diese Begriffe wird zur Verdeutlichung in diesem Kapitel ebenfalls eingegangen.

2.1. Vorhandene Strukturen zur Datenanalyse

Im Gegensatz zu allen anderen vorherigen Fusionsforschungsanlagen ist beim Wendelstein 7-X zum ersten mal das kontinuierliche Experimentieren geplant. Daher müssen neue Wege für die Datenerfassung, Steuerung und Analyse gesucht und geschaffen werden. Auf dieser Grundlage entstanden in den vergangenen Jahren am Institut definierte Strukturen, um Experimente und Analysen formal zu beschreiben. Dies wird im folgenden Abschnitt beschrieben.

2.1.1. Aufzeichnung von Experimentdaten

Das folgende Unterkapitel erläutert im Wesentlichen welche Schritte und Prozesse notwendig sind um Daten zu einem Experiment erfassen zu können.

Technische Beschreibung des Experimentes

Als Grundlage für die automatische Datenanalyse müssen zu Beginn eines Experimentes zwei Beschreibungen des Experimentes existieren. Die sind die Experimentkonfiguration

und das Experiment-Programm. Die Experiment-Konfiguration beschreibt die statische Struktur des Experimentes und die erlaubten Einstellparameter-Bereiche. Darauf aufbauend, beschreibt das Experimentprogramm die zeitliche Variation der Einstell-Parameter. Die statische und die dynamische Beschreibung werden vor einem Experiment von den Wissenschaftlern genau definiert. So wird beispielsweise beschrieben wann für die jeweiligen Diagnostiken welche physikalischen Parameter und welche Geräteparameter benötigt werden. Dadurch entsteht ein technisches Szenario zu einem bestimmten Experiment.

Datenerfassung

Bei der Datenerfassung werden nun in Abhängigkeit des technischen Szenarios die Datenströme der für das konkrete Experiment beteiligten Diagnostiken aufgezeichnet. Ein Datenstrom ist dabei eine zeitliche Abfolge von Datenobjekten, wobei die Objekttypen diagnostikspezifisch sind. Parallel zu dem Datenstrom existieren mehrere Parameterströme, die durch das technische Szenario vordefiniert sind. Der Grund dafür, das mehrere Parameterströme existieren, ist, das für die Geräte zur Daten- und Zeiterfassung jeweils eigene Parameterstrukturen benötigt werden. Ein Parameterstrom ist eine gerätspezifische komplexe Klassenstruktur, die sowohl technische als auch physikalische Parameter beinhaltet. Er ist in der Datenbank, in der die Experimentdaten aufgezeichnet werden, logisch mit dem Datenstrom verknüpft.

2.1.2. Analyse von Experimentdaten

Das folgende Unterkapitel beschreibt die vorhandenen Strukturen am Institut um erfasste Experimentdaten computergestützt auswerten zu können. Das Verständnis der im Folgenden beschriebenen Strukturen ist Vorraussetzung, um die eigentliche Vision des Kunden und die Anforderungen zu verstehen.

Das iterative Experimentieren und Analysieren

Es ist anzumerken, dass das im Folgenden beschriebene iterative Experimentieren zum Zeitpunkt der Entstehung dieser Arbeit nur an der WEGA des Teilinstituts Greifswald (sie-

he Abschnitt 2.2.5) durchführbar ist, da der Stellarator Wendelstein 7-X sich im Aufbau befindet.

Geht der Forscher gezielt und programmatisch vor, kann man das Experimentieren am Stellarator mithilfe der Offline-Analyse stark vereinfacht wie folgt beschreiben:

1. Der Forscher entwickelt ein technisches und ein physikalisches Szenario für sein Experiment. Aufgrund dieser Informationen ist nun bekannt wie das Experiment aufgebaut ist und wie genau wann welche Daten erfasst werden sollen.
2. Das Experiment wird durchgeführt und die gemessenen Daten werden abgespeichert.
3. Mithilfe von mathematischen Analysenfunktionen (siehe Abschnitt 2.2.3) erhält der Forscher Auswertungen seiner Experimentdaten und neue Informationen.
4. Sollte der Forscher noch nicht am Ziel seiner Vorstellungen sein, wird er auf Basis der gesammelten Informationen sein technisches und physikalisches Szenario abändern, auch die angewendeten Analysefunktionen können sich ändern, und der hier beschriebene Prozess beginnt von vorne.

Dieser iterative Prozess soll für den Forscher möglichst leicht, nachvollziehbar und einheitlich sein. Im besonderen deshalb, weil der Wendelstein 7-X kontinuierlich betrieben werden soll und dadurch eine große Menge von Routineanalysen benötigt wird, sollen Analyseprozesse unter Anderem möglichst automatisiert werden. Eine umfangreiche Beschreibung der Anforderungen ist im Kapitel 3 zu finden.

Bisher wurden Analysefunktionen formal definiert (siehe Abschnitt 2.2.3), sogenannte Analysemodule entworfen (siehe Abschnitt 2.2.4) und prototypisch implementiert. Mithilfe eines Analysemoduls wird eine Analysefunktion und deren Parametersätze gekapselt. Außerdem werden Abhängigkeiten zu anderen Analysemodulen definiert.

Analyseschritte und ihre Abhängigkeiten

Während eines gesamten Analyseprozesses werden nun gegebenenfalls in mehreren Analyseschritten die Daten analysiert. Dabei beginnen ein oder mehrere Analysemodule Eingangsdatenströme, dies sind im Experiment erfasste Daten, einzulesen und zu verarbeiten. Das Resultat sind Analysedatenströme die abermals versioniert und archiviert

werden. Weitere Analysemodule, die von den vorherigen Analysen abhängig sind, starten nun ebenfalls ihre spezifischen Analysen und nutzen als Eingangsdatenstrom die voranalysierten Datenströme. Diese Analyseketten bilden ein ganzes Analysenetzwerk. Dies ist in den Abbildungen 2.1 und 2.2 veranschaulicht. Letztendlich soll das Ergebnis am rechten Bildrand der Abbildung 2.1 dem Wissenschaftler Aussagen über die Machbarkeit von Fusionskraftwerken geben.

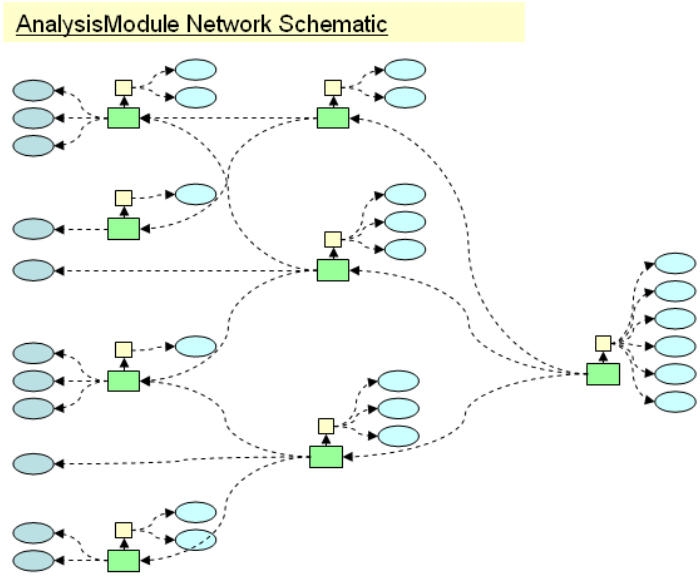

Abbildung 2.1.: Schematische Darstellung eines Datenanalysenetzwerkes

Analysemodulkette und Funktionsgruppenhierarchie

Eine Funktionsgruppenhierarchie beschreibt den jeweiligen Experimentaufbau. Dazu gehört nicht nur der Wendelstein 7-X, sondern auch die Laboreinrichtungen, an denen die Messverfahren vorbereitet und Kalibirier-Messungen für die Datenanalyse gemacht wurden. Diese Kalibrier-Messungen sind Bestandteil des Parameters p_p im grünen Kästchen der Abbildung 2.1 (siehe auch 2.2.3). In diesem Sinne hat jedes Experiment eine Funktionsgruppenhierarchie und ein Analysenetz. Die Ausgabeströme eines Analysenetzes können als Eingabeparameter in anderen Analysenetzen verwendet werden.

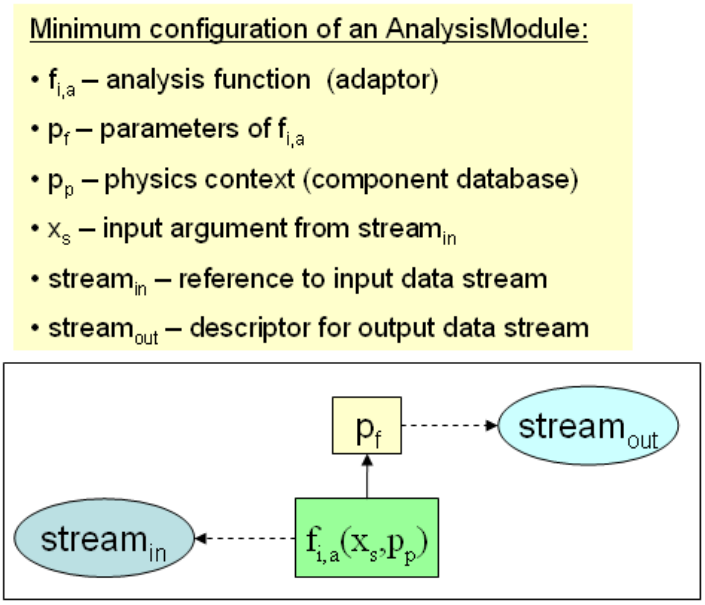

Abbildung 2.2.: Schematische Darstellung eines einzelnen Analyseknotens

2.2. Weitere Begriffe

Der folgende Abschnitt beschreibt weitere Begriffe, die zum besseren Verständnis dieser Arbeit wichtig sind.

2.2.1. Die Offlineanalyse

Bei der Offlineanalyse können parallele Daten- und Parameterströme verarbeitet werden. Es gibt bei der Offlineanalyse im Gegensatz zur Onlineanalyse im weitesten Sinne keine zeitkritischen Anforderungen, sodass die Analysen entsprechend komplex und umfangreich sein können. Der Fokus dieser Arbeit liegt auf der Offlineanalye (siehe Abschnitt 3).

2.2.2. Die Onlineanalyse

Der Begriff Onlineanalyse umfasst Analysevorgänge bei denen Daten zeitkritisch und automatisch analysiert oder für den Physiker visualisiert werden müssen.

So findet die Onlineanalyse beispielsweise Anwendung, wenn über eine synchrone Rückkopplung während des Experimentdurchlaufs aktuell entstandene Daten analysiert werden müssen, um beispielsweise bei Grenzwertüberschreitungen einen Experimentabbruch durchzuführen. Sie kann auch für Mustererkennungsmechanismen eingesetzt werden, die zur Indexbildung genutzt werden soll, um leichter in den großen entstandenen Datenmengen navigieren zu können. Außerdem findet die Onlineanalyse Anwendung bei experimentsynchronen Darstellungen von Kenngrößen der eingesetzten Diagnostikgeräte und der von ihnen gelieferten Daten und anderer weiterer Monitoring-Funktionalität.

Während der Entwicklung von Onlineanalysen werden benötigte Analysefunktionen über Offlineanalysen getestet.

2.2.3. Die Analysefunktion

Die Analysefunktion hat die Aufgabe eine Menge von gemessenen oder voranalysierten Daten zu einer Menge analysierter Daten abzubilden. Dies kann formal so beschrieben werden: $x \mapsto f(x)\ .\ x$ symbolisiert dabei die gemessenen oder voranalysierten Daten. Um die Analysefunktion flexibler anwenden zu können ist sie außerdem parametrisiert. Dadurch erweitert sich die Formel um einen weitere Variable: $x \mapsto f(x, p_f)\ .\ p_f$ beinhaltet beispielsweise Informationen zur Versionierung der Analysefunktion. Zusätzlich besitzt eine Analysefunktion experimentspezifische Abhängigkeiten, die in einem Parameter p_p definiert sind. Dieser Parametersatz ermöglicht die Beschreibung des physikalischen Kontexts zum Zeitpunkt des Experimentes als die Daten erfasst wurden und die Kapselung von Daten zur verwendeten Experimentumgebung. Dies sind typischerweise Hardwarekonfigurationen von Diagnostiken, Geometrische Daten und Weitere. $x \mapsto f(x, p_f, p_p)$ Bevor die Analysefunktion auf die Eingangsdaten x angewendet werden kann, müssen die Funktionsparameter p_f und die physikalischen Parameter p_p vollständig gesetzt sein.

2.2.4. Das Analysemodul

Das Analysemodul hat die Aufgabe eine einheitliche Schnittstelle für benutzerdefinierte Analysefunktionen (siehe Abschnitt 2.2.3) bereitzustellen. Durch die standardisierte Kapselung von spezifischen Analysefunktionen und deren Abhängigkeiten in Analysemodule ist eine Grundstruktur gewährleistet, die es ermöglicht, automatische Offline-Analysen

und Offline-Testläufe von Online-Analysen durchzuführen. Eine wichtige Anforderung an die Analysemodule war die gespeicherten Analysedaten die Abhängigkeiten der Parameter der analysierten Eingabedaten (Messdaten oder voranalysierte Daten) widerspiegeln sollten. Dafür wurde ein sogenanntes „Data Analysis Design Pattern“ am Institut entworfen. Dies ermöglicht die Konfigurationsbeschreibung eines Analysemodules. In der Abbildung 2.3 ist die derzeitige Struktur zur Konfigurationsbeschreibung dargestellt. Die Analysemodule sind durch „AnalysisModuleDescriptor“-Objekte identifizierbar und konfigurierbar. Diese Objekte sind in einer Datenbank abgelegt. Sie besitzen kausale Abhängigkeiten, die durch Referenzen spezifiziert sind. Diese befinden sich in den „ModuleSelector“-Objekten.

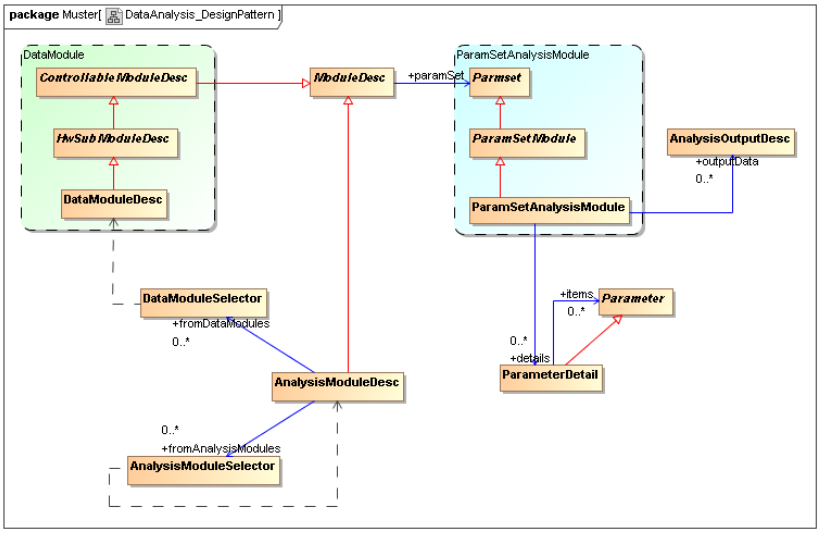

Abbildung 2.3.: UML 2.0 Klassendiagramm: Darstellung der Klassenstruktur zur Konfigurationsbeschreibung von Analysemodulen

2.2.5. Die WEGA

Die WEGA ist ein kleiner konventioneller Stellarator am Greifswalder Teilinstitut. Er wird vor Allem für Ausbildungszwecke verwendet. Außerdem werden dort Testläufe durchgeführt um beispielsweise Steuerungssoftware oder Diagnostiken zu testen. Er bildet dadurch die Testumgebung für Systemtests entwickelter Software für die Datenaufzeichnung, Analyse und Steuerung.

3. Anforderungen

In diesem Abschnitt werden die Wünsche des Kunden aus funktionaler und nichtfunktionaler Sicht betrachtet. Als Kunde werden im Rahmen dieser Arbeit Physiker am Max-Planck-Institut angesehen, die speziell mit Offline-Analysen (siehe Abschnitt 2.2.1) am Institut arbeiten werden. Ein Großteil der Anforderungen ergibt sich aus den in Abschnitt 2 beschriebenen Prozessen, die im Rahmen der Experimentanalyse am Teilinstitut Greifswald stattfinden.

3.1. Vision des Kunden

Da erwartet wird, dass die Zahl der Analysemodule (siehe Abschnitt 2.2.4) sehr groß und zum Teil umfangreiche Rechenleistung benötigen wird, und große Mengen an verteilten Rechenressourcen am Institut vorhanden sind, sollen die Analysemodule auf einem Rechnersystem als Services verteilt werden. Bei Bedarf werden die Analysemodule aktiviert und für die anschließende Analyse zur Verfügung stehen. Das Hauptziel ist die Realisierung eines Systems, das die Verteilung und Administration einer großen Anzahl von Analysemodulen auf eine Menge von Computerknoten unterstützt.

3.2. Funktionale Anforderungen

Im Wesentlichen tritt das zu entwickelnde System als Service-Mediations-System für Analysemodule auf. Die Service-Mediations-Systeme bilden das technische Rückgrat der verteilten Kommunikation zwischen den angeschlossenen Diensten und stellen Funktionalitäten zur Verfügung, die den modularen und lose gekoppelten Aufbau der Integrations-

Gesamtlösung sicherstellen. (vgl. [Mat08] S.13 f.). Darüber hinaus wird weitere Funktionalität gefordert, die hier im Folgenden kurz beschrieben wird:

Funktionen, die das Analysemodul betreffen:

- Die Belegung einzelner Analysemodule auf Computerknoten soll dynamisch konfigurierbar sein
- Die Analysemodule sollen über einen Ereignismechanismus gestartet werden: Dies ist ein Ereignis das interaktiv durch den Anwender oder automatisch nach dem Schreiben von Daten auf die Datenbank zum Beispiel durch andere Anaylsebundle stattfindet.
- Die Anzeige von Daten oder Analyseergebnissen und -zuständen soll möglich sein.
- Die Protokollierung von Analysen soll stattfinden.

Funktionen und Features, die im verteilten System automatisch stattfinden:

- Die Computerknoten, die an das Analysenetzwerk angeschlossen werden sollen, müssen automatisch detektiert und verbunden werden.
- Die Analysemodule müssen nach ihren Abhängigkeiten (siehe Abschnitt 2.1.2), zugeordnet zur Funktionsgruppe (siehe Abschnitt 2.1.2) zu anderen Analysemodulen aufgelöst werden, d.h. für eine Analysekette benötigte Analysemodule müssen ggf. automatisch deployt werden.
- Das System soll unterschiedliche Versionen von Analysemodulen verwalten können.
- Es soll sichtbar sein wo sich welches Analysemodul in welchem Zustand befindet.
- Das Erkennen von Computerknotenausfällen und Durchführung geeigneter Maßnahmen, zum Beispiel Replikation der Analysemodule des Computerknotens auf einen anderen Knoten.
- Ein Systemweites Logging.

Funktionen die zur Administration notwendig sind:

- Die Analysemodule müssen über ein Verzeichnis auswählbar und fernbedienbar auf verfügbare Computerknoten installierbar, startbar und deinstallierbar sein.

- Über eine konsolenbasierte und später über eine grafische Schnittstelle sollen Systemfunktionen aufrufbar und verschiedenste Anzeigemöglichkeiten für System- und Analyseinformationen möglich sein
- Es müssen Systemweite Einstellungen zum Beispiel zur Einrichtung von Pfaden zu Modulrepositories möglich sein
- Ein Automatisches Deployment von Analysemodulen bei:
  - dem Ausfall von Computerknoten
  - dem Aufbau einer Analysekette
  - der Überlastung eines Computerknotens (Load-Balancing)[1]
- Ein Systemweites Update von Kernmodulen bei Systemänderungen oder Systemerweiterungen ohne einen Systemneustart

Eine Liste aller Anwendungsfälle, Features und deren Beschreibung befindet sich im Anhang A, C sowie im Dokument „AWF+Features“ auf der beiliegenden CD.

3.3. Nichtfunktionale Anforderungen

Der folgende Abschnitt beschreibt die essenziellen nichtfunktionalen Anforderungen, die entscheidende Auswirkungen auf die geforderte Qualität und die zu entwerfende Architektur des Systems haben. Die Reihenfolge der beschriebenen nichtfunktionalen Anforderungen entspricht ihren Priorisierungen. Eine Tabelle der identifizierten Risiken und deren Beschreibungen befindet sich im Dokument „Risikoanalyse der nichtf. Anf.„ auf der beiliegenden CD. Die Lösungsstrategien, um die hier genannten nichtfunktionalen Anforderungen zu erfüllen werden im Abschnitt 5.1 erläutert.

[1] Mit Lastverteilung (engl. load balancing) werden Verfahren beschrieben, um bei der Speicherung, dem Transport und der Verarbeitung von Objekten vorgegebene Kapazitätsgrenzen einzuhalten.

3.3.1. Skalierbarkeit

Die wichtigste nichtfunktionale Anforderung für das System ist die Skalierbarkeit. Auf Basis eines verteilten Systems soll es möglich sein eine hohe Zahl von Computerknoten (100 und mehr) an das System anbinden und diese mit Analysemodulen automatisch oder manuell ausstatten zu können. Dabei soll sich das System „gut“ skalierbar verhalten, dass heißt bei zehnfacher Belastung soll auch nur in etwa das zehnfahe an Ressourcen verwendet werden. Eine sich daraus ebenfalls ergebende Anforderung ist die Erweiterbarkeit des Systems um beliebige Analysemodule (siehe auch Abschnitt 3.3.4).

Es soll dadurch möglich sein viele, komplexe und ressourcenraubende Analyseprozesse gleichzeitig durchführen zu können. Der Umfang ist auf weite Sicht bei einem für Jahrzehnte angelegtes Projekt, wie dem Wendelstein 7-X, schwer einzugrenzen und kaum in konkrete Zahl zu manifestieren.

3.3.2. Ausfallsicherheit

Da das System auf vielen Computerknoten laufen wird, muss während des Betriebs sichergestellt sein, das mögliche Ausfälle von Analyseknoten durch das Gesamtsystem aufgefangen werden können. Architektonische Strukturen und Maßnahmen zur Steigerung der Ausfallsicherheit finden sich im Abschnitt 5.1.

3.3.3. Plattformunabhängigkeit

Am Institut existieren eine Vielzahl unterschiedlicher Rechnersysteme und Betriebssysteme. Dies umfasst 32-Bit und 64-Bit Systeme bzw. Betriebssysteme wie beispielsweise Windows XP und Windows Vista sowie die Linux-Distributionen SUSE Linux Enterprise Server oder Redhat. Für das verteilte System sind zunächst alle verfügbaren Rechnersysteme potentielle Computerknoten, die mit verwendet werden könnten. Zur Lösung dieser Anforderung soll das System auf Java basieren. Die Programmiersprache Java bietet für alle gängigen Rechner- und Betriebssysteme eine Virtuelle Maschine[2] an.

[2] hier: eine simulierte Laufzeitumgebung für Programme

3.3.4. Erweiterbarkeit

Da das System zunächst in einem begrenzten Zeitraum im Rahmen dieser Masterarbeit entworfen wird, muss die Architektur gewährleisten, das die spätere Erweiterung des Systems hinsichtlich unterschiedlicher Architekturaspekte möglich ist. Dies umfasst beispielsweise die Erweiterung um Autorisierungs- und Authentifizierungsmechanismen oder die mögliche Erweiterung um eine Persistenzschicht, um Backup- oder Restore-Mechanismen bereitzustellen.

Des Weiteren muss das System aufgrund der funktionalen Anforderung des Hot Deployment von Analysemodulen[3] während der Laufzeit um weitere Analysemodule erweiterbar sein.

3.3.5. Anpassbarkeit

Es entstehen am Institut ständig neue Systeme und Dienste, vorhandene Schnittstellen können sich ändern. Außerdem is das Wendelstein 7-X-Projekt mit einer Laufzeit von mehreren Jahrzehnten geplant. Deshalb muss das System im besonderen Masse das Hinzufügen und Modifizieren von Funktionalität gewährleisten. Idealerweise sollte dies möglich sein ohne Ausfallzeiten des Systems zu erzeugen.

3.3.6. Benutzerfreundlichkeit

Die Hauptanwender des Systems werden Wissenschaftler am Institut sein, die ihre eigenen Analysen durchführen und mithilfe dieses Systems, ihre Analyseprozesse unter Anderem steuern, kontrollieren und überwachen möchten. Um diese Aufgaben möglichst leicht zu erlernen und einfach durchführen zu können, soll zusätzlich zu einer kommandozeilenbasierten Benutzerschnittstelle durch eine grafische Oberfläche die Benutzerfreundlichkeit erhöht werden. Weitere Informationen zum Entwurf der Benutzerschnittstelle befinden sich in Abschnitt 5.4.1.

[3] Beinhaltet installieren, aktualisieren und deinstallieren von Modulen zu Systemlaufzeit

3.3.7. Performanz

Im Raum steht ebenfalls die Frage ob sich ein derartiges System für Online-Analysen verwenden lässt. Das zu entwickelnde System ist von den eigentlichen Analysefunktionen und deren Performanceverhalten komplett entkoppelt. Deshalb ist im Rahmen eines serviceorientierten verteilten Systems vor Allem der Nachrichtendurchsatz eine entscheidende Eigenschaft für die Performanz.

3.4. Fokus

Im Rahmen dieser Arbeit wird der Fokus auf den Entwurf und die prototypische Implementierung eines Systems zur Verteilung und Administration von Analysemodulen auf einer Menge von Computerknoten gesetzt. Dies bildet die benötigte Kernfunktionalität für alle weiteren Anwendungsszenarien. Weitere Informationen finden sich in dem Dokument „Vision und Scope“ und der Tabelle „AWF+Features“ auf der beiliegenden CD.

3.5. Grenzen und Ausschlüsse

Besondere Autorisierungsmechanismen, Verschlüsselungen oder ein mögliches Backup- oder Restore-System sind nicht in den Anforderungen mit aufgenommen. Auf eine Persistenzschicht soll ebenfalls verzichtet werden.

Das System muss jedoch so konzipiert sein, das derartige Erweiterungen möglich sind. Siehe dazu auch Abschnitt 3.3.4.

4. Analyse

Im folgenden Kapitel werden zunächst Einflussfaktoren identifiziert. Anschließend wird auf vergleichbare bereits vorhandene Projekte eingegegangen, die im Rahmen von Recherchen gefunden werden konnten und mögliche Technologien und Spezifikationen betrachtet, um herauszufinden welche davon am besten für dieses System geeignet sind. Die Wahl des möglichst passenden Standards kann gegebenenfalls große Auswirkungen auf Anforderungen und der Umsetzung des Systems und seiner Aspekte (siehe auch Abschnitt 5.4) haben.

4.1. Einflussfaktoren und Randbedingungen

Zusätzlich zu den nichtfunktionalen Anforderungen existieren weitere organisatorische und technishe Einflussfaktoren und Randbedingungen, die im Folgenden kurz genannt werden.

4.1.1. Organisatorische Faktoren

Arbeiten außerhalb des Instituts

Der größte Teil der Arbeit wird nicht am Institut vor Ort geschrieben sondern von zu Hause. Es muss sichergestellt werden, dass über geeignete technische Hilfsmittel ein guter Kontakt zum Institut besteht. Die Recherchen und die damit verbundene Verwendung von Dokumentationen und anderen Quellen aus dem Intranet des IPP's muss zu jedem Zeitpunkt möglich sein.

Maßnahmen

Um Risiken zu minimieren, die durch das nicht vor Ort sein am Institut entstehen könnten, wird ein Virtual-Private-Network eingerichtet, so dass ein Arbeiten innerhalb des Intranets inclusive aller Ressourcen (Dokumentationen und Services) von außerhalb möglich ist. Des Weiteren steht am Institut ein Rechner bereit, der über einen Remote-Zugriff verfügbar ist und das Entwickeln vor Ort ermöglicht.

4.1.2. Technische Faktoren

Der folgende Abschnitt beschreibt die wichtigsten technischen Einflussfaktoren dieses Projekts.

Verwendung von Java

Am Intranet des Instituts existieren verschiedenste Rechnertypen und Betriebssysteme (siehe auch Abschnitt 3.3.3). Die Plattformunabhängigkeit soll durch den Einsatz von Standards und dem Einsatz von der Programmiersprache Java gewährleistet werden.

Vorhandene Strukturen und Schnittstellen

Die am Institut existierende abstrakte Klasse „AnalysisModule“ soll für die Adaption von Analysefunktionen und für die Entwicklung von Analyse-Services mit verwendet werden. Des Weiteren bestehen am Institut Schnittstellen, die mit eingebunden werden und sich ggf. auch während der Systementwicklung ändern können. Dies betrifft im Speziellen Schnittstellen zu Services für die Auslesung von Experimentdaten (siehe auch Abschnitt 3.3.5).

Verwendung von Standards

Es soll untersucht werden in wie weit der Einsatz des OSGi-Standards (siehe auch Abschnitt 6.1.1) und deren OpenSource-Implementierungen möglich ist. Dieser Standard soll gegebenenfalls für eine Implementierung in diesem Projekt verwendet werden.

Keine Ortstransparenz

In den meisten Fällen zeichnet sich ein verteiltes System durch Transparenz aus. Das heißt der Benutzer eines verteilten Systems weiß nicht wo sich eine Komponente eines verteilten Systems physisch befindet. Auf Basis der funktionalen Anforderungen, ist jedoch gefordert, dass der Benutzer genau weiß auf welchem Computerknoten sich Analyse-Dienste befinden. Dies muss im Besonderen bei dem Architekturentwurf berücksichtigt werden (siehe Abschnitt 5.1.4).

4.2. Vergleichbare/Ähnliche Systeme

Auch nach langer Recherche über vorhandene Systeme zur computergestützten verteilten Datenanalyse konnten praktisch keine Projekte gefunden werden, die den serviceorientierten Ansatz der Datenanalyse so verfolgen, wie er hier in Kapitel 3 beschrieben wurde. Deshalb wurde bei der Recherche zunächst der Fokus auf Systeme mit Schwerpunkt verteilter Datenanalyse und anschließend auf Projekte mit Schwerpunkt SOA gelegt. Durch die Analyse vorhandener Systeme sollen weitere Informationen für das eigene System und deren Realisierung gesammelt werden.

4.2.1. Einordnung des eigenen Projektes

Dieses Projekt soll eine mögliche Realisierung einer verteilten serviceorientierten Applikation zur Datenanalyse darstellen. Dabei soll Java verwendet werden und ggf. auf Basis einer OSGi-Plattform ein verteiltes serviceorientiertes System entworfen werden. Deshalb werden in den folgenden Abschnitten Projekte betrachtet, bei denen verteilte Anwendungen, Architekturen für verteilte Analysen, SOA-Plattformen oder bzw. und die Verwendung von OSGi eine wichtige Rolle spielt.

4.2.2. Projekte mit verteilter Datenanalyse

Die Initiative D-Grid und ein Projekt zur verteilten Datenanalyse am LHC in Cern beschreiben Ansätze zur Verteilung von Analysen und Ressourcen, die hier im Rahmen der Sys-

temanalyse betrachtet werden. Ein weiteres Projekt „BlueOx“ von der Princeton University 2003 beschreibt ein Framework zur verteilten Datenanalyse.

Die D-Grid-Initiative

Im Rahmen des vom Bundesministerium für Bildung und Forschung im März 2004 initiierten Projektes D-Grid wurde zunächst das Grundsystem realisiert, um die Infrastruktur für eine verteilte, integrierte Ressourcenplattform für Hochleistungsrechnen, großen Mengen von Daten und Informationen und entsprechenden Dienstleistungen zu schaffen (siehe [WG07] S.10 f). Auf Basis dieses Systems können nun die auf einem GRID basierenden Ressourcen serviceorientiert genutzt werden. Grid-Computing ist eine Form des verteilten Rechnens, bei der ein „virtueller Supercomputer“ aus einem Cluster lose gekoppelter Computer erzeugt wird. Unter Anderem ist auch eine Erweiterung des Systems zur Anbindung Service-orientierter Architekturen geplant.

Es existiert derzeit eine Fülle von D-Grid-Projekten. Hier ist beispielsweise das Projekt HEPCG ([WG07] S.45 ff.) zu nennen. Das Projekt beinhaltet viele Aspekte die auch in diesem Projekt gefordert sind. Dabei liegt der Schwerpunkt bei HEPCG auf der verteilten Datenverwaltung, der Überwachung von Analysejobs und dem Ressourcenverbrauch sowie der verteilten Datenanalyse. Jedoch werden Analysen selbst hier als Jobs betrachtet und treten selbst nicht als Dienst auf.

Aufgrund der serviceorientierten Verwendung der D-Grid-Plattform ist die Struktur des darüber liegenden Systems (wie zum Beispiel dem Projekt HEPCG) offen. So ließen sich D-Grid-Dienste ggf. auch an die Analysedienste, die im Rahmen dieser Arbeit entstehen, anbinden.

Distributed Data Analysis Cern LHC

Am CERN wurde ein eigenes Batch-System, das heißt im ein System zur sequentiellen Kommandoabarbeitung, entwickelt, welches Analyseprozesse verwaltet und ebenfalls ein GRID als Grundlage für intensive Rechnungen und Datenverarbeitungen nutzt [SKP08] und [JD08] S.257 ff. Hier basieren die Analysevorgänge auf Jobs und lediglich die Nutzung von GRID-Ressourcen basiert auf einer Service-Struktur. Über einen „Job-Wizard“ kann der Anwender sich beliebige Analyse-Tasks zusammenstellen, dessen Abarbeitung

dann transparent im System stattfindet. Die Ermittlung verfügbarer Ressourcen für den jeweiligen Task findet über einen Agenten statt, der mit einem GRID-Ressoure-Broker in Verbindung steht.

BlueOx

Das Projekt BlueOx [JM03] entstand an der Princeton University im Jahr 2003 und beinhaltete die Entwicklung eines Java-Frameworks für die verteilte Datenanalyse.

Der interne Prozessablauf kann folgender Maßen beschrieben werden:

- Automatische Diensterkennung (disovery)
- Dienstvermittlung (brokering)
- Analyseaufgaben durchführen (analysis job execution)

Das einzige White-Paper zu diesem Projekt stammt von 2003. Zu diesem Zeitpunkt befand sich das Framework im Teststadium. Auch hier konnten keine weiteren Informationen hinsichtlich Architekturdokumentation oder Quellcode gefunden werden.

4.2.3. Projekte mit einem SOA-basierten Ansatz

Während der Recherche wurden außerdem Projekte betrachtet, in denen der Schwerpunkt im serviceorientierten Umgang mit Daten liegt.

CoreGrid

Die Grundlage ist ein Grid das Rechen- und Speicher-Ressourcen bereitstellt. Diese sind über Services verfügbar, die in die Gruppe der „Core Services“ (Security, Data Management und Information Services) einzuordnen sind. Auf diesen aufbauend existieren die „Knowledge Grid Services“ (Ressource- und Execution Management). Auf einer weiteren Schicht befinden sich die Analyseservices. Dies sind „Ad-Hoc Services“[1], die die „Knowledge Grid Services“ für High-Level Datenanalysefunktionalitäten verwenden. Auf einer

[1] Ad-Hoc Services sind Services die bei Bedarf instanziiert werden.

letzten Schicht befinden sich dann die eigentlichen Applikationen, die die Möglichkeit haben auf alle Services zuzugreifen (siehe Abbildung 4.1).

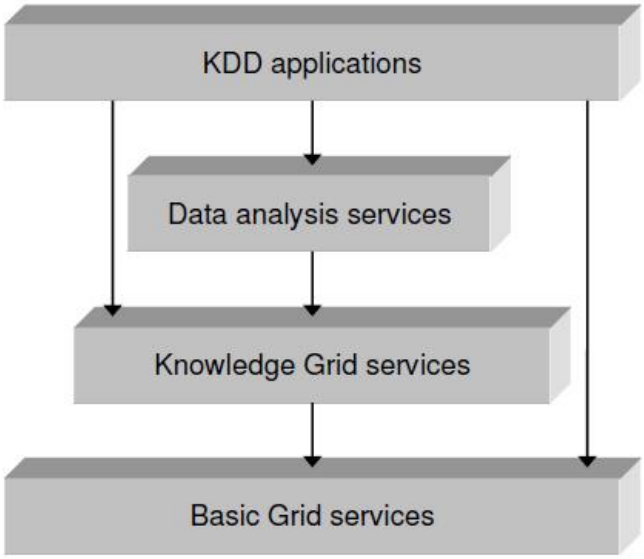

Abbildung 4.1.: Schematische Darstellung der Servicestruktur des CoreGrid

Das Projekt CoreGrid[JM07] baut auf WSRF Patterns auf und verwendet das Open Source Framework „Globus Toolkit“. Es ist selbst jedoch kein Open Source Framework.

Projekt flowSGI

Das Projekt „flowsgi“[2] bildet einen Ansatz für eine Kollaborations-Middleware für mobile Geräte und wird an der ETH Zürich entwickelt. Ein ganz klassischer Anwendungsfall für ein solches System könnte zum Beispiel das kollaborative Arbeiten an einem Dokument von mehreren physisch voneinander getrennten Personen sein. Die Grundlage für das System bildet der OSGi-Standard. Dies ist ein Industriestandard für IOC-Komponenten-Container (siehe. [Mat08] S.134 ff.). Bei der Betrachtung des flowsgi-Projektes lag der Schwerpunkt im Wesentlichen auf den verwendeten Technologien. OSGi ist hier mit dem R-OSGi Framework erweitert worden. Durch dieses Framework können die Services nicht nur auf einer OSGi-Instanz existieren, sondern über physische Grenzen hinweg verteilt, so dass die Entwicklung von echten SOA-Applikationen erst möglich wird.

[2] http://flowsgi.inf.ethz.ch/

4.3. Service-Mediations-Systeme

Unter einem Service-Mediations-System wird ein System verstanden das mehrere Servicequellen durch virtuelle Integration zusammenfasst. Die wichtigste Komponente des Systems ist ein Mediator (das entsprechende Entwurfsmuster heißt Vermittler), der Anfragen an das Gesamtsystem entgegennimmt, beantwortet und mit den eigentlichen Services kommuniziert. Ein solches System kann durch Web Service Engines, ESBs oder SOA-Plattformlösungen/Applikationen realisiert werden.

4.3.1. Formen

Im Folgenden wird kurz untersucht, welche Form eines auf SOA basierenden Mediationssystems anhand der nichtfunktionalen Anforderungen und dem Einsatzbereich die theoretisch geeignetste Struktur darstellt. Aufgrund der physikalisch verteilten Ressourcen, des zu entwickelnden Systems, spielen SOA-Plattformlösungen normalerweise eine untergeordnete Rolle. Es ist jedoch zu beachten, das SOA-Plattformlösungen ggf. erweitert werden können und so zu einer echten SOA-Applikationslösung wachsen. Ein konkretes Beispiel dafür beschreibt Abschnitt 4.2.3. Dort wird die OSGi-Plattform durch R-OSGi erweitert, sodass die Realisierung von SOA-Applikationslösungen ermöglicht wird. Deshalb werden im Folgenden derartig erweiterbare SOA-Plattformen unter dem Begriff SOA-Applikationslösungen mitbetrachtet.

4.3.2. Eigenschaften

Die wichtigste nichtfunktionale Anforderung, die Skalierbarkeit aber auch die Erweiterbarkeit kann sowohl durch Applikationslösungen auf SOA-Basis als auch durch ESB-Systeme gewährleistet werden. Jedoch bieten ESBs laut [Mat08] S. 104f. eine hohe QoS und damit einen höheren Grad der Ausfallsicherheit. Grundsätzlich sollte jede Form eines Service-Mediations-Systems über ein SOA-Managementsystem verfügen, um ein hohes Maß der Administrierbarkeit gewährleisten zu können. Weiterhin bieten sowohl Service-Applikationen als auch ESBs sehr gute Realisierungsformen für das Load Balancing[3] und

[3] Verfahren, um bei der Speicherung, dem Transport und der Verarbeitung von Objekten vorgegebene Kapazitätsgrenzen einzuhalten

dem Service-Deployment.

4.3.3. SOA-Systeme und Java

Grundsätzlich bietet Java viele ausgereifte Möglichkeiten zur Umsetzung eines SOA-Systems. Die Abbildung 4.2 zeigt einen Ausschnitt von auf Java-basierenden Technologien. Es existieren verschiedene Frameworks für die verschiednen Technologien. An dieser Stelle werden nur einige kurz genannt.

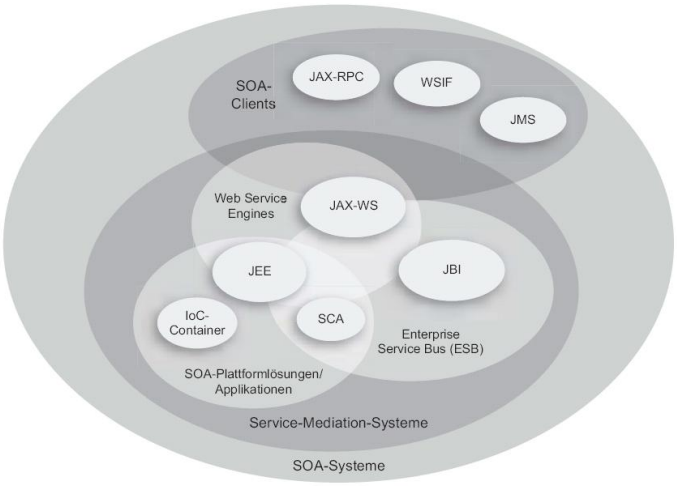

Abbildung 4.2.: Auswahl von SOA-Technologien auf der Java-Plattform [Mat08] S.133

Dies sind das Projekt „GlassFish“, das eine Referenzimplementierung des JEE 5 Standards bereitstellt. Ein weiteres Paket ist das „Java Web Services Developer Pack“, welches die Entwicklung von Web-Services benötigten XML- und Web-Service-Technologien umfasst. Ein Open Source Projekt, das einen ESB implementiert ist „OpenESB“ basierend auf der JBI-Technologie. Eine weitere Möglichkeit zur Umsetzung eines SOA-Systems sind die IoC-Komponentencontainer. Hier existieren unter Anderem Projekte wie „Pico-Container“, „HiveMind“ und „JBoss MicroContainer“.

Zusätzlich zu den reinen IoC-Komponentencontainern existieren Frameworks, die den OSGi-Industriestandard implmentieren. Da OSGi nur in einer virtuellen Maschine arbeitet, benötigen solche Frameworks entsprechende Erweiterungen, um eine vollwertige SOA-Applikationslösung darzustellen (siehe [Mat08] S.135 und S.141).

Die bekanntesten OSGi-Frameworks sind „Equinox“, „Knopflerfish’“ und „Apache Felix“. Jedes Framework besitzt zusätzlich zu den OSGi-Standards spezifische Erweiterungen, wobei „Equinox“ das etablierteste Framework darstellt. Open Source Frameworks zur Remoteerweiterung der genannten OSGi-Implementierungen sind „R-OSGi“, ein Open Source Projekt der ETH Zürich, dass zur Nachrichtenübertragung den SLP-Standad nutzt. Einen weiteren Ansatz bietet das „Newton“-Framework basierend auf OSGi und Jini. Es kapselt OSGi intern und verwendet zur Kommunikation Jini. Jini ist ein Framework zur Service-Nachrichtenübertragung und kann RMI, CORBA oder SOAP verwenden.

4.4. Zusammenfassung

Nach der Betrachtung einiger Technologien, Projekte und Frameworks, die im Zusammenhang mit verteilten Analyseprozessen stehen, wurden zusätzliche Informationen gesammelt, die zur Entscheidungsfindung bei der Technologie- und Architekturauswahl des zu entwickelnden Systems beitrugen und den Architekturentwurf in Kapitel 5 mit beeinflussen.

Grundsätzlich lässt sich feststellen, das fast alle Projekte eine sehr spezifische Lösung anbieten. Das liegt grundsätzlich daran, das zwar die Ziele und Anforderungen mit diesem Projekt vergleichbar sind, jedoch die Voraussetzungen und Randbedingungen sich stark unterscheiden. Deshalb, weil am Institut bereits konkrete Strukturen und Vorstellungen existieren, müssen, wenn auch unter Berücksichtigung von vorhandenen Projekten eigene Lösungen ermittelt werden. Die wenigen Projekte deren Schwerpunkt bei der verteilten Datenanalyse (siehe Abschnitt 4.2.2) lagen, waren nur sehr geringfügig dokumentiert und boten lediglich informativen Charakter. Es wurde dort kaum von Standards oder OpenSource-Projekten Gebrauch gemacht.

Die beiden Projekte unter Abschnitt 4.2.3 bieten jedoch vielseitige Informationen, die auf die Systemspezifikation und die Entscheidungsfindung Auswirkungen haben. Das Projekt „CoreGrid“ basiert auf einer Service-Struktur bei denen auf der einen Seite Core-Services eine Art Middleware bereitstellen und darauf aufbauend AdHoc-Analyse-Services, die die verteilte Analyse mithilfe der Core-Services bereitstellen. Dies entspricht im Wesentlichen dem Grundgedanken des Kunden und den entstandenen Anforderungen. Der Unterschied ist lediglich das im Projekt „CoreGrid“ die Hardware- und Ressourcen-Basis eine Grid-Struktur bildet. Die Hardwaregrundlage des Projektes dieser Arbeit sind jedoch

Computerressourcen, die im Intranet des Instituts bereitstehen. Dies sind beispielsweise Work-Stations mit großen Leerlaufzeiten. Deshalb fehlt eine weitere Middlewareebene zur dynamischen Anbindung dieser Computerressourcen.

Dafür bietet das Projekt „flowsgi“ (siehe Abschnitt 4.2.3) einen Ansatz. Das Projekt selbst soll eine Kollaborations-Middleware bereitstellen. Die dafür verwendeten OpenSource Frameworks (Eclipse Equinox und R-OSGi) sowie der OSGi-Standard bieten jedoch die Möglichkeit eine Middlewareebene zu realisieren, auf dessen Basis eine verteilte Analyse-Servicelandschaft im Intranet des Instituts bereitgestellt werden kann. Durch „Experimentelles Prototyping“ wurde die grundlegende Realisierbarkeit mit einem Prototypen mit minimaler Funktionalität (Remote-Deployment von AnalyseService-Mocks) überprüft.

Der Entwurf und die Architektur werden im Folgenden Kapitel 5 im Hinblick auf eine spätere Implementierung auf Basis von IoC-Komponenten-Container, dem OSGi-Industriestandard (siehe [Mat08] S.134ff.) und einer entsprechenden Remote-Erweiterung ([Mat08] S.141 Abschnitt 6.6.1) beschrieben.

5. Entwurf

Das folgende Kapitel zeigt in einem ersten Abschnitt entwurfsspezifische Maßnahmen auf, die zur Erfüllung der gegebenen nichtfunktionalen Anforderungen beitragen sollen. Es werden Entwurfsentscheidungen und der Einsatz verschiedener Entwurfsmuster beschriebenen sowie deren Auswirkung auf das System. In einem weiteren Abschnitt werden einige wichtige Architektursichten beschrieben, um das Gesamtsystem aus verschiedenen Blickwinkeln betrachten zu können. Abschließend werden die wichtigsten Architekturaspekte betrachtet.

5.1. Lösungskonzepte

Der folgende Abschnitt beschreibt die wesentlichen Maßnahmen, um die in Abschnitt 3.3 und Abschnitt 4.1 genannten Anforderungen und Bedingungen zu erfüllen. Zum Einsatz kommen verschiedene architekturelle Maßnahmen und die Verwendung bewährter Entwurfsmuster.

5.1.1. Die Skalierbarkeit

Ausschlaggebende Faktoren

Grundsätzlich ist ein verteiltes System „gut“ skalierbar, wenn es bei wachsender beispielsweise 20-facher Leistung (Nennlast) etwa mit dem 20-fachen an Ressourcen auskommt. Bei einem verteilten System ist grundsätzlich die effiziente Kommunikation ein ausschlaggebender Faktor für Performanz und für die Skalierbarkeit.

Der Entwurf des Systems findet völlig entkoppelt von der Entwicklung der Analysemodule statt. Auch während der Laufzeit hat das Mediations-System keinen direkten Einfluß auf

die Analyseprozesse. Jedoch ist während der Laufzeit des Systems zu beachten, dass die Kommunikation des verteilten Systems und der Datenaustausch der Analysemodule auf den selben Ressourcen (Ethernet) stattfindet. Dies ist ein weiterer Faktor, der eine effiziente Kommunikation des verteilten Systems in den Vordergrund stellt.

Maßnahmen

Aus architektureller Sicht wird das System so konzipiert, das entfernte Kommunikationen möglichst eingeschränkt werden. Dies bedeutet das jeder Analyseknoten und der Mediator lokale Logs und Protokollierungen durchführen wird (siehe Abschnitt 5.4.2 und [Sta08] S.240). Im Gegensatz zur zentralen Protokollierung vermindert dies die Netzlast enorm. Es erschwert jedoch auch die Konsolidierung der einzelnen Logs. Des Weiteren werden Module die jeder Analyseknoten benötigt („DeploymentService“, „AnalysisNodeInfoService“ und weitere) nicht zentral im Mediator gehalten und über einen Remotezugriff verwendet, sondern jeder Analyseknoten bekommt einen lokales Bundle der benötigen Kernmodule. Im Hinblick auf Systemweite Updates, muss jeder Analyseknoten upgedated werden. Ein weiterer Vorteil beider Maßnahmen ist die deutliche Entlastung des Mediators.

Eine weitere Maßnahme ist die Auswahl effizienter Kommunikationsprotokolle, die Daten im Binärdatenformat übertragen und wenig Overhead besitzen (siehe dazu 6.1.2 und Abschnitt 6.1.2).

5.1.2. Erhöhung der Ausfallsicherheit

Erkennen von Ausfällen

Um Ausfälle von Analyseknoten sichtbar zu machen, wird mit dem „HeartBeat“-Verfahren ([Sta08] S. 68, [LB03] S.104) periodisch festgestellt ob ein Analyseknoten noch online ist oder nicht. Bei einem erkannten Ausfall trifft das System automatisch Vorkehrungen.

Maßnahmen bei Ausfällen

Bei einem festgestellten Ausfall eines Analyseknotens wird versucht auf einem anderen Analyseknoten den ausgefallenen zu replizieren. Die ist eine Art „warm restart“-Taktik, um das System wieder in einen geregelten Zustand zu bringen. Sollte dies nicht gelingen, muss das System mit Hilfe von Rollback-Mechanismen in den letzten gültigen Zustand zurückkehren. (5.1.2). Durch sogenannte „Checkpoints"([LB03] S.103f.) werden diese letzten gültigen Zustände festgehalten.

Der Ausfall des zentralen Mediators hat zur Folge das alle Analyseknoten in einen initialen Zustand zurückfallen. Dabei beginnen die Analyseknoten automatisch auf ein neues Mediatorsignal zu warten und sich gegebenenfalls mit diesem Mediator zu verbinden. Der Mediator verbindet sich dann mit den Analyseknoten und nimmt seinen aktuellen Zustand auf. Weitere Informationen und Überlegungen im Hinblick auf einen möglichen Ausfall des Mediators befinden sich im Abschnitt 8.3.

Nebenläufigkeit

Innerhalb des verteilten Systems finden viele Prozesse gleichzeitig statt. Beispielsweise installiert ein Anwender gerade ein Analysemodul auf einem Analyseknoten, während das System festgestellt hat, das der Analyseknoten nicht mehr mit dem System verbunden ist oder zwei Anwender möchten gleichzeitig über die Web-Schnittstelle den selben Analyseservice mit einem Analysemodul belegen. Deshalb werden Transaktionen vom System verwaltet und bei möglichen Fehlerzuständen die gesamte Transaktion abgebrochen und in den vorherigen Systemzustand zurückgekehrt (Rollback). Dabei wird von der „optimistischen Steuerung der Nebenläufigkeit“ Gebrauch gemacht (vgl. [Fow03] S.84 ff., S.459 ff.).

Mit Hilfe des „Optimistic Offline Lock“ und dem „Unit Of Work“-Pattern ([Fow03] S.211 ff.) wird vor der jeweiligen Transaktion und dem damit verbundenen Funktionsaufruf entschieden, ob die Transaktion durchgeführt werden kann. Dies bedeutet in einem konkreten Beispiel das ein „Anwender A“ der gleichzeitig mit „Anwender B“ auf einen konfigurierbaren „Analyseservice 1“ zugreift und die Konfiguration des Analyseservices zuerst absendet auch den Service konfiguriert hat. Der „Anwender B“, der seine Konfiguration etwas später abgesendet hat, wird eine Nachricht erhalten, das der Analyseservice bereits konfiguriert

wurde und für ihn „gelockt“ bzw. zur Konfiguration gesperrt ist. Die Unterscheidung einzelner Anwender findet serverseitig mit Hilfe des „Server-Session-State“-Pattern ([Fow03] S.503 ff.) statt.

5.1.3. Erhöhung der Erweiterbarkeit und Modifizierbarkeit

Die Analyse-Services stellen Adaptoren dar, um beliebige Analysemodule an das verteilte System anzubinden. So kann zur Laufzeit ein entwickeltes Analysemodul dynamisch in das System mit eingefügt werden.

Das System besitzt Extensionpoints unter Anderem am „WebCommandsService“, der um beliebige Anwendungsfälle zur Laufzeit des Services erweitert werden kann. Der R-OSGi-Service besitzt Extensionpoints beispielsweise zur Erweiterung des Kommunikationskanals in einen verschlüsselten oder überwachbaren Kanal ([JSR07b] S.10 f.).

Durch ein hohes Maß an Modularisierung und Entkopplung können Module bequem ausgetauscht werden und das System so neuen Anforderungen individuell angepasst werden. Bei dem Entwurf wurde darauf geachtet, das einzelne Services möglichst kohärent und elementar sind. Eine Ausnahme bildet der zentrale „CommandsService“. Er fasst die Funktionalität vieler kleiner Services in einer Schnittstelle zusammen, sodass die lose Kopplung gefördert wird und die Komplexität des Gesamtsystems gesenkt wird, da für einen Kommandoaufruf genau ein Service zuständig ist.

Die Services zur Steuerung des Systems („CommandsService“) sowie zur Analyse („AnalysisService“) besitzen ausgelagerte Schnittstellen („Separated Interface“ [Fow03] S. 521 ff.). Dies ermöglicht eine entkoppelte Implementierung der Schnittstelle und bietet mehr Flexibilität beim Testen (Generierung von Service-Stubs siehe [Fow03] S.549 ff. und Abschnitt 7.2) und erhöht die Wartbarkeit des Systems. Außerdem kann der aufrufende Client ohne Kenntnisse der eigentlichen Serviceimplementierung auskommen.

5.1.4. Keine Ortstransparenz

Der Entwurf beinhaltet weiterhin einen „AnalysisNodeInfoService“. Dieser ermöglicht es dem Mediator umfangreiche Informationen zu einem bestimmten Analyseknoten zukommen zu lassen. In den meisten Fällen eines verteilten Systems ist gefordert, dass eine

Ortstransparenz existiert. Dass heißt, der Anwender weiß nicht wo sich ein bestimmtes Analysemodul befindet. In diesem System ist dies jedoch explizit gefordert. Der Anwender soll Informationen erhalten können, die ihm genau vermitteln wo und in welchem Zustand sich Analyseprozesse im System abspielen.

5.2. Weitere Entwurfs-Patterns

Der folgende Abschnitt beschreibt einige weitere wichtige Entwurfs-Patterns, die in die Architektur mit eingefloßen sind.

5.2.1. Das State-Pattern

Sowohl Analyseservices als auch Analyseknoten und das Gesamtsystem können zu jedem Zeitpunkt einem Zustand zugeordnet werden. Mit Hilfe des State-Pattern ([Fow03] Seite 144 f.) besitzt das entsprechende Modul nur die Funktionalität, die es in Abhängigkeit seines Zustandes haben darf.

5.2.2. Das Adapter-Pattern

Der Analyse-Service ist ein Adapter, der das abstrakte Analysemodul, dessen konkrete Implementierung jeweils unbekannt ist, zum einen an das System anbindet. Zum Anderen beinhaltet er Konfigurationsoptionen, die das Analysemodul zum fehlerfreien Ablauf benötigt.

5.2.3. Das Whiteboard-Pattern

Innerhalb des verteilten Systems können Services zu jedem Zeitpunkt nicht mehr verfügbar sein. Aufgrund dieser hohen Dynamik reicht das „Listener“-Pattern nicht aus. Fällt der zu observierende Service aus, erhält der Listener (Zuhörer oder Beobachter) darüber keine Nachricht.

Aus diesem Grund wird innerhalb des Systems das „Whiteboard“-Pattern angewandt. Hierbei existiert ein „Event“-Service im System. An diesem meldet sich das observierbare Bundle an und ebenfalls das oder die observierenden Bundle (siehe auch [GW08] S.130). Die observierbaren Ereignisse werden dann an den „Event“-Service geschickt, der diese dann an die Observer weiterreicht. Dies erhöht zusätzlich die Entkopplung, da ein observierbares Objekt seine Beobachter nicht kennt.

5.3. Die Architektursichten

Um die Vielschichtigkeit und Komplexität des Systems darzustellen, wurden einige Architektursichten an dieser Stelle dargestellt. Dadurch sollen die unterschiedlichen Aspekte des verteilten Systems sichtbar gemacht werden. Für die Darstellung kam die UML2.0-Notation zum Einsatz (siehe auch [TP04] S.144 ff.).

5.3.1. Die Bausteinsicht

Zur Beschreibung der statischen Aspekte des Systems, dies beinhaltet die Subsysteme sowie Komponenten und Schnittstellen des Systems, wird im Folgenden die Bausteinsicht des Systems kurz betrachtet. Die UML2.0-Abbildung C.1 zeigt die Kernkomponenten und ihre Verwendung untereinander. Es existieren weitere Servies, die vom OSGi-Standard bereitgestellt werden, wie zum Beispiel dem Log-Service der von allen Komponenten mit verwendet wird. Die Zentrale Komponente ist der „CommandsService“. Dieser Services zentralisiert alle Kommandos, die das System entgegennehmen kann. Bei dem Entwurf wurde vor Allem darauf geachtet, das die einzelnen Komponenten möglichst voneinander entkoppelt sind und eine hohe Kohärenz erreichen. So besitzt jeder Service seinen eigenständigen Aufgabenbereich. Der „ComamndsService“ sowie der „AnalysisService“ realisieren ein „Separated Interface“ (vgl. [Fow03] S. 521 ff.), sodass aufrufende Anwendungen lediglich die Schnittstelle und nicht die Serviceimplementierung kennen müssen.

5.3.2. Die Verteilungssicht

Der folgende Abschnitt betrachtet das System hinsichtlich seiner physischen Verteilung. In Abbildung 5.1 ist zu erkennen, dass ein zentraler Verteiler als Mediator fungiert. Dies ist der Applikationsserver. Er besitzt die Verbindung zu allen Analyseknoten. Er vermittelt und verteilt Aufgabe unter den angeschlossen Analyseknoten. Die Analyseknoten dienen selbst als Ressourcen für Analysedienste. Ein weiterer Knoten repräsentiert die Client-Workstations, die mit dem Applikationsserver verbunden sind. Über diese können Monitoring- und Administrationsaufgaben vom Anwender durchgeführt werden.

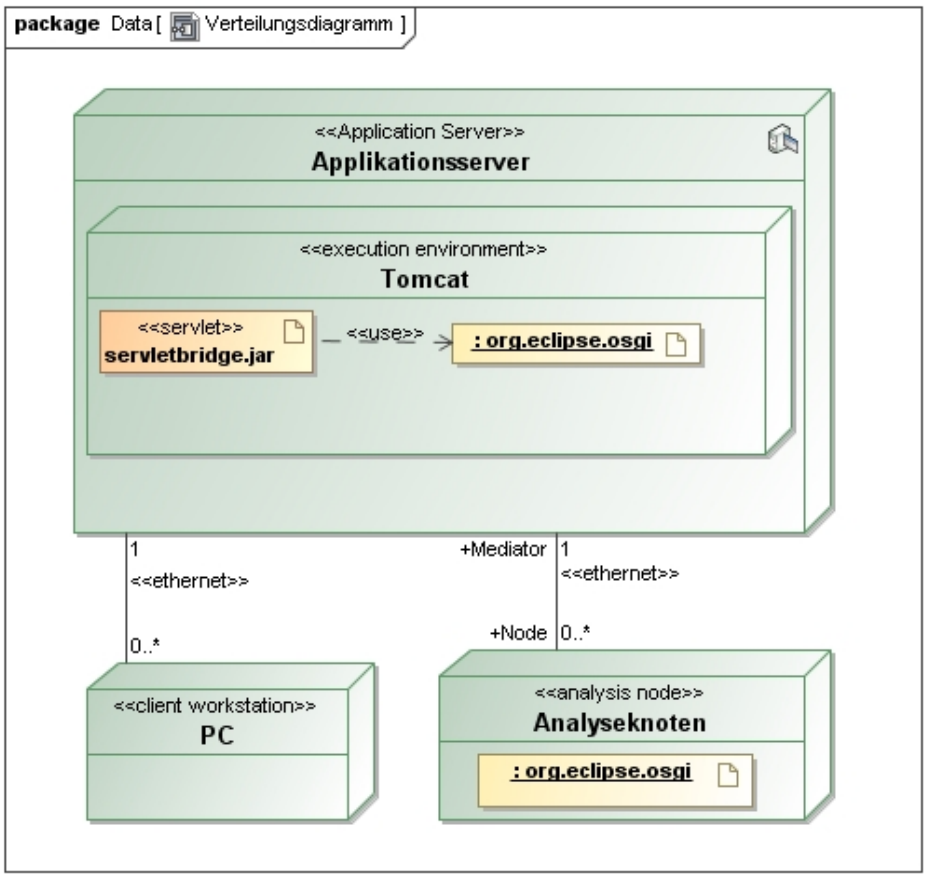

Abbildung 5.1.: UML 2.0 Verteilungsdiagramm: Die Abbildung beschreibt die Verteilungssicht auf das System

Sowohl der Mediator als auch die Analyseknoten verfügen jeweils über eine OSGi-Instanz, wobei die OSGi-Instanz des Mediators innerhalb der Umgebung eines Webapplikations-

servers (hier Tomcat) existiert und über eine sogenannte Servletbridge verbunden ist (siehe auch [GW08] S.433ff.).

5.4. Die wichtigsten Architekturaspekte

5.4.1. Die Benutzerschnittstelle

Bei dem Entwurf des Systems ist zum einen die konsolenbasierte Kommunikation direkt am Mediator vorgesehen. Außerdem existiert eine grafische webbasierte Schnittstelle, die es dem Anwender ermöglichen soll über beliebige Rechner des Instituts mit Hilfe eines Web-Browsers auf den Mediator zugreifen zu können. Der Fokus der Implementierung liegt jedoch zunächst auf der konsolenbasierten Benutzerschnittstelle.

5.4.2. Die Protokollierung

Eine wesentliche Anforderung an das System ist die Protokollierung aller Prozesse während der Systemlaufzeit. Dafür existiert ein Log-Service lokal auf jedem Knoten und auf dem Mediator. Im Gegensatz zu einem zentralen Log-Service wird dadurch die Netzlast dramatisch reduziert und der Mediator deutlich entlastet. Jedoch muss bei der Konsolidierung von Logs zusätzlicher Aufwand geleistet werden. Das Logging wird jedoch nur im Fehlerfall ausgewertet. Die Reduzierung der Netzlast und des Ressourcenverbrauchs und die damit verbundene Steigerung der Performance des Gesamtsystems ist diesem Nachteil jedoch vorzuziehen.

5.4.3. Die Verteilung und die Kommunikation

In einem verteilten System müssen zusätzliche Maßnahmen zur Fehlererkennung, Fehlerbehandlung und Fehlervorbeugung getroffen werden. Zusätzlich sollte zwischen fachlichen Fehlern und technischen (Kommunikations)-fehlern unterschieden werden. Wie bereits in Abschnitt 5.1 und Abschnitt 5.1.2 beschrieben, wurden verschiedene architektonische Maßnahmen zur Fehlererkennung, -vorbeugung und -behandlung getroffen. Die Basis dafür sind ermittelte Test- und Fehlerszenarien mit denen das System umgehen

können muss. Innerhalb des verteilten Systems wird zusätzlich zwischen fachlichen und technischen Fehlern unterschieden. Ein technischer Fehler ist beispielsweise der erkannte Ausfall eines Analyseknotens. Ein fachlicher Fehler kann durch Nebenläufigkeitseffekte, wie das gleichzeitige Konfigurieren eines Analyseservices durch zwei Anwender, auftreten.

Weiterhin sollten entfernte Methoden- und Funktionsaufrufe möglichst vermieden werden. Dies betrifft im Besonderen das lokalisierte statt zentrale Logging (siehe Abschnitt 5.4.2). Zusätzlich sollten die Beschreibungen von Schnittstellen ausgelagert werden (siehe auch Abschnitt 5.1.3).

6. Implementierung

Im folgenden Kapitel werden kurz die wichtigsten verwendeten Frameworks und die wichtigsten Aspekte der Entwicklungsumgebung beschrieben.

6.1. Verwendete Frameworks

Der folgende Abschnitt stellt kurz die zwei wichtigsten Frameworks vor, die im Rahmen dieser Arbeit verwendet wurden.

6.1.1. Eclipse Equinox

Eclipse Equinox ist ein von der Eclipse Foundation entwickeltes Java-basiertes Framework, welches zur Zeit die OSGi-R4-Kernspezifikation implementiert, und bildet das Gerüst der integrierten Entwicklungsumgebung Eclipse seit Version 3. Elipse Equinox erweitert den OSGi-Industriestandard um weitere Funktionalität wie beispielsweise der Extension Registry. In der OSGi-Spezifikation sind verschiedenste Administrationsdienste bereits vorgeschrieben, die das Konfigurieren, die Ereignisverwaltung, Benutzerverwaltung und viele weitere Funktionalitäten enthält. Des Weiteren kann das System während der Laufzeit um weitere Funktionalität erweitert werden, durch das sogenannte Hot Deployment. Es ermöglicht das Installieren, aktualisieren und deinstallieren von Bundles und Services zur Laufzeit (siehe [GW08] S.18). Aufgrund des Architektur, die eine Steuerung der Anwendung über ein Web-Interface vorsieht, wird speziell auf der Seite des Applikationsservers, der den Mediator beinhaltet eine zusätzliche Technologie benötigt. Dies wird im Folgenden verdeutlicht:

Equinox auf einem Analyseknoten

Jeder Analyseknoten besitzt eine OSGi-Instanz. Sie ist zum Einen durch R-OSGi erweitert (siehe Abschnitt 6.1.2). Auf der Anderen Seite existieren einige Kerndienste, um das automatische Verbinden von Knoten, oder das fernbediente Verwalten von Diensten zu ermöglichen. Die Instanz läuft in einer eigenen virtuellen Maschine. Die „Java Virtual Machine“ ist der Teil der Java-Laufzeitumgebung für Java-Programme, der für die Ausführung des Java-Bytecodes verantwortlich ist. Hierbei wird im Normalfall jedes gestartete Java-Programm in seiner eigenen virtuellen Maschine ausgeführt.

Equinox auf dem Mediator

Im Gegensatz zu den Analyseknoten muss beim Applikationsserver die OSGi-Instanz über eine Webschnittstelle administrierbar sein. Es gibt zwei verschiedene Möglichkeiten (siehe auch [GW08] S.433ff.) dies zu realisieren:

- Eine OSGi-Instanz, mit einem eingebetteten Web-Container, zum Beispiel Jetty[1], der als gewöhnliches OSGi-Bundle existiert,
- Die Einbettung einer OSGi-Instanz in einen Web-Container.

Da sich am Institut bereits Web-Applikationsserver, z.B. Tomcat, durchgesetzt haben, fiel die Wahl hier auf die zweite Variante. Der Aufwand ist in beiden Varianten gleich groß. Für die Einbettung einer OSGi-Instanz in einen Web-Container wurde die sogenannte „ServletBridge“ (siehe [GW08] S.433ff. und [ML06] S.27f.) verwendet. Sie ermöglicht die Einbettung einer OSGi-Instanz in eine Webapplikation innerhalb eines Web-Containers. Außerdem stellt sie einen Zugang dar, um die OSGi-Instanz über das Web zu administrieren.

6.1.2. R-OSGi

OSGi ist für sich genommen lediglich eine Service-Plattform und läuft auf einem einzigen Rechner. Durch die Erweiterung der Eclipse Equinox-Implementierung um das R-OSGi-Bundle ([JSR07b] und [JSR07a]), dass an der ETH Zürich entwickelt wurde, besitzt der

[1] Jetty ist ein in Java geschriebener Servlet/JSP-Container und Webserver.

Entwickler nun weitere Funktionalitäten zur Anbindung anderer OSGi-Instanzen. Diese können auf beliebigen anderen Rechnern laufen und lassen sich über gängige Netzwerke mithilfe von R-OSGi anbinden. Das so erweiterte Eclipse Equinox Framework stellt die Middleware für das zu implementierende dar. Der Einsatz von R-OSGi wird in Abbildung 6.1 veranschaulicht.

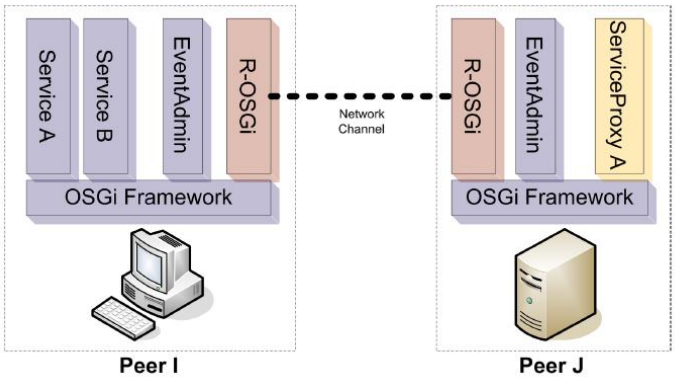

Abbildung 6.1.: Schematische Darstellung zweier OSGi-Instanzen die über R-OSGi verbunden sind

Performanz

Die Verwendung von R-OSGi ermöglicht den Remoteaufruf jedes möglichen Bundles und bietet nach Angaben der Entwickler im Vergleich zu anderen Java-Basierten Middlewarelösungen deutliche Performanzvorteile. So ist R-OSGi nach [JSR07b] S.14ff. deutlich schneller als RMI oder UPnP-Implementierungen. Für das entfernte Deployment von Analysebundles bzw. Analyseservices existiert bereits eine prototypische Implementierung die ebenfalls in diesem Projekt Verwendung findet (siehe auch [JSR07a]).

Kommunikation zwischen den Knoten

Für die Anbindung an das System kann jedes Gerät verwendet werden unter dem ein lauffähiges OSGi mit R-OSGi installiert werden kann und das eine Netzwerkanbindung besitzt. R-OSGi kommuniziert ausschließlich nachrichtenbasiert. Eine Nachricht beinhaltet einen Header zur Beschreibung der Nachricht, einige zusätzliche Attribute und einen

Body mit dem Nachrichteninhalt. Dabei sind alle Nachrichten in binärer Form enthalten. Ein Netzwerkkanal basiert unter R-OSGi per default als persistente TCP-Verbindung mit keep-Alive Option vgl. RFCC1122 S.101f.

Registrierung entfernter Services

Um entfernte Services bereitzustellen, besitzt R-OSGi eine verteilte Service-Registry. Durch einen synchronen Methodenaufruf (vgl. [JSR07b] S.7) erhält der aufrufende Client eine Menge von Servicereferenzen. Für den Aufruf wird dann ein lokaler ServiceProxy generiert, der den entfernten Dienst aufruft. Dadurch kann bei Bedarf theoretisch jedes Bündel mit einem entfernten Dienst transparent kommunizieren.

Aufruf eines entfernten Services

Wie in Abbildung 6.1 zu sehen ist, besitzt „Peer J“ einen ServiceProxy A. Dieser wurde von der R-OSGi Service-Registry erstellt, da „Peer J“ den Service A von „Peer I“ verwenden möchte. Die Verwendung entfernter Dienste auf dem lokalen Peer wird in R-OSGi grundsätzlich mit Hilfe des Proxy-Entwurfsmusters realisiert (siehe auch [Sta08] S.142f.).

Erweiterbarkeit

R-OSGi bietet verschiedene Erweiterungspunkte, sodass ein Netzwerkkanal beispielsweise überwacht oder verschlüsselt werden kann. Weitere Informationen zu diesem Aspekt befinden sich im Abschnitt 8.3.

6.1.3. JSP und AJAX

Das System beinhaltetet außer einer konsolenbasierten Benutzerschnittstelle eine Web-Schnittstelle. Auf der Serverseite wird die Webpräsentation mit Hilfe von Java-Server-Pages erstellt. Auf der Clientseite existieren JavaScript-Frameworks. Dies ist zum Einen das „Protoype“-Framework zur asynchronen Kommunikation über AJAX-Technologie mit dem Web-Server. Zum Anderen wird das „Uize“-Framework für Darstellungen verwendet.

Es bietet Templates und Widgets für die Darstellung von Daten an. Dadurch können beispielsweie dynamisch über AJAX geladene auf JSON basierende Tabellendaten in eine sortierbare Liste eingefügt werden.

6.2. Entwicklungsumgebung

6.2.1. IDE Eclipse

Im Rahmen der Programmierung wurde die „IDE Eclipse“ mit den Erweiterungen für RCP/Plugin Entwickler eingesetzt. Dadurch können viele Schritte der Plugin Entwicklung vereinfacht durchgeführt werden. So kann beispielsweise das Anlegen von Plugin Projekten mit Bundles durch zusätzliche Wizards und Editoren einfach durchgeführt werden, die Abhängigkeiten zu anderen Bundles werden dargestellt und die Bearbeitung der „plugin.xml“ sowie der „manifest.mf“-Dateien über benutzerfreundliche Editoren wird ermöglicht.

6.2.2. Virtual Private Network

Da die Entwicklung die meiste Zeit nicht am Institut stattfindet, wird über ein Virtual Private Network der Zugang auf Ressourcen des Intranets am IPP sichergestellt. Ein Virtual Private Network dient der Einbindung von Geräten eines benachbarten Netzes an das eigene Netz. So können die im Intranet des Instituts installierten Dienste auch physisch außerhalb des Intranets, beim Arbeiten von zuhause verwendet werden.

7. Qualitätssicherung und Test

Durch die Qualitätssicherung und das Testen soll zum Einen überprüft werden, ob das System die funktionalen Anforderungen erfüllt. Zum Anderen soll bewertet werden ob und wie weit die nichtfunktionalen Anforderungen erfüllt werden. Im Hinblick darauf das ein verteiltes System vorliegt

7.1. Qualitätssicherungsmaßnahmen

7.1.1. Organisatorische Maßnahmen

Um die Wartbarkeit des Quellcodes zu erhöhen, werden die Java Coding Conventions eingehalten sowie der Quellcode mit JavaDoc dokumentiert. Durch die Verwendung von Checkstyle können bei der Formatierung des Quellcodes bestimmte Regeln festgelegt und automatisch geprüft werden.

7.1.2. Statische Analysen

Ein Werkzeug für die statische Analyse ist das PMD-Eclipse-Plugin. Es gibt beispielsweise Warnungen bei leeren „try catch“-Blöcken, zu kompliziertem bzw. verschachteltem Code oder dupliziertem Quellcode aus.

7.2. Modul- und Integrationstest

Mit Hilfe von Unittests werden Module getestet sowie der Überdeckungsgrad der Tests mit entsprechenden Werkzeugen gemessen. Dabei wird ein Überdeckungsgrad von 95% angestrebt. Für die Unittests sind JUnit und EasyMock im Einsatz während für die Messung des Überdeckungsgrades Cobertura verwendet wird.

Bei dem Integrationstest werden nach und nach einzelne Bundles und Services mit einander integriert und ihr Verhalten getestet. Dabei werden spezifische Testfälle und Anwendungsszenarien durchlaufen, um das korrekte funktionale Verhalten zu überprüfen. Zusätzlich werden zustandsbasierte Tests durchgeführt. Dies bedeutet das geprüft wird, ob das System in einem bestimmten Zustand auch genau die Funktionalität besitzt, die es besitzen darf. Beispielsweise darf das System im unkonfigurierten Zustand keine Analyseanfragen annehmen. Die zustandsbasierten Tests finden im Rahmen des Integrationstests statt, da der Systemtest ein reiner Black-Box-Test ist.

7.3. Systemtest und Abnahmetest

Zum Zeitpunkt der Abgabe der Arbeit existiert noch kein komplett realisiertes Software-System. Der aktuelle Stand ist in Abschnitt 8.1 beschrieben. Bei Fortsetzung der Implementierung müssen im Systemstest in der realen Umgebung verschiedenste Testszenarien durchlaufen werden. Die Testszenarien prüfen die Korrektheit und Vollständigkeit der funktionalen Systemeigenschaften sowie die nichtfunktionalen Eigenschaften, wie das Systemverhalten bei hoher Nennlast oder bei Ausfällen von Ressourcen. Des weiteren sollen die Vollständigkeit und Korrektheit von Dokumenten geprüft werden und ein Installationstest durchgeführt werden. Der Systemtest sollte und der Abnahmetest muss in der realen Umgebung stattfinden. Im Gegensatz zum Systemtest wird der Abnahmetest ausschließlich vom Kunden durchgeführt.

8. Diskussion

Das abschließende Kapitel beschreibt zunächst den aktuellen Stand der Entwicklung sowie die während des Projektes gesammelten Erfahrungen. Es gibt weiterhin einen Ausblick über die mögliche Verwendung des Systems am Institut.

8.1. Status

Das Ziel der Arbeit war der Architekturentwurf und die prototypische Implementierung eines verteilten Systems zur Administration und Überwachung von Analysemodulen. Zum Zeitpunkt der Abgabe der Arbeit befindet sich das Projekt in einer fortgeschrittenen Implementierungsphase und Testphase. Während der Implementierung werden Funktionstests durchgeführt. Aber auch integrative Tests, durch die Verwendung mehrerer Services und Bundles kommen zum Einsatz. Die implementierte Grundfunktionalität wurde getestet. Wesentliche Anwendungsfälle die zur Verteilung und Überwachung notwendig sind wurden implementiert. Bei den Integrationstests kommen Testszenarien zum Einsatz, die anhand der Anforderungen, der Architektureigenschaften und der Implementierung erarbeitet wurden (siehe Abschnitt 7.2). Bei der Implementierung lag zunächst der Schwerpunkt auf der Realisierung der hochpriorisierten Anwendungsfälle und der Steuerung des Systems über die Kommandozeile. Weitere Informationen finden sich dazu im Dokument „Vision+Scope“ sowie „AWF+Features“ auf der beiliegenden CD. Für die Web-Schnittstelle wurde prototypisch ein Logging-Monitor implementiert. Er verwendet die „ServletBridge“ (siehe Abschnitt 6.1.1). Die Implementierung für die Web-GUI stellt einen senkrechten Prototyp dar und kann als Beispiel für weitere Implementierungen von Anwendungsfällen für die Web-GUI genutzt werden.

8.2. Gesammelte Erfahrungen

8.2.1. Unbekannte Technologie und Frameworks

Die Recherche und das Betrachten vergleichbarer Systeme (siehe Abschnitt 4.2.3 und 4.2.2) sowie die grundsätzliche Betrachtung von Service-orientierten Architekturen, insbesondere von Service-Mediations-Systemen (Abschnitt 4.3), eröffnete zusätzliche Perspektiven im Hinblick auf die Systementwicklung und möglichen Risiken.

Eine besondere Herausforderung in diesem Projekt stellte die unbekannte OSGi Technologie dar. Durch Recherchen und entsprechende Literatur, insbesondere [GW08], konnten in einem angemessenen Zeitraum mit der OSGi-Techniologie, den Frameworks Equinox (siehe Abschnitt 6.1.1) und dem R-OSGi Framework (siehe Abschnitt 6.1.2) umfangreiche Erfahrungen gesammelt werden. Für die Realisierung der Web-Schnittstelle des verteilten, auf OSGi basierenden, Systems wurde die sogenannte „ServletBridge“ (vgl. [ML06]) eingesetzt. Sie ermöglicht die OSGi-Instanz, die das eigentliche Mediations-System beinhaltet, in einen Web-Container einzubinden, so dass über „Tomcat“ und ähnliche Webapplikationsserver das Mediationssystem als Webapplikation laufen kann. Die „ServletBridge“ selbst ist jedoch kaum dokumentiert und benötigt eine gewisse Erfahrung in der Handhabung und Konfiguration.

Der Einsatz der Entwicklungsumgebung „Eclipse“ sowie die Erweiterung „Eclipse/RCP“, die eine Entwicklungsperspektive speziell für die PlugIn-Enwicklung anbietet, hat sich wie auch in vorherigen Projekten bewährt und viele Aufgaben, beispielsweise bei der Bearbeitung von XML-Dateien oder der Auflösung von Abhängigkeiten und der Arbeit mit OSGi im Allgemeinen deutlich erleichtert.

8.2.2. Vorgehen

Im Wesentlichen wurde nach dem V-Modell vorgegangen. So wurde bereits in vorhergehenden Projekten der Entwicklungsprozess praktiziert und viele Erfahrungen gesammelt. Nach der Domain- und Anforderungsanalyse (siehe Abschnitt 2 und Abschnitt 3) wurde zusätzlich nach vergleichbaren Systemen recherchiert (siehe Abschnitt 4.2.3 und 4.2.2).

Durch die prototypische Implementierung von Services, wurden Risiken im Vorfeld minimiert. Anschließend fand der Architekturentwurf statt, der mehrmals mit den Anforderungen abgeglichen und angepasst wurde. Es kommen viele bewährte Entwurfsmuster zum Einsatz, die vor Allem auf die nichtfunktionalen Eigenschaften des Systems Auswirkungen haben. Der Einsatz von Mustern hat sich als einfach und sehr effizient erwiesen. Die Verwendung einer speziell für die PlugIn-Entwicklung angepassten Entwicklungsumgebung, sowie zusätzlichen Erweiterungen für statische Codeanalysen und zur Messung von Überdeckungsmetriken erleichterte die qualitative Realisierung des Systems zusätzlich.

8.2.3. Arbeiten außerhalb des Instituts

Da der größte Teil der Arbeit von zu Hause aus stattfand, wurden Maßnahmen getroffen, um zu mindest auf technischer Ebene einen möglichst umfangreichen Zugriff auf das Intranet des Institut zu besitzen. Durch die Verwendung eines Virtual Private Network, um einen entfernten Zugriff auf Intranetressourcen des IPP zu realisieren, und einer fernbedienbaren Workstation am Institut, konnte auf technischer Ebene virtuell, wie vor Ort, gearbeitet werden. Während der Arbeit kamen aber immer wieder Fragen auf, die nur durch Gespräche real vor Ort mit dem Betreuer Georg Kühner schlüssig beantwortet werden konnten.

8.2.4. Rückschlüsse

Recherchen während der Analysephase

Die längere Recherche zu Beginn und während der Analysephase des Projektes sowie die prototyipsche Implementierung zur Prüfung der Realisierbarkeit bestimmter Anforderungen mit der ausgewählten Technologie, haben sich bewährt und positiv ausgewirkt. Es konnten so im Vorfeld Informationen gesammelt werden, die deutliche Auswirkungen auf Entscheidungen bei dem Architekturentwurf zur Folge hatten (siehe Abschnitt 5.1).

Die Verwendung von OSGi

Das Arbeiten mit der OSGi-Technologie brachte viele Vorteile. Durch den hohen Grad der Modularisierung, dem serviceorientierten Ansatz mit Hilfe einer Service-Registry sowie der Möglichkeit der Steuerung eines OSGi-Systems während der Laufzeit, kann mit OSGi eine hochgradig entkoppelte, mit wohl definierten öffentlichen Schnittstellen und Abhängigkeiten bestehende Architektur geschaffen werden. Des Weiteren zeichnet sich eine auf OSGi basierende Architektur durch hohe Erweiterbarkeit, Flexibilität, Wartbarkeit aber auch durch hohe Ausfallsicherheit und Verfügbarkeit aus. Viele Services stellt OSGi selbst bereit. Jedoch muss OSGi, das in einer einzigen Java Virtual Machine läuft, für ein physisch verteiltes System erweitert werden (siehe Abschnitt 4.3 und 6.1.2). Auch hier wurden positive Erfahrungen gemacht, da sich Frameworks zur unkomplizierten Erweiterung eines OSGi-Systems in ein System mit Remote-Eigenschaften einfach verwenden ließen. Bei der Erweiterung wurde hier das OpenSource-Framework R-OSGi verwendet, dessen Dokumentation jedoch umfangreicher hätte sein können. R-OSGi wurde das erste mal bei der „EclipseCon 2007“ präsentiert und findet als Middlewarelösung in mehreren Projekten an der ETH Zürich Verwendung, unter Anderem in dem Projekt „flowSGi“ (siehe auch Abschnitt 4.2.3).

8.3. Ausblick

8.3.1. Fortsetzung des Projektes

Bei der Fortsetzung des Projektes muss zunächst der System- und Abnahmetest erfolgreich durchgeführt werden. In einem nächsten Release ist der nächste Schritt die Implementierung der Web-GUI und der übrigen niedrigpriorisierten Anwendungsfälle sowie Features (siehe auch das Dokument „AWF+Features“ auf der beiliegenden CD).

8.3.2. Erweiterung

Bei einem tatsächlichen Einsatz des Systems sollte unbedingt über eine Erweiterung des Systems nachgedacht werden. Diese sollte eine Persistenzschicht enthalten, um Systemzustände dauerhaft speichern zu können. Dies ermöglicht zum Einen die Realisierung

von Backup- und Restore-Mechanismen. Zum Anderen kann so durch eine passive Redundanz (siehe [LB03] S.103 f.) des Mediators die Ausfallsicherheit deutlich erhöht werden. So kann eine zweite Mediator-Instanz bei einem Ausfall eines Mediators automatisch einspringen.

Des Weiteren besitzt das System Erweiterungspunkte, um Kommunikationskanäle zu verschlüsseln und überwachbar zu machen. Die Erweiterung von Autorisierungs- und Authentifizierungsmechanismen ist ebenfalls vorgesehen. Dadurch könnte ein externer sicherer Zugriff auf das Analysesystem realisiert werden. Da sämtliche Textausgaben aus einem zentralen Services stammen, ist die Internationalisierbarkeit ebenfalls mit geringem Aufwand realisierbar.

8.3.3. Verwendung

Bei einem ausgereifteren Stadium des Systems mit entsprechend umfangreichen und erfolgreich bestandenen Systemstests und abschließendem Abnahmetest (siehe Abschnitt 7.3), kann das System auch dauerhaft eingesetzt werden. Um eine breitere und schnellere Akzeptanz für das System zu erreichen, sollte auch über einen Installations-Wizard für Analyseknoten nachgedacht werden, der gegebenenfalls auch von EDV-Mitarbeitern ohne systemspezifische Kenntnisse genutzt werden könnte. Der Hauptgrund dafür ist das die Analysemodule selbst, die fast völlig entkoppelt vom verteilten System arbeiten, auf betriebssystemabhängige Bibliotheken, Pfade und Property-Dateien zugreifen müssen. Dies muss bei der Erstinstallation eines Analyseknotens bereitgestellt werden.

Die Prozesse am Institut sind oft evolutionär und Strukturen sowie Anforderungen ergeben sich nach und nach. Bei einer umfangreicheren Verwendung des Systems werden neue Anforderungen und Wünsche durch die Anwender des Systems entstehen. Um diese Anforderungen in einem späteren Release mit einfließen zu lassen, sollten entsprechende Feedbacks schon frühzeitig gesammelt werden. Dafür könnte die webbasierte Benutzerschnittstelle einen einfachen Feedbackmechanismus bereitstellen, der Anwendermeinungen und -wünsche entgegennehmen kann.

Literaturverzeichnis

[Fow03] FOWLER, MARTIN: *Patterns für Enterprise Application-Architekturen*. mitp-Verlag, 2003. ISBN 3-8266-1378-3.

[GW08] GERD WÜTHERICH, NILS HARTMANN, BERND KOLB MATTHIAS LÜBKEN: *Die OSGi Service Platform*. dpunkt.verlag, 2008. ISBN 978-3-89864-457-0.

[JD08] JÜRGEN DUNKEL, ANDREAS EBERHART, STEFAN FISCHER CARSTEN KLEINER ARNE KOSCHEL: *Systemarchitekturen für Verteilte Anwendungen*. Carl Hanser Verlag München Wien, 2008. ISBN 978-3-446-41321-4.

[JM03] J. MANS, D. BENGALI: *BlueOx: A Java Framework for Distributed Data Analysis*. Talk from the 2003 Computing in High Energy and Nuclear Physics (CHEP03), 2003. http://arxiv.org/pdf/cs/0306055.

[JM07] J. MANS, D. BENGALI: *Designing data analysis services in the Knowledge Grid*. CoreGRID Technical Report Number TR-0118, 2007. http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0118.pdf.

[JSR07a] JAN S. RELLERMEYER, GUSTAVO ALONSO, TIMOTHY ROSCOE: *Building, deploying, and monitoring distributed applications with Eclipse and R-OSGI*. In: *eclipse '07: Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, Seiten 50–54, New York, NY, USA, 2007. ACM. ISBN 978-1-60558-015-9.

[JSR07b] JAN S. RELLERMEYER, GUSTAVO ALONSO, TIMOTHY ROSCOE: *R-OSGi: Distributed Applications through Software Modularization*. Verlag Springer Heidelberg/Berlin, 2007. ISBN 978-3-540-76777-0.

[LB03] LEN BASS, PAUL CLEMENTS, RICK KAZMAN: *Software Architecture in Practice*. Addison-Wesley, 2003. ISBN-10: 0321154959.

[Mat08] MATHAS, CHRISTOPH: *SOA intern*. Carl Hanser Verlag München Wien, 2008. ISBN 978-3-446-41189-0.

[ML06] MARTIN LIPPERT, BERND KOLB, WOLFGANG WÜTHERICH: *Server-Side-Eclipse*. eclipse Magazin, 2 2006.

[Nie07] NIECKCHEN: *Das Max-Planck-Institut für Plasmaphysik im Jahresrückblick*. Max-Planck-Institut für Plasmaphysik, 2007. ISSN 1610-19 52.

[SKP08] STUART K. PATERSON, ANDREW MAIER: *Distributed Data Analysis in LHCb*. International Conference on Computing in High Energy and Nuclear Physics, Conference Series 119 (2008) 072026, 2008. http://www.iop.org/EJ/article/1742-6596/119/7/072026/jpconf8_119_072026.pdf.

[Sta08] STARKE, GERNOT: *Effektive Software-Architekturen*. Carl Hanser Verlag München Wien, 3rd Auflage, 2008. ISBN 978-3-446-41215-6.

[TP04] TORSTEN POSCH, KLAUS BIRKEN, MICHAEL GERDOM: *Basiswissen Softwarearchitektur*. Dpunkt.Verlag GmbH, 2004. ISBN-13: 978-3898642705.

[WG07] WOLFGANG GENTZSCH, HEIKE NEUROTH, MARTINA KERZEL: *Die D-Grid-Initiative*. Universitätsverlag Göttingen, 2007. ISBN 978-3-940334-01-4.

A. Übersicht aller Anwendungsfälle

Weitere detaillierte Informationen und Beschreibungen finden sich auf der CD im Dokument „AWF+Features.xls“.

Nr.	Anwendungsfall
AA01	Deployen eines Analysemodules auf einen Computerknoten
AA02	Starten eines Analysemodules auf einem Computerknoten
AA03	Stoppen eines Analysemodules auf einem Computerknoten
AA04	Undeployen eines Analysemodules von einem Computerknoten
AA05	Anzeige der Analysemodule auf einem Computerknoten
AA06	Anzeige der Analysemodule zu einer Funktionsgruppe
AA07	Anzeige eines Anlaysemodules
AA08	Anzeige des Analysemodulrepositories
AA09	Konfigurieren eines Analyseservices
CA01	Anzeige der angeschlossenen Computerknoten
CA02	Hinzufügen eines Computerknoten
CA03	Anzeige eines Computerknoten
SA01	Anzeige der Systemeigenschaften
SA02	Setzen von Systemeigenschaften
SA03	Anzeige von Log-Informationen
SA04	Hinzufügen notwendiger Module auf einen Computerknoten
SA05	Systemweites Update von Kernmodulen auf Computerknoten
SA06	Hilfefunktion

Tabelle A.1.: Übersicht aller Anwendungsfälle

B. Übersicht aller Features

Weitere detaillierte Informationen und Beschreibungen finden sich auf der CD im Dokument „AWF+Features.xls“.

Nr.	Feature
F01	Automatisches Anbinden von Computerknoten
F02	HeartBeat Detektor
F03	Automatisches Deployen und Undeployen von Analyseservices
F03a	Bei dem Aufbau einer Analysekette
F03b	Bei der Replikation eines ausgefallenen Knotens
F04	Automatisches Deployen notwendiger Kernmodule auf einen Computerknoten
F05	Load-Balancing-Mechanismus

Tabelle B.1.: Übersicht aller Features

C. Bausteinsicht

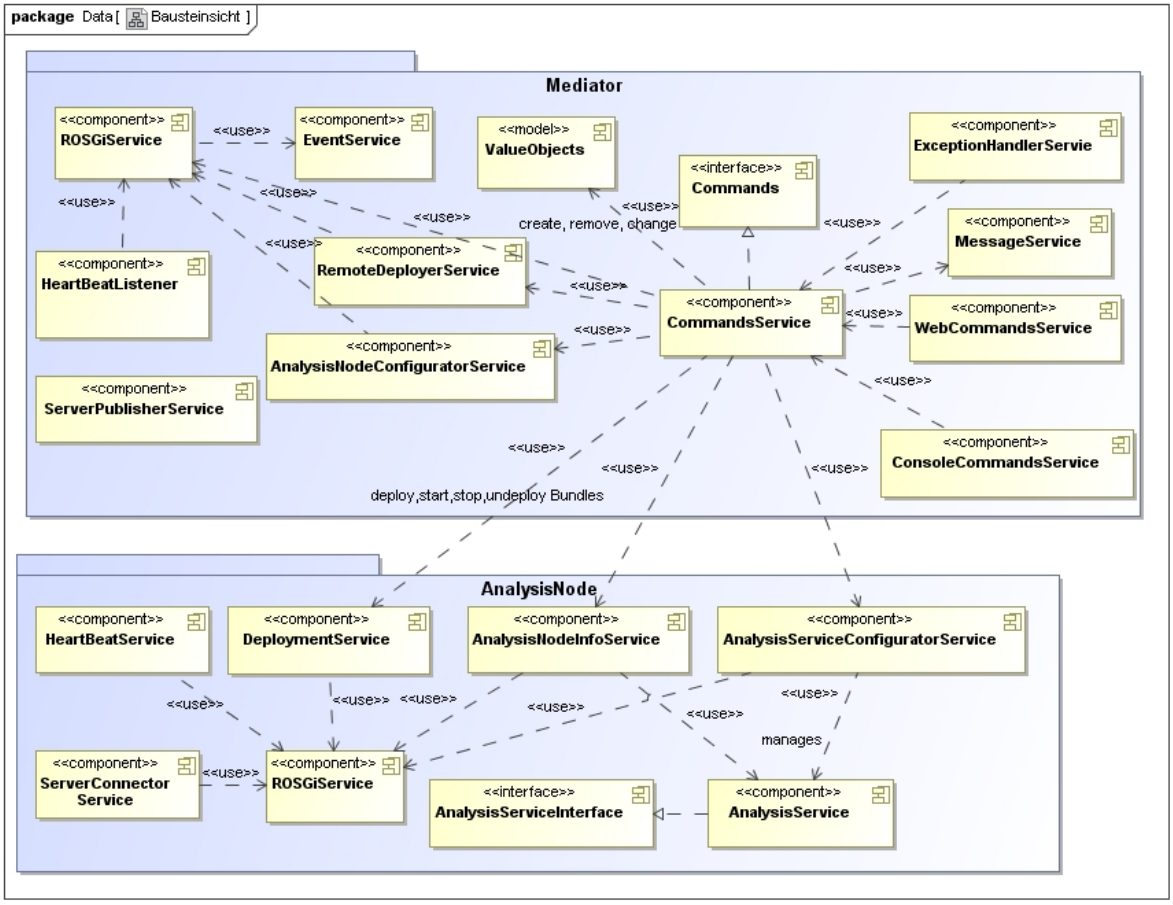

Abbildung C.1.: UML 2.0 Komponentendiagramm: Die Abbildung zeigt die Bausteinsicht des Systems

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel sind angegeben. Die Arbeit hat mit gleichem bzw. in wesentlichen Teilen gleichem Inhalt noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum Unterschrift