Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Konzeption und Implementierung eines Applikationsservers für linguistische Anwendungen

DIPLOMARBEIT

Vorgelegt von:
Karsten Böhm

Betreut durch:
Dr. habil. Uwe Quasthoff

Leipzig, Dezember 2000

"What does your master teach?" asked a visitor.
"Nothing." said the disciple.
"Then, why does he give discourses?"
"He only points the way — he teaches nothing."

Antonio de Mello, *"One minute wisdom"*

Danksagung

Mein Dank gilt zunächst meinen Eltern, die mir dieses Studium ermöglicht haben, mich in jeder Phase nach Kräften unterstützten und denen ich es maßgeblich verdanke, daß ich diese Arbeit fertigstellen konnte.

Herrn Dr. Quasthoff bin ich für seine geduldige Unterstützung während der gesamten Dauer der Arbeit dankbar und schätze seine Fähigkeit, mir bei der Arbeit den größtmöglichen Spielraum zu lassen. In Dr. Wolff fand ich immer einen Ansprechpartner, sei es bei technischen Fragestellungen oder bei der kritischen Begutachtung von Teilen der Arbeit. Schließlich danke ich Herrn Prof. Heyer, der als Leiter der Abteilung Automatische Sprachverarbeitung ein Umfeld geschaffen hat, daß durch Offenheit und Freiraum geprägt ist und mir so die Möglichkeit gab, mich voll zu entfalten.

Meinen Kommilitonen Fabian Schmidt und Timo Böhme danke ich für die zahlreichen hilfreichen Diskussionen zu Teilaspekten der Arbeit und ihre kritischen Anmerkungen zum Manuskript und Thomas Pantzer für seine Unterstützung bei verschiedenen PostScript-Problemen.

Leipzig
22. Dezember 2000

Karsten Böhm

Kurzfassung

Das Projekt “Deutscher Wortschatz” wurde Anfang der 90er Jahre am Institut für Informatik der Universität Leipzig begonnen und stellt mittlerweile eine der umfangreichsten korpuslinguistischen Datensammlungen zur deutschen Sprache dar. Der zunehmende Umfang der Sammlung und die wachsende Akzeptanz durch verschiedene Anwender aus Forschung und Wirtschaft zeigen deutlich die Grenzen der gegenwärtig verwendeten Architektur im Hinblick auf Skalierbarkeit, Performanz und Verfügbarkeit auf.

Diese Diplomarbeit beschreibt die Konzeption und Implementierung eines linguistischen Applikationsservers, der als Plattform für korpuslinguistische Anwendungen eingesetzt werden kann und insbesondere auf die Bedürfnisse des Projektes “Deutscher Wortschatz” ausgerichtet ist. Dabei handelt es sich um eine auf mehrere Rechner verteilbare, plattformneutrale Anwendung, die skalierbar ist, eine hohe Verfügbarkeit aufweist und die transparente Integration neuer Anwendungen ermöglicht. Dabei stehen die Belange linguistischer Software im Vordergrund, obwohl der Einsatz des Systems auch in anderen Aufgabenbereichen denkbar ist.

Der konzeptionelle Teil der Arbeit beschreibt zunächst die Besonderheiten und Eigenschaften, die linguistische Systeme, insbesondere solche, die mit großen Korpora arbeiten, besitzen und leitet daraus Forderungen ab, die eine Entwicklungsumgebung erfüllen muß, um diese Anwendungsklasse zu unterstützen. Ein weiterer Schwerpunkt ist die Analyse und Begriffsdefinition der Klasse der Applikationsserver – eine neue Softwaregattung die sich in einer eigenen Schicht zwischen System- und Anwendungssoftware ausgebildet hat. Im Hinblick auf den Einsatz in einem PC-Cluster fließen in die Konzeption eines linguistischen Applikationsservers auch Aspekte verteilter Systeme ein, die beim Entwurf des Systems berücksichtigt werden.

Die Vorstellung der Implementierung des vorgeschlagenen Konzepts bildet den zweiten, umfangreicheren Teil der Arbeit und stellt dem Projekt “Deutscher Wortschatz” ein neues Werkzeug zur Verfügung, das die Entwicklung verteilter, linguistischer Anwendungen erleichtert und die hohe Leistungsfähigkeit der Clusterarchitektur ausnutzt. Außerdem wird mit der Realisierung des vorgeschlagenen Konzepts dessen Praktikabilität nachgewiesen und damit gezeigt, wie in der Zukunft komplexe linguistische Softwaresysteme aufgebaut sein könnten.

Schlüsselworte: linguistische Software, Applikationsserver, verteilte Systeme, Datenbanken, Workstation Cluster, Linux, Java, JDBC, MySQL

Abstract

The "German-Vocabulary-Project" started in the early nineties at the Department of Computer Science at Leipzig University and represents one of the largest collections of linguistic data on contemporary German language available today. The rapid growth and the increasing interest among different users from various research groups and industry partners alike are pointing out the limitations of the current architecture used to process these data, in terms of scalability, performance and availability.

This Master's thesis describes the architecture and implementation of a linguistic application server that can be used as a platform for corpuslinguistic applications and is especially suited for the needs of the "German-Vocabulary-Project". The functionality of the application server can be distributed among different nodes in a cluster of workstations to satisfy the high demand of time-intensive and resource-intensive data processing. The design of the application server imposes only minimal requirements on the operating system that the nodes are running. It is therefore possible to use a wide range of workstations with heterogenous configurations. Due to the cluster architecture, the system is scalable and can be used to increase the availability of the applications that are running on the server. The provided functions are specially tailored for use by linguistic applications but the basic framework is suitable for other applications too.

The first part of the thesis identifies the properties and specific requirements of linguistic systems. In particular, we focus on systems that deploy large collections of data and derive a set of functions that must be provided to support linguistic applications. Another issue is the investigation of the term 'application server' as a new type of system software. Some systems available on the market today are evaluated to support the proposed definitions. Finally, when designing an application server for a cluster-based architecture some aspects of distributed systems have to be taken into account. The second part of the thesis describes the implementation of a new set of tools for the "German-Vocabulary-Project". These tools can be used to develop distributed linguistic applications capable of exploiting the high performance of the cluster architecture with minimal effort. The implementation can be seen as a 'proof of concept' for the outlined architecture and shows how complex linguistic systems might look like in the future.

Keywords: Linguistic Software, Application Server, Distributed Systems, Database Systems, Workstation Cluster, Linux, Java, JDBC, MySQL

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Das Projekt "Deutscher Wortschatz" beschäftigt sich seit 1994 mit dem Aufbau eines umfangreichen Korpus der deutschen Gegenwartssprache. Die in diesem Zusammenhang entstehenden umfangreichen linguistischen Daten werden in verschiedenen Systemen gespeichert und verarbeitet. Der stetige Zuwachs an Daten eröffnet dabei neue Nutzungmöglichkeiten und führt zum Anwachsen der der Benutzerzahlen auf der einen und zu einer Heterogenität der Benutzerstruktur auf der anderen Seite, die sich in unterschiedlichen Fachkenntnissen und Zielstellungen manifestiert. Die Verarbeitung von speziellen Fachwortschätzen und die schrittweise Einbeziehung weiterer indoeuropäischer Sprachen in das Projekt führt zu weiteren Engpässen bei der Verarbeitung der Daten.

Diese Randbedingungen stellen zunehmend höhere Anforderungen an die eingesetzte technische Infrastruktur, die sich parallel zu dem Projekt entwickelt und sukzessive den Anforderungen angepaßt wurde. Nachdem die Daten zunächst in einer flachen Dateistruktur gespeichert wurden, erfolgte 1997 der Übergang zu einer zentralen relationalen Datenbank, auf die mehrere Benutzer gleichzeitig zugreifen konnten. Gegenwärtig stößt dieses Konzept aufgrund steigendenden Platzbedarfs und ungenügender Geschwindigkeit des zentralen Rechners an seine Grenzen. Die 1998 realisierte Verfügbarkeit des Wortschatzes im Internet führte zu einer erneuten Verschärfung der Situation und stellte neue Anforderungen, insbesondere an die Reaktionsgeschwindigkeit des Systems auch bei hoher Belastung durch viele Nutzer.

Daher wurde nach einer Infrastruktur gesucht, die die genannten Probleme beseitigt und eine skalierbare Plattform für die zukünfig zum Einsatz kommenden Anwendungen bietet. Grundlage dieses neuen Systems ist aufgrund des äußerst günstigen Preis/Leistungs-Verhältnisses ein Cluster aus mehreren Standard-PCs[1].

Dieses Kapitel dient der Einführung in die bearbeitete Thematik und beginnt mit einer kurzen Darstellung des Wortschatzprojekts. Anschließend wird gezeigt, wie ausgehend von der gegebenen Situation eine Aufgabenstellung erarbeitet wurde, deren Lösung Gegenstand dieser Arbeit ist.

[1]Diese zunehmend häufiger anzutreffenden Systeme werden auch als *Cluster of Workstations* (COW) bezeichnet.

1.1 Das Wortschatzprojekt

Gegenstand des Wortschatz-Projekts ist der Aufbau eines umfangreichen Vollformenlexikons der deutschen Gegenwartssprache und eines großen Textkorpus für linguistische Analysen, die beide in einem bereits mehrere Jahre andauernden Sammelprozeß enstanden sind.

Die Entstehung des Lexikons basierte zunächst auf dem Wunsch, eine möglichst vollständige Sammlung deutscher Vollformen aufzubauen, die dem Sprachgebrauch der Gegenwart entspricht. Im Gegensatz zu klassischen redaktionell bearbeiteten Lexika sollte der Sammlungsprozeß, der auf der Analyse von Textdokumenten basiert, weitgehend automatisiert werden (siehe [Fis00]), um so den manuellen Aufwand möglichst gering zu halten. Sie beinhaltet daher auch umgangssprachliche Ausdrücke sowie Termini verschiedener Fachgebiete und zwangsläufig auch die in den Quellen enthaltenen Fehler. Um trotzdem eine hohe Qualität des Lexikons gewährleisten zu können, wird für die Aufnahme neuer Einträge in den Wortschatz ein zyklischer, selbstorganisierender Prozeß verwendet, der in der Lage ist, fehlerhafte Daten zu erkennen und entsprechend zu markieren (siehe [Qua98b]).

Dabei mußten die in Frage kommenden Quellen nur wenige Kriterien erfüllen, um in die Sammlung aufgenommen zu werden: Neben der offensichlichen Forderung der Maschinenlesbarkeit waren dies ein orthographisches Mindestniveau, das nicht unter dem von Zeitungstext liegen sollte und eine thematische und sprachliche Vielfalt der untersuchten Quellen, die für eine Ausgewogenheit der gesammelten Lemmata sorgt. Insbesondere das letzte Kriterium ist hierbei für die inhaltliche Struktur relevant, da es über- (z. B. Computerwissenschaften) und unterrepräsentierte (z. B. Prosa) Bereiche der deutschen Sprache gibt, was die Verfügbarkeit maschinenlesbarer Texte betrifft.

Aufbauend auf dieser Grundlage wurde die Sammlung schrittweise um weitere linguistische, beschreibende oder klassifizierende Merkmale erweitert, wie sie sich auch in klassischen Lexikontypen finden: Angaben zur Grammatik, Morphologie, Pragmatik, Sachgebietsangaben und einer Beschreibung. Damit wurde die Benutzbarkeit und die Möglichkeiten der Sammlung erheblich erweitert, obwohl die genannten Eigenschaften zunächst nicht für alle Einträge vorhanden waren. Um diesen Mangel zu beseitigen, wurden geeignete Algorithmen entworfen, die mit Hilfe verschiedener heuristischer Verfahren die fehlenden Angaben ergänzten (siehe [Qua98c, Qua98a]), so daß mittlerweile für die meisten Einträge vollständige Merkmalsangaben vorliegen, die das in Abbildung 1.1 dargestellte Aussehen haben und von den Benutzern des Wortschatzes abgerufen werden können.

Der weitere Ausbau des Lexikons erfolgte mit Hilfe von Verfahren der Korpuslinguistik (siehe [Uwe99]). Dazu war es zunächst nötig, einen Korpus zu erzeugen, dessen Ausgangspunkt die Belegstellen zu den Lexikoneinträgen bildeten. Da diese Beispielsätze allein noch keinen repräsentativen Korpus für die Verwendung eines Wortes im allgemeinen Sprachgebrauch darstellen, erfolgte eine Erweiterung des Bestandes um neues Textmaterial aus verschiedenen Quellen, mit dem dann ein ausgewogenes Verhältnis der einzelnen Einträge erreicht wurde.

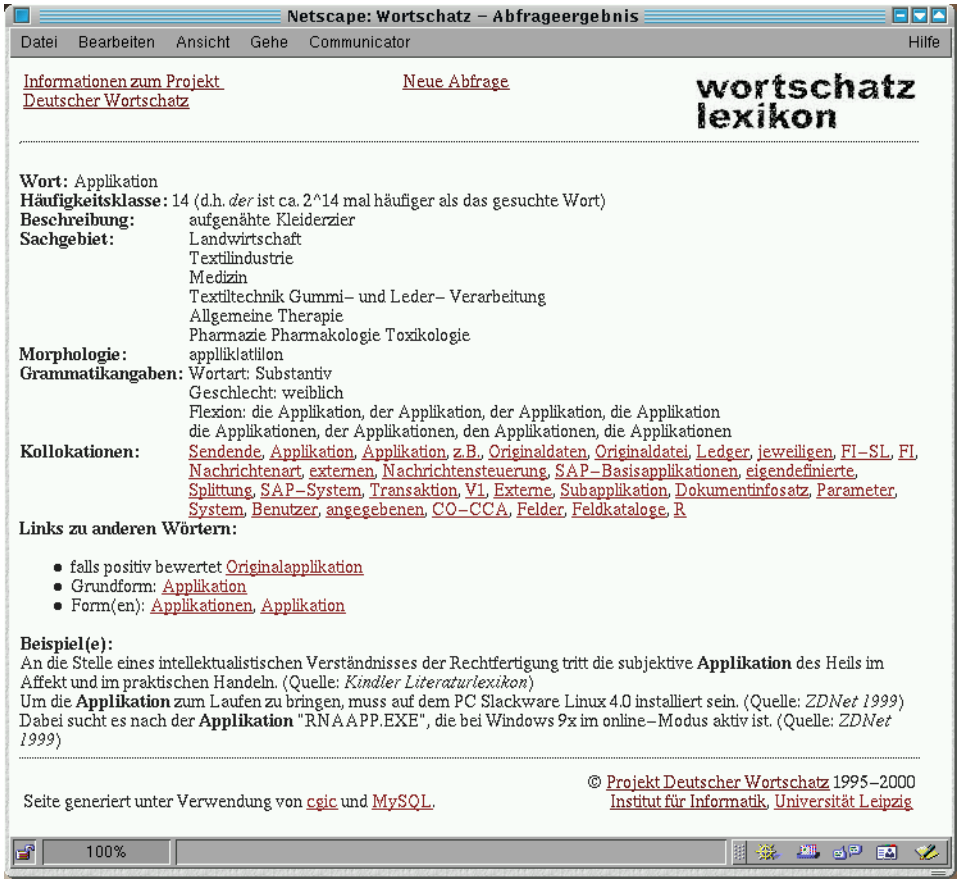


Abbildung 1.1: Abfrageergebnis einer Wortschatzabfrage

Die Analyse dieses Korpus ermöglichte die Identifikation zahlreicher neuer Beziehungen zwischen den Lemmata, beispielweise die Ermittlung von Kollokationen (siehe [Sch99]). Kollokationen beschreiben das Phänomen des gemeinsamen Auftretens von Wörtern und können für die Erkennung von semantischen Beziehungen im Text (Assoziationen) genutzt werden. Sie bilden damit eine Schlüsselrolle für das automatische Verstehen von Text ohne dedizierte Wissensbasis.

Anhand der Visualisierung der Kollokationen (im Beispiel Abbildung 1.2 zu *Stich*) ist für den Betrachter eine intuitive Bedeutungsanalyse des Begriffs und die Einordnung in einen Kontext ähnlicher Begriffe möglich. Dies kann beispielsweise zur Bedeutungsdisambiguierung genutzt werden und ermöglicht sogar die Erschließung der Bedeutung unbekannter Begriffe durch Schlußfolgerungen aus den bestehenden Assoziationen. Da diese Beziehungen direkt aus dem vorliegenden Textmaterial extrahiert wurden, spiegeln sie die reale Verwendung eines Begriffs wider und reflektieren nicht die möglicherweise subjektiven Vorstellungen und Ansichten eines Autors, wie dies z. B. bei manuell erstellten Ontologien der Fall ist.

Das Beispiel der Kollokationen zeigt sehr anschaulich, welche Vorteile das enstandene Lexikon im Gegensatz zu traditionellen Nachschlagewerken hat, da erst durch die elektronische Verfügbarkeit und die große Datenmenge die Analyse einer Rei-

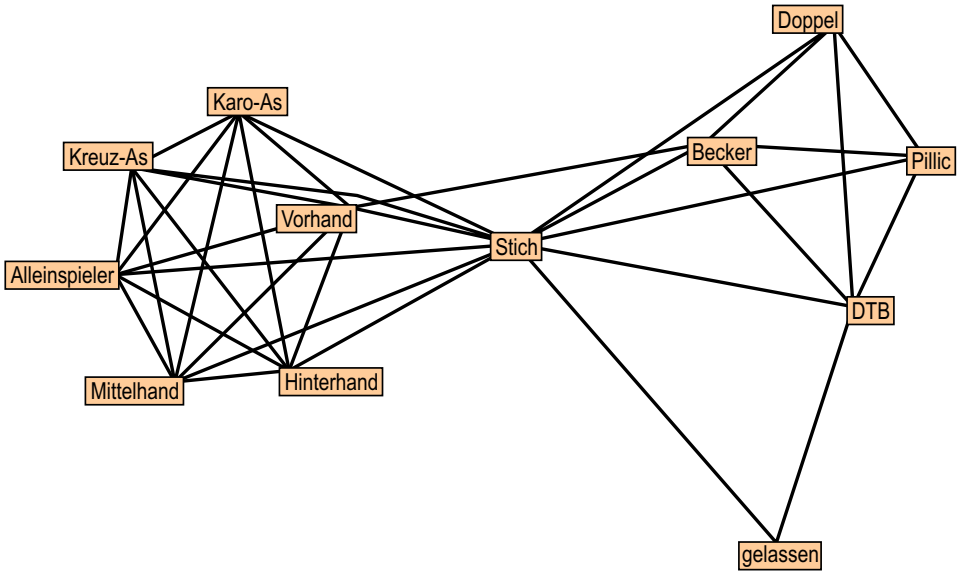


Abbildung 1.2: Beispiel für einen Kollokationsgraphen

he von Fragestellungen möglich wird, die ein in Buchform gehaltenes Lexikon nicht bietet. In diesem Sinne ist das im Rahmen des Projekts erstellte Lexikon als Ergänzung und nicht als Ersatz traditioneller Nachschlagewerke anzusehen.

Eine weitere Eigenschaft des Projektes ist die Tatsache, das die Ergebnisse eines Analyseverfahrens in vielen Fällen als Ausgangsdaten für weitere Untersuchungen genutzt werden können. Bezogen auf das Beispiel der oben erläuterten Kollokationen lassen sich schnell weitere sinnvolle Anwendungen ableiten[2]:

- Die Beschränkung der Kollokationen eines Wortes Einträge mit bestimmten Eigenschaften (Beispielsweise die Einschränkung auf Eigennamen oder Nomen) stellt eine Filterfunktion dar und ermöglicht bei großen Kollokatsmengen eine anders strukturierte Sichtweise auf die gleichen Daten.

- Die Kollokationsmengen selbst können ebenfalls noch weiter analysiert werden: Die Schnittmenge zweier Kollokatsmengen ermöglicht Aussagen über die semantische Nähe zweier unabhängiger Begriffe und kann auf die Bildung von Kollokationen von Kollokationen ausgedehnt werden (siehe [Wit00a, Wit00b]).

- Die graphische Darstellung semantischer Relationen eröffnet ganz neue Möglichkeiten und erlaubt den Einsatz 'artfremder' Algorithmen für linguistische Analysen. Denkbar wäre beispielsweise ein Disambiguierungsverfahren, das mit Hilfe von Verfahren der Graphentheorie ausgehend vom analysierten Begriff verschiedene Cluster von zusammenhängenden Konzepten isoliert, die auf die verschiedenen Bedeutungen eines Begriffs hinweisen (im

[2]Die ersten beiden Anwendungen sind mittlerweile schon im Wortschatzprojekt umgesetzt, während das letztgenannte Beispiel noch nicht implementiert wurde.

in Abbildung 1.2 dargestellten Beispiel lassen sich zwei solche Cluster identifizieren). Durch die Analyse der Sachgebietsinformationen der in den Clustern enthaltenen Begriffe ist so sogar eine noch genauere automatische Zuordnung des untersuchten Konzepts möglich.

Zusammenfassend kann festgestellt werden, daß sich das Wortschatzprojekt ausgehend von einfachen Wortlisten zu einem umfangreichen Werkzeug entwickelt hat, das vielfältige Analysemöglichkeiten bietet und ein hohes Entwicklungspotenial besitzt. Einen maßgeblichen Anteil am Erfolg des Projekts hat hierbei die Tatsache, daß die Ergebnisse jeder neuen Analysemöglichkeit in das Lexikon einfließen und damit dessen Wert erhöhen.

1.2 Motivation und Aufgabenstellung

Die bereits angeführten Leistungsengpässe, die durch die Entwicklung des Wortschatzprojekts in den letzten Jahren entstanden, waren zusammen mit dem prognostizierten weiteren Ansteigen der Nutzerzahlen und der Korpusgröße Anlaß dafür, nach einer neuen leistungsfähigeren Infrastruktur zu suchen, die zukünftig als Fundament für das Projekt dienen kann und mittelfristig die derzeit verwendete Lösung ablösen soll.

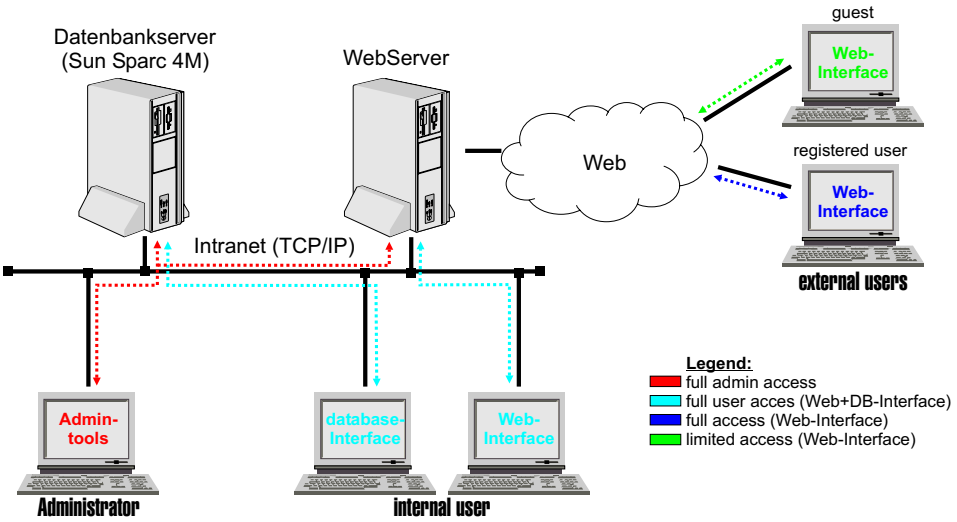


Abbildung 1.3: derzeitige Architektur des Wortschatzprojekts (aus [Böh99])

Gegenwärtig erfolgt sowohl die Datenspeicherung als auch die Abarbeitung wesentlicher Bearbeitungsprozesse auf einem einzelnen zentralen Rechner (siehe Abbildung 1.3). Obwohl es sich dabei um ein leistungsfähiges Mehrprozessorsystem handelt, kommt es bei aufwendigen Prozessen oder hohen Benutzerzahlen zu unvertretbaren Wartezeiten, die die Interaktivität der Benutzerschnittstelle zum Wortschatz beeinflussen und Abstimmung bei der Durchführung umfangreicher Analyseprozesse erfordern. Hinzu kommt, daß die verwendete Maschine nicht exklusiv

für das Wortschatzprojekt verwendet werden kann, sondern auch für andere Projekte eingesetzt wird. Der naheliegende Ansatz, bei Beibehaltung der verwendeten Infrastruktur die Leistungsfähigkeit des betroffenen Systems durch bessere Hardware zu steigern, bzw. dieses auszutauschen scheitert an den zu hohen finanziellen Kosten und beseitigt das Problem nur kurzfristig.

Deshalb wurde nach einer alternativen Lösungsansätzen gesucht, die dann in Form einer Clusterlösung gefunden worden, die in Abbildung 1.4 dargestellt ist. Dabei handelt es sich um eine Anzahl meist gleichwertiger PCs, die aus Standardkomponenten aufgebaut sind und über ein Netzwerk miteinander verbunden sind. Diese Lösung ist *kostengünstig, skalierbar* und ermöglicht eine *verteilte Verarbeitung*, wodurch die Entkopplung von parallel laufenden unabhängigen Prozessen möglich wird.

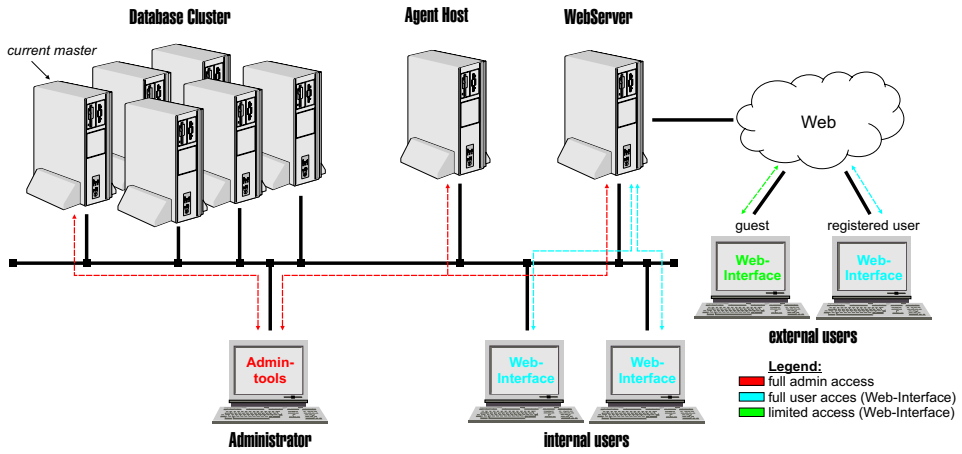


Abbildung 1.4: vorgeschlagene Architektur des Wortschatzprojekts (aus [Böh99])

Ausgehend von einer solchen Architektur wurde zunächst ein Projektvorschlag (siehe [Böh99]) ausgearbeitet, der die gegenwärtige Situation analysiert und ein Konzept vorsah, bei dem das verwendete relationale Datenbanksystem vollständig redundant auf die einzelnen Clusterknoten verteilt werden sollte. Grundlage für diesen Ansatz war hierbei die Annahme, daß die Leistungsoptimierung des Datenbanksystems, das den Kernpunkt des Wortschatzprojektes darstellt, maßgeblichen Einfluß auf das Reaktionsverhalten des Gesamtsystems hat.

Für die Realisierung dieses Konzepts war es notwendig, einen Verteilungsmechanismus zu entwerfen, der in einer Schicht oberhalb der Knoten-DBMS angesiedelt ist, mit den Client-Anwendungen kommuniziert und die eingehenden Anfragen möglichst gleichmäßig auf die einzelnen Knoten verteilt. Der durch diese zusätzliche Ebene und weitere Nebenbedingungen (z. B. Synchronitätsforderungen, ACID-Eigenschaften) entstehende Mehraufwand wurde hierbei in Kauf genommen, da aufgrund der speziellen Abfragestruktur der Wortschatzanwendungen (im wesentlichen handelt es sich um lesende Abfragen, die in der Regel Bearbeitungszeiten im Sekundenbereich haben) mit geringen Verlusten zu rechnen ist. Um die Flexibilität der Verteilungsmechanismen zu gewährleisten und eine Anpassung auf sich

ändernde Umgebungsbedingungen[3] zu erreichen, sah das Konzept eine modulartiges System auf der Basis von Softwareagenten vor.

Während der Beschäftigung mit dem Projektvorschlag und den ersten Arbeiten zu dessen Umsetzung wurde schnell klar, daß der Cluster ein weit höheres Anwenungspotential hat, als zunächst angenommen und daß eine Ausnutzung dieser Fähigkeiten für viele Anwendungen des Wortschatzes von Vorteil wäre. So entstand ein umfangreicher Anforderungskatalog, dessen Umsetzung eine Revision des Konzepts erforderte und zum Entstehen des Applikationsservers als tragende Infrastruktur für den Cluster führte. Dieser Applikationsserver sollte zum einen die Wortschatzanwendungen integrieren und die benötigten Funktionen bereitstellen, gleichzeitig jedoch so flexibel angelegt ein, daß eine Erweiterung des Systems problemlos möglich ist.

Die Analyse bereits am Markt befindlicher Applikationsserver resultierte jedoch in der Feststellung, daß keines der Produkte trotz des sehr großen Funktionsumfanges für den geplanten Einsatz im Wortschatzprojekt besonders gut geeignet ist. Aus diesem zunächst überraschenden Ergebnis ließ sich schließen, daß für den Entwurf eines Applikationsservers immer die geplante Zielgruppe zu berücksichtigen ist – im konkreten Fall galt es also, die besonderen Eigenschaften linguistischer Software zu erarbeiten, um eine optimale Unterstützung gewährleisten zu können.

Die endgültige Konzeption berücksichtigt diese Gesichtspunkte, und stellt ein Framework zur Verfügung, in das sich die Anwendungen des Wortschatzprojektes einbetten, um so mit minimalem Aufwand von der Leistungsfähigkeit des Clusters zu profitieren. Die anschließende Implementierung sollte die Praktikabilität dieses Ansatzes zeigen und eine Nutzung des Clustes im Rahmen des Projekts ermöglichen.

1.3 Aufbau der Arbeit

Der Rest der Arbeit gliedert sich in zwei thematische Teile: Der konzeptionelle Teil, der im folgenden Kaptiel enthalten ist, beschäftigt sich mit den Grundlagen für die Umsetzung der gestellten Aufgabenstellung. Hierbei gilt es zunächst anhand allgemeiner Betrachtungen und praktischer Beispiele zu klären, was allgemein unter einem Applikationsserver zu verstehen ist. Im Anschluß daran werden die Eigenschaften linguistischen Software untersucht und schließlich das Konzept für den zu realisierenden linguistischen Applikationsserver vorgestellt. Der zweite Teil der Arbeit beschreibt die Implementierung eines linguistischen Applikationsservers, der im Projekt “Deutscher Wortschatz” eingesetzt werden soll und das vorgeschlagene Konzept umsetzt. Dabei wird in Kapitel 3 auf Seite 46 zunächst die allgemeine Laufzeitumgebung vorgestellt, während in Kapitel 4 auf Seite 84 die auf der Plattform aufbauenden Dienste im Detail beschrieben werden. Die Zusammenfassung in Kapitel 5 auf Seite 133 zieht ein Resümee über die gewonnenen Erkentnisse und den erreichten Entwicklungsstand und weist auf Bereiche hin, in denen noch weiterer Forschungsbedarf besteht.

[3] Hierzu zählen das Lastverhalten der einzelnen Knoten, die typischen Anfragestrukturen der einzelnen Anwendungen und der Aufbau der Datenbasis.

Kapitel 2

Konzeption eines linguistischen Applikationsservers

Dieses Kapitel beschreibt die Entwicklung einer Konzeption für einen linguistischen Applikationsserver, der für die Anwendungen innerhalb des Wortschatzprojektes prädestiniert ist.

Zur Erreichung dieses Ziels wird zunächst in Abschnitt 2.1 die Softwarekategorie, die durch den Begriff Applikationsserver beschrieben wird, vorgestellt und anhand ihrer Eigenschaften und Funktionen pragmatisch definiert. Da es sich bei Applikationsservern um eine neue Gattung von Software handelt, ist eine Einordnung dieses Begriffs zunächst problematisch, da ihm, ähnlich wie dem Begriff 'Multimedia', eine klare Definition fehlt und damit eine *genaue* Positionierung schwerfällt. Durch die Einbettung des Begriffs in die zentrale Infrastruktur moderner Informationssysteme wird jedoch dessen Bedeutung klarer und kann als Grundlage für die Entwicklung einer allgemeinen Definition dienen. Zur Herstellung eines Praxisbezugs werden dann zwei Vertreter dieser neuen Softwaregattung gegenübergestellt und deren unterschiedlicher Charakter herausgearbeitet. Die Ableitung eines allgemeinen Bewertungsmaßes aus den gewonnenen Erkenntnissen bildet den Abschluß des Überblicks und zeigt den weiteren Forschungsbedarf auf.

Im Anschluß daran wird in Abschnitt 2.2 die Spezifik linguistischer Software untersucht, bewertet und die Alleinstellungsmerkmale dieser Klasse von Anwendungen herausgearbeitet. Hierbei konzentrieren wir uns auf die korpuslinguistische Software, da diese in dem Wortschatzprojekt dominiert und besonders hohe Anforderungen an die Laufzeitumgebung stellt.

Aufbauend auf diese notwendigen Vorbetrachtungen wird im Abschnitt 2.3 schrittweise das Konzept des Applikationsservers entwickelt und dessen Bestandteile dargestellt. Die zu beachtenden Einflußfaktoren, die hauptsächlich aus der verwendeten Cluster-Architektur resultieren, spielen hierbei eine besondere Rolle und grenzen den konzipierten Entwurf von existierenden Systemen ab.

Ein Resümee der gewonnenen Erkenntnisse bildet den Abschluß des Kapitels und leitet zu der Implementierung des Systems über, der die anschließenden Kapitel gewidmet sind.

2.1 Applikationsserver — ein Überblick

Der Begriff des Applikationsservers ist innerhalb der Informatik weit verbreitet und hat durch das Wachstum des Internets in den letzten Jahren einen besonderen Aufschwung erlebt.

Interessanterweise bleibt es trotzdem unklar, was genau sich hinter dem Begriff verbirgt, denn offensichtlich gehen die Ansichten weit auseinander, wie das folgenden Zitat aus [Mik98] besipielhaft illustriert:

> "Software makers agree: Application servers could be the hottest new product category the software business has seen in years.
>
> Analysts agree: Application servers already make up a half-billion dollar market and will drive sales of roughly $2 billion in the coming four to five years.
>
> And technology buyers concur: Application servers are needed to help them tame the hairball of connections needed for the typical Web app and to get new systems up-and-running in a hurry. Sixty-two percent of execs surveyed by Forrester Research say application servers will be a strategic part of their development environments by year's end.
>
> With all of this head nodding, you'd think that it would be easy to find consensus on what an application server is. No chance — ask the two dozen or so application server vendors and you'll get a wide range of answers. (...) Like any booming product category, confusion is reigning."

Obwohl diese Beschreibung bereits vor zwei Jahren entstand, hat sich an der Situation nicht viel geändert; es scheint vielmehr durch die vielen verschiedenen Produkte, die mittlerweile am Markt sind und die Bezeichnung Applikationsserver tragen, eine noch größere Verwirrung um den Begriff an sich entstanden zu sein.

Um zu einer allgemeinen Definition des Begriffs Applikationsserver aus heutiger Sicht zu kommen, gehe ich daher folgendermaßen vor: Zunächst betrachte ich die modernen Informationssysteme im Kontext ihrer historischen Entwicklung, wobei die Eigenschaften, die zur Entstehung von Applikationsservern beigetragen haben, von besonderem Interesse sind. Aufbauend darauf werde ich anhand dieser Entwicklung ein allgemeingültiges Paradigma formulieren, das die Evolution von komplexen Informationsystemen beschreibt. Schließlich werden die Ursachen für die Entstehung der Softwaregattung Applikationsserver angegeben und eine Definition des Begriffes vorgestellt, die die Charakteristik dieser Systeme beschreibt.

2.1.1 Historische Einordnung

Um die Struktur moderner Informationssysteme zu verstehen, ist es notwendig, einen Blick auf die Entwicklung der Informationstechnik in den letzten fünfzig Jahren zu werfen. Dabei ist nicht so sehr die rasante technische Entwicklung der Geräte von Interesse, als vielmehr die Infrastruktur solcher Systeme, die sich ausgehend von einfachen monolithischen Systemen hin zu einem komplexen Geflecht von interagierenden Komponenten entwickelt hat. Die wesentlichen Schritte in dieser Entwicklung sind in Abbildung 2.1 dargestellt und werden im folgenden vorgestellt und kommentiert, wobei auf eine genaue zeitliche Einordnung aus Gründen der Übersichtlichkeit verzichtet und auf [Geo94], Seite 23 verwiesen wird.

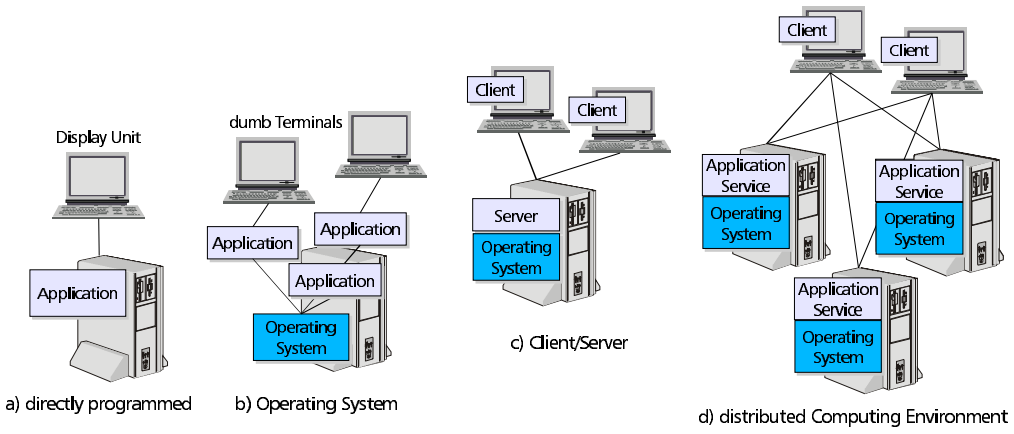


Abbildung 2.1: Evolution moderner Informationssysteme

In den Anfängen des Informationszeitalters[1] wurden Rechenmaschinen (Computer) direkt programmiert (siehe Abbildung 2.1a), d.h. für jede zu lösende Aufgabe wurde ein Programm entworfen, in den Rechner eingegeben und von diesem abgearbeitet. Alle benötigten Funktionen mußten jedesmal neu programmiert werden und belasteten den Entwurf von Programmen mit der immer wieder notwendigen Lösung von Standardaufgaben wie Ein- und Ausgabefunktionen oder Fehlerbehandlungsroutinen. Obwohl diese Art der Programmierung auf den ersten Blick antiquiert erscheint, ist sie auch heute noch im Einsatz – in Waschmaschinen und anderen Konsumgütern, die im wesentlichen über eine festgelegte Steuer- oder Regelfunktion verfügen.

Um sich dieser lästigen Standardaufgaben zu entledigen, entstand bald die Idee, die grundlegenden Funktionen in einem eigenen Programm – dem Betriebssystem – zu kapseln und dieses zusammen mit der eigentlichen Anwendung auf dem Rechner ablaufen zu lassen (siehe Abbildung 2.1b). Aus diesem Ansatz entwickelte sich eine Vielzahl von Betriebssystemen für die unterschiedlichsten Anforderungen, und obwohl sie heute ungleich leistungsfähiger als ihre Vorgänger

[1] Als Beginn des modernen Informationszeitalters wird der Zeitpunkt betrachtet, zu dem die ersten elektronischen Universalrechner verfügbar waren und mit dem von-Neumann-Prinzip ein Paradigma geschaffen wurde, das den Aufbau universeller Computer ermöglicht und das auch noch in heutigen Systemen verwendet wird.

sind, nach wie vor die gleiche prinzipielle Funktion erfüllen, nämlich die Bereitstellung systemspezifischer, anwendungsneutraler Funktionen, die von allen Applikationen genutzt werden können. Der interessanteste Punkt ist in diesem Zusammenhang die erstmals auftretende Aufteilung von Funktionen in verschiedene Schichten oder Ebenen (engl. *tiers*). Diese Unterscheidung zwischen System- und Anwendungssoftware hat auch heute noch Bestand und wird von [Bal96] wie folgt definiert:

> ***Systemsoftware**, auch Basissoftware genannt, ist Software, die für eine spezielle Hardware oder eine Hardwarefamilie entwickelt wurde, um den Betrieb und die Wartung dieser Hardware zu ermöglichen bzw. zu erleichtern. Zur Systemsoftware zählt man immer das Betriebssystem, in der Regel aber auch Compiler, Datenbanken, Kommunikationsprogramme und spezielle Dienstprogramme.*
>
> ***Anwendungssoftware** (application software), auch Applikationssoftware genannt, ist Software, die Aufgaben des Anwenders mit Hilfe eines Computersystems löst. Anwendersoftware setzt in der Regel auf der Systemsoftware der verwendeten Hardware auf, bzw. benutzt sie zur Erfüllung der eigenen Aufgaben.*

Bis zu diesem Zeitpunkt erfolgte die Nutzung der Rechner durch eine einzelne Person (dem Operator), der die abzuarbeitenden Programme von den eigentlichen Anwendern entgegennahm und nacheinander ausführte. Später wurde durch die Einführung von Multi-User-Betriebssystemen eine direkte Nutzung der teuren, zentralen Rechenanlagen (*Mainframes*) durch mehrere Benutzer möglich (der Operator trat als Verwalter des Systems in den Hintergrund), die durch Ein/Ausgabeeinheiten (*Terminals*) mit dem System verbunden waren.

Mit der Einführung des *Personal Computers (PC)* durch IBM wurde diese zentralisierte Form der Datenverarbeitung erstmals aufgebrochen und ermöglichte es den Anwendern, dezentral zu arbeiten, ohne von einem Rechenzentrum abhängig zu sein. Da die finanziellen Aufwendungen für ein solches System ungleich geringer waren, verbreiteten sich die PCs ungeachtet ihrer anfangs bescheidenen Leistungen rasch. In Unternehmen und größeren Arbeitsgruppen entstand aber bald die Notwendigkeit, Daten untereinander auszutauschen oder gemeinsam zu bearbeiten. Das führte zur Einrichtung von zentralen Rechnern (Servern), die durch Netzwerke mit den Arbeitsplatzrechnern verbunden waren. Dies schien zunächst wieder ein Weg zurück zur zentralen Datenverarbeitung zu sein, im Unterschied zu dieser waren die Arbeitsplatzrechner aber jetzt nicht nur für die Benutzerinteraktion zuständig, sondern übernahmen die Ausführung der eigentlichen Anwendungsprogramme. Die Server stellten zunehmend Dienste zur Verfügung (Datei- und Druckdienste, Datenbankdienste), die von den Anwender-PCs in Anspruch genommen wurden (siehe Abbildung 2.1c). Bemerkenswert ist hier die erneute Schichtenbildung, die sich nun sogar über mehrere Ebenen und Systeme erstreckt: Betriebssystem des Servers, angebotene Dienste auf dem Server, Betriebssystem des Klienten und schließlich das Anwendungsprogramm auf dem Arbeitsplatzrechner. Diese sich schnell verbreitende Infrastruktur wird als *Client/Server-Modell*

bezeichnet und wird durch zwei Hauptkomponenten charakterisiert: dem *Server*, der einen Dienst anbietet und dem *Klienten (engl. Client)*, der den Dienst in Anspruch nimmt. [Uma97] definiert die Client/Server-Architektur folgendermaßen:

1. *Clients and servers are functions modules with well defined interfaces (i.e. they hide internal information). The functions performed by a client and a server can be implemented by a set of software modules, hardware components, or a combination thereof. Clients and servers typically reside on seperate machines connected through a network.*

2. *Each client/server relationship is established between two functional modules when one module (client) initiates a service request and the other (server) chooses to respond to the service request.*

3. *Information exchange between clients and servers is strictly through messages (i.e. there is no information exchanged through global variables).*

Die breite Anwendung dieses Konzepts zunächst für einige wenige Dienste (hauptsächlich Datei- und Druckdienste, die von sogenannten *Fileservern* zur Verfügung gestellt wurden) entwickelte sich rasch zu einem dichten Netzwerk verschiedener Dienste für eine breite Palette von Anwendungen (Mail, News, HTTP[2], SNMP[3], NIS[4], um nur einige zu nennen), die zur Ausbildung einer verteilten DV-Umgebung führten (*distributed computing environment*, siehe Abbildung 2.1d) und unter dem Begriff *Middleware* zusammengefaßt werden. Nach [Uma97] definiert sich diese Klasse von Software in der folgenden Weise:

> ***Middleware*** *is a set of common business-unaware services that enable applications and end users to interact with each other across a network. In essence, middleware is the software that resides above the network and below the business-aware application software.*

Diese Entwicklung von Client/Server-Systemen und die Entstehung von Middleware ist bemerkenswert, weil sie starke Parallelen zu der weiter oben beschriebenen Entwicklung von Betriebssystemen hat. Da es in anderen Bereichen der Informationstechnik, beispielsweise der Hardwareentwicklung ähnliche Prozesse gibt (Entwicklung von speziellen Coprozessoren für numerische Berechnungen (FPU[5]), Entwurf quasiautonomer leistungsfähiger Subsysteme wie Grafikkarten (AGP[6]) oder I/O-Systeme (SCSI), liegt die Vermutung nahe, daß dieser Prozeß der Entwicklung komplexer Informationssysteme inheränt ist.

[2]HTTP: das Hypertext Transfer Protocol wird für die Kommunikation zwischen Web-Browsern und -Servern benutzt

[3]SNMP: Simple Network Management Protocol dient der Verwaltung von Geräten in einem Netzwerk

[4]NIS: Network Information System, ist ein Dienst für die netzwerkweite Verwaltung von Benutzerinformationen

[5]FPU: Floating Point Unit

[6]AGP: Accelerated Graphics Port

2.1.2 Evolution moderner Informationsysteme

Wie im letzten Abschnitt illustriert, führt die fortschreitende Entwicklung moderner Informationssysteme zu immer komplexeren Strukturen, die zu handhaben es immer ausgefeilterer Methoden bedarf. Vergleicht man die Entwicklungen der letzten Jahre, so stellt man ein gemeinsames Muster fest, das in der folgenden Hypothese beschrieben wird:

*Die **Entwicklung komplexer Informationssysteme**, die auf der Interaktion weitgehend unabhängiger Komponenten basiert, erfolgt zyklisch und ist durch zwei sich wiederholende Entwicklungsstufen charakterisiert:*

1. ***Spezialisierung:** In einer ersten Phase erfolgt eine Ausdifferenzierung der Funktionalität einzelner Komponenten bis zu einem kritischen Komplexitätsgrad. Steigt die Funktionalität weiter, zerfällt die Komponente in funktionale Teilkomponenten, die sich ebenfalls weiterentwickeln können. Dieser Prozess wiederholt sich so lange, bis die gewünschte Funktionalität oder eine maximale Komponentenanzahl erreicht ist.*
2. ***Abstraktion:** Konnte die angestrebte Funktionalität nicht erreicht werden, setzt die zweite Phase ein, die funktional zusammengehörige Gruppen in eine einzige Einheit (oder Ebene) zusammenfasst. Dabei wird dieses Objekt wie eine einzelne Komponente behandelt, die über Schnittstellen mit ihrer Umgebung kommuniziert. Die mit diesem Prozeß einhergehende Verringerung der Komplexität stimuliert erneut den Spezalisierungsprozeß und vollendet damit den Zyklus.*

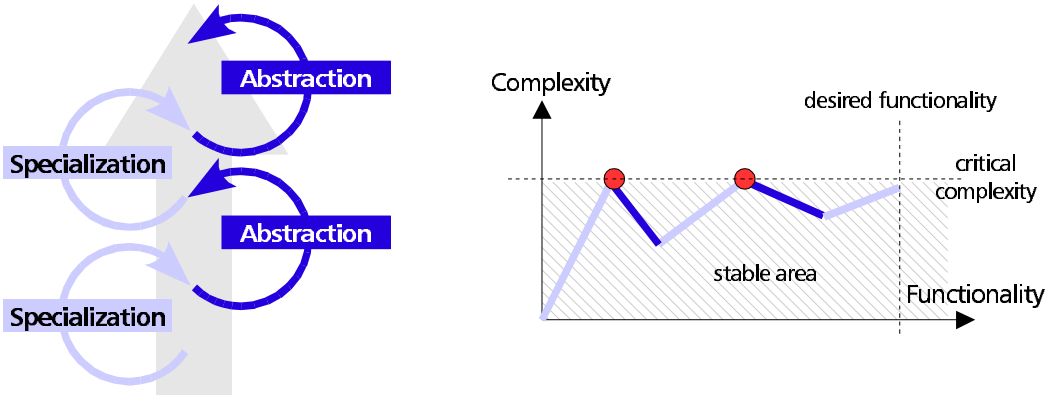


Abbildung 2.2: zyklisches Wachstum durch Spezialisierung und Abstraktion

Wie in Abbildung 2.2 dargestellt, wird durch dieses Prinzip die Funktionaliät des Systems schrittweise bis zu dem gewünschten Umfang vorangetrieben, wobei sich die Komplexität des Systems innerhalb eines handhabbaren Bereichs (*stable area*) bewegt. Bei Erreichen der Komplexitätsgrenze (*critical complexity*) erfolgt durch Einführung einer neuen Abstraktionsebene eine Reduktion, die ein unbeschränktes Wachstum der Systemkomplexität verhindert.

Diese so entstehenden Systeme verfügen über eine Reihe von Eigenschaften, die sie mit anderen komplexen Systemen in Natur, Technik und Wirtschaft vergleichbar machen:

- Die entstehenden Systeme entwickeln sich *evolutionär*, d.h. die Funktionalität entsteht erst bei Bedarf und immer aufbauend auf bereits funktionierenden Komponenten. In gewisser Weise wachsen sie damit 'organisch' mit ihren Anforderungen, wobei sie im Gegensatz zu den üblichen ingenieurtechnischen Entwicklungsparadigmen stehen, die Entwurf, Realisierung und Betrieb als separate sequentielle Entwicklungsschritte betrachten.
- Die Komponenten der Systeme zeigen *symbiotisches Verhalten*, d.h. ihr Funktionieren hängt in zunehmend stärkerem Maße von anderen Komponenten ab (Beispiel: heute wird kaum ein Programm entworfen, daß kein Betriebssystem benötigt).
- Die Entwicklung ist *selbstverstärkend*, *selbstregulierend* und *unumkehrbar* und die entstehenden Systeme sind trotz steigender Komplexität *stabil*. Nach einer initialen Entwicklungsphase setzen die oben beschriebenen Wirkprinzipien eine Entwicklungsspirale in Gang, die zwar ständig neue Möglichkeiten schafft, dabei aber gleichzeitig neue Bedürfnisse weckt. Wie das Beispiel Internet zeigt, kann das entstehende Produkt durchaus komplexer Natur sein und trotzdem ein stabiles Verhalten zeigen.

Die Entwicklung komplexer Informationssysteme auf die oben dargestellte Weise entspricht nicht den klassischen Prinzipien des Softwareentwurfs [Bal96], der nach wie vor die Konstruktion von Informationssystemen durch mehrere sequentielle Entwicklungsphasen beschreibt. Da zukünfig die Entwicklung von Software selten ohne Einflußnahme bereits existierender Systeme erfolgen wird, werden die angesprochenen Kriterien immer bedeutsamer werden. Hierbei ist sicherlich die Einbeziehung anderer Wissensgebiete (z. B. der Biologie und der Kybernetik) sinnvoll, die ähnlichen Problemstellungen gegenüberstehen. Anregungen dazu finden sich in [Kel94] oder [Ves99].

2.1.3 Begriffsfindung

Der Begriff des Applikationsservers verweigert sich einer Definition im herkömmlichen Sinne, da zum einen kaum theoretische Untersuchungen zu diesem Thema vorliegen und die verschiedenen in diesem Marktsegment aktiven Unternehmen das Aufgabengebiet sehr unterschiedlich umreißen und jeweils auf das entsprechende Produkt zuschneiden. Zum anderen ist zum Thema kaum fundierte allgemeine Literatur in Buchform verfügbar, die primäre Informationsquelle sind also Zeitschriften und das Internet.

Die historischen Betrachtungen und die allgmeinen Informationen in den letzten Abschnitten zeigen, daß die heutigen Informationssysteme einen Schichtenaufbau haben, sich auf mehrere Systeme verteilen und einem allgemeinen Prinzip folgen, das bei steigender Funktionalität zu hohe Kapselung durch Abstraktion und Ebenenbildung vermeidet. Diese Systeme beruhen auf dem Client/Server-Prinzip, wobei die Grenzen zwischen Server und Klient[7] zunehmend mehr verschwimmen, da eine Komponente als Klient Dienste eines Servers in Anspruch nehmen kann und diese Informationen, möglicherweise in modifizierter Form, gleichzeitig anderen Komponenten als Server zur Verfügung stellen kann (beispielsweise kann ein Web-Server auf ein DBMS als Klient zugreifen, während er die Daten an Web-Browser in aufbereiteter Form weitergibt und aus deren Sicht als Server auftritt).

2.1.3.1 Ursache für die Entwicklung von Appliktionsservern

Recht klar definieren läßt sich hingegen die Ursache für die Entstehung von Applikationsservern, einer Idee, die ihre Wurzeln in der Entwicklung der verteilten Systeme hat und nicht prinzipiell neu ist. Durch die rasche Verbreitung des Internet als Kanal für die Abwicklung von Geschäftsvorgängen wurde es notwendig, die bereits vorhandenen unternehmensinternen Applikationen (auch als Legacy-Applikationen bezeichnet) nach außen zu öffnen und miteinander zu kombinieren, um das Internet in dieser Form als neues Medium zu nutzen. Bei dem Öffnungsprozeß galt es weiterhin, die Geschäftsabläufe in die neue Präsentationsform zu integrieren, die Risiken einer solchen Öffnung des Unternehmens nach außen zu minimieren und auf einem prinzipiell unsicheren Übertragungmedium eine Lösung zu schaffen, die die sichere Abwicklung von Transaktionen gewährleistet. Andererseits stellt die Entwicklung neuer Anwendungen hohe Anforderungen an die Entwickler, da das Schichtenmodell, die Verteilung der Anwendung auf mehrere Systeme und die notwendige Interaktion mit einer Vielzahl von Subsystemen den Softwareentwurf stark verkomplizieren und den Einarbeitungsaufwand erhöhen.

Diese beiden wesentlichen Funktionen sind Abbildung 2.3 dargestellt. Zusammengefasst lassen sich die Ursachen für die Entwicklung in der folgenden Art und Weise beschreiben:

[7] Die verschiedenen Schreibweisen für "Klient" mögen anfänglich verwirren. Normalerweise werden ich das deutsche Wort (Klient) verwenden, sollte es jedoch in einem Fachbegriff in englischer Schreibweise (Client) vorkommen (wie in Client/Server-Architektur), so behalte ich diese bei.

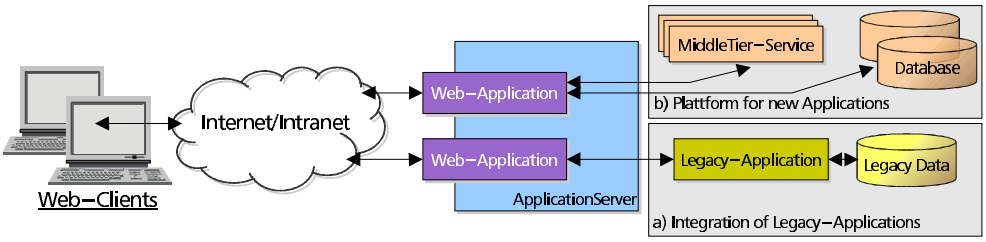


Abbildung 2.3: Haupteinsatzgebiete für Applikationsserver

Ursache für die Entstehung von Applikationsservern: *Der Grund für die Ausbildung von Applikationsservern als neuer Softwaretypus, der in die Kategorie Middleware einzuordnen ist, liegt in dem Bedürfnis nach neuer Funktionalität, die durch die Verbreitung des Internets als neues, globales Informationsmedium maßgeblich bestimmt ist.*

Die benötigten Funktionen lassen sich in zwei Kategorien einordnen:

1. ***Unterstützung von Legacy-Anwendungen:*** *Die Notwendigkeit der Integration von bestehenden in neue Anwendungen, bzw. die Realisierung eines Datenaustauschs zwischen solchen Anwendungen unter der Vorrausssetzung, daß die betroffenen Anwendungen selbst nicht verändert werden. Meist handelt es sich hierbei um den Entwurf einer kommerziellen Web-basierten Anwendung, die die für die Abwicklung der Geschäftsprozesse verwendete bestehende Software verwenden soll oder muß.*
2. ***Plattform für neue Anwendungen:*** *Die Entwicklung neuer Anwendungen, speziell für den Einsatz im Internet, ist durch eine hohe Komplexität gekennzeichnet, die zum einen durch die Vielzahl der zu berücksichtigenden Standards entsteht und zum anderen dadurch gekennzeichnet ist, daß verschiedenste Kommunikationsvorgänge mit Komponenten aller Schichten notwendig sind. Schließlich wird die Entwicklung der Anwendungen durch die Aufteilung in mehrere Schichten – meist sind es mindestens drei – erschwert. Applikationsserver haben in diesem Kontext die Aufgabe, den Entwickler zu unterstützen, indem die Komplexität und Heterogenität der Entwicklungsumgebung weitgehend verborgen wird, wodurch er sich auf die anwendungsspezifische Funktionalität konzentrieren kann.*

2.1.3.2 Zur Begiffsdefinition von Applikationsservern

Ausgehend von der intendierten Funktionalität kann man die Applikationsserver als anwendungsneutral und der Kategorie der Middleware zugehörig betrachen. Da sie, basierend auf einem bereits etablierten Paradigma, eine neue Funktionalität bereitstellen, handelt es sich um eine neue Softwaregattung. Dies führt zur ersten

vorläufigen Definition des Begriffs, die häufig in der Literatur anzutreffen ist, z. B. in [Mik98]:

Definition Applikationsserver (I): *Applikationsserver sind eine Softwaregattung, die zu der Gruppe der Middleware gehört und aufbauend auf Basisfunktionen der Middleware-Schicht anwendungsneutral neue Funktionen realisiert und als Anwendungsplattform dient.*

Charakteristisch für Applikationsserver ist, daß die bereitgestellten Funktionen keine Funktionalität bereitstellen, die vom Endanwender direkt genutzt wird, sondern erst die Grundlage für anwenderspezifische Programme bilden, die die notwendige Funktionalität bereitstellen. Im Unterschied zu den bereits vorhandenen Middleware-Basisdiensten sind die Funktionen bereits auf eine spezielle Klasse von Anwendungen zugeschnitten, abstrahieren von den Basisfunktionen und vereinheitlichen den Zugriff über standardisierte Schnittstellen. Dies läßt sich definitorisch folgerdermaßen fassen:

Definition Applikationsserver (II): *Applikationsserver implementieren eine Anwendungsplattform für eine spezielle Klasse von Anwendungen. Durch teilweise Aufgabe der Universalität und das Zusammenfassen verschiedener Dienste entsteht so eine Schnittstelle, die an eine bestimmte Klasse von Anwendungen besonders gut angepaßt ist.*

Der Aspekt der Abhängigkeit von der Klasse der unterstützten Anwendungen wird üblicherweise nicht besonders hervorgehoben, ist aber meiner Meinung nach der entscheidende Faktor, der zu der Entwicklung vieler – auch funktional verschiedener – Applikationsserver geführt hat. Generell kann man die Spezialiserung auf eine bestimmte Klasse von Anwendungen und die Universalität eines Applikationsservers als entscheidende, aber gegensätzliche Parameter eines solchen Systems identifizieren. Je besser die bereitgestellte Fuktionalität auf eine Klasse von Anwendungen ausgerichtet (spezialisiert) ist, desto leichter wird die Implementierung entsprechender Applikationen fallen. Andererseits wird dadurch die universelle Anwendung des Applikationsservers beeinträchtigt. Ein ausgewogenes Verhältnis zwischen diesen beiden Faktoren ist also entscheidend für den Erfolg eines solchen Produkts. Die betont universelle Auslegung der Funktionalität führt im Extremfall zu der Entwicklung von Basisdiensten, die von allen Anwendungen genutzt werden können, während eine zu hohe Spezialisierung unter Umständen nur von wenigen speziellen Anwendungen genutzt werden kann und in der extremen Ausprägung eher ein Subsystem der entsprechenden Anwendung darstellt.

Definition Applikationsserver (Endfassung): *Applikationsserver sind eine neue Gattung der Systemsoftware und implementieren eine Anwendungsplattform für eine spezielle Klasse von Anwendungen. Obwohl der Applikationsserver keine Anwendungslogik enthält, ergibt sich durch die Spezialisierung eine Abhängigkeit zu bestimmten Klassen von Anwendungen.*

Applikationsserver laufen in der mittleren Schicht einer mehrschichtigen Architektur und gehören damit zur Klasse der Middleware-Anwendungen. Innerhalb dieser

> *Schicht sind sie jedoch oberhalb der Basisdienste angesiedelt, die abstrahiert, in geeigneter Weise homogen integriert und durch definierte Schnittstellen den Anwendungen zur Verfügung gestellt werden.*
>
> *Durch die gegensätzlichen Faktoren **Spezialisierung** und **Universalität** wird sowohl die Anwendungsspezifik des Applikationsservers als auch dessen Nähe zu den verwendeten Basisdiensten definiert.*

Die obige Definition setzt die Applikationserver in Beziehung zu den anderen Komponenten einer modernen Informationsarchitektur und ordnet ihn sowohl funktional als auch strukturbezogen ein. Die Entstehung dieser neuen Gattung wird mit der Notwendigkeit motiviert, Anwendungen zu schaffen, die in die zunehmend komplexer werdende, offene DV-Infrastruktur integriert werden müssen, entspricht dem vorgeschlagenen allgemeinen Entwicklungsprozeß und ist die logische Konsequenz aus der bisherigen technischen Entwicklung. Die Diversität der in diesem Sektor vorhandenen Produkte wurde analysiert und begründet.

2.1.4 Beispiele für Applikationsserver

Nachdem die allgemeinen Eigenschaften von Applikationsservern ausführlich dargestellt wurden, werden in diesem Abschnitt stellvertretend für die große Zahl von verschiedenen Applikationsservern zwei Vertreter kurz vorgestellt und deren wesentliche Eigenschaften umrissen. Da eine umfassende Evaluierung aller am Markt befindlichen Systeme den Rahmen dieser Arbeit sprengen würde, konzentriere ich mich im folgenden auf die schwerpunktmäßige Darstellung zweier gegensätzlicher Produkte, die einerseits die Diversität der Softwaregattung deutlich werden lassen und deren Konzept andererseits Grundlage vieler anderer Applikationsserver ist.

Bemerkenswert für die Entwicklung dieser Softwarekategorie ist hierbei, daß in vielen Fällen der Einsatz von Applikationsservern zunächst primär auf dem Einsatz im Internet ausgerichtet ist (siehe [Die99]) und erst später verallgemeinert und auf die umfassende Unterstützung der Anwendungsentwicklung für bestimmte Anwendungsspektren ausgerichtet wurde (siehe [Ber99], [Beh99]).

2.1.4.1 Enhydra Java Application Server

Der Applikationsserver *Enhydra*[8] von *Lucent Technologies* dient primär der Entwicklung von Anwendungen, die mit Hilfe von Internet-Technologien entworfen und im Inter- und Intranet[9] eingesetzt werden sollen. Dabei handelt es sich um eine komplette, in sich abgeschlossene Entwicklungsumgebung, die durch eine Laufzeitumgebung für die fertigen Anwendungen ergänzt wird und deren schematischer Aufbau in Abbildung 2.4 dargestellt ist.

[8] Weitere Informationen zu dem Produkt finden sich unter `www.enhydra.org`

[9] Die Unterscheidung zwischen *Internet* oder *Intranet* bezieht sich nicht auf die verwendeten Technologien, die in beiden Fällen die gleichen sind, sondern auf die geographische Ausdehnung des Netzwerkes. Während es sich beim Internet um ein globales, öffentliches Netz handelt, ist das Intranet meist lokal beschränkt und nicht oder nur in eingeschränktem Maße öffentlich zugänglich.

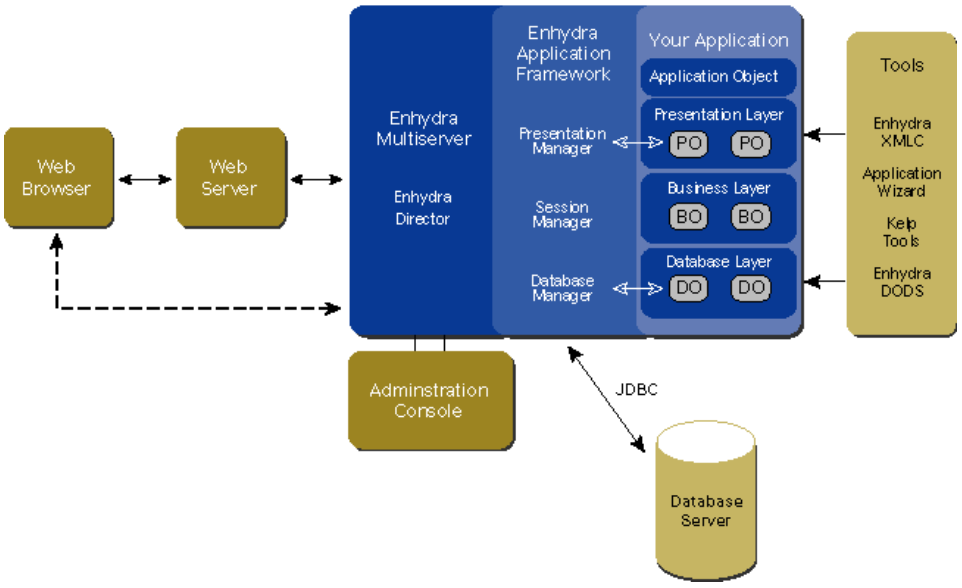


Abbildung 2.4: Architektur des Enhydra-Applikationsservers (aus [Lut00])

Der Ansatz von Enhydra besteht darin, dem Anwender ein fertiges Framework zu liefern, daß ihm alle notwendigen Funktionen bietet, um schnell mehrschichtige, webbasierte Anwendungen in einer wohldefinierten Umgebung zu entwickeln. Das Resultat dieser Bemühungen ist ein komplettes, homogenes System, das intern dem Schichtenmodell folgt und in das die zu entwerfende Anwendung integriert wird. Dabei kapselt der Applikationsserver die Anwendungen und integriert alle Ebenen in einer Laufzeitumgebung und interagiert mit externen Systemen (Web-Server, DBMS usw.), sofern notwendig.

Die Struktur des Applikationsservers orientiert sich an dem üblichen Aufbau von Web-Applikationen und teilt die Laufzeitumgebung in drei Schichten auf, die für die Darstellung, Anwendungslogik bzw. Datenhaltung zuständig sind. In [Lut00] werden diese Ebenen wie folgt charakterisiert:

Presentation Objects contain the logic that presents information to an external source and obtains input from that source. The presentation logic generally provides menues of options to allow the user to navigate through the different parts of an application, and it manipulates the input and output fields on the display device. Frequently the presentation component also perfoms a limited amount of data validation.

Business Objects contain the application logic that governs the business function and process. Business objects are invoked either by a presentation component when a user requests an option, or by another business function. The business functions generally perform some type of data manipulation.

Data Objects contain the logic that interfaces with a data storage system, such as database systems or hierarchical file systems, or with some other type of ex-

> ternal data source such as a data feed or an external application system. Business objects invoke Data Objects to save persistent state.

Diese Aufteilung ermöglicht die Modularisierung der zu entwerfenden Anwendung und vereinfacht so den Entwurfsprozeß. Beispielsweise ließen sich durch verschiedene Darstellungskomponenten (Presentation Objects) verschiedene Sichten auf den gleiche Anwendung realisieren oder unter Verwendung verschiedener Data Objects Informationen in verschiedenen Systemen speichern. Die so erreichte Unbahängigkeit der eigentlichen Anwendung von konkreten Darstellungsformen oder Datenspeichersystemen erhöht die Portabilität, Flexibilität und Wartungsfreundlichkeit der Anwendung.

Wesentlich bei diesem Konzept ist die Zusammenfassung des gesamten Applikationsservers in einem monolithischen System, das auf einem einzelnen Rechner läuft. Durch die Integration aller benötigten Komponenten in einem System bleibt der Einarbeitungsaufwand gering, da keine externen Systeme oder APIs berücksichtigt werden müssen. Dies kommt einem schnellen Entwicklungsprozeß zugute.

Andererseits ist trotz der funktionalen Schichtenaufteilung eine Verteilung auf verschiedene Systeme im Sinne des traditionellen Client/Server-Paradigmas nicht möglich. Außerdem erfolgen alle Anbindungen an externe Systeme ausschließlich über die Datenebene; die Interaktion der Applikationslogik (Business Object) mit anderen Systemen wird nicht direkt unterstützt. Damit wird die Integration des Enhydra in eine bestehende Infrastruktur erschwert.

Zusammenfassend läßt sich feststellen, daß sich der Enhydra-Applikationsserver besonders gut für die Erstellung neuer Web-Anwendungen eignet, die sich in die dreischichtige Architektur aus Daten-, Applikations- und Präsentationsebene einbinden lassen. Durch die Kapselung in ein Laufzeitsystem wird eine homogene Entwicklungsplattform bereitgestellt, die die Entwicklung erleichtert, aber eine Verteilung der Ebenen auf verschiedene Rechner nicht vorsieht.

Damit eignet sich der Applikationsserver gut für die Aufbereitung von Daten für das Internet, etwa in Form von Katalogen oder anderen neuen datenorientierten Diensten. Für den Einsatz in großen, transaktionsbasierten Systemen (beispielsweise E-Commerce-Anwendungen) ist der Server aber wegen seiner eingeschränkten Integrationsfähigkeit weniger geeignet.

Obwohl Enhydra durch die Integration von Entwicklungs- und Laufzeitumgebung zu den vollständigsten Lösungen dieser Art gehört, existieren weitere Systeme, die sich in Bezug auf die Architektur in diese Klasse einordnen lassen: das *Struts-Framework-Project*[10] des Jakarta-Projects, das *Turbine-Framework*[11] der Apache Software Foundation und der *Zope Application Server* von Digital Creations[12] [Teg99], um nur einige Beispiele zu nennen.

[10] `jakarta.apache.org/struts/index.html`
[11] `java.apache.org/turbine/fsd.html`
[12] `www.zope.org`

2.1.4.2 Enterprise Java Beans und die Java 2 Plattform, Enterprise Edition

Für die Realisierung einer Applikationsserverfunktionalität auf der Basis von Java ist Sun Microsystems einen besonderen Weg gegangen, indem die Grundlagen der Architektur und die Beziehungen aller betroffenen Komponenten zunächst analysiert und in einer Spezifikation publiziert wurden, ohne diese an eine konkrete Implementierung zu koppeln[13]. Die Umsetzung der Spezifikation in entsprechende Produkte erfolgte dann durch andere Hersteller, die ihre bereits vorhandenen Systeme auf die Spezifikation anpassten.

Die Spezifikation beschreibt den Aufbau von *Enterprise Java Beans (EJB)*; das sind funktional abgschlossene, austauschbare Software-Komponenten, die typischerweise für die Modellierung von Geschäftsvorgängen (*Business Logic*) im Unternehmen eingesetzt werden. In [Sun00a] wird die EJB-Architektur in der folgenden Weise charakterisiert:

> "The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional and multi-user secure."

Die EJB-Module laufen innerhalb einer Laufzeitumgebung, dem *Container*, der als einheitliche Schnittstelle zu der verwendeten Implementierung fungiert und damit die Austauschbarkeit des EJBs sichert. Außerdem wird die von den EJB bereitgestellte Funktionalität von anderen Softwarebausteinen – den Klienten – genutzt. Ein Klient kann in diesem Zusammenhang ein Anwendungsprogramm im klassischen Sinne oder ein anderes EJB sein, das die Funktionalität im Rahmen einer größeren Anwendung nutzt. Die Beziehungen zwischen den verschiedenen Komponenten werden in der Spezifikation als *Contract*[14] bezeichnet und für die unterschiedlichen Ausprägungen genau spezifiziert (siehe Abbildung 2.5).

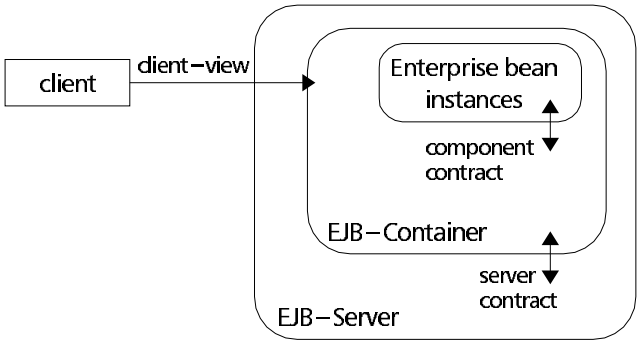


Abbildung 2.5: Beziehungen zwischen den EJB-Komponenten (nach [Sun99b])

[13] Zwar stellt Sun zusammen mit der Spezifikation eine Referenz-Implementierung zur Verfügung, jedoch ist diese nicht für den produktiven Einsatz, sondern als Entwicklungsstudie gedacht.

[14] In diesem Sinne kann ein Contract als eine Menge von Schnittstellen (*Interfaces*) betrachtet werden, die die Kommunikation zwischen Server und Klient definieren.

Ein weiterer Schwerpunkt der Spezifikation ist die Berücksichtigung der einzelnen Rollen im Entwicklungszyklus eines Systems (Entwicklung (*development*), Installation (*deplomyment*) und Administration (*administration*)). Da diese Rollen bei großen Systemen üblicherweise auch innerhalb der einzelnen Phasen von verschiedenen Personen oder Dienstleistern in Anspruch genommen werden, ist hierbei eine genaue Aufgabenverteilung von Vorteil, um eine problemlose Integration der Module in verschiedenste Systeme zu ermöglichen. Tabelle 2.1 gibt in Anlehnung an [Sun00a], Kapitel 3 einen Überblick über die verschiedenen Einsatzszenarien, die im Rahmen der Spezifikation abgedeckt werden.

Rolle	**Erläuterung**
Enterprise Bean Provider	stellt in einem Archiv EJBs bereit, die domänenspezifische Funktionalität enthalten. Die Beans werden in einem Archiv zusammengefaßt und mit strukturellen Informationen versehen, die die Verteilung der Module ermöglichen und Abhängigkeiten zu externen Komponenten festlegen.
Application Assembler	kombiniert die EJBs verschiedener Hersteller zu einer größeren spezifischeren Anwendung, wobei die Funktionalität der Beans in die entstehende Applikation integriert wird. Die Anwendung wird inklusive der enthaltenen Komponenten verteilt und enthält die entsprechenden Deployment-Informationen.
Deployer	sorgt für die Verteilung eines EJBs oder einer Anwendung in einer bestimmten Laufzeitumgebung (Container). Neben dem eigentlichen Installationsprozeß obliegt ihm die Auflösung der externen Referenzen der EBJs und die Erstellung containerspezifischer Schnittstellen zur Verwaltung der Module.
EJB Server Provider, EJB Container Provider	bilden die Ablaufumgebung für die EJBs und stellen die in der Spezifikation festgelegten Dienste bereit. Außerdem liefert der Provider die notwendigen Werkzeuge für die Verwaltung und Verteilung der verschiedenen Beans. Obwohl Container- und Server-Provider nicht notwendigerweise identisch sein müssen, fallen diese Rollen in der Praxis oft zusammen.
Persistence Manager Provider	ist verantwortlich für die Persistenz sogenannter Entity Beans mit Hilfe von Datenbanken oder anderen Systemen zur Datenspeicherung.
System Administrator	sorgt für die Konfiguration und Administration der installierten EJBs mit Hilfe geeigneter Werkzeuge.

Tabelle 2.1: Rollen innerhalb der EJB-Spezifikation

Der Vorteil dieser genauen Rollenverteilung liegt in der exakten Identifikation der Verantwortlichkeiten und Bedürfnisse der beteiligten Parteien und ermöglicht so die gezielte Entwicklung entsprechender Werkzeuge (Tools), die die jeweiligen Entwicklungsphasen unterstützen und stellt in dieser Hinsicht im Vergleich zu anderen verteilten Infrastrukturen wie CORBA oder DCE einen erheblichen Fortschritt dar.

Die für den Anwendungsentwickler interessanten Fähigkeiten sind die vom EJB-Container bereitgestellten Kernfunktionalitäten. Da diese relativ umfangreich sind, muß an dieser Stelle auf eine ausführliche Darstellung aller Funktionen verzichtet werden und auf die entspechenden Spezifikationen verwiesen werden ([Sun99a], [Sun00a], [Sun99b]. Stattdessen werden wir wesentliche Eigenschaften des Systems charakterisieren, die auf das anvisierte Anwendungsspektrum schließen lassen:

Transaktionen: Eine der wichtigsten Funktionen der EJB-Architektur ist die Unterstützung von *verteilten flachen Transaktionen* ([Sun00a], Kapitel 16). Dabei handelt es sich um eine Programmiertechnik, bei der die Datenbearbeitung in einzelne logisch zusammengehörige Abschnitte (Transaktionen) eingeteilt wird. Dabei wird sichergestellt, daß alle Datenzugriffe innerhalb einer solchen Transaktion entweder erfolgreich ausgeführt, oder im Falle eines Fehlers innerhalb eines solchen Abschnittes alle bereits ausgeführten Änderungen rückgängig gemacht werden. Der Anwendungsprogrammierer kann hierbei davon ausgehen, daß alle Datenzugriffe seriell und unbeeinflußt von anderen Benutzern erfolgen. Verteilte Transaktionen erlauben die Kapselung von Datenzugriffen auf meherere unabhängige Systeme (beispielsweise mehrere Datenbanken) in einer einzelnen Transaktion. Für den EJB-Entwickler wird hierbei die Komplexität des verteilten Tranaktionskonzepts verborgen, da diese vollständig von dem EJB-Container übernommen wird. Als zentrale Schnittstelle kommt hierbei die *Java Transaction API* (JTA, siehe [Sun99e]) zum Einsatz, die für die Markierung der Grenzen einer Transaktion innerhalb der Anwendung dient.

Persistenz: Für die Datenspeicherung und den Zugriff auf existente Datenquellen aus DBMS oder anderen bereits vorhandenen proprietären Anwendungssystemen wird von der Architektur ein spezielles System für den einheitlichen objektorientierten Zugriff bereitgestellt. Diese als *Entity-Beans* bezeichneten Module können auf verschiedene Art und Weise auf die Datenquellen zugreifen und dabei verschiedene Persistenzmechanismen benutzen, die entweder vom Bean selbst (*Bean Managed Persistence, BMP*) oder vom Container (*Container Managed Persistence, CMP*) überwacht und gesteuert werden.

Sitzungsunterstützung: Der Zugriff der Klienten auf die EJBs eines Containers, der oft als Sitzung (engl. Session) bezeichnet wird, kann unterschiedlichen Paradigmen folgen, die von der Anwendung abgebildet werden müssen. Im einfachsten Fall kann jeder Klient auf die gleiche Instanz eines EJB zugreifen. Das entsprechende Bean wird in diesem Fall als *zustandsfrei (stateless)* bezeichnet. Im umgekehrten Fall muß für jeden Klienten eine neue Instanz des EJB erzeugt werden, die dann spezifische Daten für diesen Klienten enthal-

ten kann und daher als *zustandsabhängig (stateful)* bezeichnet wird. Die EJB-Architektur unterstützt die beschriebenen Verbindungsparadigmen ([Sun00a], Kapitel 6) und ermöglicht darüber hinaus eine vorübergehende Deaktivierung einer Sitzung (*Passivation*), die zu einem späteren Zeitpunkt fortgesetzt werden kann (*Activation*). Die Sitzungsunterstützung wird durch den EJB-Container realisiert und steht dem Anwendungsentwickler über einen Contract zur Verfügung.

verteilte Verarbeitung und Interoperabilität: Da komplexe Geschäftsanwendungen selten auf ein einzelnenes isoliertes System beschränkt bleiben, ist die Unterstützung einer verteilten Verarbeitung der verschiedenen Module notwendig, und da oft die Interaktion mit bereits existierenden Systemen erforderlich ist, sollte das System den entstehenden Interoperabilitätsforderungen Rechnung tragen. Die aktuelle Spzifikation widmet sich in Kapitel 18 diesem Thema und spezifiziert die zu unterstützenden Technologien. Für die verteilte Verarbeitung wird angeführt, daß diese bereits durch das zum Einsatz kommende Kommunikationsprotokoll (RMI) unterstützt wird und damit in Zusammenarbeit mit einem geeigneten Verzeichnisdienst eine verteilte Verarbeitung auf Objektebene ermöglicht. Die Interoperabilität des EJB-Containers soll mit Hilfe von CORBA sichergestellt werden. Die Spezifikation fordert deshalb zwingend die Unterstützung des IIOP-Protokolls durch den Container und ermöglicht damit die Integration in eine CORBA-Infrastruktur. Die Einbindung der Enterprise Java Beans erfolgt durch eine Abbildung der RMI-basierten Schnittstellen auf IDL (RMI to IDL-Mapping), die in einer separaten API definiert ist ([Sun99c]). Die Spezifikation betrachtet darüberhinaus weitere Funktionen in Bezug auf deren Interoperabilität, beispielsweise Transaktionen, Sicherheitskonzepte und interoperable Namensdienste, und greift dabei meist auf die entsprechenden CORBA-Funktionen zurück.

Sicherheit: Die Implementierung einer geeigneten Sicherheitsstrategie wird bereits in der EJB-Spezifikation festgelegt ([Sun00a], Kapitel 20) und formuliert damit ein Framework, das die einzelnen Komponenten für den Zugriff auf sicherheitsrelevante Funktionen verwenden können. Das Besondere an diesem Konzept ist dabei die Aufteilung der Verantwortlichkeiten für die Sicherung der sensitiven Funktionalitäten auf die einzelnen Rollen im EJB-Entwurfsprozeß.

Der Entwickler der Komponenten (Bean-Provider) hat hierbei im Gegensatz zum traditionellen Ansatz nicht die Aufgabe, die notwendigen Sicherheitsmechanismen oder Zugriffshierarchien zusammen mit der Anwendungslogik zu implementieren, sondern delegiert sie auf andere Ebenen außerhalb der Komponente, was die Entwicklung der Anwendungslogik und deren Anpassbarkeit an veränderte Sicherheitsanforderungen positiv beeinflußt. Die notwendigen Mechanismen werden vom EJB-Container bereitgestellt und fallen damit in den Zuständigkeitsbereich des Container-Providers. Die Realisierung einer geeigneten Sicherheitsstrategie erfolgt bei der Konfiguration der Anwendung oder des Beans in deklarativer Weise in einer Konfigurationsdatei und ist damit Aufgabe des Application-Assemblers und des Deploy-

ment-Providers. Hierzu kommt ein Rollenkonzept (*security roles*) zur Anwendung, das die zur erfolgreichen Ausführung der Anwendung notwendigen Zugriffsrechte in einer Rolle zusammenfasst. Für jede Rolle können deklarativ die Zugriffsrechte auf die Methoden der EJB-Beans (*method permissions*) definiert werden. Diese Aufgabe übernimmt in der Regel der Application-Assembler, während die Zuordnung der verschiedenen Rollen zu den vorhandenen Benutzern dem Deployment-Provider obliegt.

Dieser Ansatz ermöglicht zum einen eine einfache Abdeckung von Standardszenarien mit Hilfe entsprechender Rollen, die durch den Anwendungsentwickler definiert werden und bietet andererseits durch die Festlegung von Zugriffsrechten auf Methodenebene eine sehr differnzierte Einstellung des Sicherheitskonzeptes an, das auch bei der Installation der Anwendung noch angepaßt werden kann. Durch die Entkopplung des Sicherheitskonzepts von der eigentlichen Anwendungslogik wird der Einsatz verschiedenster Sicherheitsstrategien möglich, die von dem verwendeten Container bereitgestellt werden.

Softwareverteilung und -Installation: Durch den modularen Aufbau der EJB-Infrastruktur aus weitgehend unabhängigen Modulen, die zu Anwendungen zusammengefügt und dann in einer Ablaufumgebung installiert werden, ergeben sich besondere Anforderungen an die Verteilung der EJBs und die Installation und Konfiguration der Anwendungen in dem Container. Insbesondere sollte sichergestellt werden, daß das Verhalten der Module in den einzelnen Containern gleich ist und eine leichte Austauschbarkeit der Laufzeitumgebungen verschiedener Hersteller möglich ist. Außerdem beeinflussen die vielfältigen Konfigurationsmöglichkeiten der verwendeten Beans die Anwendung in entscheidender Weise und erfordern deshalb eine zentrale Konfiguration, die zusammen mit der Anwendung installiert und an die spezifischen Bedürfnisse des Anwenders angepaßt wird. Während frühe Versionen der EJB-Spezifikationen diesen Punkt unbetrachtet ließen und damit proprietäre Eigenentwicklungen der Hersteller provozierten, widmet sich die aktuelle Version diesem Problem ausführlich und definiert einen *Deplomen-Descriptor* ([Sun00a], Kapitel 21), der alle notwendigen Konfigurationsinformationen in einer XML-Datei ablegt, deren Format durch die Spezifikation vorgegeben ist. Zusammen mit dem ebenfalls in der EJB-Spezifikation festgelegten Archiv-Format für EJB-Anwendungen ([Sun00a], Kapitel 22) wird die Installation kompletter Anwendungen in verschiedenen Containern verbindlich festgelegt und damit erheblich vereinfacht.

Laufzeitumgebung: Für die Implementierung der Funktionalität der EJBs stellt der Container eine Laufzeitumgebung zur Verfügung, die im wesentlichen aus einer Reihe von APIs besteht. Dabei handelt es sich dabei um die Basis-APIs der Java 2 Plattform und einige sogenannte Standard-Erweiterungen[15], die die Funktionalität der grundlegenden APIs ergänzen. Im einzelnen be-

[15] Diese APIs integrieren sich in die Pakethierarchie der Java-Plattform unterhalb des gemeinsamen Knotens `javax` und erweitern damit die API-Hierarchie der Java-Plattform.

steht die Laufzeitumgebung aus den folgenden APIs, die in [Sun00a], Kapitel 23 beschrieben werden:

- Java 2 Plattform, Standard Edition, v1.3 APIs
- EJB 2.0 Standard Extension
- JDBC 2.0 Standard Extension
- JNDI Standard Extension
- JTA 1.0.1 Standard Extension
- JMS 1.0.2 Standard Extension
- JAXP 1.0

Wesentlich ist neben der Spezifikation der bereitzustellenden APIs auch die Festlegung von Programmierrestriktionen, denen der Entwickler von EJBs unterliegt. Nicht alle Funktionen, die von den APIs angeboten werden, können von dem Entwickler auch benutzt werden[16]. So ist beispielsweise die Erzeugung neuer Threads ebenso untersagt wie der Zugriff auf Dateidienste, die Verwendung von TCP/IP-Sockets oder der grafischen Benutzerschnittstelle (AWT). Diese Einschränkungen wurden hauptsächlich aus Sicherheitsgründen vorgenommen und sollen außerdem verhindern, daß der Container die Kontrolle über die in ihm laufenden EJBs verliert.

Die EJB-Spezifikation liegt seit kurzem in der Version 2.0 [Sun00a] vor, die meisten Produkte implementieren momentan noch die ältere Version 1.1 [Sun99a]. Der Markt für EJB-Container ist noch immer in Entwicklung begriffen und bietet eine Reihe verschiedener Produkte an, ohne daß ein deutlicher Marktführer erkennbar wäre. Interessant ist hierbei oft die unterschiedliche Herkunft der Hersteller[17], die auf die Ausrichung des Produktes schließen läßt und damit dessen Eignung für konkrete Projekte entscheidend bestimmt. Ein Überblick über verfügbaren EJB-Container bieten [Mer00b] und [Mer00a], während [Ste00] verschiedene Entwicklungsumgebungen vorstellt, die die Erstellung von EJB-Komponenten unterstützen.

Wie die überblickartige Darstellung der EJB-basierten Applikationsserver zeigt, konzentrieren sich diese ausschließlich auf die Implementierung der Anwendungslogik, wobei alle Aspekte der Präsentation oder Benutzerinteraktion unberücksichtigt bleiben. Um diese Bedürfnisse abdecken zu können, wird das Konzept der EJBs in eine umfassende Infrastruktur eingebettet, die, basierend auf der Java 2 Plattform, ein Gesamtkonzept für die Entwicklung unternehmensweiter Anwendungen bildet. Diese als *Java 2 Plattform Enterprise Edition* bezeichnete Plattform wird in einer eigenen Spezifikation beschrieben (siehe [Sun99b]) und lagert die typischen Anwendungsschichten in verschiedene Container aus, die miteinander kommunizieren.

[16] Für die Umsetzung dieser Restriktionen wird das Policy-Konzept verwendet, das standardmäßig in der Java 2 Plattform enthalten ist.

[17] Die meisten Anbieter von EJB-basierten Applikationsservern stammen aus dem Datenbankbereich, stellen Object Request Broker her oder bieten Transaktionsmonitore an.

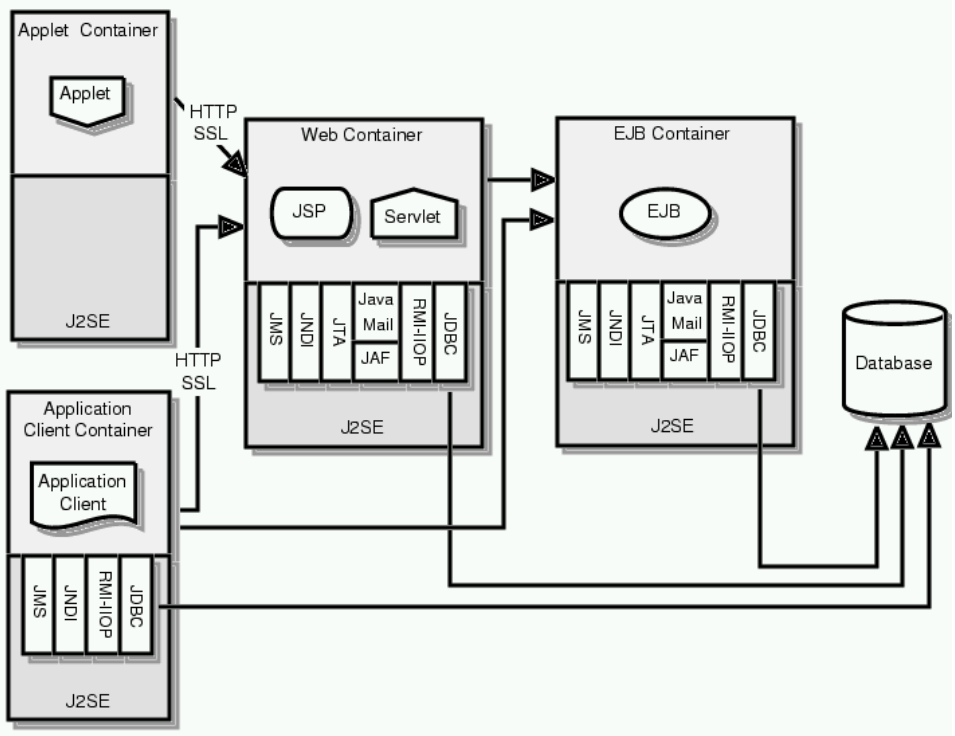


Abbildung 2.6: Aufbau der Java 2 Plattform, Enterprise Edition (aus [Sun99b])

Abbildung 2.6 zeigt diese Aufteilung, bei der der EJB-Container eine zentrale Rolle spielt und die zu realisierende Anwendungslogik implementiert. Für die Interaktion mit dem Benutzer und die Präsentationsebene kommen ebenfalls verschiedene Container zum Einsatz, die auf die verschiedenen Frontends zugeschnitten sind. Wie aus der schematischen Darstellung deutlich wird, ist kann auf die Anwendungen im EJB-Container gleichzeitig auf verschiedene Arten zugegriffen werden, entweder direkt durch eine entsprechende Client-Anwendung oder über einen Web-Container, der die darzustellenden Inhalte für eine Browser-orientierte Darstellung aufbereitet. Die Anwendungen erhalten damit – auf Kosten steigender Komplexität durch die Verwendung weiterer unabhäniger Komponenten und Schichten – ein hohe Flexiblität für alle möglichen Interaktionsformen. Durch die Verwendung verschiedener Container ist weiterhin eine bessere Verteilung der Anwendung möglich, was eine Lastbalancierung stark belasteter Komponenten möglich macht.

Die Applikationsserver auf der Basis von Enterprise Java Beans stellen ein umfangreiches und komplexes Grundgerüst für die Entwicklung transaktionsorienterter, verteilter Geschäftsanwendungen dar, das sich auf eine umfangreiche Spezifikation stützt, deren Entwicklung allerdings noch nicht abgeschlossen ist. Damit werden dem Anwendungsentwickler sowohl wichtige funktionelle Einheiten zur Verfügung gestellt, als auch der Entwicklungsprozeß der Anwendung selbst vom Entwurf bis zur Administration aktiv unterstützt. Dem gegenüber stehen die hohe Einarbeitungszeit in diese Systeme, die aufgrund ihrer Komplexität sehr kosten-

intensiv sind. Die angestrebte, aber momentan immer noch lückenhafte Kompatibilität der einzelnen Produkte macht die Auswahl eines bestimmten Applikationsservers zu einer strategischen Entscheidung, dessen Chancen und Risiken im Einzelfall gegeneinander abgewogen werden müssen (siehe [Pet00]).

2.1.4.3 Abschließende Bemerkungen

Der Markt für Applikationsserver ist noch neu und in Entwicklung befindlich. Obwohl vom Ansatz her nicht völlig neu, verfolgen die Applikationsserverhersteller recht unterschiedliche Strategien, die jeweils die Unterstützung einer bestimmten Klasse von Anwendungen zum Ziel haben. Dies erschwert die Einordnung und den Vergleich der verschiedenen Produkte bzw. deren Evaluierung für den Einsatz in konkreten Projekten.

Die beiden wesentlichen, sich gegenseitig beeinflussenden Faktoren sind hierbei die Breite der unterstützten Anwendungspalette und die Komplexität des Applikationsservers selbst, die mit zunehmender Funktionalität steigt und dem erklärten Ziel der einfachen Anwendbarkeit des Applikationsservers entgegenwirkt.

Die Entwickler solcher Systeme haben hierbei die schwierige Aufgabe, eine überschaubare Menge von Funktionen zu wählen, die einerseits *anwendungsneutral* (universal) sind, d.h. keine spezifische Anwendungslogik enthalten und trotzdem für die anvisierte Gruppe von Applikationen *anwendungsnah* (spezialisiert) sind, d.h. von einer Gruppe von Anwendungen benötigt werden. Wird die Forderung nach Anwendungsneutralität nicht erfüllt, so schränkt dies die Anwendbarkeit des Applikationsservers ein – im Extremfall artet der Applikationsserver in eine reine Anwendung aus. Andererseits sorgt die Forderung nach Anwendungsnähe für eine Spezifik des Applikationsservers für bestimmte Anwendungen, deren Implementierung damit erheblich vereinfacht wird, und die verhindert, daß durch die Unterstützung zu vieler Funktionen eine hohe Komplexität entsteht und die durchschnittliche prozentuale Nutzung der angebotenen Features durch die einzelnen Applikationen sinkt. Im Extremfall entsteht so eine weitere Schicht, die eher dem Betriebssystem zuzuordnen ist.

Die gewonnenen Erkenntnisse aus der allgemeinen Betrachtung von Applikationsservern als Werkzeugkategorie und die beispielhafte Darstellung einiger realer Produkte läßt sich in der folgenden Weise zusammenfassen:

Korrolar 1: *Applikationsserver* sind ein Werkzeug für Anwendungsentwickler, die für eine ausgewählte Menge von potentiellen Anwendungen Funktionen bereitstellen, die bei minimaler Systemkomplexität und maximaler Abstraktion von den verwendeten Basissystemen (Hardware, Betriebssystem usw.) den Forderungen nach Anwendungsneutralität und Anwendungsnähe genügen.

Bei der Analyse verschiedener Applikationsserver stellte sich schnell heraus, daß ein objektiver Vergleich verschiedener Produkte nur in den seltensten Fällen möglich war. Selbst wenn die Funktionalität des Applikationsservers durch eine Spezifikation festgelegt war (wie bei den EJB-Containern), sicherte dies nicht immer eine Vergleichbarkeit der entsprechenden Produkte, da die Implementierung der

Spezifikation nur teilweise umgesetzt wurde oder andere Funktionen zusätzlich von dem Produkt zur Verfügung gestellt werden. Eine direkte Vergleichbarkeit – möglicherweise über die angestrebte Zielgruppe hinaus – erscheint schwierig und wird von den Autoren entsprechender Vergleichstests nur in den seltensten Fällen versucht. Desweiteren fehlt jegliches Maß, das die Eignung eines Applikationsservers für eine bestimmte Anwendungsklasse angibt oder verschiedene Produkte evaluierbar macht. Letztlich bleibt es damit der Praxis überlassen, die Eignung und nachhaltige Nützlichkeit von Applikationsservern zu beweisen.

Trotz dieser Schwierigkeiten bei der objektiven Einordnung verschiedener Applikationsserver halte ich die Entwicklung eines entsprechenden Qualitätsmaßes für die Eignung eines Applikationsservers in einem dedizierten Anwendungsspektrum für notwendig, um für die projektbezogene Evaluierung verschiedener Produkte Anhaltspunkte zu haben, die sich im nachhinein überprüfen lassen. Ein möglicher Ansatz für ein solches allgemeines Bewertungsmaß sieht wie folgt aus:

Korollar 2: Die *Eignung von Applikationsservern für ein Anwendungsspektrum* ist meßbar und drückt sich in dem Verhältnis der durchschnittlich von den Anwendungen des Spektrums genutzten Funktionen und der Anzahl der vom Applikationsserver angebotenen Funktionen aus. Idealerweise wird hierbei ein Verhältnis von 1:1 angestrebt, stark abweichende Werte weisen auf ein Ungleichgewicht zwischen Anwendungsneutralität und -nähe hin.

Da die Qualität des Applikationsservers mit der Anzahl der im Spektrum eingesetzten Anwendungen steigt, ist zusätzlich eine entsprechende Gewichtung des angegebenen Koeffizienten sinnvoll, um einerseits die Eignung eines Servers für verschiedene Spektren zu analysieren und andererseits die Eignung verschiedener Server in unterschiedlichen Spektren zu vergleichen.

Formal läßt sich die Berechnung des Eignungskoeffizienten (u) in der folgenden Art und Weise darstellen, wobei die Anzahl der vom Server angebotenen Funktionen mit fu_{app} notiert wird, während die von den n Anwendungen benutzten Funktionen als fu_i in die Berechnung einfließen:

$$u = \frac{\left(\left(\sum_{i=1\ldots n}^{n} fu_i\right)/n\right)}{fu_{app}} \cdot n$$

Auf der Basis eines solchen Eignungskoeffizienten kann mit Hilfe entsprechender Beispielprogramme oder bereits vorhandener Anwendungen ein Benchmark für spezifische Anwendungsspektren entwickelt und die Einsatzfähigkeit verschiedener Applikationsserver in diesen Bereichen überprüft und bewertet werden. Da eine solche Studie der am Markt befindlichen Produkte jedoch den Rahmen dieser Arbeit sprengen würde, wird dieser Aspekt hier nicht weiter vertieft. Allerdings wird deutlich, daß es in diesem Bereich noch erheblichen Forschungsbedarf gibt; zumindest ist mir derzeit keine Studie bekannt, die die Klasse der Applikationsserver auf die skizzierte allgemeine Art und Weise vergleicht.

2.2 Linguistische Software

Der zu entwickelnde Applikationsserver soll für den Einsatz linguistischer Software zugeschnitten sein. Um diesen Anspruch erfüllen zu können, wird in diesem Abschnitt untersucht, was die Kategorie linguistische Software ausmacht und die speziellen Eigenschaften und Verhaltensweisen ausgearbeitet, die für sprachverarbeitende Software typisch ist.

2.2.1 Entwicklung und Kategorisierung

Die Linguistik bzw. allgemeine Sprachwissenschaft befaßt sich mit der Erforschung der Strukuren der menschlichen Sprache und versucht dieses für den Menschen in der Natur einzigartige Phänomen zu erklären. Traditionell wird hierbei versucht, für die verwendeten Sprachen ein möglichst allgemeines Modell (eine Sprachtheorie) zu finden, das die Struktur einer Sprache sowie ihre Funktionsweise und Gebrauch genau erläutert[18] (siehe [Gün94], Seite 15ff.). Da es sich dabei um einen sehr komplexen Untersuchungsgegenstand handelt, und die Sprache eine Ebenenstruktur aufweist, kommt es zur Unterscheidung verschiedener spezialisierter Teilbereiche (theoretische Sprachwissenschaft, Sozio-, Psycho-, Korpus- und Computerlinguistik usw.) und Sprachebenen (Phonetik, Phonologie, Morphologie, Syntax, Semantik und Pragmatik), die aufeinander aufbauen, aber häufig isoliert voneinander untersucht werden.

Seitdem in der Linguistik Rechentechnik für die Analyse von im wesentlichen geschriebener Sprache verwendet wird, gewinnen Untersuchungen realer Materialien[19] zunehmend an Bedeutung. Dadurch wird die automatische Untersuchung großer Textmengen möglich und erlaubt sowohl die Überprüfung vorhandender Theorien als auch das Auffinden neuer Mechanismen und Zusammenhänge. Die Teilbereiche, die sich hiermit beschäftigen, sind die *Computerlinguistik*, die die Entwicklung und den Einsatz linguistischer Algorithmen und Verfahren für den Einsatz auf Computersystemen betreibt und die *Korpuslinguistik*, die mit der strukturellen und funktionellen Untersuchung von Sprache anhand entsprechender Belegsammlungen (Korpora) befaßt ist und dafür auf den Einsatz moderner Computertechnik angewiesen ist (siehe [Dou98]).

In diesem Kontext entwickelten sich in den letzten Jahren verschiedenste Anwendungen, die hier unter dem Begriff *Linguistische Software* zusammengefaßt werden können. Besonders hervorzuheben ist hierbei, daß die eingesetzten Verfahren oft nicht nur rein linguistisches Wissen einsetzen, sondern auch auf Erkenntnisse anderer Wissenschaftszweige zurückgreifen, um so bessere Ergebnisse zu erreichen. Zusammenfassend läßt sich diese Softwarekategorie in der folgenden Weise definieren:

[18] Im Gegensatz hierzu untersuchen die Einzelphilologien (Germanistik, Slawistik, Anglistik usw.) eine bestimmte Sprache oder Sprachfamilie mit den prinzipiell gleichen Zielen wie die allgemeine Sprachwissenschaft.

[19] Gemeint sind hiermit solche Materialien, die aus Quellen stammen, die die sprachliche Verwendung widerspiegeln (beispielsweise Zeitungstexte) im Gegensatz zu konstruierten Beispielen, wie sie in der linguistischen Grundlagenforschung eingestzt werden.

Linguistische Software *Die Verarbeitung sprachlicher, insbesondere textueller Daten ist Gegenstand linguistischer Software. Für die Analyse dieser Daten werden sowohl Theorien und Erkenntnisse der allgemeinen Sprachwissenschaft bzw. Linguistik und ihrer Teilgebiete, als auch Algorithmen und Verfahren aus der Statistik, Informatik und den entsprechenden Fachbereichen wie der Computerlinguistik herangezogen.*

2.2.2 Eigenschaften und Besonderheiten linguistischer Software

Die besondere Problematik bei der Verarbeitung von Sprache besteht in einer gewissen 'Unschärfe' des untersuchten Objekts, die eine exakte algorithmische Darstellung unmöglich erscheinen läßt[20]. Zwar verfügt Sprache zweifellos über eine detaillierte Strukturierung, die sich über verschiedene aufeinander aufbauende Ebenen erstreckt und von der Sprachwissenschaft seit Jahrhunderten untersucht wird; allein eine widerspruchsfreie Darstellung wichtiger sprachlicher Zusammenhänge ist nach wie vor unklar. Das wird besonders bei dem Versuch deutlich, diese Zusammenhänge automatisch mit Hilfe von Computern zu untersuchen oder als neuen Kommunikationsweg zwischen Mensch und Maschine zu nutzen.

Dies hat dazu geführt, daß sich sprachliche Daten schlecht auf exakte algorithmische Art und Weise bearbeiten lassen und sich somit der traditionellen ingenieurtechnischen Herangehensweise entziehen, die auf der Zerlegung eines komplexen Problems in kleinere handhabbare Teile basiert. Durch dieses Herangehen kann zwar die Erklärung von Phänomenen der einzelnen Teilgebiete gelingen, jedoch kann das sprachlichen Wissens in seiner Gesamtheit nicht vollständig erfaßt werden.

Der Grund hierfür ist im Komplexitätsgrad der Sprache an sich zu suchen. Komplexes Verhalten ist überall dort zu beobachten, wo sich die Funktionsweise eines Systems nicht nur aus der Funktion ihrer Komponenten ergibt, sondern ganz wesentlich auf der Interaktion der Komponenten untereinander basiert[21]. Übertragen auf die Linguistik lassen sich ähnliche Phänomene beobachten, die die Grenzen zwischen den Teilgebieten verwischen und auf komplexes Verhalten schließen lassen. Das folgende Beispiel illustriert exemplarisch diese Zusammenhänge: Die semantische Bedeutung eines Satzes wird klassisch kompositional von der Bedeutung der einzelnen Satzbestandteile abgeleitet, was aber in der Praxis nur selten zum Erfolg führt. Die Gründe hierfür sind vielfältig und liegen u.a. in der Kontextbezogenheit der Äußerung, der Intention des Sprechers und der multiplen Bedeutungsstruktur einzelner Satzkomponenten. Betrachtet man nur die semantische Bedeutung der einzelnen Komponenten und ignoriert andere Einflußfaktoren (Kontextualität und

[20] Zumindest ist derzeit keine allgemeine Theorie bekannt, die in der Lage wäre, das menschliche Sprachverständnis umfassend und widerspruchsfrei zu erklären, geschweige denn nachzubilden.

[21] Komplexe Systeme lassen sich in der Natur häufig beobachten, sind aber in der Regel nicht unmittelbar als solche zu erkennen und noch schwerer algorithmisch zu beschreiben. Sobald diese Wechselwirkungen bekannt sind, wird die Funktionsweise des Gesamtsystens offenbar. Beispielsweise versuchte Craig Raynolds Ende der achziger Jahre das Schwarmverhalten von Vögeln nachzubilden und scheiterte mit einem algorithmischen Ansatz. Erst als er erkannte, daß das beobachtete komplexe Verhalten dadurch entsteht, daß die einzelnen Individuen einfachen Regeln gehorchen, war eine erfolgreiche Simulation des Schwarmverhaltens möglich (siehe [Rey87]).

Allgemeinwissen) oder betrachtet sie als Bestandteil eines anderen Teilgebietes (Intentionen gehören in die Domäne der Pragmatik), so entsteht eine einseitige Überfrachtung der Bedeutungsstruktur einzelner Worte, mit der versucht wird, die entstandenen Defizite auszugleichen. Das Resultat dieser Entwicklung sind komplexe Bedeutungspostulate, die aufgrund ihrer komplizierten Struktur schwer anwendbar sind und trotzdem oft zu widersprüchlichen oder falschen Satzbedeutungen führen. Ein allgemeines maschinelles Sprachverständnis, das über die Bedeutungsanalyse einzelner Wörter hinausgeht, kann damit nicht realisiert werden. Andererseits existieren derzeit noch keine Theorien, die die ganzheitliche Erfassung einer Satzbedeutung erklären können und damit automatisches Textverstehen ermöglichen.

Dieser Tatsache Rechnung tragend beschränkt sich die Untersuchung linguistischer Korpora meist auf die Untersuchung spezieller Phänomene oder Strukturen innerhalb des gegebenen Materials. Dabei wird im Gegensatz zu anderen Gebieten der Informatik aufgrund der oben angeführten Probleme wesentlich weniger Wert auf Vollständigkeit, Widerspruchsfreiheit und deterministisches Verhalten gelegt. Die daraus resultierenden Systeme sind daher meist auch Speziallösungen, die sich meist nur für eine konkrete Aufgabenstellung verwenden lassen und oft auf eine bestimmte Datenbasis festgelegt sind. Dabei sind die entstehenden Systeme oft autark, d.h. nur selten erfolgt der Rückgriff auf etablierte Software für die Lösung von Standardproblemen etwa für das Markup oder die Datenspeicherung, bzw. genügen die Anwendungen den Interoperabilitätsanforderungen, um die Integration in ein größeres Projekt zu ermöglichen.

Linguistische Software beruht also auf zwei wesentlichen Paradigmen, die einander entgegenstehen und – wie oben dargestellt – jeweils ihre spezifischen Vor- und Nachteile haben:

- Der *holistische Ansatz* betrachtet Sprache als komplexes Phänomen, dessen Analyse maßgeblich von der Interaktion einzelner Komponenten bestimmt wird. Dadurch wird die Erkennung von vielfältigen Beziehungen zwischen den einzelnen Komponenten möglich, allerdings erschweren hohe Informationsdichte und fehlende Strukturierung die Erkennung von Wirkprinzipien und die Entwicklung entsprechender Analyseverfahren.

- Der *spezialisierte Ansatz* nutzt die Ebenenstruktur von Sprache für die spezialisierte Analyse einzelner sprachlicher Eigenschaften. Dies sorgt für eine radikale Vereinfachung der zu untersuchenden Strukturen und führt damit schnell zu nachvollziehbaren Ergebnissen. Allerdings besteht die Gefahr, daß durch die Spezialisierung wichtige Zusammenhänge unerkannt bleiben und so die gewonnenen Erkenntnisse im sprachlichen Gesamtsystem keinen Bestand haben.

Eine gemeinsame Eigenschaft beider Ansätze besteht darin, daß keiner der beiden allein erfolgversprechend zu sein scheint. Eine alternative Herangehensweise besteht in der Entwicklung von *datengetriebenen Verfahren*, die in einer Reihe von Teilprozessen versuchen, linguistische Features zu isolieren und gemeinsam mit den sprachlichen Daten zu speichern. Diese dienen dann wiederum anderen Prozessen

als Grundlage für die umfangreichere Analyse solcher Fakten, die auch wieder auf die gespeicherten linguistischen Daten einwirken. Dabei können auf einer Ebene durchaus auch mehrere Verfahren parallel eingesetzt werden, die mit unterschiedlichen Mitteln den gleichen Zweck verfolgen, etwa die Extraktion eines bestimmten grammatischen Merkmals. Die ermittelten Ergebnisse können dann entweder durch ein Entscheidungsverfahren gegeneinander abgewogen werden und nur das vermeintlich beste Ergebnis in der Datenbasis abgelegt werden, oder die Informationen werden redundant gespeichert und stehen übergeordneten Algorithmen zur Verfügung.

Ein solches Verfahren hat den in Abbildung 2.7 dargestellten Aufbau und ist *aggregierend, iterativ und hierarchisch*. Es ermöglicht einen strukturellen Aufbau der linguistischen Informationen zusammen mit den eigentlichen Quellen und gewährt trotzdem allen Ebenen den Zugriff zum gesamten Informationsspektrum.

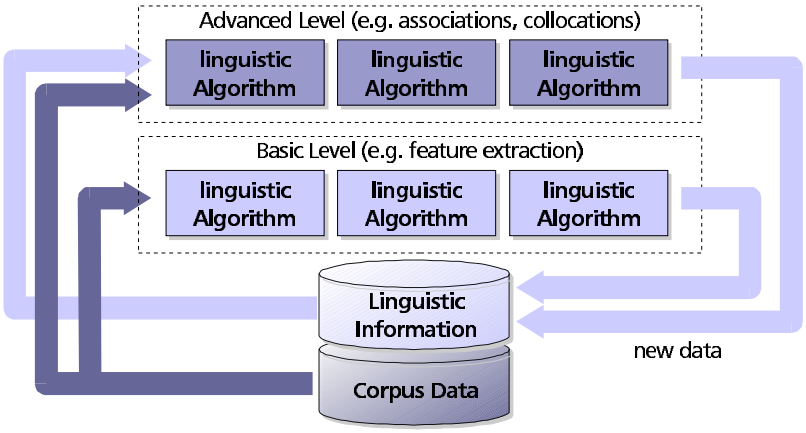


Abbildung 2.7: Aufbau aggregierender, datengetriebener linguistischer Software

Zentraler Bestandteil eines solchen Systems ist die gemeinsam genutzte Datenbasis, die den Anwendungen aller Ebenen zur Verfügung steht und von den Algorithmen verändert werden kann. Die Datenbasis selbst entsteht dabei zunächst aus einem linguistischen Korpus, der wiederum automatisch aus zur Verfügung stehenden textuellen Quellen aufgebaut und ergänzt wird. Im Unterschied zum Ansatz in der Korpuslinguistik geht die Datenbasis über die Idee eines Korpus im klassischen Sinne[22] hinaus, da sie automatisch erstellt ist, mit Ausnahme der Quellenauswahl nicht redaktionell bearbeitet wird und keine feststehende Sammlung ist, sondern durch die verschiedenen Anwendungen ständig erweitert wird.

Durch diesen Prozeß entsteht so im Laufe der Zeit ein linguistischer Wissenspeicher mit sehr heterogenem Inhalt. Zum einen enthält er rein sprachliche Daten, die aus den Textquellen stammen. Außerdem werden durch die verschiedenen linguistischen Werkzeuge Informationen gewonnen und ebenfalls in der Datenbasis ge-

[22] Eine umfangreiche Beschreibung klassischer korpuslinguistischer Verfahren mit Referenzen auf gebräuchliche Korpora findet sich in [Dou98].

speichert. Schließlich enthält die Datenbasis noch einen weiteren Teil, der als nicht expliziertes Wissen bezeichnet werden kann[23]. Darunter verstehe ich sprachliche Informationen, die zwar in der Datenbasis enthalten sind, aber noch nicht durch ein entsprechendes Werkzeug erfaßt und verarbeitet wurde und damit vom Benutzer nicht explizit abrufbar ist. Dieser letzte Teil erscheint zunächst wenig sinnvoll und sorgt für scheinbar unnötige Redundanz, stellt aber sicher, daß zukünftigen Analyseprozessen die kompletten Ausgangsdaten zur Verfügung stehen, um daraus neue Erkenntnisse gewinnen zu können.

Ein wesentliches Merkmal der beschriebenen datenzentrierten Systeme ist das Vorhandensein einer unteren Schranke für das zur Verfügung stehende Material, da viele linguistische Verfahren erst bei ausreichend vielen Belegstellen sinnvolle Ergebnisse liefern. Hingegen ist die Größe der Datensammlung nach oben theoretisch unbegrenzt; meist setzen hier die technischen Möglichkeiten ein Limit. Der Sammlungsprozeß führt außerdem zu einer Konzentration von Daten, was die Zugriffsgeschwindigkeit und die Analysedauer ungünstig beeinflussen kann. Um diesen Engpaß zu umgehen, ist eine Dezentralisierung der gesammelten Daten oder die Verwendung alternativer Speicherungsformen (beispielsweise die Verwendung von speziellen Baumstrukturen anstelle eines relationalen DBMS) denkbar, die für unterschiedliche Anwendungen einen schnellen Datenzugriff ermöglichen.

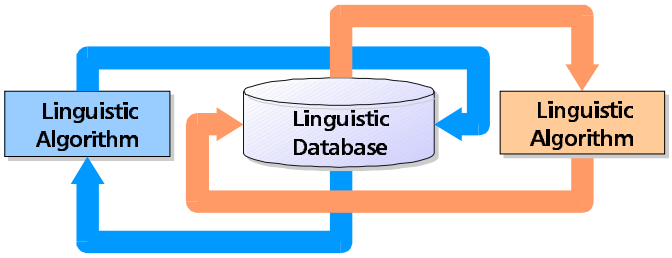


Abbildung 2.8: Analyseverfahren mit Hilfe linguistischer Applikationen

Durch die in Abbildung 2.8 skizzierte iterative und zyklische Anwendung linguistischer Verfahren, die meist durch die Veränderung des Korpus ausgelöst werden, dem neue Daten hinzugefügt werden, erfolgt eine schrittweise Anpassung der in der Datenbasis gespeicherten linguistischen Informationen. Die Verwendung einer gemeinsamen Datenbasis erzeugt außerdem eine implizite Abhängigkeit der verschiedenen Algorithmen zueinander, die zu unerwarteten Ergebnissen führen kann. Obwohl jedes einzelne der verwendeten Verfahren deterministisch ist, kann so der Eindruck entstehen, daß das Gesamtsystem durch die beschriebenen Seiteneffekte ein stochastisches Verhalten zeigt, das erst im nachhinein erklärt werden kann. Dies betrifft insbesondere Informationen, die von Verfahren erzeugt wurden, die auf der Analyse von Daten beruhen, die nicht direkt aus dem Korpus stammen, beispielsweise linguistische Merkmale oder Häufigkeiten, die für die Ermittlung

[23] Vergleicht man dieses linguistische Analyseverfahren mit dem menschlichen Sprachsystem, läßt sich das nicht explizierte Wissen am ehesten mit dem passiven Wortschatz eines Sprechers vergleichen.

von semantischen Relationen zwischen einzelnen Wörtern verwendet werden und durch Unausgewogenheit der Quelldaten verfälscht werden können.

Das folgende Beispiel aus dem Wortschatzprojekt, zu dessen Quellen unter anderem verschiedene Tageszeitungen gehören, illustriert einen solchen Fall:

> Zu den Angeboten von Interliant gehören Lösungen für die folgenden Bereiche: Website-Hosting, Messaging und Wissensmanagement, Sicherheit, E-Commerce, Kunden-beziehungspflege, dezentralisiertes Lernen und Internet-Mietapplikationen via AppsOnline.com.
>
> Zu den Angeboten von Interliant gehören Lösungen für die folgenden Bereiche: Webseiten-Hosting, Messaging und Wissensmanagement, Sicherheit, E-Commerce, Kunden-beziehungspflege, dezentralisiertes Lernen und Internet-Mietapplikationen via AppsOnline.com.
>
> Die Angebote von Interliant schließen Lösungen in den folgenden Bereichen mit ein: Website Hosting, Messaging und Knowledge-Management, Sicherheit, E-Commerce, Kunden-service, Verteiltes Lernen, und Web-basierte Rental-Anwendungen via AppsOnline.com.
>
> Interliants Angebote beinhalten Lösungen in den folgenden Bereichen: Website-Hosting, Nachrichtenübermittlung und Wissensmanagement, E-Commerce, Verwaltung der Kunden-beziehungen, distribuiertes Lernen und Web-Mietapplikationen via AppsOnline.com.
>
> Interliants Angebote beinhalten Lösungen in den folgenden Bereichen: Webseiten-Hosting, Nachrichtenübermittlung und Wissensmanagement, E-Commerce, Verwaltung der Kunden-beziehungen, distribuiertes Lernen und Web-Mietapplikationen via AppsOnline.com.

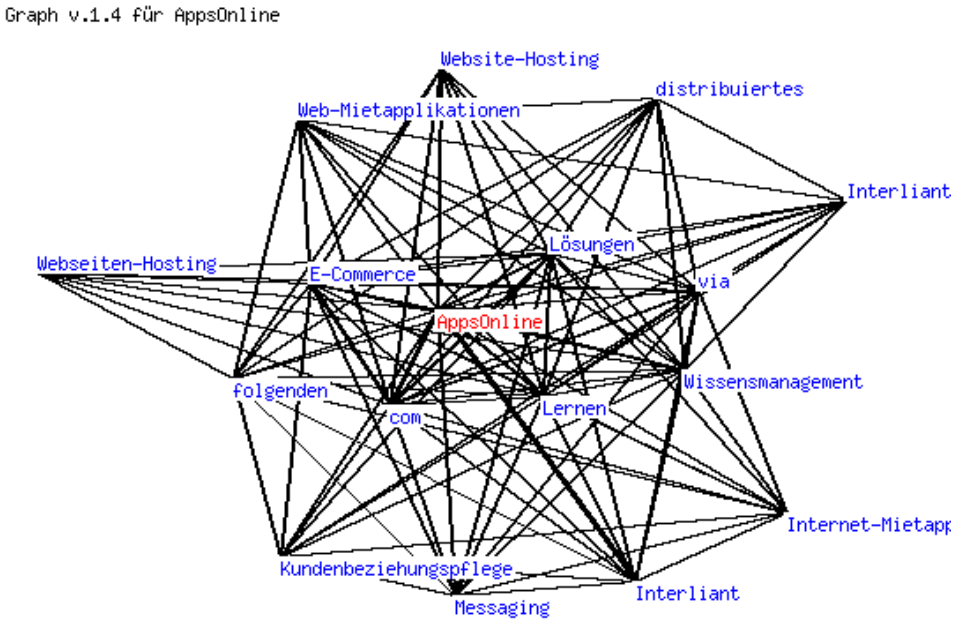


Abbildung 2.9: verfälschter Assoziationsgraph für den Begriff 'AppsOnline'

Hierbei handelt es sich um fünf verschiedene Versionen der gleichen Pressemitteilung, die in verschiedenen Tageszeitungen erschienen ist und von den Redaktionen in ganz unterschiedlicher Weise bearbeitet wurden. Das Resultat dieser Meldungsvarianten im Korpus führt zu einer Verfälschung des Assoziationsgraphen für die in allen Meldungen auftauchenden und bezogen aud den Gesamtkorpus niederfrequenten Begriffe (etwa den Firmennamen 'AppsOnline'). Ursache sind in diesem Fall die inkorrekt ermittelten Häufigkeiten der Begriffe, die durch die Annahme entstehen, es handle sich bei den Beispielen um fünf unterschiedliche Verwendungen, obwohl offensichtlich immer dieselbe Meldung zugrunde liegt. Wie aus dem Assoziationsgraphen für den Begriff 'AppsOnline' in Abbildung 2.9 zu ersehen ist, führt dies u.a. zur Überrepräsentation einzelner Begriffe (vgl. die starke Assoziation zu 'Lösungen', die in allen Meldungen unverändert vorhanden ist), der Aufteilung eines Begriffs auf mehrere Knoten im Graphen (vgl. 'Interliant') und den verschiedenen Übersetzungsvarianten, die im Graphen auftauchen (vgl. 'Website-Hosting' vs. 'Webseiten-Hosting').

An dem angeführten Beispiel läßt sich gut demonstrieren, welchen Einfluß der dynamisch aufgebaute Korpus auf alle nachfolgenden Prozesse hat, andererseits zeigt es auch den durchaus positiven Effekt, daß sich sprachbildende Prozesse (wie etwa die nicht festgelegte Verwendung eines fremdsprachlichen Fachbegriffs) direkt in der Datenbasis widerspiegeln und somit untersucht werden können.

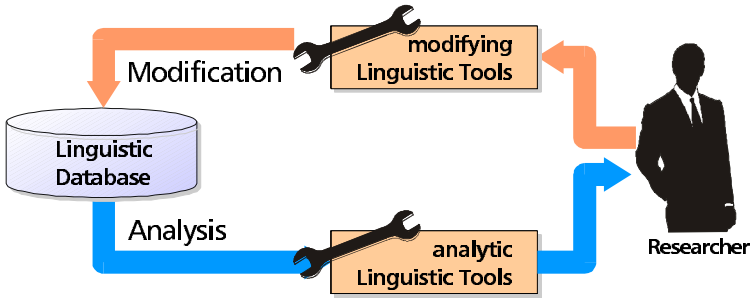


Abbildung 2.10: Analyseverfahren mit Hilfe linguistischer Applikationen

Weiterhin wird deutlich, daß diese Anomalien, die zunächst als zufällige Seiteneffekte entstanden sind, für den Gesamtkorpus nur automatisch mit geeigneten speziell angepaßten Verfahren untersucht werden können (im konkreten Fall müßte ein Verfahren entwickelt werden, daß die ähnlichen Satzbedeutungen trotz der Übersetzungsvarianten erkennen kann). Die anschließend notwendige Veränderung der Datenbasis (beispielsweise eine 'Bereinigung' verfälschter Assoziationsgraphen durch zusammenfassen identischer Knoten) ist ebenfalls nur mit Hilfe geeigneter Werkzeuge möglich.

Der Benutzer des Wortschatzes, insbesondere der interessierte Computerlinguist, wird damit, wie in Abbildung 2.10 dargestellt, direkt in den Kreislauf der Verfahren zum Aufbau des linguistischen Wissensspeichers eingebunden, indem er mit Hilfe geeigneter Analysewerkzeuge die bereits enthaltenen Informationen untersucht und bei Bedarf entsprechend seiner Zielstellung verändert. Aufgrund der großen Datenmenge sind sowohl Analyse als auch Modifikation nur mit Hilfe automati-

scher Verfahren möglich. Da viele linguistische Probleme nur schwer vollständig algorithmisch erfaßbar sind, wird bei dem dargestellten werkzeugunterstütztem Redaktionsprozeß oft ein Teil des Korpus nicht erfaßt werden können, was jedoch tolerierbar ist, solange der Anteil der nicht erfaßten Einträge gegenüber der Größe des Korpus gering bleibt.

2.2.3 Anforderungen an eine linguistische Infrastruktur

Nachdem die Struktur linguistischer Software umrissen und deren allgemeine Eigenschaften in den letzten Abschnitten dargestellt wurde, gilt es nun, in diesem Abschnitt diejenigen Punkte herauszustellen, die für eine Infrastruktur benötigt werden, die diese Klasse von Software unterstützen soll.

Obwohl der Einsatz von Computern in der Linguistik mitterweile üblich ist, wird bei der Entwicklung entsprechender Software oft auf Eigenentwicklungen zurückgegriffen, da es keine Infrastruktur gibt, die die Entwicklung linguistischer Software unterstützt. Dies resultiert oft aus der Tatsache, daß die bearbeiteten Probleme in der Regel so speziell sind, daß die gestellten Aufgaben am besten durch ein eigens hierfür entworfenes Programm gelöst werden. Aufgrund der Spezialisierung der verwendeten Algorithmen ist das Potential für eine unterstützende Infrastruktur gering[24] und beschränkt sich auf die Bereitstellung geeigneter Datenspeicherungsverfahren und eines Kommunikationsprotokolls, falls für eine verteilte Verarbeitung die Kommunikation der Programme untereinander notwendig ist.

Die oben dargestellte datengetriebene Verarbeitung bietet hier durch die notwendige Interaktion mehrerer Algorithmen auf einer gemeinsamen Datenbasis ein weitaus höheres Potential.

Zum einen sind alle Analyseprozesse auf einen Datenzugriff auf die Korpora angewiesen, so daß auf dieser unteren Ebene bereits eine Unterstützung sinnvoll ist, etwa in der Bereitstellung eines oder mehrerer generischer Zugangswege bzw. Schnittstellen zu den Daten. Diese Funktionalität bietet bereits ein herkömmliches DBMS. Allerdings ist für den konkreten Anwendungsfall auch der Zugriff auf mehrere unterschiedlich strukturierte Datenquellen notwendig (flache Dateisysteme, Datenbankdaten, spezialisierte Speicherstrukturen), so daß auf der Datenebene eine Integration dieser verschiedenen Dienste wünschenswert ist. Idealerweise sollte diese Integration so transparent sein, daß die darauf aufgebauenden Anwendungen nicht auf die zu verwendende Struktur angepaßt werden müssen.

Einen weiteren Schwerpunkt bildet die Prozeßhaftigkeit komplexer linguistischer Analyseprozesse, die in der Regel nicht aus einer einzigen Anwendung bestehen, sondern mehrere spezielle Programme in einer Prozeßkette kombinieren, um so das gewünschte Ergebnis zu erhalten. Dabei kann es durchaus zu Iterationen einzelner Programmaufrufe oder ganzer Prozeßketten kommen. Eine Automatisierung verschiedener Analyseprozesse kann für die Qualitätssicherung bzw. -stei-

[24] Im Sinne der oben angegebenen Definition für Applikationsserver (siehe Abschnitt 2.1.4.3 auf Seite 28) ist die Anzahl potentiell gemeinsam genutzter Funktionen gering und bedingt dadurch nur einen geringen Eignungskoeffizienten für einen diese Anwendungsspezifik unterstützenden Applikationsserver.

gerung des Datenbankinhaltes sinnvoll sein und verlangt nach Möglichkeiten zur zyklischen Ausführung der entsprechenden Prozesse bzw. für deren Start aufgrund bestimmter Ereignisse (beispielsweise dem Hinzufügen neuer Daten). Dies erfordert eine Unterstützung auf Anwendungsebene, die von einer entsprechenden Infrastruktur bereitgestellt werden kann. Da ein solcher aus mehreren Komponenten bestehender Analyseprozeß für den Anwender wie eine einzelne Anwendung aussieht, erfüllt die Infrastruktur auch eine Integrationsfunktion auf Anwendungsebene.

Durch die Verwendung mehrerer ineinandergreifender Anwendungen entsteht die Frage nach der Notwendigkeit der Kommunikation dieser Prozesse untereinander und der Unterstützung einer solchen Funktionalität durch die Infrastruktur. Hierfür gibt es verschiedene Möglichkeiten:

- Eine indirekte Art der Kommunikation ist der Datenaustausch über die gemeinsame Datenbasis, bei der die von einer Anwendung generierten Daten als Eingangsdaten für eine weitere Anwendung verwendet werden. Da diese Anwendung wiederum Daten erzeugen oder modifizieren kann, die die erste Anwendung verwendet, ist auch eine bidirektionale Kommunikation über solche Kreislaufprozesse möglich.
- Durch die Prozeßsteuerung kann eine Anwendung Einfluß auf das Ablaufverhalten anderer Komponenten nehmen und übt damit eine Art unidirektionale Kommunikation aus.
- Die direkte Kommunikation verschiedener Anwendung erfordert eine Unterstützung durch die Infrastruktur und ist besonders bei der verteilten Ausführung von Analyseprozessen notwendig, um Daten auszutauschen oder die Ausführung bestimmter Funktionen zu synchronisieren.

2.2.4 Das Wortschatzprojekt als linguistische Software

Das Projekt "Deutscher Wortschatz" besteht aus einer Reihe verschiedenster Werkzeuge zur Bearbeitung und Analyse von sprachlichen Daten, die aus vorhandenen Textquellen extrahiert wurden. Obwohl hauptsächlich die deutsche Sprache untersucht wird, ist eine solche Festlegung nicht zwingend notwendig. Anhand anderer europäischer Sprachen wurde bereits die Anwendbarkeit der verwendeten Analysemethoden überprüft, so daß sich das Projekt schrittweise in einen multilingualen Wortschatz verwandelt. Außerdem wurden unabhängig von der allgemeinen Datenbasis, die Texte aus der deutschen Gegenwartssprache enthält, weitere Datensammlungen angelegt, die spezifische Fachtexte enthalten und für die Extraktion wesentlicher konzeptueller Zusammenhänge der betrachteten Fachgebiete dienen und damit die Recherche in den Fachtexten unterstützen sollen.

Im Hinblick auf die Charakterisierung linguistischer Software ist die Einordnung des Wortschatzprojektes von besonderem Interesse, da der zu konzipierende Applikationsserver insbesondere auf die Bedürfnisse des Projektes zugeschnitten sein soll. Bezogen auf die oben angeführten Kategorien ist das Worschatzprojekt als *datengetriebene Analyse von umfangreichen sprachlichen Daten mit Hilfe automatischer*

Verfahren zu beschreiben, wobei intensiver Gebrauch von korpus- und computerlinguistischen Analyseverfahren gemacht wird. Für eine das Wortschatzprojekt unterstützende Infrastruktur wird also – wie in Abschnitt 2.2.3 dargestellt – eine Unterstützung auf Daten- und Anwendungsebene benötigt, die die ablaufenden linguistischen Prozesse unterstützt und eine Integrationsfunktion hat.

Durch die Verwendung eines Clustersystems aus mehreren Knoten muß die Infrastruktur aber noch eine weitere wichtige Funktion erfüllen: die Realisierung einer Ablaufumgebung, die von der verwendeten verteilten Cluster-Architektur des Systems abstrahiert und für die Anwendungen die gleichen Bedingungen bereitstellt, die diese auf einem einzelnen System vorfinden würden. Dabei sind folgende Schwerpunkte zu unterscheiden:

- *Abstraktion von einer Verteilung auf Datenebene:* Um einen möglichst schnellen Zugriff auf die gespeicherten Informationen zu ermöglichen und die Speicherkapazität der Knoten ausnutzen zu können, werden die Daten auf die verschiedenen Knoten des Clusters verteilt. Dies ermöglicht die parallele Bearbeitung mehrerer Anfragen an die Datenbank und sorgt insbesondere bei aufwendigen Anfragen für eine erhebliche Geschwindigkeitssteigerung.

  Die Daten werden auf den einzelnen Knoten redundant gespeichert. Da jedoch die Größe der Datensammlung immer weiter zunimmt, ist damit zu rechnen, daß die Kapazität eines einzigen Knotens für die Speicherung der gesamten Datenbank nicht mehr ausreicht. In diesem Fall werden nur noch häufig benötigte Daten redundant gespeichert, während die übrigen Daten auf die einzelnen Knoten verteilt werden und nur dort zur Verfügung stehen.

  Aufgabe der Abstraktionsebene ist es, die Verteilung der Anfragen auf die DBMS der einzelnen Knoten zu koordinieren und dabei die Verteilung der Daten auf den einzelnen Clusterknoten zu berücksichtigen.

- *Abstraktion von einer Verteilung auf Anwendungsebene:* Da die Gesamtrechenleistung des Clusters auf die einzelnen Knoten verteilt ist, steht sie einer Anwendung nicht unmittelbar zur Verfügung. Um bei aufwendigen Analyseprozessen eine hohe Auslastung des Clusters zu erreichen, ist eine Verteilung der einzelnen unabhängigen Teilprozesse auf die verfügbaren Knoten notwendig und von der Abstraktionsebene entsprechend einer vorgegebenen Prozeßbeschreibung zu realisieren.

  Für Anwendungen, die nicht für den Einsatz in einer Clusterumgebung entworfen wurden, ist ebenfalls eine Anpassung notwendig, um einen Ablauf in einer solchen verteilten Umgebung zu realisieren. Da diese Anpassungen nicht an der Anwendung selbst durchgeführt werden sollten[25], muß diese Aufgabe ebenfalls von der Abstraktionsebene übernommen werden.

Die Abstraktion von der eingesetzten Clusterarchitektur ist ein wesentlicher Bestandteil der Infrastruktur für die Unterstützung des Wortschatzprojektes, da durch

[25] Wird die Anwendung speziell für den Einsatz auf dem Cluster angepaßt, kann dies erheblichen Entwicklungsaufwand nach sich ziehen, da dadurch zwei Varianten der Anwendung entstehen, die weiterentwickelt und gepflegt werden müssen.

die Einführung einer neuen Ebene eine Transparenz für die Wortschatzanwendungen erzeugt wird, die den Einsatz existierender Anwendung ohne gravierende Änderungen möglich macht und die Entwicklung neuer Applikationen für den Cluster erleichtert.

2.3 Konzeption eines Applikationsserver für linguistischer Systeme

Dieser Abschnitt beschreibt aufbauend auf den in den vorangegangenen Abschnitten ausgearbeiteten Grundlagen die Konzeption eines linguistischen Applikationsservers, der insbesondere die Anwendungen des Wortschatzprojektes in geeigneter Weise unterstützt. Zunächst wird hierfür die zu realisierende Aufgabenstellung präzisiert, bevor dann die Architektur des Systems schrittweise vorgestellt wird.

2.3.1 Einflußfaktoren

Bei der Konzeption eines linguistischen Applikationsservers für das Wortschatzprojekt sind zum einen die in Abschnitt 2.2 angeführten Eigenschaften linguistischer Informationssysteme zu berücksichtigen und andererseits ist die Architektur des Systems so zu wählen, daß es die eingesetzte Cluster-Architektur optimal ausnutzt. Außerdem muß der Tatsache Rechnung getragen werden, daß nahezu die gesamte Datenbasis in einem relationalen Datenbanksystem gespeichert ist, das auf absehbare Zukunft ein Kernbestanteil des Systems bleiben wird, obwohl für spezielle Anwendungsfälle auch alternative Systeme zur Verarbeitung und Speicherung der linguistischen Daten zum Einsatz kommen.

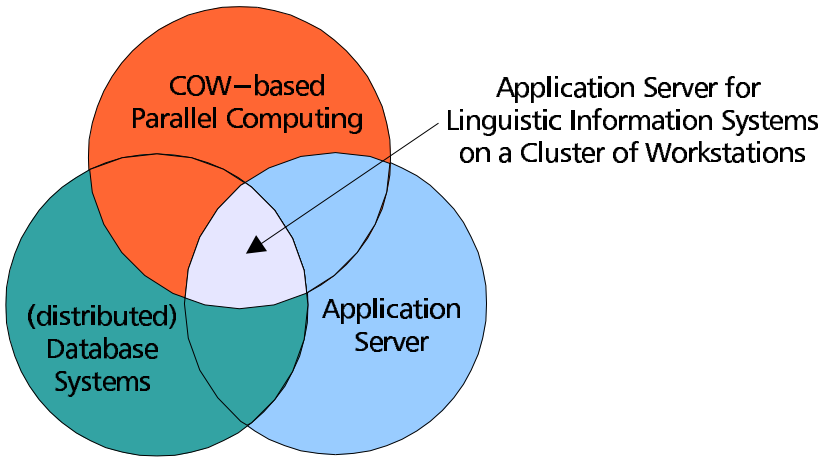


Abbildung 2.11: Applikationsserverkonzept in Relation zu verwandten Technologien

Bei genauerer Betrachtung der Aufgabenstellung fällt auf, daß die zu realisierende Konzeption verschiedene Teilbereiche berührt, aber nicht eindeutig einem bestimmten Gebiet zuzuordnen ist. Dies macht zunächst eine eindeutige Einordnung

der zu lösenden Problemklasse schwierig, da in jedem Teilgebiet wichtige Aspekte der Aufgabenstellung ausgeschlossen bleiben. Für eine klare Positionierung ist daher ein anderer Ansatz notwendig. Charakteristisch für die zu lösende Aufgabe ist die Überschneidung der verschiedene Gebiete, so daß die Aufgabenstellung im Zentrum eines Spannungsfelds zwischen den einzelnen Teildispziplinen angesiedelt ist. Abbildung 2.11 stellt die betroffenen Bereiche zueinander in Beziehung und illustriert den Schwerpunkt der zu realisierenden Funktionalität, die im Schnittpunkt der drei angegebenen Teilgebiete liegt.

Die Verwendung eines Datenbanksystems positioniert die Anwendung zunächst im Datenbanksektor, speziell im Bereich verteilter oder Mehrrechnerdatenbanken. Allerdings ist auch innerhalb dieses Gebiets keine eindeutige Zuordnung möglich: Die Tatsache, daß auf jeder Knoten der Datenbankinhalt vorgehalten wird, die Knoten über ein Netzwerk kommunizieren und keine Ressourcen (Hauptspeicher, Prozessor oder Festplatten) teilen spricht zunächst für ein Mehrrechnerdatenbanksystem auf Shared-Nothing-Basis (Eine Übersicht über die Klassifizierung von Mehrrechnerdatenbanksystemen findet sich in [Rah94] oder [Dad96]). Betrachtet man jedoch den Fakt, daß die Clusterknoten jeweils ein vollständiges DBMS betreiben, das von einer übergeordneten Instanz koordiniert wird und daß außerdem die Relationen der Datenbank nicht fragmentiert[26], sondern tabellenweise vollständig repliziert werden, neigt man dazu das System als förderativ zu klassifizieren ([Con97] widmet sich ausführlich der Beschreibung solcher Systeme).

Der Wunsch, die vorhandene Infrastruktur auch für andere, nicht datenbankzentrierte Anwendungen nutzen zu wollen, verschiebt den Schwerpunkt vom Datenbankserver in Richtung Applikationsserver. Die hierfür benötigte Infrastruktur muß erheblich allgemeiner sein als im Falle eines verteilten DBMS, um eine breite Palette von möglichen Anwendungen unterstützen zu können. In der Regel bieten die einzelnen Systeme hierfür eine Laufzeitumgebung (oft als Container bezeichnet) an, die die Anwendungen kapseln und die notwendigen Funktionen bereitstellen. Wie die Evaluierung verschiedener Systeme gezeigt hat liegt dabei der Schwerpunkt oft auf der Erstellung Web-basierter oder transaktionsorientierter Anwendungen und sind damit nicht für die Anwendung im Wortschatzprojekt geeignet (vgl. Abschnitt 2.1.4 auf Seite 18). Außerdem kommt noch hinzu, daß die Container sich nur bis an die Grenzen eines einzelnen Rechners erstrecken. Zwar ist eine Kommunikation von Anwendungen in verschiedenen Containern meist möglich, für den Einsatz im Cluster wäre jedoch eine Laufzeitumgebung wünschenswert, die von dem Clusterknoten abstrahiert und einen Container auf der Ebene des Clustersystems zur Verfügung stellt.

Die eingesetzte Clusterarchitektur mit einer variablen Anzahl von meist homogenen Knoten macht es notwendig, weitere Aspekte zu berücksichtigen, die typischerweise aus dem Umfeld paralleler Systeme stammen (Beispiel: verteilte Aufgabenbearbeitung mit variablen Synchronisationspunkten). Hierbei ist sowohl der Entwurf des Systems, als auch administrative Fragen von Interesse. Im Rahmen

[26] Unter der Fragmentierung einer Relation versteht man die Aufteilung einer Tabelle auf mehrere Knoten eines verteilten DBMS. Man unterscheidet horizonale Fragmentierung, bei der die Relation ein disjunkte Teilmengen zerlegt wird und vertikale Fragmentierung, die eine Relation spaltenweise zerlegt (siehe [Rah94], S. 59ff.).

des *Beowulf-Projekts*[27] sind hierfür für Cluster auf der Basis von Linux zahlreiche Softwarepakete zur Verwaltung und Administration entwickelt worden. Für die Unterstützung paralleler Programmiertechniken unter Ausnutzung einer Cluster-Architektur existieren ebenfalls frei verfügbare Subsysteme[28]. Allerdings wird hierbei hauptsächlich die verteilte Berechnung aufwendiger numerischer Prozesse betrieben, eine Unterstützung für datenintensive Verfahren, wie sie im Wortschatzprojekt zu finden sind, wird nicht angeboten.

Die dargestellten verschiedenartigen Einflußfaktoren stellen eine Besonderheit des zu konzipierenden Applikationsservers dar und erklären gleichzeitig, warum eine Verwendung eines Standardproduktes, beispielsweise eines parallelen Datenbanksystems oder eines Applikationsservers, keine Alternative darstellt.

2.3.2 Aufbau des konzipierten Applikationsservers

Die Realisierung der Zielstellung unter Berücksichtigung der oben angeführten Einflußfaktoren erfordert eine generische Architektur für den (zu entwickelnden) Applikationsserver, der die verschiedenen Gesichtspunkte in einer funktionalen Einheit kapselt.

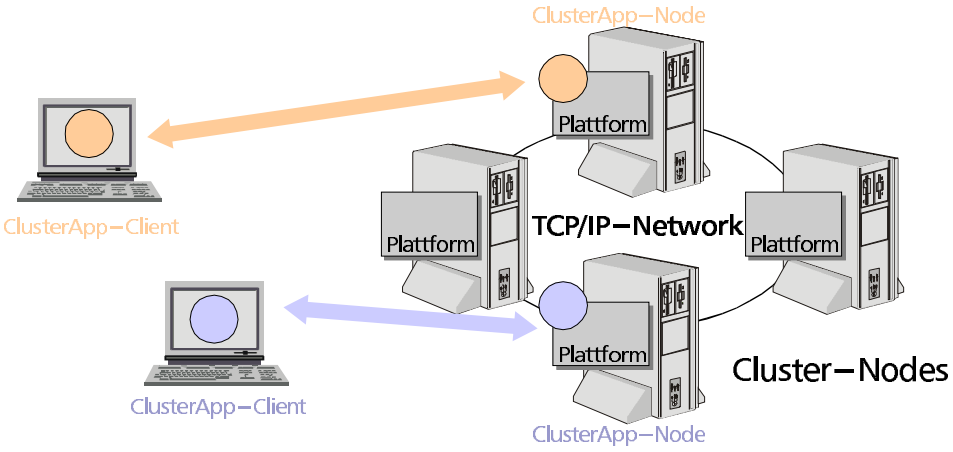


Abbildung 2.12: MetaServer-Plattform als Grundlage für Clusteranwendungen

Der Grundgedanke ist hierbei eine sich über alle Clusterknoten erstreckende *homogene Laufzeitumgebung*, in denen Module eingebettet werden, die die gewünschte Funktionalität implementieren. Die Laufzeitumgebung wird als *MetaServer-Plattform*, die einzelnen Module als *Server* oder *Services* bezeichnet. Wie in Abbildung 2.12 dargestellt, wird die Homogenität der Plattform dadurch erreicht, daß auf jedem Knoten des Clusters eine Instanz des MetaServers läuft und damit überall die gleiche Umgebung zur Verfügung steht. Für die einzelne Anwendung ist es dadurch unerheblich, auf welchem Knoten sie ausgeführt wird.

[27] Weitere Informationen zu diesem Projekt finden sich unter `http://www.beowulf.org`.

[28] Der bekannteste Vertreter dieser Software-Gattung dürfe die *Parallel Virtual Machine*, kurz PVM sein, die unter `http://www.epm.ornl.gov/pvm/pvm_home.html` verfügbar ist.

Diese flache Struktur führt dabei zwar zu hoher Universalität des Konzepts, trägt aber nicht unbedingt zu dessen Übersichtlichkeit bei, insbesondere wenn man beachtet, daß die von der Laufzeitumgebung angebotenen Grundfunktionen ebenfalls als Service implementiert sein können.

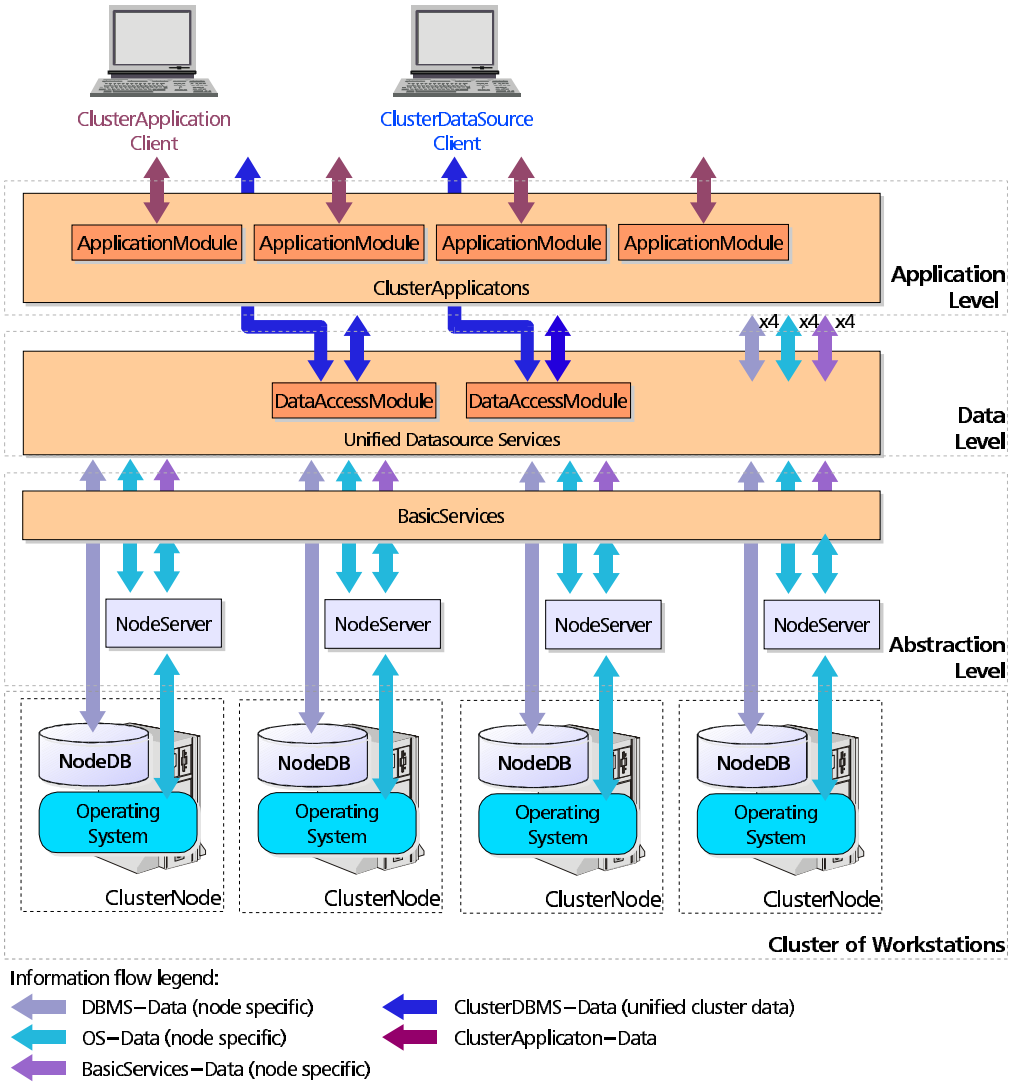


Abbildung 2.13: Schichtenaufbau des konzipierten Applikationsservers

Die Realisierung einer bestimmten Anwendungsfunktionalität ergibt sich üblicherweise aus dem Zusammenwirken mehrerer Module in einer Hierarchie, die wiederum teilweise oder vollständig von einer anderen Anwendung mitgenutzt wird. Für die Breitstellung eines bestimmten Zugangspfades zu den in den Cluster-DBMS gespeicherten Daten wird neben dem Modul, das das zu verwendende Protokoll implementiert, auch noch eine Synchronisationskomponente und ein Scheduler benötigt, die dann zusammen eine Anwendung bilden. Wird ein weiterer Zugangspfad zu den selben Daten gewünscht, so ist nur die Implementierung eines neuen

protokollspezifischen Moduls nötig, während alle anderen Module aus der oben erwähnten Anwendung wiederverwendet werden können.

Die Module können außerdem hinsichtlich ihrer Funktionalität kategegorisiert werden und bilden zusammen mit der angeführten Anwendungshierarchie eine Ebenenstruktur, die in Abbildung 2.13 dargestellt ist und den konzeptionellen Aufbau des Applikationsservers reflektiert:

- Die *Abstraktionsschicht* baut auf der zugrundeliegenden Clusterstruktur auf und abstrahiert von der Funktionalität des einzelnen Knotens. Hierbei werden Informationen von Betriebsystem und DBMS (z. B. Auslastungsfaktoren) extrahiert und den übergeordneten Schichten generisch zur Verfügung gestellt. Weiterhin werden betriebssystemspezifische Funktionen (z. B. Datenübertragungen zwischen Knoten, der Aufruf von externen Programmen) gekapselt und ermöglichen durch diese Abstraktion den Einsatz heterogener Cluster, die verschiedene Betriebssysteme oder DBMS einsetzen.

  Ebenfalls dieser Schicht zugehörig sind verschiedene Basisdienste, die als Module ausgebildet sind und unabhängig von den einzelnen Clusterknoten grundlegende Funktionen zur Verfügung stellen (beispielsweise Verzeichnisdienste, Scheduler, Benutzer-Management usw.) und teilweise von den verwendeten Betriebsmitteln abstrahieren (z. B. die Bereitstellung benutzerspezifischer Verbindungsbündel statt einzelner Verbindungen zum DBMS durch den ClientManager[29]).

- Die *Datenschicht* hat die Aufgabe, den Datenzugriff auf die verschiedenen Datenquellen innerhalb der Clusterstruktur zu realisieren. Dabei kann es zugunsten einer höheren Benutzerakzeptanz notwendig sein,verschiedene Zugangswege zu den gleichen Daten zu realisieren[30] oder verschiedenartige Datenspeicherverfahren zu integrieren und mit einer einheitlichen Schnittstelle zur Verfügung zu stellen.

  Die Datenschicht nutzt außerdem die Basisdienste der Abstraktionsschicht, um die verteilte Datenhaltung im Cluster vor der Client-Anwendung zu verbergen.

- Die *Anwendungsschicht* bildet eine weitere Ebene für Module, die die von dem Applikationsserver bereitgestellte Infrastruktur nutzen bzw. dessen Funktionspektrum erweitern. Dabei wird für die Umsetzung der entsprechenden Funktionalität auf die einzelnen Clusterknoten auf die Basisdienste zurückgegriffen. Die Benutzung der Datenebene ist hierbei möglich, aber nicht notwendig.

  Beispielsweise würde eine clusterfähige Version des Programms zur Erzeugung von Assoziationsgraphen auf der Anwendungsebene angesiedelt werden und die notwendigen Daten über einen von der Datenebene angebo-

[29] Der ClientManager wird in Abschnitt 3.5.6 auf Seite 66 beschrieben

[30] Bei der Umsetzung des Konzepts wurden zwei Zugangswege zu den in einem relationalen Datenbanksystem gespeicherten Daten implementiert, die die Verwendung des proprietären Protokolls oder der JDBC-API ermöglichen.

tenen Zugangsweg extrahieren und damit von der Funktionalität des Clusters Gebrauch machen. Andererseits nutzen andere Anwendungen die Möglichkeiten zur verteilten Verarbeitung im Cluster, indem für ein a priori nicht für den Cluster entworfenes Programm durch eine Anwendung auf Applikationsebene entsprechend gesteuert wird (vgl. BinaryServer).

Der SkriptServer illustriert beispielhaft die Erweiterung der Applikationsserverfunktionalität unter Ausnutzung einer Reihe von Basisdiensten, indem er die verteilte, parallele Ausführung von komplexen Prozessen ermöglicht, die durch eine Prozeßbeschreibung formuliert wurden.

2.3.3 Umsetzung

Der verbleibendene Teil der Arbeit ist der Beschreibung der Implementierung des in diesem Kapitel konzipierten verteilten linguistischen Applikationsservers gewidmet, die den umfangreichsten Teil der vorliegenden Diplomarbeit darstellt. Der Konzeption folgend teilen sich die Erläuterungen zu der vorliegenden Software in zwei Kapitel auf: Das folgende Kapitel 3 beschreibt zunächst den Aufbau und die Funktion der MetaServer-Plattform, die als Grundlage für alle auf der Plattform laufenden Anwendungen dient. In Kapitel 4 werden verschiedene Dienste und Anwendungen dargestellt, die für den Einsatz im Wortschatzprojekt entworfen wurden und die Einsatzfähigkeit des Applikationsservers für linguistische Belange untermauern sollen. Als Gesamtheit betrachtet, stellen diese beiden Hauptkomponenten die Umsetzung der in diesem Kapitel vorgestellten Konzeption dar.

Kapitel 3

Die MetaServer-Plattform — Grundlage für einen verteilten linguistischen Applikationsserver

In diesem Kapitel wird, der vorgeschlagenen Konzeption folgend, zunächst die *MetaServer-Plattform* vorgestellt, die den Kern des implementierten Applikationsservers darstellt. Die Aufgabe dieses Systems ist die Bereitstellung der notwendigen System- und Basisdienste, die unabhängig von der Art der Anwendung benötigt werden. Darüberhinaus ist die Bereitstellung einer flexiblen Laufzeitumgebung für die Anwendungen des Applikationsservers ein weiterer Schwerpunkt, der bei der Implementierung des MetaServers eine wichtige Rolle spielt.

Für die Realisierung einer Plattform, die der obigen Spezifikation genügt, waren zunächst die notwendigen Vorraussetzungen durch den Aufbau einer geeigneten Hardwareumgebung zu schaffen, verbunden mit der Installation der notwendigen Systemsoftware und der Einbindung dieses Systems in die abteilungsinterne Infrastruktur. In Abschnitt 3.1 wird der Aufbau und die Konfiguration der entstandenen Hardware-Architektur beschrieben, während in Abschnitt 3.2 die Kriterien angeführt werden, die für die Auswahl der geeigneten Programmierumgebung ausschlaggebend waren.

Auf diesen Grundlagen aufbauend konnte die Implementierung des Applikationsservers erfolgen, dessen Aufbau in Abschnitt 3.5 beschrieben wird. Nachdem die allgemeine Struktur des MetaServers vorgestellt wurde, wird auf verschiedene Teilaspekte wie die Lastverteilung (Abschnitt 3.5.5), die Konfiguration verschiedender Systemparameter (Abschnitt 3.5.2) und die Benutzerverwaltung (Abschnitt 3.5.6) eingegangen. Die zu der Plattform gehörenden, aber eigenständigen Komponenten Verzeichnisdienst und Logging werden separat erläutert (Abschnitt 3.3 bzw. Abschnitt 3.4). Die Idee einer modularen Architektur betrifft auch die Basisfunktionalität der Plattform und findet ihre Ausprägung in der Implementierung der Funktionen für den Zugriff auf die Cluster-Knoten (3.7) und zur Synchronisation der Dateninhalte der Knoten-DBMS (3.6). Die Vorstellung des für die MetaServer-Plattform realisierten Frontendkonzepts (Abschnitt 3.8) rundet das Ka-

pitel ab und komplettiert die Beschreibung der Grundlagenfunktionalität der Implementierung.

3.1 Aufbau der Clusterumgebung

Um die verteilte Clusterumgebung in der Praxis testen zu können, wurde ein physischer Cluster aus derzeit vier PC-Systemen aufgebaut, die von der Firma *MaxData* geliefert wurden und derzeit als Testumgebung für den entwickelten Applikationsserver dienen. Der gesamte Hardwareaufbau findet auf einem transportablen Geräteträger Platz, der in Abbildung 3.1 dargestellt ist und alle benötigten Hardware-Komponenten inklusive Netzwerk- und Stromversorgung enthält. Nach Abschluß der Test- und Erprobungsphase ist die schrittweise Ablösung der zur Zeit verwendeten zentralen Architektur durch den im Cluster laufenden verteilten linguistischen Applikationsserver vorgesehen, der im Rahmen dieser Arbeit entstand.

Abbildung 3.1: derzeitiger Aufbau für die verwendete Clusterumgebung

Im ersten Schritt wird die Funktionalität der Web-Schnittstelle von dem Cluster übernommen werden, da es sich um eine in Bezug auf die Datenintegrität des Wortschatzes unkritische Anwendung handelt und die erzeugte Last durch diese Anwendung gut auf die einzelnen Knoten verteilt werden kann.

Wenn sich diese Anwendung im Praxiseinsatz bewährt hat, können zu einem späteren Zeitpunkt auch andere Dienste auf den Cluster übertragen werden, die der weiteren Entlastung der zentralen Infrastruktur dienen, etwa die zeit- und datenintensiven Prozesse, die für Indexerstellung und Kollokationsberechnung durchgeführt werden müssen, aber auch Aufgaben niedrigerer Priorität wie die dynamische Erzeugung der Kollokationsgraphen oder die Berechnung von Anagrammen. Der Grad der Integration des Clusters in das Wortschatzprojekt hängt dabei auch stark von der Akzeptanz des angebotenen Applikationsservers durch die Entwickler der Wortschatzanwendungen ab.

Bei den eingesetzten Systemen handelt es sich um baugleiche, handelsübliche PC-Systeme, die zum Zeitpunkt der Anschaffung (Anfang 2000) ein gutes Preis/Leistungsverhältnis boten. Bei der Auswahl der Systeme wurde besonders darauf geachtet, daß kritische Ressourcen (Hauptspeicher und Festplattenkapazität) möglichst flexibel erweiterbar sind. Für die Netzwerkanbindung wurden schnelle Fast-

Ethernet Karten verwendet, die die Knoten in einem sternförmigen Netzwerk über einen Hub an das Universitätsnetz anbinden. Zwischen den einzelnen Knoten kann dabei die volle Geschwindigkeit von 100 MBit/Sekunde verwendet werden, während für die Verbindung zum übrigen Netzwerk auch die Geschwindigkeit von 10 MBit/Sekunde verwendet werden kann, falls keine Fast-Ethernet-Verbindung zur Verfügung stehen sollte. Die technischen Eckdaten der Systeme werden in Tabelle 3.1 zusammengefaßt.

Als Betriebssystem wurde bei allen Knoten Linux eingesetzt, da es sich für den Einsatz als Server-Betriebsystem durch umfangreiche Unterstützung einer Vielzahl von Diensten und gute Administrierbarkeit besonders gut eignet. Darauf aufbauend kommt als Datenbankmanagementsystem MySQL zum Einsatz, das in der verwendeten Version bereits die Aufteilung von Datenbanktabellen auf mehrere physische Dateien im Dateisystem unterstützt und damit die betriebssystemseitig bestehende Beschränkung der maximalen Dateigröße von 2 GB umgeht.

Kenndaten der Hardware:
• 4 MaxData PC-Systeme als Clusterknoten: - Intel Celeron Prozessor, 400 MHz Taktfrequenz - Hauptspeicher 256 MB RAM, erweiterbar bis 768 MB - Maxtor IDE-Festplatte Kapazität 30 GB, Transferrate ca. 15 MB/sec. - Netzwerkkarte 3Com, 100 MBit Fast Ethernet, Twisted Pair - IDE CD-ROM in einem System (für Installationszwecke) • Netzerwerkanbindung der Knoten: - über einen 8fach DualSpeed-Hub von C-Net (10/100 MBit) - 100 Mbit Uplink über einen Switch an das Universitätsnetz
Kenndaten der Systemsoftware:
• Betriebssystem: - RedHat Linux 6.1, Kernel-Version 2.0.36 • Datenbanksystem: - MySQL 3.23.14

Tabelle 3.1: Kenndaten der Hard- und Systemsoftware für den Clusteraufbau

Durch die Ausführung des Clusters als *Shared-Nothing*-Architektur müssen die zur Verfügung stehenden kritischen Ressourcen (insbesondere Hauptspeicher und Festplattenkapazität) in allen Clusterknoten vorhanden sein und können nicht gemeinsam genutzt werden. Die damit erzeugte Redundanz kommt zwar der Unabhängigkeit der einzelnen Systeme zugute, führt aber zu einer möglicherweise ungleichmäßigeren Ressourcenauslastung. Obwohl beispielsweise 1 GB Hauptspeicher und 120 GB Festplattenkapazität im Gesamtsystem zur Verfügung stehen, ist die Ausführung von sehr speicherintensiven Anwendungen nicht möglich, da jedem Kno-

ten nur ein Viertel dieser Kapazität zur Verfügung steht. Eine Möglichkeit, trotz dieser Einschränkung ressourcenintensive Anwendungen ausführen zu können, besteht in einer geeigneten Partitionierung des Problems und der Verteilung der Teilaufgaben auf die Clusterknoten. Die Durchführbarkeit dieses Ansatzes ist jedoch problemabhängig, da die Aufteilung eines Problems in mehrere unabhänige Teilprobleme mit möglichst gleichem Berechnungsaufwand nicht immer realisierbar ist.

Der Vorteil der realisierten Architektur besteht hingegen in der relativ losen Kopplung der einzelnen Knoten, die die Integration weiterer Systeme möglich macht, ohne daß hierbei spezielle Hard- oder Softwareanforderungen (abgesehen von einem installierten DBMS und der Clustersoftware) erfüllt werden müßten. Diese Integration muß außerdem nicht notwendigerweise statisch sein, sondern kann je nach Auslastung der einzelnen Knoten dynamisch erfolgen und so beispielsweise die brachliegenden Ressourcen nichtbenutzter Workstations erschließen.

3.2 Kriterien für die Auswahl der Programmierumgebung

Bei der Auswahl der Programmiersprache für die Implementierung der Clustersoftware war vor allem eine weitgehende Plattformunabhängigkeit ausschlaggebend, da auf dem Clusterknoten möglichst keine Abhängigkeiten zu der verwendeten Systemsoftware (Betriebssystem und DBMS) entstehen sollten[1]. Weitere Gesichtspunkte waren eine möglichst umfangreiche Unterstützung bekannter Technologien und APIs[2], um die Anbindung an bestehende Systeme und Dienste zu erleichtern.

Die Wahl fiel schließlich auf Java[3], eine objektorientierte Sprache, die von Sun Microsystems entworfen wurde und mittlerweile breite Anwendung und Unterstützung findet. Zu den maßgeblichen Vorteilen der Sprache zählen zum einen die Plattformunabhänigkeit, die durch die Übersetzung des Quelltextes in einen plattformneutralen Binärkode erreicht wird, der auf der Zielplattform von einem Laufzeitsystem interpretiert und ausgeführt wird. Außerdem wird mit der Sprache gleichzeitig eine Enwicklungs- (JDK[4]) und eine Laufzeitumgebung (JRE[5]) ausgeliefert, die mit Hilfe umfangreicher APIs viele nützliche Datenstrukturen und Algorithmen zur Verfügung stellt und Anbindungen an viele Dienste bietet, die häufig in verteilten Systemen vorzufinden sind – der einheitliche Zugriff auf Datenbanken mit Hilfe der JDBC-API ist hierbei der sicher am häufigsten genutzte, jedoch nicht der einzige Dienst dieser Art. Eine ausführliche Betrachtung weiterer Aspekte von Java findet sich beispielsweise in [Böh97].

[1] Um die Spezifika der verwendeten Systeme zu verbergen, wurden geeignete Abstraktionsebenen (siehe u.a. *NodeServer*, Abschnitt 3.7) eingeführt, die einen einheitlichen Zugriff auf unterschiedliche Systeme ermöglichen.

[2] API: Application Programming Interface

[3] Die primäre Quelle für Dokumentationen, Software und Weiterentwicklungen ist die Website von Sun Microsystems: `http://www.javasoft.com`

[4] JDK: Java Development Kit

[5] JRE: Java Runtime Environment

Daß die Plattformunabhängigkeit auf Kosten der Ausführungsgeschwindigkeit erkauft werden muß und sich dadurch in einen erheblichen Nachteil verwandelt, ist oft als Schwäche der Java-Plattform angeführt worden und disqualifiziert sie für den Einsatz bei extrem kritischen oder rechenintensiven Aufgaben. Obwohl die die Ausführungsgeschwindigkeit begrenzenden Faktoren oft nicht beim Programm selbst liegen, sondern durch langsame externe Datenspeicher, Netzwerke oder umfangreiche Datenbankzugriffe verursacht werden, stehen mittlerweile Mittel zur Verfügung, die Geschwindigkeit von Java-Programmen erhöhen.

Zunächst hat Sun versucht, die Geschwindigkeit mit Hilfe von *JIT-Compilern*[6] zu steigern, die den Binärkode häufig ausgeführter Programmteile zur Laufzeit in den Maschinenkode der verwendeten Plattform übersetzen. Durch die *Hotspot-Technologie* (siehe [Sun00b, Mel99]) wird diese Technik verfeinert und läßt weitere Geschwindigkeitssteigerungen erwarten. Da diese Übersetzungen transparent innerhalb der Laufzeitumgebung erfolgen, bleibt die Plattformunabhängigkeit des Java-Binärkodes erhalten.

Ein anderer Ansatz beruht darauf, den Java-Quellkode bereits in den Maschinenkode der verwendeten Plattform zu übersetzen (siehe [Red00]). Wie bei herkömmlichen Compilern geht hierbei zwar die Plattformunabhängigkeit des entstehenden Kodes verloren, allerdings erhöht sich die Ausführungsgeschwindigkeit noch einmal beträchtlich, da kein Laufzeitsystem für Interpretation und Optimierung des kompilierten Kodes benötigt wird.

Obwohl die Sprachspezifikation und die JDKs von Sun entwickelt und propagiert wurden, gibt es mittlerweile eine Reihe von alternativen Implementierungen, sowohl von kommerziellen Herstellern, z. B. IBM (siehe [IBM00]), als auch aus der Open-Source-Gemeinschaft ([Tra00, Bla00]). Für das im Cluster verwendete Betriebssystem (Linux) stehen derzeit vier verschiedene Java-Laufzeitumgebungen zur Verfügung, was den Einsatz von Java als Entwicklungsplattform begünstigt.

Der Entwurf verteilter Anwendungen wird von der Java-Plattform durch die RMI-API[7] (siehe [Sun98]) bereits unterstützt, beschränkt sich jedoch auf Java-Programme und erlaubt nicht die Integration von Systemen, die in einer anderen Programmiersprache geschrieben worden sind. Da diese Einschränkung jedoch der Einfachheit und Geschwindigkeit des Systems entgegenkommt und alle verteilten Komponenten der Clustersoftware in Java geschrieben sind, haben wir uns für den Einsatz von RMI entschieden. Sollte dennoch später die Integration in eine universellere verteilte Systemarchitektur wie CORBA[8] notwendig werden, so wird hierfür von Sun ein entsprechender Migrationspfad angeboten, der den Austausch von RMI-Objekten über IIOP[9] ermöglicht. Vergleiche der Geschwindigkeit des RMI-Systems haben gezeigt, daß auch hier Universalität und Flexibilität die Geschwindigkeit des Gesamtsystems beeinträchtigen. Obwohl dies für die Anwendung innerhalb des Clusters unkritisch ist, zeigt die Analyse von J. Maassen, vgl. [Maa99], daß eine alternative Implementierung des RMI-Paradigmas eine ähnliche Geschwindigkeit

[6] JIT-Compiler: Just-in-time-Compiler
[7] RMI: Remote Method Invocation
[8] CORBA: Common Object Request Broker Architecture
[9] IIOP: Internet InterORB Protocol, ist das zwischen *Object-Brokern* in einer CORBA-Infrastruktur verwendete Protokoll zum Austausch von Nachrichten

wie spezielle RPC-Lösungen erreicht und sich damit auch für den Einsatz in parallelen Systemen eignet.

Für die Entwicklung wurde die Version 1.x der Java-Plattform verwendet, da die aktuelle Version 1.3 der Java2-Plattform nicht auf allen Systemen verfügbar ist und die neuen Eigenschaften der aktuellen Version nicht zwingend benötigt wurden. Da die aktuelle Version jedoch abwärtskompatibel ist, ist der Einsatz der Cluster-Software auf allen Java-Plattformen gewährleistet.

3.3 Verzeichnisdienst

Um in einem verteilten System vorhandene Dienste zu identifizieren, ist es notwendig, entweder die Adresse (Hostnamen, Portnummern usw.) zu kennen, unter der der Dienst erreichbar ist, oder einen Verzeichnisdienst (*Registry*) zur Verfügung zu haben, in dem die entsprechenden Ressourcen nachgeschlagen werden können. Da innerhalb der MetaServer-Plattform Dienste mobil und unter Umständen nicht immer verfügbar sind, wurde der Einsatz einer Registry favorisiert. Der Einsatz von Verzeichnisdiensten ist bei verteilten Systemen weit verbreitet. Es existieren eine Reihe verschiedenster Systeme für die unterschiedlichsten Ressourcen und Topologien: LDAP[10], NDS[11], NIS[12], DNS[13] usw.

Da für die verteilten Objekte innerhalb der MetaServer-Plattform hauptsächlich RMI (siehe [Sun98]) eingesetzt wird, bot es sich an, den integrierten einfachen Verzeichnisdienst zu verwenden, der in dem Java-Paket `java.rmi.registry` enthalten und bereits Bestandteil des JDK ist. Leider konnte die Implementierung von Sun nicht verwendet werden, da sie zwei wesentliche Nachteile hatte, die ihren Einsatz verhinderten:

- **nur lokaler Zugriff:** In der Registry können sich nur Dienste registrieren, die auf dem gleichen Rechner wie der Verzeichnisdienst selbst laufen. Um den Dienst für die geplante Architektur einsetzen zu können, wäre es notwendig geworden, auf jedem Clusterknoten eine eigene Registry zu starten und bei jeder Anfrage alle aktiven Verzeichnisdienste nach dem gewünschten Dienst abzusuchen.
- **Persistenzproblem:** Da das zu verwaltende Verzeichnis der Registry nur im Speicher gehalten wird, gehen bei einem Neustart alle Einträge verloren. Da sich die Dienstanbieter üblicherweise nur einmal während des Startvorgangs registrieren, ist auch eine Wiederherstellung der Einträge durch die Registry nicht möglich.

Um das Konzept der Registry in der RMI-Umgebung trotzdem beizubehalten und nicht auf einen externen, meist komplexen Verzeichnisdienst ausweichen zu müs-

[10] LDAP: Lightweight Directory Access Protocol
[11] NDS: Novell Directory Service
[12] NIS: Network Information System
[13] DNS: Domain Name Service

sen, wurde die Registry komplett neu implementiert, um den ersten Nachteil beheben zu können. Die Neuimplementierung verfügt über die gleiche Funktionalität wie die ursprünglichen Version, so daß sie ohne Änderungen an den Anwendungen, die sie benutzen, gegen das Original ersetzt werden kann. Die Erweiterungen betreffen vor allem das neue Sicherheitskonzept, das jetzt die Benutzung des Dienstes netzwerkweit ermöglicht und durch eine Property-Datei weitgehend konfigurierbar ist, und die Verwendung des Syslog-Subsystems für die Ausgabe von Fehlermeldungen. Darüberhinaus implementiert die neue Registry das *Embedded-Server*-Interface und ermöglicht so auch die direkte Integration in die MetaServer-Plattform.

Property-Datei für die Registry der MetaServer-Plattform:

```
rmiPort=3005
logHost=woclu1.informatik.uni-leipzig.de
denyHosts=ALL
allowHosts=localhost.localdomain,                  \
        woclu.informatik.uni-leipzig.de, woclu,    \
        woclu1.informatik.uni-leipzig.de, woclu1, \
        woclu2.informatik.uni-leipzig.de, woclu2, \
        woclu3.informatik.uni-leipzig.de, woclu3, \
        woclu4.informatik.uni-leipzig.de, woclu4
```

Die Auswahl der Systeme, die den Registry-Dienst nutzen können, ist über zwei Eigenschaften definiert, die ein System für die Benutzung des Dienstes qualifizieren (`allowHosts`) oder deren Nutzung verbieten (`denyHosts`). Je nach Konfiguration kann ein System so recht offen (indem allen Systemen der Zugriff erlaubt und nur einzelnen Rechnern der Zugriff verwehrt wird) oder eher restriktiv (bis auf die explizit angegebenen Systeme wird allen anderen Systemem der Zugriff verwehrt) konfiguriert werden, wie im obigen Beispiel angegeben.

Eine Lösung des Persistenzproblems wäre die zyklische Speicherung aller Verzeichniseinträge mit den zugehörigen RMI-Funktionsrümpfen (stubs) auf einem externen Datenträger. Da jedoch die Wiederherstellung der Referenz auf Dienstanbieter problematisch ist, wurde bei der durchgeführten Implementierung ein anderer Ansatz gewählt. Mit der Klasse `cluster.rmi.RegistryManager` wird anderen Anwendungen ein Objekt zur Verfügung gestellt, das die exportierte Schnittstelle eines Dienstanbieters selbstständig unter dem angegebenen Namen in die Registry einträgt und den Eintrag zyklisch überwacht. Hierbei werden alte inaktive Einträge erkannt und überschrieben, während aktive Einträge mit dem gleichen Namen erhalten bleiben und eine Warnung im Logfile des Managers erzeugen. Sollte der Verzeichnisdienst neu gestartet werden oder zeitweise nicht verfügbar sein, so wird der Eintrag von dem RegistryManager aufgefrischt, sobald der Verzeichnisdienst wieder erreichbar ist. Da der RegistryManager in einem eigenen Ausführungsstrang (Thread) läuft und die Intervalle, in denen der Verzeichniseintrag überprüft wird, einstellbar sind, ist sowohl die Prozessor- als auch die Netzwerkbelastung gering.

Der Verzeichnisdienst bietet in seiner gegenwärtigen Implementierung alle für die MetaServer-Plattform notwendigen Funktionen an und genügt den gestellten An-

forderungen. Falls zu einem späteren Zeitpunkt die Anbindung anderer Dienste notwendig werden sollte, bietet sich mit der JNDI-API[14] (siehe [Sun99d]) ein geeigneter Migrationsweg an, da er die bestehende Lösung integrieren kann und gleichzeitig die Einbindung weiterer populärer Verzeichnisdienste erlaubt.

3.4 Logging

Da ein verteiltes System aus einer Reihe mehr oder weniger unabhäniger – meist nicht interaktiver – Komponenten besteht, ist es aus Gründen der Übersicht notwendig, einen systemweiten Dienst zur Aufzeichnung von System- und Fehlermeldungen einzurichten, der den Entwicklungsprozeß unterstützt und später dem Administrator des Systems die Möglichkeit gibt, Probleme zu erkennen, richtig zu diagnostizieren und schließlich zu beheben. Oft ist ein entsprechendes Logfile die einzige Möglichkeit, den Grund für ein Fehlverhalten des Systems im Nachhinein zu erkennen. Da die Zahl der Quellen für Systemnachrichten in einem verteilten System schnell ansteigen kann, ist es wichtig, diese in einem zentralen Punkt zusammenzuführen und zentral zu überwachen. Andererseits soll die Zentralität eines solchen Systems nicht so kritisch sein, daß der Ausfall dieses Subsystems systemweite Auswirkungen hat und daher der Forderung nach ausreichender Redundanz genügt.

Für die Implementierung des Applikationsservers mußten die verschiedenen Alternativen gegeneinander abgewogen werden, die für die Realisierung eines Systems, das den angeführten Bedingungen genügte, zur Auswahl standen: Eine bereits vorhandene eigene Implementierung wurde als unzureichend verworfen und wäre nur mit hohem Aufwand erweiterbar gewesen. Im nächsten Schritt wurden verschiedene, bereits vorhandenen Systeme evaluiert, die im Java-Umfeld zur Verfügung standen und diese Standardaufgabe lösen. (*JavaLog* von Grace Software (siehe [Gra00]) und das *Logging Toolkit for Java* von Chris Barlock (siehe [Chr00])). Leider konnte keines dieser Systeme überzeugen, da durch zu hohe Belastung für die nutzenden Applikationen, fehlende Verteilungseigenschaften oder ungenügende Flexibilität die gestellte Zielstellung nicht erreicht werden konnte.

Schließlich versprach das Logging-System des BSD-UNIX, das heute als *Syslog*-Service auf nahezu allen UNIX-Plattformen verwendet wird, genau die benötigten Eigenschaften zu besitzen: es handelt sich um ein verteiltes System zur zentralen Erfassung von Systemnachrichten verschiedenster Dienste vom Betriebssystem-Kernel bis zum Anwendungsprogramm, wobei die Nachrichten nach verschiedenen Kriterien zentral sortiert und in verschiedene Logdateien geschrieben werden, deren Aufbau in Abbildung 3.2a dargestellt ist.

Die Kommunikation zwischen den Anwendungen und dem Subsystem erfolgt hierbei lokal über Systemaufrufe oder das Netzwerk mit Hilfe einer Datagramm-orientierten Verbindung (UDP). Das Subsystem versieht die Systemmeldungen mit einem Zeitstempel und dem Namen des meldenden Prozesses und ordnet sie nach einer festgelegten Kategorisierung einer bestimmten Logdatei zu. Die Logdateien

[14] JNDI: Java Naming and Directory Interface

müssen nicht unbedingt lokal auf dem jeweiligen System abgelegt werden, sondern können auch an einen anderen Rechner mit Syslog-System weitergeleitet werden, wodurch ein zentrales Logfile entsteht, daß die Meldungen verschiedener Systeme zusammenfaßt. Sollte dies ausfallen oder nicht mehr über das Netzwerk erreichbar sein, so bleiben die einzelnen Systeme immer noch lokal verfügbar und können die Systemmeldungen lokal aufzeichnen.

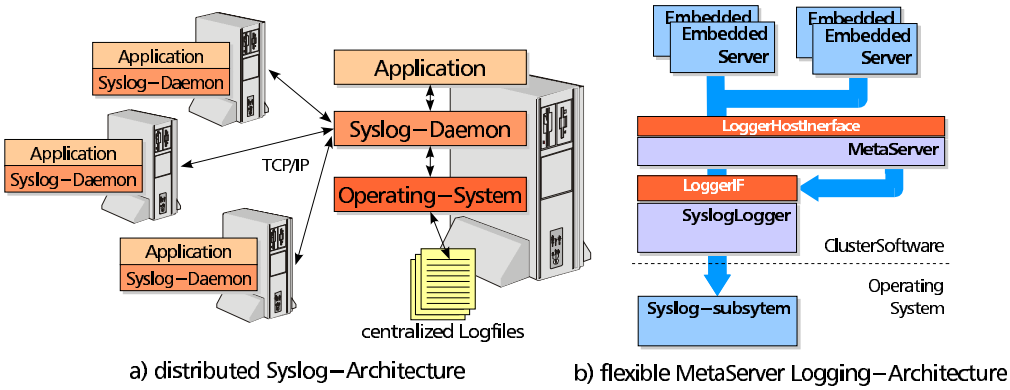


Abbildung 3.2: Syslog-Subsystems und Integration in die MetaServer-Plattform

Der einzige Nachteil des Systems scheint in der fehlenden Verfügbarkeit des Syslog-Subsystems auf nicht-UNIX-Plattformen zu liegen, die beispielsweise die Verwendung von Microsoft™-Betriebssystemen ausschließen, da hier ein anderes System zum Loggen von Systemnachrichten verwendet wird. Allerdings läßt sich dieser vermeintliche Nachteil durch den Einsatz entsprechender serverseitiger Komponenten von Drittanbietern kompensieren, und da sogar eine freie Java-Implementierung eines Syslog-Servers verfügbar ist (siehe [Tim00]), ist der Dienst auf allen Plattformen verfügbar, die diese Programmiersprache unterstützen. Um aber trotzdem von dem verwendeten Logging-Subsystem unabhängig zu sein, wurde bei der Implementierung durch die Verwendung entsprechender Interfaces von dem zugrundeliegenden System abstrahiert und damit dessen Austauschbarkeit gewährleistet. Wie in Abbildung 3.2b dargestellt, wird die Loggingfunktionalität von einem eigenen Modul, dem `SyslogLogger` implementiert, das die Syslog-Client-Objekte von Tim Endres ([Tim00]) verwendet, um auf das Syslog-Subsystem zugreifen zu können. Die Schnittstelle zu der MetaServerPlattform wird durch das `LoggerInterface` definiert, über das der Zugriff auf die von dem Logger bereitgestellten Funktionen geregelt wird. Um auch den einzelnen Anwendungen des Servers ein Logging zu ermöglichen, implementiert der MetaServer das `LoggerHostInterface`, welches für alle EmbeddedServer des Systems einen einheitlichen Zugriff auf das zentrale Logging-System ermöglicht.

Damit realisiert die Implementierung ein flexibles Logging-System, welches allen Komponenten zur Verfügung steht und auf transparente Weise an das Syslog-Subsystem angebunden ist, das für die Aggregation der Systemnachrichten an einer zentralen Stelle zuständig ist. Sollte sich das Syslog-System ändern oder gegen ein anderes System ausgetauscht werden, so erfordert dies nur eine Neuimplementierung eines neuen Logging-Moduls, das das `LoggerInterface` implementiert.

Außerdem können spezielle Funktionen, die das Subsystem nicht direkt anbietet, ebenfalls in dem Logger-Modul untergebracht werden[15].

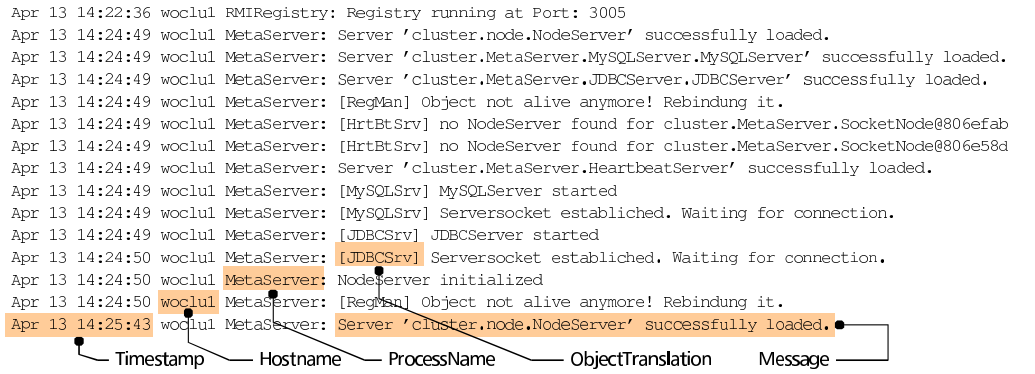


Abbildung 3.3: Ausschnitt der Syslog-Meldungen der MetaServer-Plattform

Die Abbildung 3.3 zeigt einen Ausschnitt aus der Log-Datei des MetaServers und illustriert die verschiedenen Komponenten der Einträge. Der Zeitstempel wird ebenso wie der Hostname automatisch beim Eintreffen generiert, während der Name des Prozesses von der jeweiligen Anwendung vorgegeben werden kann. Da dies im Falles des MetaServers noch keine ausreichende Diskriminierung der Nachrichtenquelle ermöglicht, kann von dem Logging-Modul noch zusätzlich eine Identifikation des Prozesses eingefügt werden. Dies wird als *ObjectTranslation* bezeichnet, da eine Objektinstanz in eine aussagefähige Beschreibung umgeformt wird.

3.5 Die MetaServer-Plattform

Die MetaServer-Plattform ist der Kern des implementierten Applikationsservers und stellt grundlegende Dienste und Funktionen zur Verfügung, die unter dem Begriff *Basisdienste* (*Services*) subsumiert werden. Aufbauend auf dieser Grundlage wird die Plattform von verschiedenen *Nutzdiensten* (*Applications*) als Laufzeitumgebung verwendet, um eine anwendungsspezifische Funktionalität auf der Clusterarchitektur zu realisieren.

Der folgende Abschnitt beschreibt die Architektur der Plattform, während die sich anschließenden Abschnitte wesentliche Komponenten des MetaServers detailliert erläutern. Dabei handelt es sich in Abschnitt 3.5.2 um den *PropertiesManager*, den *Mobility-* und *HeartbeatServer* in Abschnitt 3.5.3 bzw. Abschnitt 3.5.4 sowie um den *ClientManager*, dem Abschnitt 3.5.6 gewidmet ist. Auf den *Scheduler* und die zur Ressourcenverteilung verwendeten Algorithmen wird in Abschnitt 3.5.5 eingegangen. Die einzelnen Nutzdienste werden in diesem Kapitel nur erwähnt und in Beziehung zu den Basisdiensten gesetzt. Die ausführliche Beschreibung der einzelnen Komponenten erfolgt in Kapitel 4 auf Seite 84ff.

[15]Ein Beispiel für einen solchen Fall sind die dynamisch eingefügten Objektsignaturen innerhalb einer Systemnachricht, die diese dem entsprechenden Subprozeß zuordnen.

3.5.1 Architektur

Um ihre Aufgabe erfüllen zu können, muß die MetaServer-Plattform alle Ressourcen und Funktionen (*Basisdienste*) bereitstellen, die von den verschiedenen Nutzanwendungen (*Applications*) benötigt werden. Die Plattform selbst sollte hierbei so flexibel sein, daß sowohl die Integration weiterer Nutzdienste, als auch eine funktionale Erweiterung der Plattformfunktionalität selbst möglich ist. Abbildung 3.4 zeigt den Aufbau der MetaServer-Plattform, mit der dieses Ziel erreicht werden soll.

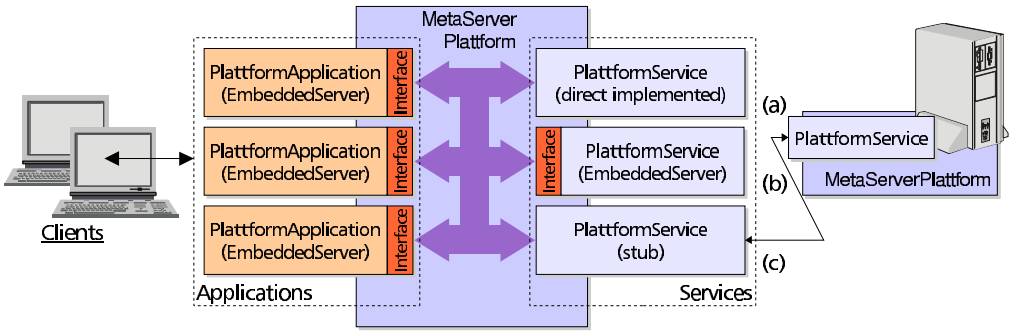


Abbildung 3.4: Architektur der MetaServer-Plattform

Um die flexible Integration der Nutzdienste zu gewährleisten, wurde eine Schnittstelle (*ServerInterface*) definiert, über die die Dienste mit der Plattform interagieren können. Über ein weiteres Interface (*EmbeddedServer*) wird der Zugriff der Plattform auf die Dienste[16] definiert und steuert damit deren Ausführung, die in einem eigenen Thread erfolgt und in einem speziellen Objekt (`ServerThread`) gekapselt ist.

Tabelle 3.2 enthält die Aufstellung der verschiedenen Nutz- und Basisdienste, die in der Implementierung verwendet werden, zusammen mit einer kurzen Beschreibung. In den folgenden Abschnitten werden diese Kompomenten noch genauer erläutert werden.

Bei der Implementierung der Basisdienste ergeben sich verschiedene Möglichkeiten, unter denen für die entsprechenden Funktionen jeweils eine geeignete Alternative ausgewählt werden muß.

Die einfachste Variante (siehe Abbildung 3.4a) ist die direkte Implementierung der gewünschten Funktionalität innerhalb der Plattform selbst, die für essentiell notwendige Funktionen verwendet wird. Dem Vorteil der ständigen Verfügbarkeit stehen erhöhte Komplexität der Plattform, fehlende Mobilität der Funktion und eine möglicherweise nicht gewünschte Redundanz entgegen, wenn mehrere MetaServer-Plattformen innerhalb der Umgebung laufen. Der PropertiesManager, das Logging-Modul und der Zugriff auf die Registry sind Beispiele für diese Implementierungsvariante.

[16] Aufgrund ihrer Einbettung in die Plattformarchitektur des MetaServers werden sowohl Nutz- als auch Basisdienste teilweise als *EmbeddedServer* bezeichnet.

Nutzdienste	Basisdienste
MySQLServer: Zugriff auf die Knoten-DBMS für eine Vielzahl von MySQL-Anwendungen durch Verwendung des proprietären Protokolls *JDBCServer:* Zugriff auf die Knoten-DBMS für Java-Applikationen durch Verwendung der JDBC-API und eines entsprechenden Treibers *BinaryServer:* Unterstützung von monolithischen Anwendungen, die keine Client/Server-Architektur aufweisen, bei der verteilten Ausführung auf dem Cluster. *ScriptServer:* Bereitstellung eines Subsystems zur verteilten Verarbeitung batchorientierter Anwendungen	*NodeServer:* Bereitstellung einer einheitlichen Zugriffsebene auf das Betriebssystem und das DBMS der einzelnen Knoten und Realisierung knotenspezifischer Dienste *SynchroServer:* Synchronisation von modifizierenden DB-Zugriffen unter den Clusterknoten, inklusive Replikationsverfahren *Scheduler:* Verteilung aller anfallenden Tasks der Nutzdienste über die zur Verfügung stehenden Clusterknoten *HeartbeatServer:* zyklische Überwachung der Verfügbarkeit der einzelnen Knoten und der dort laufenden Dienste *MobilityServer:* Transport von Serverdiensten zwischen verschiendenen MetaServer-Plattformen

Tabelle 3.2: Einordnung der verschiedenen *Embedded Server* in Basis- und Nutzdienste

Wie in Abbildung 3.4b dargestellt, kann ein Basisdienst auch als EmbeddedServer implementiert sein und unterscheidet sich damit nur dadurch von den Nutzdiensten, daß er für Anwendungen außerhalb des Clusters nicht direkt zur Verfügung steht. Beispiele für die Anwendung dieses Paradigmas sind der Mobility- und der HeartbeatServer, aber auch hybride Dienste wie die Registry oder der NodeServer, die sowohl innerhalb der Plattform als auch selbstständig lauffähig sind. Durch die Verwendung von Schnittstellen wird die Funktionalität von der eigentlichen Plattform entkoppelt und trägt damit zur besseren Strukturierung des Gesamtsystems bei. Außerdem sind die so implementierten Dienste besser konfigurierbar (sie können beispielsweise bei Bedarf ge- oder entladen werden) und potentiell mobil.

Eine letzte Variante ist der in Abbildung 3.4c dargestellte Einsatz von Funktionsrümpfen (stubs), die statt der eigentlichen Basisdienste verwendet werden und in Verbindung mit einem Basisdienst stehen, der auf einer anderen MetaServer-Plattform läuft und die benötigte Funktionalität bereitstellt. Diese Alternative findet hauptsächlich bei Basisdiensten Verwendung, die nur einmal im Cluster laufen sollen, also beispielsweise dem Scheduler oder dem SynchroServer.

3.5.2 PropertiesManager

Um eine hohe Flexibilität der MetaServer-Plattform zu gewährleisten, sind eine Reihe von Eigenschaften in einer zentralen Konfigurationsdatei einstellbar. Darüber hinaus verfügen auch die verschiedenen EmbeddedServer, die innerhalb der Plattform laufen, über Eigenschaften, die einstellbar sind und über die Konfigurationsdatei gesteuert werden, die das folgende Aussehen hat:

Ausschnitt der zentralen Property-Datei für "woclu2":

```
# ----- General properties, shared among all components ---
logHost=woclu1.informatik.uni-leipzig.de
Registry.host=woclu1.informatik.uni-leipzig.de
Registry.port=3005
# ----- Properties for the MetaServer ----------------------
Server.name=ExperimenatalMetaServer
Server.subServers=\
        cluster.node.NodeServer, \
        cluster.MetaServer.SynchroServer.SynchroServer, \
        cluster.MetaServer.MySQLServer.MySQLServer, \
        cluster.MetaServer.JDBCServer.JDBCServer, \
        cluster.MetaServer.HeartbeatServer
        cluster.MetaServer.MobilityServer
Server.DBPoolHosts= woclu1, woclu2, woclu3, woclu4

# Properties for the MobilityServer
MobilityServer.name=MobilityServer
MobilityServer.port=3006
# ----- Properties for the MySQLServer ---------------------
MySQLServer.name=NativeMySQLServer
MySQLServer.port=4000
MySQLServer.maxConnections=10
MySQLServer.DBPoolServerLogin=metaserver
MySQLServer.DBPoolServerPassword=*************
MySQLServer.clusterUserPrefix=C
MySQLServer.clusterUserPassword=*************
```

Für die zentrale Konfigurierbarkeit ist der *PropertyManager* verantwortlich, der auf der Grundlage von *Properties* arbeitet, einer Standard-Java-Klasse, die eine Liste aus Attribut-Wert-Paaren verwaltet und über den Attributnamen einen Zugriff auf die entsprechenden Werte gestattet. Eine Vorbelegung der Attribute mit Standardwerten (default-values) ist ebenso möglich wie das Einlesen von Properties aus Konfigurationsdateien.

Der PropertyManager steht unmittelbar nach der Initialisierung der Plattform allen Komponenten zur Verfügung und kann für die Konfiguration der einstellbaren Parameter verwendet werden. Während der Initialisierung werden die Standardwerte für alle wichtigen Properties gesetzt bzw. automatisch ermittelt (wie der Hostname der Plattform, der zwar von den Komponenten als Eigenschaft abgefragt, jedoch nicht statisch in der Konfigurationsdatei festgehalten wird).

Für einen komfortablen Zugriff auf die Werte unterstützt der PropertyManager nicht nur den Datentyp `String`, sondern darüber hinaus auch die Verwendung von `Boolean`-, `Integer`- oder `Float`-Werten, indem die notwendigen Konvertierungen automatisch durch den PropertiesManager erfolgen. Um die Fehlerbehandlung zu erleichtern, werden alle auftretenden Ausnahmen in einer eigenen Klasse (`PropertyManagerException`) zusammengefaßt, die von der Anwendung ausgewertet werden kann.

3.5.3 MobilityServer

Um den Transfer von eingebetteten Servern zwischen mehreren MetaServer-Plattformen zu gewährleisten, steht der *MobilityServer* zur Verfügung. Die dadurch erreichte Mobilität der Plattform-Anwendungen ist eine administrative Fähigkeit, die die Verlagerung von Applikationen zwischen den verschiedenen Clusterknoten während des laufenden Betriebes ermöglichen soll und darf nicht mit den weitaus flexibleren Mobilitätseigenschaften von Agentensystemen verwechselt werden.

Sofern ein innerhalb der Plattform laufender Server als mobil gekennzeichnet ist, kann der MobilityServer mit Hilfe eines Management-Frontends veranlasst werden, ihn auf eine andere Plattform zu transferieren. Hierfür wird der Server auf der aktuellen Plattform gestoppt und dann in serialisierter Form mit Hilfe einer TCP-Verbindung über das Netzwerk an den MobilityServer der Zielplattform übertragen. Dort wird der Server wieder deserialisiert, einem Serverthread zugeordnet und gestartet. Hierbei erhalten alle serverinternen nicht transienten Variablen ihren alten Wert, während die Umgebung des Servers mit den Werten der neuen Plattform initialisiert wird. Dies betrifft beispielsweise die vom PropertyManager der neuen Plattform bereitgestellten Werte der konfigurierbaren Parameter sowie die Referenzen zu den entsprechenden Plattformdiensten und kann dazu führen, daß sich der transferierte Dienst auf der neuen Plattform anders verhält als auf der Quellplattform.

Bei dem beschriebenen Verfahren wird die Methode der Objektserialisierung verwendet, die von der Java-Umgebung ab Version 1.1.x bereitgestellt wird. Dabei handelt es sich um ein Verfahren, mit dem eine Instanz eines als serialisierbar gekennzeichneten Objekts[17] nach einem festgelegten Protokoll in eine Binärform umgewandelt (*Serialisierung*), oder aus dieser wieder eine Objektinstanz erzeugt werden kann (*Deserialisierung*). In Verbindung mit Datenströmen (*Streams*) kann dieses Verfahren verwendet werden, um Objekte dauerhaft zu speichern oder in einem Netzwerk zu übertragen.

[17] Diese Kennzeichnung erfolgt durch Implementierung des methodenfreien Interfaces `Serializable`

3.5.4 HeartbeatServer

Der HeartbeatServer ist ein Bestandteil der MetaServer-Plattform, der als nichtmobiler EmbeddedServer realisiert ist. Er hat die Aufgabe, die vorhandenen Clusterknoten auf ihre Verfügbarkeit hin zu überwachen und jede Veränderung an die Dienste, die diesen Knoten benutzen, weiterzugeben. In der derzeitigen Konfiguration ist dies insbesondere der Scheduler (siehe Abschnitt 3.5.5), der die Verteilung von Tasks auf die einzelnen Knoten regelt, und der SynchroServer (siehe Abschnitt 3.6), der für die konsistente Replikation der Datenbankinhalte zuständig ist.

Der Server überprüft die Knoten zyklisch in einem konfigurierbaren Intervall und stellt die Ergebnisse über eine Schnittstelle anderen Anwendungen zur Verfügung. Für die Ermittlung der notwendigen Knoteninformationen greift der HeartbeatServer auf die *NodeServer* der einzelnen Knoten zurück. Momentan werden nur die Datenbanken der Knoten auf Erreichbarkeit geprüft, obwohl der NodeServer auch andere Daten zur Verfügung stellt.

Die Anwendungen können die Verfügbarkeit der Knoten mit einem exportierten Methodenaufruf bei Bedarf direkt abfragen, oder durch Anwendung des *Observer-Patterns* automatisch über jede Statusänderung der Knoten informiert werden.

Das *Observer-Pattern* gehört zu den Entwurfsmustern (*Software-Pattern*), bei denen es sich um häufig verwendete Interaktionen von Objekten handelt, die immer einem bestimmten Muster (*pattern*) folgen. Dieses mittlerweile weit verbreitete Entwicklungsparadigma der objektorientierten Programmierung ist in [Gam94] sowohl allgemein, als auch anhand konkreter Anwendungsfälle, ausführlich beschrieben.

Bei dem hier verwendeten Muster überwacht eine Reihe von Objekten (die *Observer*) den Status eines einzelnen Objekts (des *Observable Objects*). Statt zyklisch den Status abzufragen, registrieren sich die Observer bei dem beobachteten Objekt und dieses benachrichtigt die Beobachter, sobald eine Statusänderung stattgefunden hat.

3.5.5 Scheduler

Der Scheduler ist ein zentraler Bestandteil der MetaServer-Architektur und hat die Aufgabe, die Verteilung der *Tasks* der verschiedenen Dienste auf die einzelnen Knoten zu koordinieren. Bei den Tasks kann es sich hierbei um ganz verschiedene Aufgaben handeln, z. B. eine Anfrage, die von einem MySQL-MetaServer bearbeitet wird, aber auch eine Replikationsanforderung, die vom SynchroServer angefordert wird. Für die Abarbeitung der Tasks müssen ihnen Clusterknoten zugewiesen werden, wobei darauf zu achten ist, daß alle Knoten möglichst gleichmäßig ausgelastet sind und eventuelle Abhängigkeiten konkurrierender Tasks berücksichtigt werden.

Die Betriebsmittelvergabe (Scheduling) ist ein umfangreicher und komplexer Forschungsgegenstand, der seit langem in verschiedenen Bereichen innerhalb der Informatik und auch in anderen Wissenschaftsgebieten (z. B. den Wirtschaftswissenschaften) bearbeitet wird und zu dem es bereits eine Vielzahl von Veröffentlichun-

gen und Vorschlägen gegeben hat (einen Überblick über verschiedene Scheduling-Aspekte, speziell im Datenbankenbereich enthält [The99]). Aufgrund der zahlreichen unterschiedlichen Anwendungen und den Anforderungen an ihr Verhalten unter Last bzw. bei Betriebsmittelknappheit ist hier auch in Zukunft nicht mit einer allgemeingültigen Lösung zu rechnen.

Im konkreten Anwendungfall der MetaServer-Plattform kommt hinzu, daß die Betriebsmittelvergabe oberhalb anderer komplexer Systeme (Betriebssystem, DBMS) operiert und es sich darüber hinaus um ein verteiltes System handelt, das heißt, daß die Anwendungen auf verschiedenen Rechnern laufen, die über ein lokales Netzwerk verbunden sind – einem weiteren Betriebsmittel, das Einfluß auf eine optimale Lastverteilung hat, aber nicht gezielt beeinflußt werden kann. Weiterhin ist maßgeblich, daß die Struktur der Anwendungen, die die MetaServer-Architektur nutzen, im Hinblick auf ihre Anforderungen sehr unterschiedlich ist und somit ein variables Anpassen des Schedulers je nach Anwendungsfall erforderlich macht.

Da der Scheduler von allen angebotenen Diensten der Serverplattform in Anspruch genommen werden muß, ist es wichtig, die Zeit, die für das Scheduling eines Tasks benötgt wird, zu minimieren, da sie maßgeblich die Performanz des gesamten Clusters beeinflußt.

3.5.5.1 Die Scheduling-Umgebung

Für die Architektur des Schedulers ist es hierbei wichtig, daß der Entwurf ausreichend universell ist, um eine Verteilung unterschiedlichster Tasks durchführen zu können. Hierbei sollte je nach Art der Anwendung zwischen den zur Auswahl stehenden Verfahren der am besten geeignete Algorithmus ausgewählt werden. Eine weitere Eigenschaft dieser Umgebung ist die Tatsache, daß nicht immer alle Ressourcen allen Tasks zugeordnet werden sollen (ein Benutzer soll möglicherweise nur auf eine Teilmenge der verfügbaren Clusterknoten zugreifen können). Dies muß vom verwendeten Schedulingverfahren berücksichtigt werden.

Der Ansatz der Implementierung besteht deshalb zunächst darin, eine Umgebung zu schaffen, in die verschiedene Scheduling-Algorithmen eingebettet und je nach Bedarf verwendet werden können. Die Entwicklung und der Test geeigneter Verfahren für die einzelnen Anwendungen ist dabei für den Aufbau eines effizienten Systems wichtig. Wie dies in der vorliegenden Implementierung umgesetzt wurde, zeigt Abbildung 3.5.

Die in unserem Fall zu verteilenden Betriebsmittel (Ressourcen) sind die zur Verfügung stehenden Cluster-Knoten, die durch ein *Node*-Objekt innerhalb des Schedulers modelliert werden und eine Referenz auf den *NodeServer* des entsprechenden Clusterknotens enthalten, über den es möglich ist, aktuelle Lastinformationen abzurufen. Die Verfügbarkeit aller Knoten wird ständig vom *HeartbeatServer* (siehe Abschnitt 3.5.4) überwacht und jede Veränderung an den Scheduler übermittelt. Bei der Zuordnung der Ressourcen zu den Tasks ist aus der Menge der für diesen Task verfügbaren Cluster-Knoten derjenige auszuwählen, der die gestellte Aufgabe am besten erfüllen kann.

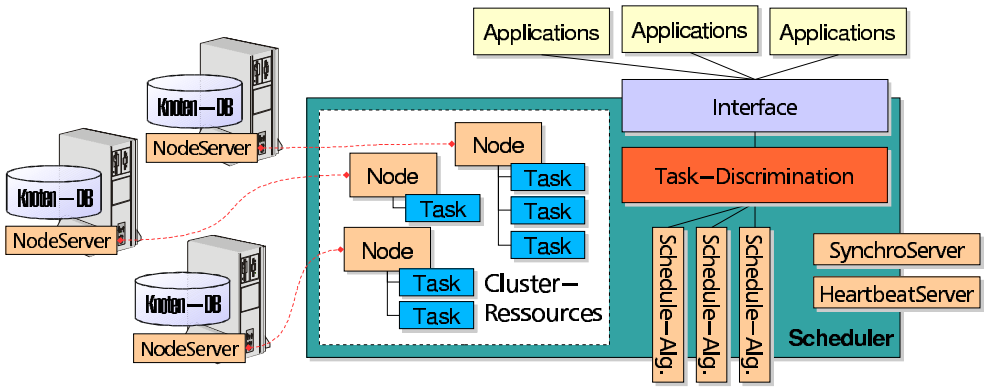


Abbildung 3.5: Aufbau der Schedulerumgebung

Hierfür ist eine Beschreibung des Tasks notwendig. Diese Beschreibung ist als eigenes Objekt (*TaskDescription*) ausgebildet und enthält eine Typklassifizierung sowie allgemeine Informationen zu dem Task. Diese Informationen sind fakultativ, werden in Stringform übergeben und unterliegen keiner weiteren Maßgabe (es könnte sich hierbei z. B. um die SQL-Anfrage handeln, wenn die Anforderung an den Scheduler vom MySQL-MetaServer kommt). Während die Klassifikation der schnellen Identifikation des Anfragetyps dient, werden die restlichen Informationen bei der Auswertung innerhalb der einzelnen Schedulingstrategien verwendet. Wenn es notwendig sein sollte, applikationsbedingt weitere Informationen in der Beschreibung unterzubringen, so kann dies mit Hilfe geeigneter Spezialisierungen des *TaskDescription*-Objekts erfolgen. Für den Fall des MySQL-MetaServers wurde dies genutzt, um in der Task-Beschreibung Informationen über die Knoten zu übermitteln, auf denen der Benutzer angemeldet ist.

Die Kommunikation zwischen den Anwendungen und dem Scheduler erfolgt über ein Interface und besteht im wesentlichen aus vier Schritten, die in Abbildung 3.6 dargestellt sind.

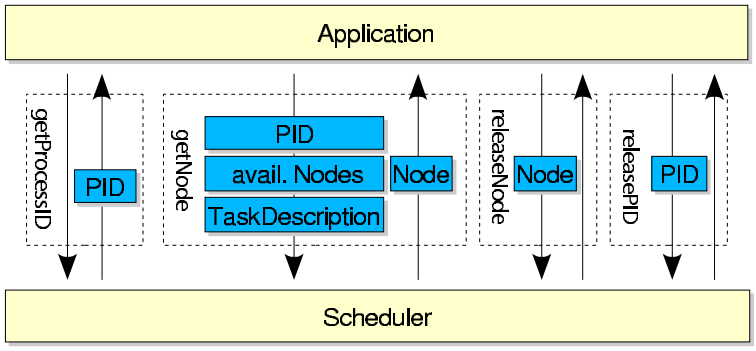


Abbildung 3.6: Interaktion zwischen Applikation und Scheduler

- **Schritt I:** Jede Applikation, die den Scheduler benutzen will, muß sich zunächst mittels der `getProcessID()`-Methode registrieren und erhält dar-

aufhin eine Identifikationsnummer (PID), die bei jeder weiteren Anfrage an den Scheduler angegeben werden muß.

- **Schritt II:** Benötigt eine Applikation für die Abarbeitung einen Clusterknoten, so fordert sie diesen mit Hilfe der Methode `getNode()` an und beschreibt in einer *TaskDescription* die Spezifika des auszuführenden Tasks. Innerhalb des Schedulers wird hierauf anhand der Beschreibung ein entsprechender Clusterknoten ausgewählt und als Referenz an die Applikation zurückgegeben. Gleichzeitig wird ein internes *Task*-Objekt erzeugt, das dem ausführenden Knoten zusammen mit der PID der Applikation zugeordnet wird.
- **Schritt III:** Sobald die Applikation den Clusterknoten nicht mehr benötigt, sollte sie ihn mit Hilfe der Methode `releaseNode()` wieder freigeben. Bei der Freigabe wird die Zuordnung des Tasks zu dem Knoten wieder aufgehoben und steht damit der Vergabe erneut zur Verfügung.
- **Schritt IV:** Wenn die Applikation ihren Dienst einstellt, so sollte sie dies dem Scheduler durch Aufruf der Methode `releaseProcessID()` mitteilen, der dann auch alle durch diesen Prozeß belegten Knoten wieder freigibt.

Bei diesem System der Zuteilung der einzelnen Clusterknoten basiert die Zusammenarbeit zwischen den einzelnen Applikationen und dem Scheduler auf der Annahme gegenseitiger Kooperation, d.h. der Scheduler vertraut darauf, daß die Applikationen einen Knoten nur so lange beanspruchen wie unbedingt notwendig und daß die Freigaben von Knotenobjekten und PIDs auch erfolgen. Sollte dies nicht geschehen, hat der Scheduler zunächst keine Möglichkeit, die Zuteilung einer Ressource zu überwachen oder rückgängig zu machen. Dieser Ansatz wurde gewählt, da ein exklusiver Zugriff auf die Clusterknoten über den Scheduler nicht realisierbar ist, da die Dienste der Clusterknoten prinzipiell allen Plattformanwendungen zur Verfügung stehen.

3.5.5.2 Scheduling-Verfahren

In der implementierten Umgebung können verschiedene Scheduling-Algorithmen eingebunden werden, die je nach Applikation alternativ eingesetzt werden können. Welche Strategie hierbei die effektivste ist, entscheiden viele Parameter, die durch die Struktur des Clusters und die verwendeten Applikationen vorgegeben werden. In der gegenwärtigen Implementierung sorgt ein einfaches statisches Verfahren für die Verteilung der Tasks auf die einzelnen Knoten; im produktiven Einsatz ist hier aber eine auf die verwendeten Applikationen angepasste Implementierung vorzusehen.

Die zahlreichen Scheduling-Verfahren können in die folgenden Gruppen eingeordnet werden, die nachfolgend beschrieben werden: Die *statischen* Algorithmen sind meist einfach implementierbar und eignen sich für homogene Systeme, in denen eine Ausgewogenheit von Ressourcen und Tasks herrscht. *Dynamische* Systeme hingegen versuchen, auf Last- und Leistungunterschiede im System zu reagieren und ein optimales Schedule zu finden und sich auf wechselnde Bedingungen anzupassen. Eine weitere Untergruppe bilden die *problemorientierten* Verfahren, die durch

die Verwendung von problemspezifischem Wissen ein besseres dynamisches Verhalten zeigen.

Statische Verfahren

Die einfachsten Verfahren beruhen auf einer statischen Verteilung der Ressource an die einzelnen Verbraucher, die sich während der Ausführung des Algorithmus nicht ändert.

Eine solches Verfahren ist das *Round-Robin Scheduling*, bei dem alle Clusterknoten als gleichwertig in Bezug auf Leistungsfähigkeit und momentane Belastung angesehen und die Tasks in einer festen Reihenfolge gleichmäßig auf die einzelnen Knoten verteilt werden[18]. Da die speziellen Eigenschaften eines Knotens nicht berücksichtigt werden, kann es hierbei bei großen Leistungsunterschieden der einzelnen Knoten oder bei unterschiedlichen Task-Laufzeiten zu einer ungleichmäßigen Auslastung des Clusters kommen. Eine Implementierung dieses Verfahrens ist jedoch einfach, da auf keine externen Parameter Rücksicht genommen werden muß.

Um der unterschiedlichen Leistungsfähigkeit einzelner Knoten Rechnung zu tragen, kann man als Variante eine Gewichtung der einzelnen Knoten vornehmen und so unter Ausnutzung der Leistungsunterschiede eine bessere Performanz des Gesamtsystems erreichen. Hierfür wird jedem Knoten ein ganzzahliges positives Gewicht zugeordnet. Die feste Reihenfolge bei der Zuteilung bleibt erhalten, jedoch erhält jeder Knoten maximal so viele Tasks, wie sein Gewicht angibt. Die Umlaufdauer eines Schedules ergibt sich bei einer Knotenanzahl von n damit zu $W = \sum_{i=1}^{n} w_i$, wobei die Verteilung der Tasks auf den Knoten i dem Verhältnis w_i/W entspricht.

Beispiel: Angenommen, der Cluster besteht aus vier Knoten A, B, C, D, die mit den Gewichten 4, 3, 2, 2 versehen sind. Bei einer festen Durchlaufreihenfolge von $A - D$ ergibt sich damit eine Durchlauflänge von 11 und der folgende Schedule: $ABCDABCDABA$. Damit werden die Tasks zu ca. 36% auf A, zu ca. 27% auf B und jeweils zu ca. 18% auf C und D verteilt.

Mittlerweile hat sich das Round-Robin Scheduling zu einer ganzen Familie von Verfahren entwickelt, die jeweils für verschiedene Anwendungsfälle optimiert sind. Neben einem Überblick stellt [Hem99] eine Reihe solcher Optimierungen vor.

Dynamische Verfahren

Wenn das Verhalten des Schedulers nicht durch eine bestimmte Konfiguration festgelegt ist, sondern anhand von Laufzeitparametern bestimmt wird, so handelt es sich um ein dynamisches Scheduling-Verfahren. Ein eher einfaches Verfahren, das

[18] Der Begriff 'Round Robin' stammt übrigens aus der Piraterie des 18. Jahrhunderts. Bei Meutereien auf Segelschiffen wurden die verschworenen Mannschaftsmitglieder in einer Liste geführt, und um die Anführer des Aufstandes im Falle eines Verrats nicht identifizieren zu können, wurden die Namen kreisförmig notiert, da man fürchtete, die übliche Listenform würde die Rangfolge der Meuterer erkennen lassen.

in diese Klasse einzuordnen ist, ist das *Queue Length-Scheduling*, das für die Verteilung der Tasks auf die Knoten die Anzahl der bereits auf dem Knoten laufenden Tasks in Betracht zieht. Wenn bei einer Knotenanzahl von n Knoten die Anzahl der bereits auf dem Knoten i laufenden Tasks (die Länge der Task-Warteschlange) mit l_i gegeben ist, so wird der nächsten Task demjenigen Knoten zugeordnet, dessen Warteschlange am kürzesten ist, für den also gilt: $l_j = min\{l_i\}$ für $i = 1 \ldots n$.

Auch hier werden zunächst die Leistungsunterschiede der einzelnen Knoten nicht berücksichtigt und damit unter Umständen eine ungünstige Verteilung der einzelnen Tasks erreicht. Durch die Einführung eines ganzzahligen positiven Gewichts für jeden Knoten (notiert mit w_i) kann der unterschiedlichen Leistungsfähigkeit der Knoten Rechnung getragen werden. Bei dem *weighted Queue Length-Scheduling* gilt folgende Vorschrift für die Zuweisung des nächten Tasks: $l_j = min\{l_i/w_i\}$ für $i = 1 \ldots n$.

Beispiel: Angenommen, der Cluster besteht aus vier Knoten A, B, C, D, die mit den Gewichten 1, 2, 3, 1 versehen sind. Bei einer Durchlaufreihenfolge von $A - D$ ergibt sich nach sieben verteilten Tasks[19] die typische Verteilung $ABCDCBC$, die der vorgegebenen Gewichtung entspricht und sich zyklisch wiederholt. Damit werden die Tasks zu ca. 14% auf A und D, zu ca. 29% auf B und zu ca. 43% auf C verteilt.

Der bei diesem Verfahren ausgewertete Parameter liegt innerhalb der Schedulerumgebung vor, spiegelt aber nicht die tatsächliche Last des Knotens wider. Der *NodeServer* (siehe Abschnitt 3.7) stellt als Schnittstelle zum Knotenbetriebssystem und -DBMS Informationen bereit, die den realen Belastungen des Knotens entsprechen und für ein lastorientiertes dynamisches Scheduling verwendet werden können.

Problemorientierte Verfahren

Den oben angeführten Verfahren ist gemeinsam, daß sie bei der Verteilung der Tasks keine Differenzierung hinsichtlich ihres Aufwands machen. In der implementierten Umgebung liegen innerhalb der *TaskDescription* Informationen vor, die von dem Scheduling-Algorithmus dazu verwendet werden können, eine Abschätzung über Aufwand und Laufzeit des entsprechenden Tasks zu machen. Besonders geeignet scheint hierbei die Optimierung der Verteilung von SQL-Anfragen an die Knoten-DBMS zu sein, da einerseits aus der Anfrage recht gut auf den Aufwand geschlossen werden kann und andererseits die Ausführungszeiten der Abfragen stark variieren.

Eine effektive Klassifizierung der Abfragen kann zunächst erreicht werden, indem die Abfragen in verschiedene Klassen eingeordnet und entsprechend ihres Aufwandes auf die einzelnen Knoten verteilt werden, um eine gleichmäßige Auslastung des Clusters zu erreichen. Diese Klassifizierung kann manuell durchgeführt und in der Konfiguration des Schedulers festgehalten werden, allerdings ist sie dann abhängig von der Datenbankstruktur und den zugrundeliegenden Applikationen und kann nicht auf Veränderungen an der Struktur reagieren (z. B. das nachträgliche Anlegen einen Indexes für Tabellen).

[19] wobei angenommen wird, daß während der Verteilung kein Task beendet wurde

Besser, weil adaptiv und selbstoptimierend, ist ein Verfahren, das die Klassifizierung entsprechend der Tasklaufzeiten für die einzelnen Abfragen kategorisiert. Hierbei sind die in Abbildung 3.7 dargestellten Ansätze denkbar.

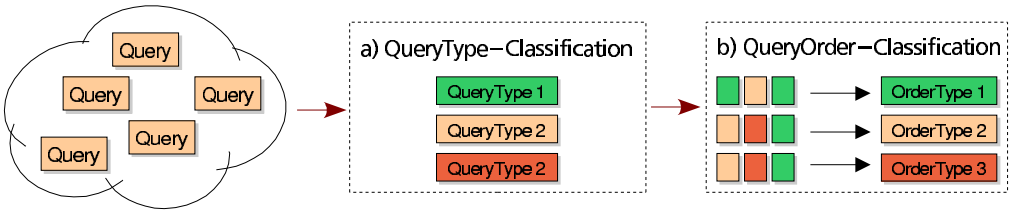


Abbildung 3.7: mehrstufige Abfrageklassifikation

Eine solche Kategorisierung der einzelnen Abfragen (*QueryType-Classification*, Abbildung 3.7a) entspricht in gewisser Weise einem Mustererkennungsverfahren, da typische variable Elemente einer Abfrage (etwa Tabellennamen) zu klassifizieren sind. Andererseits ist auch der im wesentlichen unveränderliche Aufbau der Anfrage relevant, da schon hieraus Hinweise auf die Komplexität der Anfrage abgeleitet werden können (`JOIN`-Abfragen oder eine `ORDER BY`-Anweisung sind in der Regel aufwendig). Der Aufwand für die Klassifizierung der Abfragen ist hierbei der Verzögerung des Anfrageschedulings gegenüberzustellen und abzuwägen.

In einem zweiten Schritt können in einem ähnlichen Prozeß die Beziehungen einzelner (bereits typisierter) Abfragen zueinander erfaßt werden, indem die Abfolge einer Anzahl von Anfragen betrachtet und bezüglich ihres Aufwandes klassifiziert wird (*QueryOrder-Classifikation*, Abbildung 3.7b). Dies kann sinnvoll sein, um Gruppen von Anfragen, wie sie z. B. von dem Web-Frontend gestellt werden, zu erkennen und positive Effekte, die sich durch die Abfragereihenfolge ergeben, auszunutzen (z. B. Ausnutzung des Caches innerhalb des DBMS bei mehreren Abfragen auf gleiche Tabellen).

3.5.6 ClientManager

Der ClientManager ist ein Bestandteil der MetaServer-Plattform und stellt seine Dienste über das Serverinterface anderen Komponenten zur Verfügung. Er hat die Aufgabe, die für den Cluster notwendige Benutzerverwaltung zu realisieren, sofern dies nicht schon von den eingesetzen Serverkomponenten (DBMS) übernommen wird. Die einzelnen Benutzer werden hierbei durch ihren Anmeldenamen und durch den Rechner, von dem sie sich zum Cluster verbunden haben, unterschieden. Eine detaillierte Beschreibung der Benutzerverwaltung erfolgt in Abschnitt 3.5.6.1.

Eine weitere Aufgabe des ClientManagers ist die Verwaltung und Optimierung der offenen Verbindungen der einzelnen Klienten zu den DBMS der einzelnen Knoten. Dieses als *ServerSide Connection Caching* bezeichnete Verfahren wird in Abschnitt 3.5.6.2 beschrieben und soll insbesondere bei steigender Knotengröße und wachsenden Benutzerzahlen dafür sorgen, daß der Anmeldeprozess am Cluster nicht unnötig viel Zeit verbraucht.

3.5.6.1 Benutzerverwaltung im Cluster

Für die Speicherung von Benutzerinformationen, die für den Einsatz im Cluster relevant sind, ist es notwendig, eine geeignete Form zu finden, diese abzulegen. Da eine Integration in die bestehenden Basissysteme (z. B. DBMS) zu neuen Abhänigkeiten von den jeweiligen Systemen führen würde, werden diese Informationen separat abgelegt und ergänzen so die bereits vorhandene Benutzerverwaltung um genau die Informationen, die zusätzlich benötigt werden.

Die Informationen über die einzelnen Nutzer werden in Form von Eigenschaften (Properties) in normalen Textdateien abgelegt. Zum gegenwärtigen Zeipunkt ist dies aufgrund der überschaubaren Anzahl von Benutzern ohne Probleme möglich. Ein solcher Eintrag hat dann die in Abbildung 3.8 dargestellte Struktur.

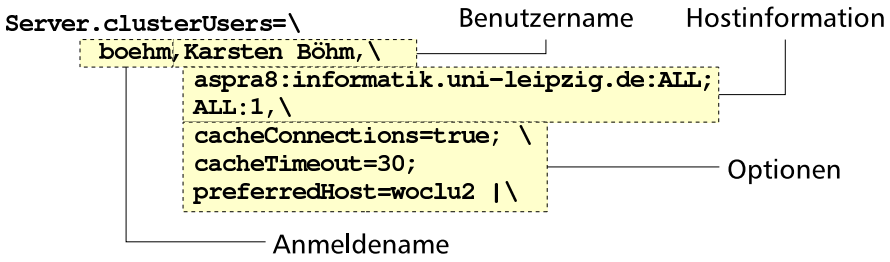


Abbildung 3.8: Property für die Definition von Benutzerrechten

Hierbei werden für jeden Benutzer Attribute definiert, die untereinander durch "," getrennt werden. Sofern die Attribute mehrwertig sind, erfolgt hier eine Abgrenzung mit Hilfe von ";". Ein abschließendes "|"-Zeichen beendet die Definition des Nutzereintrages. Für jeden Benutzer wird der Anmeldename (im Beispiel: `boehm`), unter dem er sich auf dem System anmeldet, sowie ein 'beschreibender' Name (im Beispiel: `Karsten Böhm`) festgelegt.

Es folgt eine Reihe von Zugangsbeschränkungen, die angeben, wie viele Clusterknoten dem Benutzer von einem bestimmten System aus zur Verfügung stehen. Hierbei ist mindestens der Standardfall zu definieren, der verwendet wird, wenn keiner der anderen Fälle angewendet werden kann und der mit dem Schlüsselwort `ALL` beginnt. Im Beispiel in Abbildung 3.8 wurde festgelegt, daß dem Benutzer `boehm` nur dann alle Clusterknoten zur Verfügung stehen[20], wenn er sich von `aspra8` aus anmeldet. Von allen anderen Rechnern aus wird ihm nur ein einziger Knoten zugeordnet.

In der nächsten Sektion können für einen Benutzer noch eine Reihe von Optionen angegeben werden, die das Verhalten der Verbindungen für den Nutzer weiter parametrisieren. Insbesondere kann hier angegeben werden, ob und wie lange Verbindungsinformationen zwischengespeichert und welche Clusterknoten bevorzugt verwendet werden sollen. Im aufgeführten Beispiel wird die Verbindung des Benutzers noch 30 Sekunden nach dessen Abmeldung zwischengespeichert und wiederverwendet, sollte sich der Nutzer innerhalb dieser Zeitspanne erneut am

[20] dies wird ebenfalls durch das Schlüsselwort `ALL` erreicht, das an Stelle eines realen Rechnernamens notiert wird

System anmelden. Außerdem wird der Knoten mit Namen `woclu2` bevorzugt verwendet, sofern er verfügbar ist.

3.5.6.2 ServerSide Connection Caching

Die Anmeldung eines Benutzers an einem DBMS ist ein zeitaufwendiger Prozeß – zunächst muss die Identität des Benutzers festgestellt und überprüft und dann die notwendigen Ressourcen bereitgestellt werden, also Speicher alloziert und neue Threads gestartet werden.

Durch den Einsatz des Clusters steigt dieser Aufwand noch beträchtlich, da der Klient auf allen Knoten, die ihm zur Verfügung stehen sollen, angemeldet werden muß. In einigen Fällen ist auch noch eine vorherige separate Authentifizierung notwendig, die den Benutzer zunächst identifiziert (dies ist beim MySQL-MetaServer der Fall). Hinzu kommt weiterhin, daß die Zeit, die für das Übertragen der entsprechenden Pakete benötigt wird, an verschiendenen Stellen in die Kalkulation einfließt. Die Zeit, die für die Anmeldung eines Klienten benötigt wird, läßt sich wie folgt formalisieren:

$$t_{cluster} = t_{dispatch} + \left\lceil \underbrace{t_{login} + 2 \cdot t_{net}}_{\text{authentication}} \right\rceil + \underbrace{n \cdot (t_{login} + 2 \cdot t_{net})}_{\text{login}}$$

Hierbei ist t_{login} die durchschnittliche Zeit, die für das Anmelden des Benutzers auf dem realen DBMS benötigt wird, während mit t_{net} die mittlere Übertragungszeit eines Daten-Paketes und mit n die Anzahl der Knoten im Clustet notiert wird. Unabhängig vom Ausbau des Clusters und der verwendeten Systeme ist noch die Zeit zu berücksichtigen, die zur Bearbeitung innerhalb des MetaServers benötigt wird ($t_{dispatch}$).

Für die Effizienz der Anmeldung sind hierbei im wesentlichen zwei Faktoren ausschlaggebend:

- die Anzahl der Knoten im Cluster
- die durchschnittliche Dauer einer Verbindung

Im ungünstigen Fall (bei einer großen Knotenzahl und einer kurzen Verbindungsdauer) kann der Aufwand für die Anmeldung leicht die Zeit, die für die Abfrage der Daten benötigt wird, übersteigen.

Da beide Parameter je nach Anwendung und Konfiguration des Clusteraufbaus sehr unterschiedlich ausfallen können, wurde es notwendig, hier Möglichkeiten für eine geeignete Parametrisierung zu bieten, um den oben dargestellten zusätzlichen Zeitaufwand zu minimieren.

Die Knotengröße, notiert durch n, die besonders bei großen Clusterkonfigurationen den bestimmenden Faktor für den beim Anmeldevorgang entstehenden Zeitaufwand darstellt, läßt sich insofern beeinflussen, als daß für jeden Klienten ange-

geben werden kann, wie viele der verfügbaren Knoten ihm maximal zugeordnet werden sollen.

Die durchschnittliche Verbindungsdauer hingegen läßt sich außerhalb der Applikation nur schwer abschätzen oder beeinflussen, weswegen hier nach einer anderen Möglichkeit gesucht wurde. Insbesondere war der Fall, daß eine Applikation zwar viele Knoten anfordert, aber nur kurz am System angemeldet bleibt, zu betrachten. Dieser Fall ist insofern interessant, als daß Web-Anwendungen, die auf dem CGI-Interface basieren, für jede Anfrage eine neue Datenbanksitzug initiieren, die anschließend sofort wieder beendet wird.

Gelöst wurde dieser Konflikt, indem die pro Klient geöffneten Verbindungen zu dem Cluster auch nach der Abmeldung des Benutzers nicht sofort getrennt werden, sondern noch eine bestimmte Zeit geöffnet beiben. Verbindet sich der Benutzer innerhalb dieser frei wählbaren Zeitspanne erneut mit dem Cluster, werden ihm die bestehenden Verbindungen ohne Zeitverlust wieder zugewiesen. Dieses als *Server Side Connection Caching* bezeichnete Verfahren arbeitet für den Benutzer völlig transparent und kann für jede Verbindung individuell konfiguriert werden. Es ist außerdem unabhängig von der gewählten Verbindungsart und steht in der derzeitigen Implementierung bei allen Zugangsarten zum DBMS zur Verfügung.

Mit den angegebenen implementierten Optimierungen läßt sich die für den Anmeldeprozeß benötigte Zeit bereits erheblich verringern. Bei einer sehr großen Knotenanzahl kann eine weitere Verbesserung der Performanz erreicht werden, indem der Anmeldevorgang selbst parallelisiert wird. Der resultierende Aufwand verändert sich dann folgendermaßen:

$$t_{cluster} = t_{dispatch} + \left\lceil \underbrace{t_{login} + 2 \cdot t_{net}}_{\text{authentication}} \right\rceil + \underbrace{max((t_{login} + 2 \cdot t_{net})) + t_{par}}_{\text{login}}$$

Der durch die Parallelisierung entehende Mehraufwand wird hierbei durch t_{par} notiert und kann bei großen Clusterumgebungen zum bestimmenden Faktor werden. Eine weitere Optimierungsmöglichkeit bestünde in der vorzeitigen positiven Rückmeldung an den Klienten, nachdem eine Mindestanzahl von Knoten verfügbar ist. In der vorliegenden Implementierung werden diese Möglichkeiten jedoch aufgrund der geringen Clustergröße nicht umgesetzt.

3.6 SynchroServer

Beim Entwurf der Clusterarchitektur unter Verwendung von vollständig replizierten DBMS auf jedem einzelnen Knoten entsteht das Problem der Konsistenzhaltung der redundanten Datenbestände, wenn der Datenbankinhalt oder deren Struktur durch entsprechende Anfragen verändert wird. Außerdem muß trotz der Mehrbenutzerfähigkeit des Systems gewährleistet werden, daß modifizierende Anfragen verschiedener Nutzer zeitlich synchronisiert werden und dieselbe Wirkung haben wie bei Verwendung eines einzelnen DBMS. Hierbei sind insbesondere die ACID-Eigenschaften[21] des eingesetzten DBMS zu berücksichtigen. Je nach Art des verwendeten Systems kann dies einen unterschiedlich hohen Aufwand bedeuten. Da das verwendete MySQL-DBMS keine Transaktionen unterstützt[22] und das Sperren von Daten nur auf Tabellenebene möglich ist, hält sich der Aufwand in Grenzen und beschränkt sich darauf, die Abfragen in der richtigen Reihenfolge zu serialisieren und auf die einzelnen Knoten zu propagieren. Ein weiterer Gesichtspunkt sind die verschiedenen Zugangsmöglichkeiten (JDBC oder MySQL-Protokoll), die innerhalb des Synchronisationsmechanismus vereinheitlicht und gemeinsam bearbeitet werden müssen. Einen guten Überblick über die verschiedenen Synchronisationsverfahren (auch Replikationsverfahren oder engl. *Replication*) findet man in [Geo94], Kaptitel 11. Der dort getroffenen Einordnung folgend, wird in der vorliegenden Implementierung der *Gossip-Approach* verwendet, der weiter unten detailliert vorgestellt wird.

Der *SynchroServer* berücksichtigt die oben angeführten Bedingungen an das Synchronisationssystem der MetaServer-Plattform und ist als *EmbeddedServer* in der Plattform implementiert. Gleichzeitig bildet er eine wesentliche Komponente des Gesamtsystems. Abbildung 3.9 gibt einen Überblick über die einzelnen Komponenten des Systems und veranschaulicht deren Zusammenwirken.

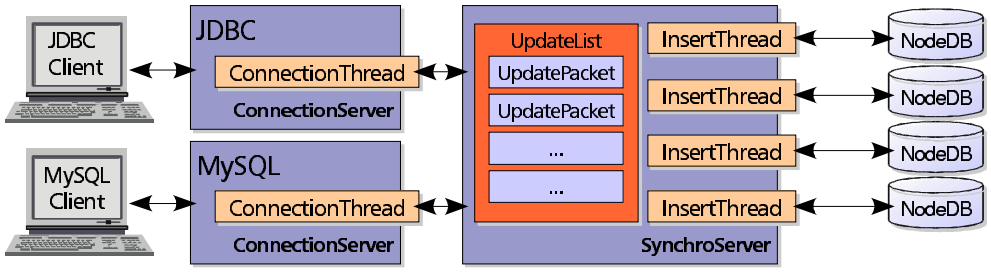


Abbildung 3.9: Architektur des SynchroServers

Die *ConnectionThread*s der einzelnen *ConnectionServer*, die jeweils einen Zugang zu den Knoten-DBMS realisieren, haben die Aufgabe, diejenigen Anfragen zu isolie-

[21] Die ACID-Eigenschaften zählen zu den wichtigsten Wirkprinzipien in einem Multiuser-DBMS: Atomicity, Consistency, Isolation, Durability. Eine detaillierte Beschreibung dieser Eigenschaften findet sich in [Rah94]

[22] Ein transaktionsfähiges MySQL-DBMS mit Hilfe der BerkleyDB-Bibilothek befindet sich derzeit in Entwicklung.

ren, die den Inhalt der Datenbank oder deren Struktur verändern, und diese in einem mehrstufigen Verfahren auf alle Knoten-DBMS zu übertragen:

1. In dem ersten Schritt fordert der ConnectionServer vom Scheduler einen Masterknoten an, auf dem die Abfrage ausgeführt und damit die Persistenz der Datenänderung sichergestellt wird.

   Die Implementierung arbeitet mit einem einzelnen Masterknoten, der von dem Scheduler diese Rolle zugewiesen bekommt. Auch bei einem Ausfall des Masterknotens bleibt das System stabil, solange noch Knoten mit konsistenten Datenbanken verfügbar sind, da der Scheduler in diesem Falle den defekten Knoten deaktivieren und die Rolle des Masters einem anderen Knoten zuordnen kann. Die Verwendung von mehr als einem Masterknoten mit dem Ziel einer höheren Datensicherheit und Verfügbarkeit ist ebenfalls denkbar, allerdings muß dann durch Verwendung eines geeigneten zentralen Synchronisationsverfahrens sichergestellt werden, daß die Datenbankknoten konsistent bleiben (eine vergleichende Beschreibung verschiedener Synchronisationsverfahren ist in [Rah94], Kapitel 8 enthalten).

2. Das Ergebnis der Abfrage wird an den Client übermittelt.

3. Sofern die Operation auf dem Masterknoten erfolgreich war, wird anschließend die Anfrage zusammen mit weiteren Informationen (z. B. Benutzerinformationen und die verwendete Zieldatenbank) an den SynchroServer übergeben und von diesem unabhängig von dem ConnectionThread, der für weitere Anfragen zur Verfügung steht, auf alle verbleibenden Clusterknoten propagiert.

Im SynchroServer werden diese Pakete dann in der Reihenfolge ihres Eintreffens und ungeachtet ihres Typs einer speziellen Liste, der *UpdateList*, hinzugefügt. Für jeden Knoten läuft innerhalb des SynchroServers ein eigener *InsertThread*, der für die Aktualisierung der Daten des Knoten-DBMS zuständig ist, dessen Status überwacht und bei Bedarf eine Replikation des Knotens einleitet.

3.6.1 Funktionsweise der UpdateList

Die zentrale Komponente des SynchroServers ist die UpdateList, die die noch zur Bearbeitung ausstehenden Pakete enthält. Sie hat mehrere Aufgaben zu erfüllen:

- Herstellung einer Ordnung über die eintreffenden Anfragen, entsprechend ihres zeitlichen Eintreffens, unabhägig von der Art der Verbindung zu dem Klienten.

- dynamische Größenanpassung der Liste bis zu einem definierbaren Maximum. Bei Überschreitung dieser oberen Grenze wird das weitere Einfügen von Elementen verzögert und damit Speicherproblemen vorgebeugt, die durch ein starkes Mißverhältnis zwischen ConnectionThread und InsertThreads entstehen und zu einem ungebremsten Wachstum der Liste führen können.

- Entkopplung der Abhängigkeiten der InsertThreads von den eigentlichen ConnectionsThreads durch die Bereitstellung entsprechender synchronisierter Methoden[23] zum Einfügen von neuen Elementen und Sicherstellung des Zugriffs auf das jeweils aktuellste Element.
- Verwaltung der externen Referenzen auf die Listenelemente durch ein einfaches Benachrichtigungsschema. Ein ausschließlich inkrementeller Zugriff auf die Referenzen stellt sicher, daß die Listenelemente nicht mehrfach abgefragt werden können.

Realisiert werden diese Funktionen in der Implementierung durch eine eigene Klasse, der *ReferencedList*, die eine spezielle Listenart implementiert, die für die zu bewältigenden Aufgaben besonders gut geeignet ist. Es handelt sich dabei um eine Liste in Form eines dynamisch wachsenden FIFO-Speichers[24], der einen sequentiellen Zugriff auf die Listenelemente in der Reihenfolge erlaubt, in der sie eingefügt wurden. Auf diese Weise können mehrere *Produzenten* (in unserem Fall die ConnectionThreads) neue Elemente in die Liste einfügen, während die *Konsumenten* (die in unserer Anwendung durch die InsertThreads repräsentiert werden) die Elemente der Liste in sequentieller Reihenfolge verarbeiten (*Producer-Consumer*-Pattern entsprechend [Gam94]). Während die Behandlung von mehreren Produzenten hierbei kein Problem ist, ist der Fall mehrerer Konsumenten nicht ohne weiteres durch die üblicherweise verwendeten Datenstrukturen (allgemein als *Warteschlangen* oder *Queues* bezeichnet) abbildbar. Die ReferencedList verwendet deshalb für jeden einzelnen Konsumenten eine *Referenz*, die auf das aktuelle Listenelement für diesen Konsumenten zeigt. Greift der Konsument auf das Element zu, so wird das nächste Element der Liste referenziert, bis das Ende der Liste erreicht ist. Wenn alle Konsumenten ein Listenelement bearbeitet haben, also keine Referenzen mehr auf dieses Element verweisen, so gilt es als vollständig bearbeitet und kann aus der Liste entfernt werden.

Da die Produzenten der Liste zu einem beliebigen Zeitpunkt neue Elemente hinzufügen können, wird noch ein System benötigt, daß die Konsumenten über die Veränderung des Listeninhalts informiert. Hierbei wird unter Anwendung des *Observer-Pattern*s nach [Gam94] von der ReferencedList – dem beobachteten Objekt (observable object) – eine Nachricht an alle InsertThreads geschickt, die hierbei als Beobachter (observer) fungieren. Auslöser der Benachrichtigung ist das Einfügen eines neuen Elements in die Liste.

Das ReferencedList-Objekt realisiert also ein Warteschlangensystem für eine beliebige Anzahl von Produzenten und Konsumenten, wobei die virtuellen Warteschlangen aller Konsumenten in einer realen Warteschlange zusammengefaßt werden. In Abbildung 3.10 wird die Anwendung eines solchen Objekts für die Up-

[23] Synchronisierte Methoden verhindern Seiteneffekte, die durch den gleichzeitigen modifizierenden Zugriff mehrerer Threads auf ein Datum entstehen. Java unterstützt eine Synchronisation dieser kritischen Daten, die mit dem Schlüsselwort `synchronized` gekennzeichnet sind, indem jeweils nur ein Thread exklusiven Zugriff auf die Daten erhält, während andere Threads auf deren Freigabe warten müssen.

[24] FIFO: First in, first out

dateList des SynchroServers zusammen mit den Stati der einzelnen Listenelemente dargestellt.

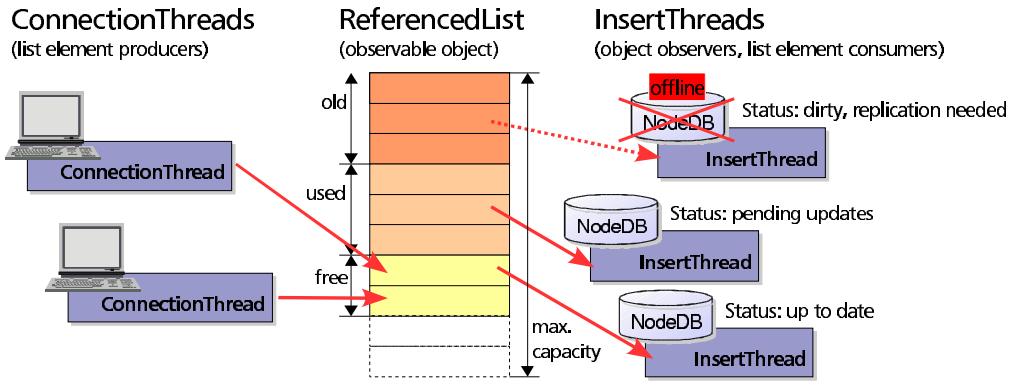


Abbildung 3.10: Aufbau und Funktion des *ReferencedList*-Objekts

Nach der Initialisierung besteht die Liste aus einer Reihe von freien Listeneinträgen (`free`), die von den Produzenten mit Elementen gefüllt werden und dann so lange als benutzt (`used`) gelten, bis sie von allen Konsumenten bearbeitet und nicht mehr referenziert werden. Ihr Status wechselt dann zu `old`, was dazu führt, daß die Elemente vom SynchroServer aus der Liste entfernt werden. Sollten für das Einfügen von neuen Elementen keine freien Plätze in der Liste mehr zur Verfügung stehen, werden zunächst alle alten Elemente entfernt und dann bei Bedarf die Liste bis zu einer definierbaren maximalen Größe erweitert. Sollte auch dann noch kein Platz für neue Einträge in der Liste vorhanden sind, wird der Einfügevorgang so lange verzögert, bis wieder Platz für neue Elemente vorhanden ist.

Um die verschiedenen Formen der Anfragen von den einzelnen ConnectionsThread zu verarbeiten, wird von den InsertThreads als Listenelement ein Objekt erwartet, daß das `UpdatePacket`-Interface implementiert und die notwendigen Informationen enthält. Anhand der Klassenzugehörigkeit des Elements wird dann von den InsertThreads entschieden, welche Methoden für die Abarbeitung des Pakets verwendet werden. Momentan werden hierfür die Klassen `JDBCUpdatePacket` und `MySQLUpdatePacket` akzeptiert und verarbeitet. Alle anderen Listenelemente werden ignoriert und erzeugen eine Fehlermeldung im Logfile.

Wie in Abbildung 3.10 angedeutet, ist ein Knoten, der die einzelnen Listenelemente nur sehr langsam oder überhaupt nicht mehr abarbeitet, die Ursache für ein stetes Anwachsen der Liste und führt zur Beeinflussung der anderen Konsumenten, sobald das Maximum erreicht ist.

Der SynchroServer muß diesen Fall erkennen und entscheiden, welcher Clusterknoten deaktiviert werden muß, um ein effektives Arbeiten weiterhin sicherzustellen. Eine solche Deaktivierung ist immer dann notwendig, wenn das Verhältnis zwischen benutztem und freien Platz in der Liste auch nach der Beseitigung aller alten Elemente und der zulässigen Kapazitätserweiterungen einen Schwellwert ($th_{offline}$) überschreitet. Im Zusammenhang mit der Deaktivierung einzelner Knoten ist auch ein Kriterium anzugeben, nachdem deaktivierte Knoten wieder in den Synchronisationsprozeß eingebunden werden können. Dieses Kriterium ist er-

reicht, wenn das oben angegebene Verhältnis den Schwellwert th_{online} unterschreitet:

Offline-Kriterium:	Online-Kriterium:
$100\% \cdot \frac{p_{used}}{p_{free}} > th_{offline}$	$100\% \cdot \frac{p_{used}}{p_{free}} \leq th_{online}$
Standardwert: $th_{offline} = 90\%$	Standardwert: $th_{online} = 10\%$

Von der Deaktivierung betroffen ist immer der Knoten mit dem größten Abstand zum aktuellsten Listeneintrag. Somit wird gewährleistet, daß er Platzgewinn in der UpdateList maximiert wird. Falls notwendig, wird dieser Prozeß solange wiederholt, bis alle nichtreagierenden Knoten aus dem Synchronisationsprozeß entfernt wurden. Im Extemfall können hiervon, abgesehen vom Master-DBMS, alle Cluster-Knoten betroffen sein.

Jeder deaktivierte Knoten wird als `offline` markiert und an den Scheduler gemeldet. Außerdem wird der zugehörige InsertThread angehalten und dessen Referenz aus der UpdateList entfernt, um die von ihm belegten Einträge freizugeben.

Sobald die Auslastung der UpdateList sinkt und das oben angegebene Online-Kriterium erfüllt ist, initiiert der SynchroServer in den den deaktivierten Knoten zugeordneten InsertThreads den im folgenden Abschnitt beschriebenen Replikationsprozeß. Wenn dieser erfolgreich abgeschlossen ist, wird der InsertThread wieder als Observer der UpdateList registriert und propagiert die folgenden Update-Anfragen auf den ihm zugeordneten Knoten.

3.6.2 Replikation von Datenbankinhalten

Um die redundanten Datenbestände der Knoten-DBMS auf einen einheitlichen Stand zu bringen, ist es notwendig, Knoten, die zeitweise inaktiv waren, mit den aktuellen Daten des Master-DBMS zu aktualisieren, das heißt, die Daten von dieser Quelle auf die veralteten Datenbestände zu replizieren. Prinzipiell bieten sich hier zwei Vorgehensweisen an:

- **logbasierte Replikation:** Hierbei dient eine Log-Datei, in der alle Abfragen aufgezeichnet werden, die den Datenbankinhalt oder dessen Struktur verändern, als Grundlage für die Replikation. Die Aktualisierung besteht in diesem Fall darin, alle Abfragen, die im Log ab dem Deaktivierungszeitpunkt enthalten sind, in der gleichen Reihenfolge auf den Zielsystem auszuführen.

  Die Vorzüge dieses Verfahrens bestehen darin, daß die Datenbank zur Replikation nicht außer Betrieb genommen werden muß und für weitere Anfragen zur Verfügung steht[25] und daß es für wenige Abfragen sehr effektiv ist. Nachteilig ist der zusätzliche Speicheraufwand, der für das Anlegen der Logdatei entsteht und die Problematik, die Deaktivierungszeitpunkte für die einzelnen Knoten exakt identifizieren zu können. Außerdem ist der Aufwand für

[25] Allerdings kann es hier zu Datenbankanomalien kommen (lost update, non repeatable read, dirty read, phantom read), siehe hierzu [Rah94], Seite 133 ff.

die Replikation schwer abschätzbar, da die Ausführung der modifizierenden Abfragen durch vorhandene Indextabellen sehr aufwendig sein kann[26] und in jedem Fall die gesamte Historie sequentiell abgearbeitet werden muß, um den aktuellen Stand zu erreichen[27].

- **dateibasierte Replikation:** Das zweite Verfahren arbeitet außerhalb der Datenbank auf Dateiebene und kopiert alle zu einer Tabelle einer Datenbank gehörenden Dateien auf das zu aktualisierende Zielsystem[28]. Im Falle von MySQL sind das für jede Tabelle genau drei Dateien: die Strukturbeschreibung, die Daten und der Index.

  Der Vorteil dieses Verfahrens besteht darin, daß der Aufwand für die Replikation einer Datenbank gut abschätzbar ist, da er nur von dessen Größe und der Geschwindigkeit der verwendeten Netzwerkverbindung abhängig ist. Außerdem ist das Verfahren recht einfach anwendbar und funktioniert auf allen DBMS, die auf Dateiebene arbeiten[29]. Je umfangreicher die Veränderungen an den zu replizierenden Daten sind, desto effektiver ist das Kopieren der ganzen Tabelle. Ein Nachteil dieses Ansatzes besteht darin, daß das DBMS für die Replikation heruntergefahren werden muß und in dieser Zeit nicht für Abfragen zur Verfügung steht. Außerdem ist das Kopieren ganzer Tabellen nicht effektiv, wenn die Änderungen an den Daten nur gering sind. Das trifft insbesondere auf große Tabellen zu.

Bei der Gegenüberstellung der beiden Alternativen für die Replikation wird deutlich, daß jedes Verfahren seine Vor- und Nachteile hat und es von der aktuellen Situation abhängig ist, welche Variante besser geeignet ist. In der Implementierung wird eine dateibasierte Replikation eingesetzt, da diese robuster gegen schwankende Umgebungsbedingungen (Auslastung des Clusters, Änderungsgrad der zu replizierenden Daten) ist. Bei der Replikation arbeitet der jeweilige InsertThread eng mit dem NodeServer des entsprechenden Knotens zusammen. Der Prozeß gliedert sich hierbei in die folgenden drei Schritte:

- Herunterfahren des Knoten-DBMS durch den NodeServer mit Hilfe eines entsprechenden Betriebssystemskripts.
- Replikation der Daten mit Hilfe des FTP-Services des NodeServers, wobei die zu kopierenden Dateien von dem InsertThread gesteuert werden.

[26] Ein Index auf einer Datenbanktabelle beschleunigt zwar den lesenden Zugriff auf die Daten; alle Schreiboperationen werden jedoch durch die notwendige Aktualisierung des Indexes behindert.

[27] Beispielsweise kann man sich eine Situation vorstellen, in der zunächst viele neue Einträge der Datenbank hinzugefügt und zu einem späteren Zeitpunkt mit einem einzelnen Kommando gelöscht werden

[28] Es müssen bei der Replikation nur diejenigen Tabellen kopiert werden, die von Änderungen seit dem letzten Deaktivierungszeitpunkt betroffen sind. Da der Änderungszeitpunkt der Dateien kein zuverlässiger Indikator ist, muß der SynchroServer die Informationen über die geänderten Tabellen verwalten.

[29] Im Gegensatz zu dateibasierten DBMS gibt es auch Systeme, die auch die Dateiverwaltung selbst übernehmen und sie nicht dem Betriebssystem überlassen (z. B. Oracle™). In diesem Fall liegen alle Daten in einem oder mehreren großen Datenbereichen, die nicht weiter zu unterscheiden sind.

- Neustart des Knoten-DBMS durch den NodeServer mit einem weiteren Betriebssystemskript.

Nach erfolgreichem Abschluß der Replikation der veränderten Daten kann der Status des Knotens in `online` verändert werden und dies an den Scheduler gemeldet werden, der dann eintreffende Anfragen auf diesen Knoten weitergeben kann. Gleichzeitig wird der Knoten durch die Registrierung des InsertThreads bei der UpdateList wieder in den Synchronisationsprozeß einbezogen.

3.6.3 Einschränkungen

Die Realisierung des beschriebenen Replikationsprozesses stellt einen Eingriff in die Architektur des zugrundeliegenden DBMS dar und führt an einigen Stellen zu einer Einschränkung der Funktionalität des Systems, obwohl es Ziel des Entwurfs war, die Replikation aus Benutzersicht als möglichst transparenten Prozeß zu implementieren.

Um die Arbeit mit automatischen Zählern (`AUTO-INCREMENT`-Spalten), die von dem DBMS verwaltet werden, zu erleichtern, bietet MySQL eine spezielle Funktion (`last_insert_id()`) an, die den letzten automatisch generierten Wert liefert, der für jede Datenbankverbindung separat geführt wird. Oftmals werden solche Zähler für die automatische Erzeugung von numerischen Primärschlüsseln verwendet und müssen in anderen Tabellen als Fremdschlüsel referenziert werden. In dem folgenden Beispiel sei angenommen, daß Daten für ein Wort in einer Tabelle `wort` enthalten seinen, die mit einem solchen automatisch generierten numerischen Primärschlüssel versehen ist. Die Relationen des Wortes zu anderen Daten (z. B. Kollokationen, Beispielsätze usw.) seien in einer weiteren Tabelle `wortbeschreibung` enthalten und müssen über den gerade erzeugten Primärschlüsselwert miteinander verknüpft werden. In MySQL läßt sich dies mit Hilfe der angegebenen Funktion elegant lösen:

```
INSERT INTO wort VALUES(NULL, 'Testwort','...');
INSERT INTO wortbeschreibung VALUES(LAST_INSERT_ID(), '...');
```

Für den Cluster kann diese Konstruktion nicht verwendet werden, da nicht sichergestellt werden kann, daß beim zweiten Einfügebefehl die gleiche Datenbankverbindung benutzt wird (der Scheduler könnte die Rolle des Masterknotens neu vergeben haben) und somit den falschen Wert liefert.

Ein weiteres Problem ergibt sich aus der Tatsache, daß die UpdateList des SynchroServers ausschließlich im Hauptspeicher geführt wird. Sollte der SychroServer abstürzen oder der Knoten, auf dem der Server läuft, durch einen Hardwaredefekt ausfallen, so sind alle in der Liste enthaltenen unbearbeiteten Update-Pakete verloren und die betroffenen Knoten-DBMS in einem inkonsistenten Zustand, der nur durch Replikation vom Masterknoten beseitigt werden kann. Durch eine zyklische Sicherung der UpdateList auf einen externen Datenträger oder das Führen eines Logfiles könnte zumindest Softwarefehlern wirksam vorgebeugt werden.

Bei dem SynchroServer handelt es sich um eine zentrale, systemkritische Komponente der Plattform, die aufgrund ihrer Funktion als zentraler Synchronisationspunkt als Singleton[30] implementiert werden muß und nicht in der verteilten Architektur auf mehreren Systemen gleichzeitig ausgeführt werden kann. Damit ist der Synchronisationspunkt eine zentrale Schwachstelle der Plattform (*single point of failure*), deren Ausfall Auswirkungen auf alle Knoten des Clusters hat. Zwar ist eine Implementierung mehrerer SynchroServer denkbar, die ihrerseits gegeneinander abgeglichen werden und durch die entstehende Redundanz eine höhere Verfügbarkeit versprechen, jedoch steht dem ein hoher Implementationsaufwand und ein zusätzlicher Geschwindigkeitsverlust durch die notwendige externe Synchronisation mit Hilfe eines geeigneten ausfallsicheren Protokolls entgegen.

3.7 NodeServer

Innerhalb der MetaServer-Plattform muß von verschiedenen Anwendungen auf unterschiedliche Weise auf das Betriebssystem und das Datenbankmanagementsystem der Clusterknoten zugegriffen werden. Um eine weitgehende Plattformunabhänigkeit zu gewährleisten, wird dieser Zugriff nicht direkt von den einzelnen Komponenten ausgeführt, sondern über einen *NodeServer* abgewickelt, der auf jedem Clusterknoten verfügbar ist und die zentrale Schnittstelle zu den Diensten und Ressourcen der Knoten bildet.

Der NodeServer kann dabei als Bestandteil der Metaplattform betrieben werden (er arbeitet dann als *EmbeddedServer*), oder er kann unabhängig von anderen Komponenten auf Knoten laufen, die keine eigene MetaServer-Plattform betreiben. In diesem Fall können Applikationen auf den NodeServer über die Registry zugreifen, in der sich der Dienst selbstständig registriert.

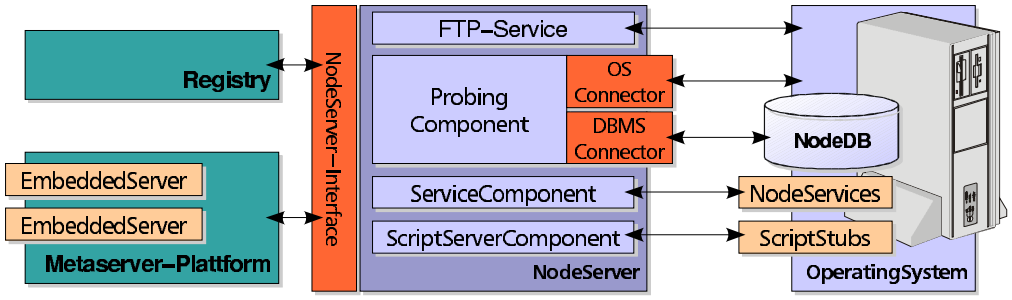


Abbildung 3.11: Architektur des NodeServers

[30] Ein Singleton ist nach der Theorie der Software Pattern eine Klasse, die nur einmal instantiiert werden kann. Siehe [Gam94]

Die Aufgaben, die der NodeServer dabei zu erfüllen hat, gliedern sich in die folgenden Bereiche:

- Bereitstellung verschiedener Systemparameter (probing values) des Knotenbetriebssystems und des Knoten-DBMS zur Abschätzung der Systemlast und Verfügbarkeit eines Knotens. Diese Werte werden vor allem vom Scheduler verwendet, um eine bessere Lastverteilung der Tasks auf die einzelnen Clusterknoten zu erreichen.
- Implementierung einer betriebssystemunabhängigen Schnittstelle für die auf den Knoten laufenden *BinaryServices*, die das Starten, Stoppen und Überwachen der entsprechenden Prozesse auf dem Knotenbetriebssystem steuern.
- Realisierung der Schnittstelle des Skriptservers (siehe Abschnitt 4.3) zu den einzelnen Clusterknoten. Sie sorgt dafür, daß der Name eines Statements in einen betriebssystemspezifischen Programmaufruf umgesetzt, ausgeführt und das Ergebnis zurück an den SkriptServer geliefert wird.
- Bereitstellung eines FTP-Klient-Dienstes, der den effizienten Transfer großer Datenmengen ermöglicht und von dem SynchroServer zur Replikation benutzt wird. Die Implementierung innerhalb des NodeServers hat den Vorteil, daß sie zum einen eine gewisse Unabhängigkeit von den betriebssystemseitig angebotenen Diensten gewährleistet und andererseits die Steuerung des Transferprozesses erheblich vereinfacht, da etwaige Übertragungsfehler oder Verbindungsabbrüche direkt erkannt werden können und eine entsprechende Reaktion (etwa die Operation zu wiederholen, auf einen Timeout zu warten oder eine Fehlermeldung zurückzuliefern) ermöglichen.

Um eine möglichst hohe Unabhängigkeit von dem verwendeten Betriebs- und Datenbanksystem zu erreichen, wurde der Zugriff auf die Systemparameter mit Hilfe von Interfaces realisiert (`DatabaseConnector` bzw. `SystemConnector`), die für alle zu verwendenden Systeme implementiert werden müssen. Gegenwärtig existieren solche Implementierungen für *Linux* als Betriebssystem (`LinuxSystemConnector`) und *MySQL* (`MySQLDatabaseConnector`) als Datenbankmanagementsystem. Dabei müssen diese Implementierungen die benötigten Daten nicht notwendigerweise direkt aus Betriebs- oder Datenbanksystem extrahieren, sondern können stattdessen auf installierte Systems- Management-Systeme zurückgreifen, die diese Informationen aufbereiten und für Applikationen zur Verfügung stellen. Als Beispiel für ein solches System sei das Tivoli-Framework der IBM angeführt, das die Überwachung einer Vielzahl verschiedener Plattformen gestattet (siehe [Tiv00]) und das über eine CORBA-Infrastruktur auch den Zugriff für systemfremde Applikationen ermöglicht.

Die von den Schnittstellen zur Verfügung gestellten Systemparameter erheben keinen Anspruch auf eine Vollständigkeit in Bezug auf die von einem Clusterknoten zu ermittelnden Daten, sondern stellen das Minimum dessen dar, was für die verschiedenen Komponenten der Metaserver-Plattform benötigt wird. Die Spezifikation der Interfaces definiert die folgenden Parameter:

- momentane prozentuale CPU-Benutzung (gemittelt über eine Minute)[31], die einen groben Anhaltspunkt für die Auslastung des Knotens liefert. Die Analyse dieses Parameters ohne Betrachtung weiterer Eckdaten ist jedoch häufig irreführend, da beispielsweise Prozesse geringer Priorität eine hohe Systemlast suggerieren können, obwohl das System durchaus in der Lage ist, weitere Prozesse auszuführen, die dann die niedrigpriorisierten Prozesse verdrängen. Andererseits kann von einer niedrigen CPU-Auslastung nicht auf ein unbelastetes System geschlossen werden, da es möglicherweise durch I/O-Operationen bereits vollauf beschäftigt ist.

- Anzahl der Zugriffe pro Sekunde auf die Festplatten, auf der das Betriebssystem liegt (einschließlich Swap-Bereich). Dieser Wert dient der Ermittlung der I/O-Last, die auf das Betriebssystem entfällt, etwa durch Auslagern von Prozessen bei Hauptspeichermangel.

- Anzahl der Zugriffe pro Sekunde auf die Festplatten, auf der sich das Datenbanksystem einschließlich der Datenbanken befindet. Interessant ist dieser Wert insbesondere für die Erkennung von umfangreichen I/O-Aktivitäten des DBMS, etwa einem linearen Table-Scan, der zu einer hohen Auslastung des Datenbanksystems führt.

- Zugriffszeit auf das Datenbanksystem in Millisekunden. Dieser Wert dient einerseits der Überprüfung der Verfügbarkeit des Knoten-DBMS und ermöglicht außerdem auch Rückschlüsse über die Auslastung des Datenbanksystems, deren Verläßlichkeit abhängig von der jeweiligen Implementierung ist[32]. Momentan verwendet der *HeartbeatServer* diese Informationen, um die Erreichbarkeit der einzelnen Knoten zu prüfen.

[31] momentan werden nur Systeme mit einer einzelnen CPU unterstützt, da nur ein einzelner Wert zurückgeliefert wird, bei Mehrprozessorsysteme könnte die Gesamtlast näherungsweise ermittelt werden, indem die Lastfaktoren der einzelnen Prozessoren addiert und durch deren Anzahl dividiert werden

[32] MySQL bietet hierfür ein spezielles `PING`-Kommando an; im allgemeinen kann eine entsprechende Überprüfung mit der Abfrage `'SELECT 1'` erfolgen.

3.8 Frontendkonzept und Administration der Plattform

Die beschriebenen Anwendungen für die MetaServer-Plattform können über verschiedene Kommunikationswege (meist über TCP/IP oder RMI) mit den Klient-Anwendungen, die außerhalb der Cluster-Umgebung laufen, kommunizieren. Für einige dieser Anwendungen existieren bereits solche Frontends, die für die Interaktion mit der Clustersoftware genutzt werden können, beispielsweise die MySQL-Anwendungen und -Bibliotheken für zahlreiche Programmiersprachen oder die JDBC-API, die den Zugriff für Java-Applikationen genau spezifiziert. Andere Anwendungen hingegen verfügen noch über kein entsprechendes Frontend, so daß deren Entwicklung noch bevorsteht. Beispiele hierfür sind der BinaryServer mit der implementierten Anagramm-Funktionalität oder der ScriptServer, der durch ein entsprechendes Frontend gesteuert werden soll.

Darüber hinaus sind sowohl die Plattform als auch die verfügbaren Anwendungen der Plattform zur Laufzeit konfigurier- und administrierbar und müssen daher eine geeignete Schnittstelle bieten, um diese Funktionalität für externe Administrationsprogramme zur Verfügung zu stellen und so die Verwaltung des Clusters zu ermöglichen.

Der dritte Schwerpunkt, der beim Entwurf der Frontend-Architektur zu beachten ist, ist die Gewährleistung einer weitestgehenden Plattform- und Darstellungsunabhängigkeit, die die Verwendung verschiedener Administrationswerkzeuge erlauben, seien sie nun konsolen- oder Web-orientiert oder in einer GUI integriert.

Schließlich ist zu berücksichtigen, daß die Anforderungen der einzelnen Anwendungen hinsichtlich der für die Administrierbarkeit benötigten Funktionalität sehr unterschiedlich ausfallen werden und damit eine zentralisierte Verwaltung durch ein einzelnes Werkzeug auf lange Sicht erschwert wird.

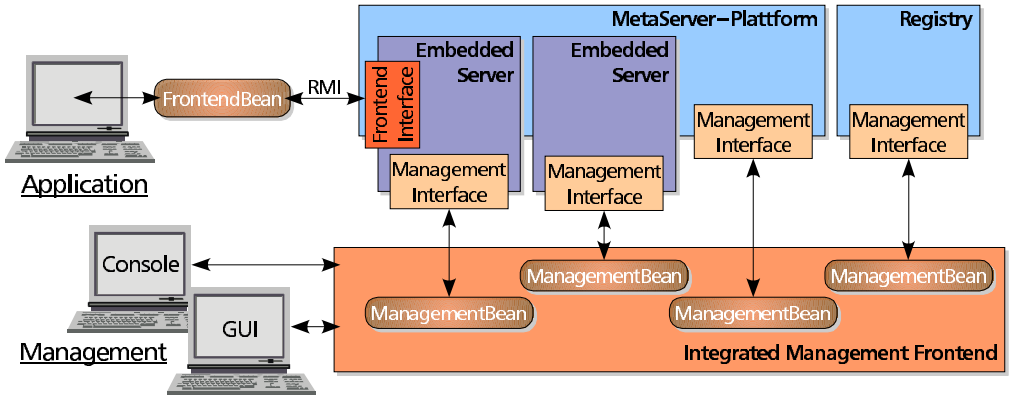


Abbildung 3.12: Darstellung des Frontend-Konzepts

Das realisierte Konzept für die Frontends zur Administration der einzelnen Komponenten der Plattform ist in Abbildung 3.12 skizziert und basiert wesentlich auf der Java-Beans-Architektur von Sun Microsystems™ [Sun97a]. Bei Java-Beans handelt es sich um abgeschlossene, automatisch analysierbare, konfigurierbare Software-Komponenten, die in andere Anwendungen integriert werden können und

eine visuelle oder nichtvisuelle Funktionalität zur Verfügung stellen, die von der jeweiligen Anwendung genutzt werden kann.

Die realisierten Beans sind ausnahmslos nichtvisuell und stellen eine Schnittstelle zu der zu verwaltenden Komponente her. Die Kommunikation zwischen dem Bean und dem entsprechenden Server erfolgt hierbei über eine Schnittstelle, auf die über RMI zugegriffen werden kann. Für das Management der einzelnen Komponenten wird hierfür jeweils ein spezifisches *AdministrationBean* bereitgestellt, das auf das *AdministrationInterface* des entsprechenden Servers zurückgreifen kann. In analoger Weise wird für Klienten einer Plattform-Applikation ein *FrontendBean* zur Verfügung gestellt, das über das *FrontendInterface* Zugriff auf die Funktionalität der entsprechenden Anwendung gewährt und gleichzeitig die Details der Klient/Server-Kommunikation verbirgt.

Ein Werkzeug zur Verwaltung des Clusters oder einzelner Anwendungen muß sich nun lediglich um die Aspekte der Darstellung und der Benutzerinteraktion für das verwendete Medium kümmern, während die Durchführung der notwendigen Aktionen dem entsprechenden Bean überlassen wird, das zu diesem Zweck in die Anwendung eingebunden wird. Durch die Separation der Administrionsfähigkeit in mehrere anwendungsspezifische Schnittstellen wird der Aufbau einer Werkzeugpalette mit variabler Funktionalität ermöglicht, die jeweils nur soviele Beans einbinden muß, wie für gewünschte Funktionalität benötigt werden. Somit ist sowohl der Aufbau eines einzelnen universellen Administrationswerkzeuges, als auch der Entwurf mehrerer unabhäniger Tools für die einzelnen Funktionen möglich, wobei nicht alle vom Bean angebotenen Funktionen benutzt werden müssen.

Für eine flexible Administration des Clusters ist es wünschenswert, verschiedene Zugangswege zu dem verwalteten Server zur Verfügung zu haben, um z. B. Bandbreitenrestriktionen oder Plattformabhängigkeiten Rechnung tragen zu können. Bei einer langsamen Netzwerkanbindung (beispielsweise über ein Modem) ist einem konsolenbasierten Werkzeug sicher einer entsprechenden GUI-Variante der Vorzug zu geben, obwohl letztere zweifellos einen höheren Komfort bietet. Soll die Administration von einer Plattform aus erfolgen, auf der die zur Verfügung stehende GUI nicht lauffähig ist (weil möglicherweise kein JDK zur Verfügung steht), kann unter Umständen auf eine Web-basierte Variante ausgewichen werden. Abbildung 3.13 zeigt eine solche alternative Realisierung eines Administrationswerkzeuges zur Darstellung des aktuellen Inhalts der Registry der Plattform, als zeichenorientierten Anwendung in einer Shell des Betriebssystems und als GUI-Variante, die mit Hilfe des Swing-Toolkits[33] realisiert wurde.

Innerhalb der Implementierung für die MetaServer-Plattform sind die meisten EmbeddedServer, der Verzeichnisdienst (Registry) und auch die Plattform selbst mit einer ausbaufähigen Schnittstelle für die Administration versehen worden. Für die Verwaltung dieser Komponenten wurde ein konsolenorientiertes Administrationswerkzeug (*Cluster Management Shell*) entworfen, das die Plattform zentral verwalten hilft und für den Test der einzelnen ManagementInterfaces genutzt werden kann. Um eine flexible Handhabung unterschiedlichster Management-Schnitt-

[33] Das Swing-Toolkit ist eine von Sun spezifizierte API, die ein plattformneutrales grafisches Benutzerinterface (GUI) zur Verfügung stellt

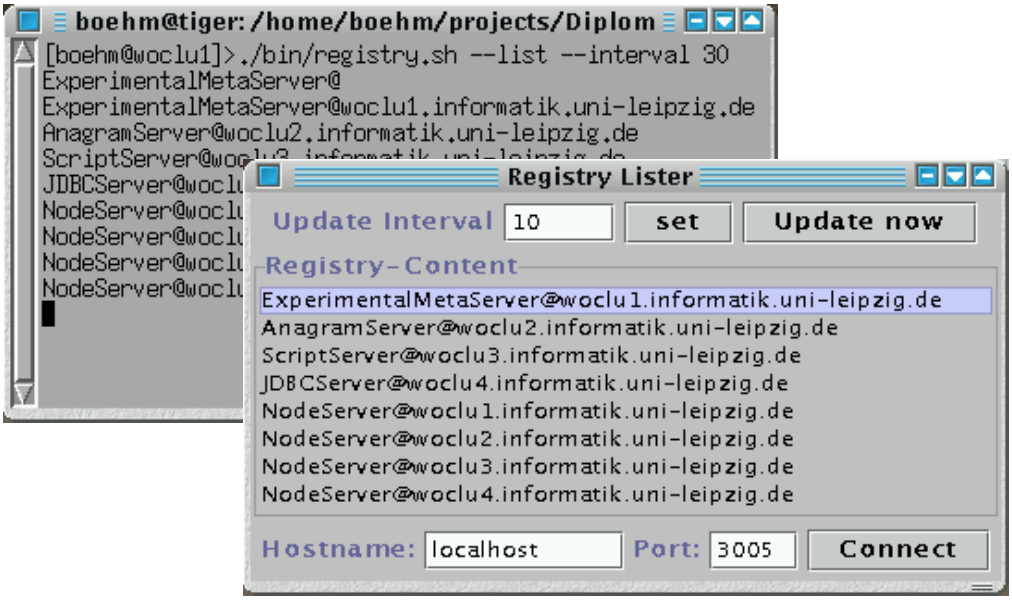


Abbildung 3.13: Frontend für die Registry als Konsolen- und GUI-Version

stellen zu gewährleisten, ist der Kommandointerpreter (*shell*) in mehrere Ebenen (*subshells*) aufgeteilt, die jeweils einem ManagementBean zugeordnet sind und ineinander verschachtelt verwendet werden können.

In Abbildung 3.14 wird anhand einer kurzen Beispielsitzung der Umgang mit dem Werkzeug und die Handhabung der verschiedenen Ebenen veranschaulicht. Nach dem Starten der ManagementShell wird zunächst in den `registry`-Modus gewechselt, wobei im Hintergrund das entsprechende ManagementBean geladen wird, das den Zugriff auf den zentralen Verzeichnisdienst ermöglicht. Mit einer `connect`-Anweisung wird dann innerhalb der Subshell eine Verbindung mit der Registry hergestellt und deren Inhalt mit `list` angezeigt. Im dargestellten Fall werden zwei durch den Verzeichnisdienst registrierte MetaServer-Plattformen und ein weiterer Dienst (JDBCServer) angezeigt, der über RMI angesprochen werden kann. Um die Plattform zu administrieren, wird mit dem Kommando `use ExperimentalMetaServer@lion` eine weitere Subshell aufgerufen, die die Administration auf Plattformebene gestattet. Auf dieser Ebene zeigt das `list`-Kommando die auf dieser Plattform laufenden EmbeddedServer an. Da es sich um verschiedene Subshells handelt, ist die Verwendung gleicher Kommandonamen mit unterschiedlicher Funktionalität hier kein Problem. Wie aus dem Sitzungsprotokoll ersichtlich ist, werden innerhalb der Plattform weitere Dienste angezeigt, die im Verzeichnisdienst nicht aufgeführt waren. Die Ursache hierfür liegt darin, daß diese Dienste entweder über andere Kommunikationswege erreicht werden können (dies betrifft den `NativeMySQLServer`) oder nur für interne Aufgaben vorgesehen sind (wie der `MobilityServer`). Dies illustriert die Tatsache, daß durch die verschiedenen Ebenen die Komplexität des Gesamtsystems weitgehend verborgen wird — ein durchaus gewollter Effekt.

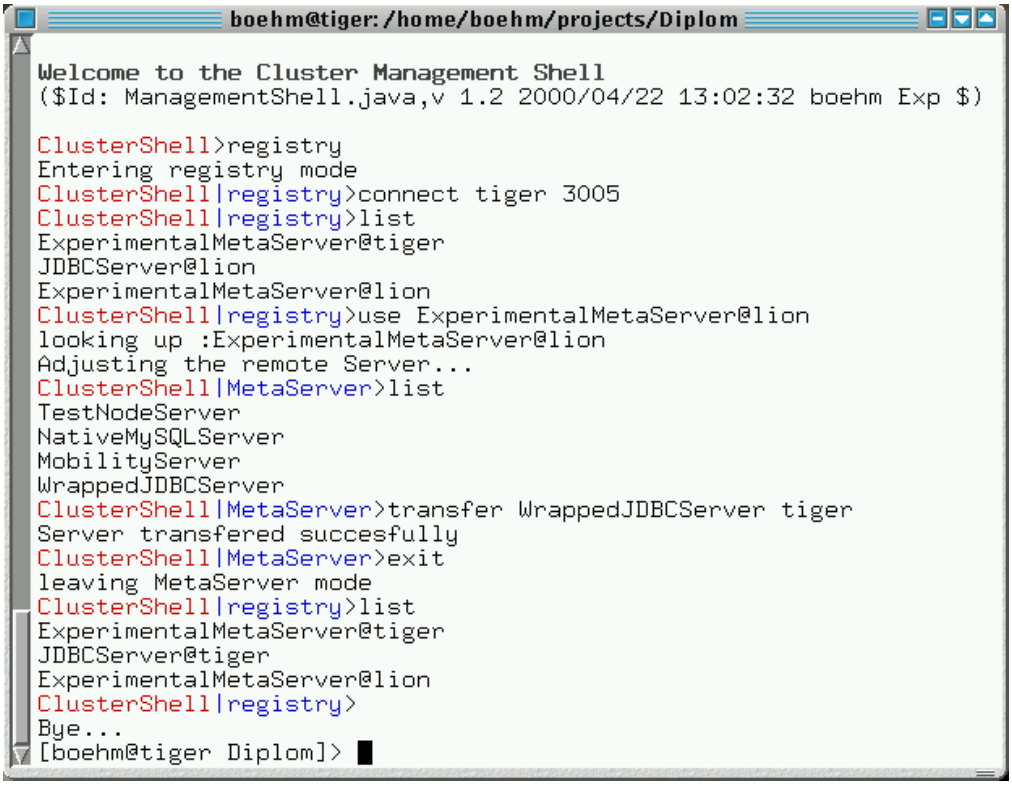


Abbildung 3.14: Konsolenbasierte Verwaltung der MetaServer-Plattform

Im nächsten Schritt wird der JDBCServer zu einer anderen Plattform transferiert, um zu zeigen, wie die Funktionalität eines EmbeddedServer als Bestandteil der MetaServer-Plattform genutzt und über das ManagementInterface gesteuert werden kann. Anschließend wird die Subshell verlassen, was automatisch zur Registry-Shell zurückführt, in der wir uns zuvor befunden haben. Die erneute Anzeige des Verzeichnisinhaltes zeigt, daß der transferierte Server jetzt auf der anderen Plattform über einen neuen Namen erreichbar ist.

Kapitel 4

Anwendungsspezifische Dienste für die MetaServer-Plattform

Die Implementierung der eigentlichen Funktionalität des Applikationsservers ist Gegenstand dieses Kapitels. Aufbauend auf der grundlegenden Funktionalität der MetaServer-Plattform, die im vorangegangenen Kapitel ausführlich dargestellt wurde, reichern die im folgenden dargestellten Komponenten die Architektur des Servers mit applikationsspezifischer Funktionalität an, die in verschiedene Module aufgeteilt ist.

Unter Ausnutzung der durch die einzelnen Komponenten bereitgestellten Funktionen können Anwendungen auf die Clusterarchitektur zugreifen und deren Dienste nutzen. Ein wesentlicher Gesichtspunkt ist hierbei die Tatsache, daß für die Nutzung der Clusterdienste keine Programmänderungen an den bereits vorhandenen Anwendungen nötig sind und damit die Verwendung der verteilten Cluster-Architektur für den Anwender völlig transparent bleibt, wobei gleichzeitig die Komplexität und die Besonderheiten des Systems verborgen bleiben. Außerdem erfolgt keine Festlegung auf die durch den Cluster vorgegebene Infrastruktur, da die Applikationen auch weiterhin in ihrem ursprünglichem Umfeld einsetzbar bleiben.

Die realisierten Anwendungen verfolgen das Ziel, die momentan innerhalb des Wortschatzprojektes eingesetzten oder in Entwicklung befindlichen Anwendungen zu unterstützen. Durch die Kapselung der bereitgestellten Funktionalität in einzelne Module und deren Integration in eine gemeinsame Plattform ist auch eine zukünfige Erweiterung des Applikationsservers um weitere Funktionen möglich.

Wie bereits bei der Analyse korpuslinguistischer Software in Kapitel 2 festgestellt, sind die verwendeten Verfahren sehr stark datenzentriert und verwenden für die Datenhaltung meist eine (relationale) Datenbank. Aus diesem Grund wurde der Realisierung eines transparenten Zugriffs auf die Daten eines auf mehreren Knoten replizierten Datenbanksystems besondere Aufmerksamkeit geschenkt. Um die Implementierungsalternativen hinsichtlich ihrer Vor- und Nachteile gegeneinander abwägen zu können und einer breiten Palette von Anwendungen Zugriff auf die im Cluster zur Verfügung gestellten Daten bieten zu können, wurden zwei verschiedene Zugangspfade zur den Datenbankinhalten realisiert: Der *MySQL-Server*, der

in Abschnitt 4.1 beschrieben wird, verwendet für die Kommunikation mit den Klienten das datenbankabhängige proprietäre Protokoll des MySQL-DBMS, während der *JDBC-Server* dessen Darstellung in Abschnitt 4.2 auf Seite 102 erfolgt, auf der von Sun Microsystems standardisierten API fußt und damit auch die Verwendung anderer DBMS möglich macht.

Für die Abarbeitung ressourcen- und zeitintensiver, nichtinteraktiver Applikationen wurde mit der Implementierung des *ScriptServers* eine Anwendung geschaffen, die anhand einer Prozeßbeschreibung die einzelnen Arbeitsschritte des Prozesses isoliert, auf die zur Verfügung stehenden Betriebsmittel im Cluster verteilt und damit die Ausführungszeit optimiert. Aufbau und Funktionsweise dieser Anwendung werden in Abschnitt 4.3 auf Seite 117 erläutert.

Den Abschluß des Kapitels bildet die Beschreibung der Implementierung des *BinaryServers* in Abschnitt 4.4, durch den der realisierte Applikationsserver auch Anwendungen unterstützen kann, die ursprünglich nicht für die Ausführung in einer Client/Server-Umgebung vorgesehen waren. Anhand eines Programms zur Berechnnung von Anagrammen[1] wird die Praxistauglichkeit dieses Konzepts demonstriert.

4.1 MySQL-Server

Eine wesentliche Anforderung an den Cluster stellt der transparente Zugriff auf das MySQL-Datenbanksystem dar. Dabei sollten möglichst viele verschiedene Programmiersprachen und Plattformen auf die Daten zugreifen zu können, ohne die bereits exisitierenden Anwendungen für den Einsatz auf dem Cluster anpassen zu müssen. Außerdem soll möglichst die gesamte Funktionalität des MySQL-DBMS auch im Cluster uneingeschränkt zur Verfügung stehen.

Um diese Vorgaben zu erreichen, wurde mit dem *MySQL-Server* eine Anwendung für die MetaServer-Plattform geschaffen, die Klienten durch Verwendung des proprietären Protokolls, das von der MySQL-Datenbank verwendet wird, Zugriff auf die Datenbankinhalte der Knoten-DBMS bietet. Damit wird erreicht, daß sich die Clustersoftware anwenderseitig wie ein MySQL-DBMS verhält und damit die breite Palette von Treibern und Bibliotheken zum Einsatz kommen kann, die für die Anbindung an solche Systeme zur Verfügung stehen.

Den Vorteilen, die dieser Ansatz bietet, stehen allerdings auch Nachteile entgegen. Zunächst wird durch die Festlegung auf ein bestimmtes Protokoll eine direkte Abhängigkeit zu dem MySQL-DBMS geschaffen, die sich später nur mit hohem Aufwand eliminieren läßt, was im Falle eines DBMS-Wechsels schwerwiegende Folgen haben kann. Da das verwendete Protokoll zudem proprietär ist und keinem Standard entspricht, wird eine korrekte Implementierung durch ungenügende Dokumentation erschwert und eine ständige Anpassung an eventuelle Protokolländerungen notwendig.

[1] Ein Anagramm ist ein Wort, das durch Vertauschen von Buchstaben innerhalb eines Ursprungswortes entstanden ist.

4.1.1 Architektur

Zur Umsetzung der oben dargestellten Anforderungen, wurde ein MySQL-MetaServer entworfen, der als EmbeddedServer innerhalb der MetaServer-Plattform läuft und unter Verwendung des proprietären MySQL-Protokolls sowohl auf die DBMS der einzelnen Knoten zugreift, als auch mit den Klienten kommuniziert.

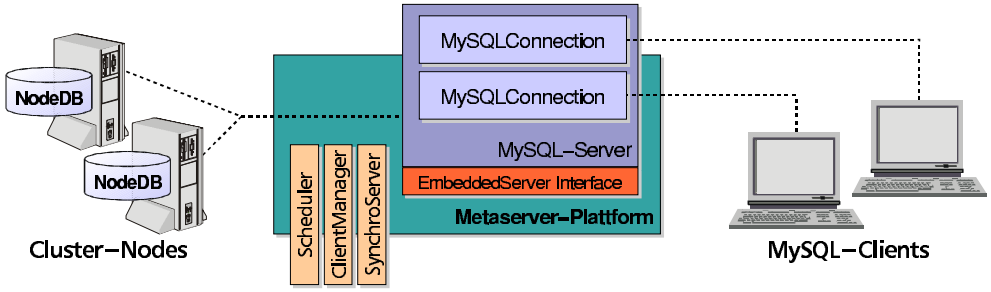


Abbildung 4.1: Architektur des MySQL-MetaServers

Dem Server kommt dabei hauptsächlich die Aufgabe zu, die Anfragen von den verschiedenen Klienten an die verfügbaren Clusterknoten zu vermitteln, sowie alle weiteren Aufgaben zu realisieren, die in diesem Zusammenhang entstehen. Hierzu zählen beispielsweise der Anmeldevorgang oder die Serialisierung von schreibenden Zugriffen auf die Datenbanken. Dabei werden diese sekundären Funktionen nicht im Server selbst realisiert, sondern es wird auf von der MetaServerplattform angebotene Dienste zurückgegriffen. Im einzelnen betrifft dies die folgenden Komponenten:

- **Scheduler**: stellt für eine zu vermittelnde Anfrage des Klienten einen freien Clusterknoten bereit. Dabei werden sowohl Lastverteilungs- und Verfügbarkeitsaspekte als auch Spezifika der jeweiligen Anfrage berücksichtigt.
- **ClientManager**: stellt Informationen über die Verbindungseigenschaften der Benutzer bereit und implementiert einen Cache-Speicher für verfügbare Verbindungen, der das Anmeldeverfahren beschleunigen soll.
- **SynchroServer**: dient der Serialisierung der schreibenden Zugriffe der Klienten auf die Datenbank und stellt deren Konsistenz untereinander sicher.

Eine Grundvoraussetzung für die Realisierung der oben skizzierten Architektur war zunächst die Analyse und das Verständnis des verwendeten Protokolls, dessen Darstellung Gegenstand des nächsten Abschnitts ist. Anhand entsprechender Schemata und realer MySQL-Pakete, die als Beispiele dienen, wird demonstriert, wie die Kommunikation zwischen Server und Klient funktioniert.

4.1.2 Das MySQL-Protokoll

Datenbankanwendungen können auf den MySQL-Server entweder lokal über Unix-Sockets oder über TCP/IP-Sockets bei Verwendung eines Netzwerkes zugreifen. Für unsere Anwendungen ist primär der Zugriff über das Netzwerk von Interesse, bei dem durch die Verwendung des TCP-Protokolls (eine Beschreibung des Protokolls findet sich beispielsweise in [Tan92], Seite 518 ff.) bereits sichergestellt wird, daß Unzuverlässigkeiten des Netzwerkes wie Übertragungsfehler durch verlorene oder mehrfach gesendete Pakete bereits auf Transportebene erkannt und vermieden werden.

Auf der Applikationsebene erfolgt die Kommunikation mit dem DBMS über ein proprietäres Protokoll, das leider nur spärlich dokumentiert ist. Primäre und lange Zeit einzige Informationsquelle zu Bedienung und Funktionalität des DBMS ist die mitgelieferte Online-Dokumentation [MyS]. Aufgrund steigender Popularität des DBMS sind mittlerweile auch einige Bücher erschienen, die die Arbeit mit MySQL mehr oder weniger detailliert beschreiben, beispielsweise [Ran99] und [DuB99, DuB00]. Allen Publikationen ist jedoch gemeinsam, daß meist nur die Bedienung des DBMS im allgemeinen und der Zugriff von verschiedenen Plattformen und Programmiersprachen aus detailliert erläutert werden, während ausführliche Informationen über grundlegende Funktionen des Systems keine Erwähnung finden. Auch eine Recherche im Internet und den verschiedenen Newsgruppen, die sich mit MySQL beschäftigen, lieferte keine direkt verwertbaren Informationen:

> Martin Ramsch (m.ramsch@computer.org): *"As far as I know you have to refer to the source code..."*
>
> Tõnu Samuel (tonu@mysql.com): *"It's not documented elsewhere than in source. But it is very easy to understand. Protocol is simple."*
>
> Mark Matthews (mmatthew@worldserver.com): *"The documentation is the Mysql source. There is no formal documentation that outlines the protocol. The protocol itself is really simple, the hardest part being the handshaking and authentication. The rest is easy as all that is sent back and forth are strings (zero-terminated or length-specified), ints, and longs (for status/error reports)."*

Den Empfehlungen der Entwickler folgend habe ich das Protokoll dann mit Hilfe der verfügbaren Quelltexte[2] und durch Beobachtung des Datenverkehrs zwischen Klient und Server analysiert.

Das MySQL-Protokoll baut auf einer TCP-Verbindung auf und stellt damit eine Verbindung zwischen Klient und Server zur Verfügung, die wie ein normaler Datenstrom behandelt werden kann, auf den lesend und schreibend zugegriffen wird. Auf dieser Grundlage tauschen Server und Klient wechselseitig Informationen aus, die in spezifischen Datenpaketen gekapselt werden. Die Kommunikation erfolgt

[2] Der Quelltext des MySQL-DBMS ist zwar frei verfügbar, aber relativ umfangreich und aufgrund der vielen Plattformabhängigkeiten nur schwer lesbar. Aus diesem Grund stellt die JDBC-Treiber-Implementierung von Mark Matthews die Grundlage der Analyse dar

dabei immer nur in eine Richtung, d.h. solange der Server Daten an den Klient übermittelt, kann dieser keine weiteren Anforderungen an den Server schicken. Dies gilt auch für den umgekehrten Fall[3]. Bei umfangreicheren Datenübertragungen kann es durchaus vorkommen, daß die Daten in mehrere Pakete aufgeteilt werden, die dann als Paketsequenz übertragen werden (siehe Abbildung 4.2).

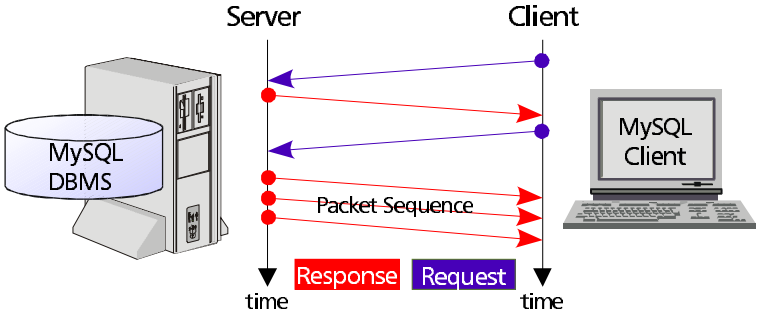


Abbildung 4.2: Datenaustausch zwischen MySQL-Server und -Klient

Die aktuellen Versionen des MySQL-DMBS[4] verwenden die Protokoll-Version 10, während MySQL in der Version 3.21.xx noch die Protokoll-Version 9 einsetzt. Die Definitionen der Protokollversionen wird nicht strikt gehandhabt, da kleinere Änderungen am Protokoll nicht notwendigerweise zu einer Veränderung der Version führen[5]. Im folgenden beziehe ich mich, sofern nicht anders angegeben, immer auf die aktuelle Protokoll-Version.

4.1.2.1 Datentypen

Für die Übertragung der Daten benutzt MySQL verschiedene Datentypen, die in Tabelle 4.1 aufgeführt sind. Hierbei handelt es sich lediglich um die Datentypen, die innerhalb des Protokolls für den Transport der eigentlichen Nutzdaten verwendet werden – es wird jedoch keine Aussage über die im DBMS verfügbaren Datentypen getroffen.

[3] Es gibt durchaus Fälle, in denen eine solche echte bidirektionale Kommunikation wünschenswert wäre; z. B. könnte so die Übertragung einer umfangreichen Ergebnismenge vorzeitig auf Wunsch des Klienten abgebrochen werden.

[4] Momentan sind zwei verschiedene Versionsfamilien von MySQL aktuell: Version 3.22.xx ist die vom Hersteller als stabil bezeichnete Variante und wird, abgesehen von der Beseitigung enthaltener Fehler, nicht weiterentwickelt; Version 3.23.xx ist die Entwicklerversion, in die alle Verbesserungen und Neuentwicklungen einfließen

[5] Bei der Implementierung trat dieser Fall für das `processlist`-Kommando innerhalb verschiedener 3.23.xx-Versionen auf

Datentyp	Länge in Bit	Erläuterung
byte	8	repräsentiert einen vorzeichenbehafteten Byte-Datentyp mit einem Wertebereich von -127...126
int	16 oder 24	ein Datentyp für ganzzahlige vorzeichenlose Werte im Bereich von 0...65.535 (16 Bit) bzw. 0...16.777.215 (24 Bit).
long	32 oder 64	ein Datentyp zur Darstellung von großen ganzzahligen Werten. Die 32 Bit breite Darstellung ist vorzeichenlos und erlaubt Werte von 0...4.294.967.295, während die 64 Bit Variante vorzeichenbehaftet ist und Werte im Bereich von −9.223.372.036.854.775.808...9.223.372.036.854.775.807 zuläßt.
string	$n+1$ oder n	Für diesen Datentyp gibt es zwei verschiedene Formate: Bei der längenbegrenzten Variante enthält das erste Byte des Datums die Länge des Strings und begrenzt damit die mögliche Gesamtlänge auf 256 Byte. Die zweite Variante arbeitet mit einem terminierenden Null-Byte, das das Ende der Zeichenkette signalisiert. Hier gibt es keine prinzipielle Längenbeschränkung. In einigen Fällen wird auf eine spezielle Terminierung verzichtet; in diesem Fall ist mit dem Ende des Pakets auch das Ende der Zeichenkette erreicht.

Tabelle 4.1: Datentypen des MySQL-Protokolls

4.1.2.2 Genereller Aufbau der Pakete

Das Protokoll verwendet spezifische Datenpakete, die den in Abbildung 4.3 dargestellten Aufbau haben.

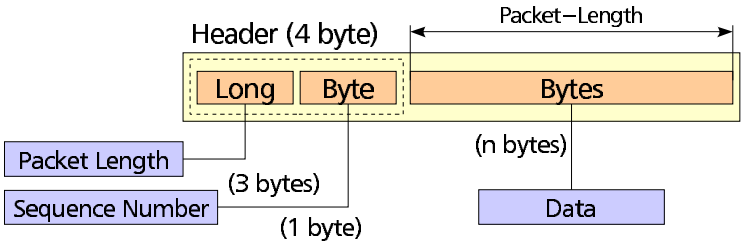


Abbildung 4.3: genereller Aufbau eines MySQL-Pakets

Das Paket besteht aus einem Header, der Informationen über die Länge des Paketes (Packet Length) enthält, und einer Sequenznummer. Diese Zahl dient der Nummerierung der einzelnen Pakete innerhalb einer Sequenz und beginnt für jeden Übertragungszyklus erneut bei 0 und wird schrittweise erhöht. Da die richtige Reihenfolge der Pakete bei TCP generell garantiert ist, erscheint diese zusätzliche Nummerierung redundant, wird aber wahrscheinlich dazu verwendet, um das Ende eines Übertragungsabschnitts zuverlässig zu erkennen[6].

[6]Daß das Sequenzpaket tatsächlich für die Reihenfolgebestimmung genutzt wird, wurde experimentell überprüft, indem die Sequenznummer für alle Pakete einer Sequenz auf null gesetzt wur-

Da die Sequenznummer ein Bytewert ist, sind eigentlich nur Paketsequenzen bis zu einer maximalen Länge von 256 Paketen möglich. Es gibt jedoch durchaus Fälle, in denen längere Sequenzen übertragen werden müssen (z. B. umfangreiche Ergebnismengen einer Anfrage, die vom Server an den Klienten geschickt werden). Die Beobachtung des DBMS zeigte, daß in diesen Fällen der Zähler auf null gesetzt und dann wie gewohnt inkrementiert wird. Dem vier Byte langen Header folgt der Datenteil, der die eigentlichen Nutzdaten enthält und dessen Länge dem Headerwert Packet Length entspricht. Die maximale Größe des Datenpakets ergibt sich einerseits durch den maximalen Wertebereich eines Long-Wertes (siehe Tabelle 4.1) und andererseits durch die Parametrisierung des DBMS[7].

In den folgenden Darstellungen von Paketen wird der Header nicht immer explizit angegeben, sondern als implizit vorhanden vorausgesetzt.

4.1.2.3 Fehlerpakete

Tritt bei der Bearbeitung der Anfrage des Klienten ein Fehler auf, so sendet der Server ein spezielles Fehlerpaket zurück, das die in Abbildung 4.4 dargestellte Struktur hat.

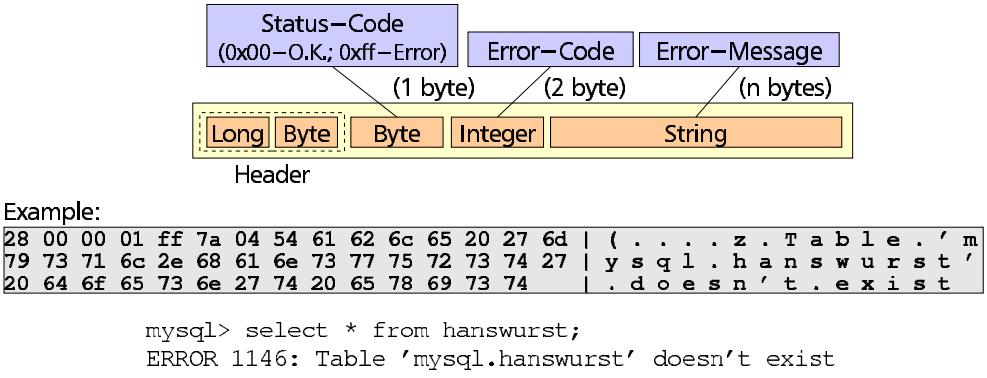


Abbildung 4.4: MySQL-Fehlerpaket

Dieses Fehlerpaket enthält zunächst einen Statuscode, der das Paket als solches identifiziert. Dies ist notwendig, da der Klient in einigen Fällen vom Server eine Rückmeldung über den Erfolg oder das Fehlschlagen einer Aktion erwartet. Im Fehlerfall hat das Statusfeld den festen Wert `0xff`, ansonsten den Wert `0x00`.

Sollte es sich wirklich um einen Fehler handeln, folgt ein zwei Byte langer Fehlercode, ergänzt durch eine aussagefähige Fehlerbeschreibung. Dabei handelt es sich um eine Zeichenkette, die ohne Terminierung bis zum Ende des Pakets reicht.

den. In diesem Fall wurden die Pakete nicht mehr akzeptiert. Daraus läßt sich schließen, daß man annimmt, daß die Sicherstellung der Paketreihenfolge über 256 fortlaufende Pakete in der Praxis ausreichend ist.

[7] Der einstellbare Parameter `max_allowed_packet` gibt die maximale Größe eines MySQL-Pakets an.

4.1.2.4 Loginprozeß

Die Anmeldeprozedur des Klienten an der MySQL-Datenbank erfolgt durch ein mehrstufiges Request-Response-Verfahren und stellt sicher, daß sich der Benutzer richtig authentifiziert. Dabei werden einige Sicherheitsmaßnahmen beachtet, die den unbefugten Zugriff auf die Daten des Systems von vornherein verhindern sollen.

Wenn sich der Klient zu dem Server verbindet, sendet letzterer zunächst ein initiales Paket, dessen Aufbau in Abbildung 4.5 dargestellt ist.

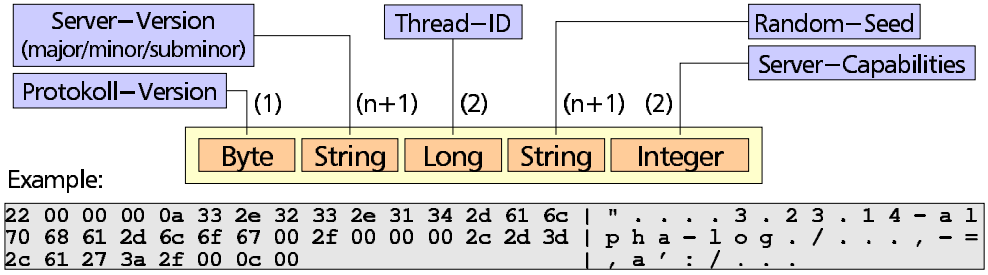


Abbildung 4.5: initales Paket vom Server bei Klientverbindung

Dieses Paket enthält zunächst Versionsinformationen wie etwa die verwendete Protokollversion und die Version des Serverprozesses. Die Thread-ID identifiziert den Thread innerhalb des MySQL-Servers, der für die aufgebaute Verbindung zuständig ist und kann bei Bedarf dazu verwendet werden, um die Sitzung zu beenden (mit dem `kill`-Kommando). Die Server-Capabilities liefern Informationen über die speziellen Fähigkeiten dieses Servers, beispielsweise darüber, ob eine Komprimierung des Datenverkehrs zwischen Server und Klient unterstützt wird. Die für den Klienten wichtigste Information ist jedoch der *Seed*, eine Zeichenkette fester Länge, die vom Server bei jeder Verbindungsanfrage neu generiert wird und vom Klienten zur Verschlüsselung des Paßwortes benutzt wird. Der Seed soll verhindern, daß ein Unbefugter Zugriff auf den Server erhalten kann, indem er die Pakete, die während des Loginprozesses zwischen Server und Klient ausgetauscht werden, abfängt und später dazu benutzt, um sich mit Hilfe der abgefangenen Pakete mit dem Server zu verbinden[8].

Im nächsten Schritt sendet der Klient ein Paket zum Server zurück, dessen Struktur in Abbildung 4.6 wiedergegeben ist und das die Anmeldeinformationen des Benutzers enthält. Dieses Paket muß innerhalb einer bestimmten Zeitspanne an den Server zurückgesendet werden, da dieser sonst die Verbindung automatisch trennt[9].

Der Benutzername wird hierbei im Klartext übertragen und das Paßwort wird mit Hilfe des vom Server gesendeten Seeds verschlüsselt. Optional kann in dem Paket noch der Name der Datenbank angegeben werden, die der Klient verwenden möchte. Bei allen Zeichenketten handelt es sich um null-terminierte Strings, wo-

[8] Diese Variante des Einbruchs in ein System wird als “man in the middle attack” bezeichnet

[9] Diese Trennung erfolgt einerseits aus Sicherheitssgründen und stellt andererseits sicher, daß der Server nicht unnötig viele Socket-Verbindungen aufrecht erhält.

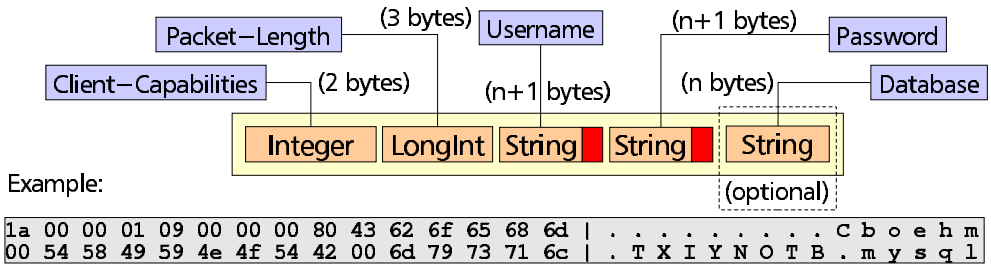


Abbildung 4.6: Aufbau des Anmeldepaketes des Klienten

bei bei der jeweils letzten Zeichenkette (also dem Paßwort oder dem Datenbanknamen) das terminierende Nullbyte fehlt.

Die Client-Capabilities spezifizieren die variablen Protokolleigenschaften, die der Klient unterstützt bzw. benutzen möchte. Hierzu zählen unter anderem die Art der Paßwortverschlüsselung[10] und die Angabe, ob die zu übertragenden Daten komprimiert werden sollen oder nicht.

Wie aus der schematischen Darstellung zu erkennen ist, wird die Paketlänge bei dem Anmeldepaket innerhalb der Daten erneut angegeben, obwohl dies eigentlich nicht notwendig ist, da diese Angabe ja schon im Header enthalten ist. Hinzu kommt, daß einige Klienten hier nicht die korrekte Länge, sondern einen festen Wert angeben, der, sofern er groß genug gewählt ist, vom Server auch akzeptiert wird. So sendet z. B. der MySQL-Monitor die feste Bytefolge `0x00 0x00 0x80`, während der MySQLTreiber von Mark Matthews die Länge korrekt berechnet.

Im letzten Schritt wird das gesendete Paket vom Server ausgewertet und der Klient authentifiziert, wenn die Anmeldeinformationen korrekt sind. Im Fehlerfall wird der Klient mit einer Fehlermeldung abgewiesen. Das vom Server gesendete Antwortpaket hat hierbei die übliche Struktur eines Fehlerpaketes (siehe Abbildung 4.4 auf Seite 90).

Anhand des Status-Codes kann der Klient feststellen, ob seine Anmeldung erfolgreich war (Code: `0x00`) oder nicht (Code: `0xff`). Wenn die Anmeldung fehlgeschlagen ist, wird die Verbindung zu dem Klienten vom Server aus Sicherheitsgründen automatisch getrennt, so daß sich der Klient erneut eine Verbindung aufbauen muß, um die Anmeldung zu wiederholen. Außerdem registriert der Server jeden fehlgeschlagenen Anmeldeversuch und verweigert dauerhaft jeglichen Zugriff auf die Datenbank, sobald die Anzahl der abgewiesenen Anmeldungen pro Zeiteinheit einen gewissen Wert überschreitet[11].

[10] MySQL unterstützt zwei verschiedene Verfahren der Paßwortverschlüsselung: ein “altes” Verfahren, das von Servern der Version 3.21.xx benutzt wird und ein “neues” Verfahren, welches von den aktuellen Versionen verwendet wird.

[11] Diese Sperrung kann mit Hilfe des Administrationskommandos `flush hosts` durch den Serververwalter wieder aufgehoben werden

4.1.2.5 Kommandos

Anfragen an das MySQL-DBMS erfolgen durch eine Reihe von Kommandos, die durch einen speziellen Code gekennzeichnet werden und unter Umständen weitere Informationen für die Abarbeitung des Kommandos enthalten. Tabelle 4.2 zeigt die verschiedenen Kommandos:

Tabelle 4.2: MySQL-Kommandos

Kommando	Code	zusätzl. Daten	Erläuterung
sleep	0	—	Operation ohne weitere Funktion.
quit	1	—	Beendet die bestehende Sitzung und trennt die Verbindung zwischen Klient und Server
init DB	2	Datenbankname	initialisiert die Datenbank, die für die weiteren Anfragen verwendet wird
query	3	Query	führt eine Anfrage auf der Datenbank aus. Dabei kann es sich sowohl um eine lesende (`SELECT`) als auch um eine schreibende Abfrage (`UPDATE` oder `DELETE`) handeln.
field list	4	Tabellenname	liefert die für die angegebene Tabelle definierten Felder. Dies wird z. B. vom MySQL-Monitor dazu benutzt, die Feldnamen einzulesen, um eine automatische Ergänzung auf der Kommandozeile zu ermöglichen
create DB	5	Datenbankname	erzeugt eine neue Datenbank mit dem angegebenen Namen
drop DB	6	Datenbankname	löscht eine Datenbank incl. aller in ihr enthaltenen Tabellen und Daten
reload/flush	7	Ziel	setzt Parameter, die aus Geschwindigkeitsgründen vom DBMS im Speicher vorgehalten werden, zurück bzw. liest sie erneut vom externen Datenträger ein. Der Parameter gibt hierbei die betroffene Zielgröße an. Dies betrifft z. B. die in Datenbanktabellen gespeicherten Benutzerinformationen, die Logdateien und andere Parameter.
shutdown	8	—	initiiert das Herunterfahren des DBMS und schließt anschließend die Verbindung zum Klienten.
statistics	9	—	liefert statistische Informationen über die aktuelle Verbindung zum Server (z. B. Thread-ID, Server- und Protokoll-Version, laufende Abfragen, offene Tabellen etc.)
process info	10	—	liefert eine Übersicht über die laufenden Prozesse auf dem DBMS

connect	11	Datenbankname	verbindet den Klienten mit den bekannten Anmeldeinformationen erneut zu der Datenbank mit dem angegebenen Namen. Die Sitzung erhält eine neue Thread-ID und entspricht damit einer neuen Verbindung[12].
process kill	12	Prozess-ID	beendet den Prozeß mit der angegebenen ID sofort, ohne auf eventuelle Datenverluste Rücksicht zu nehmen
ping	14	—	Kann verwendet werden, um die Verfügbarkeit des DBMS zu überprüfen

Ein typisches Kommandopaket enthält neben dem Header also den Kommandocode, der die Art der Anfrage klassifiziert, und optional einen weiteren Datenbereich, in dem zusätzliche Anfrageinformationen in einem null-terminierten String abgespeichert sind. Diese Struktur hat das in Abbildung 4.7 dargestellte Aussehen.

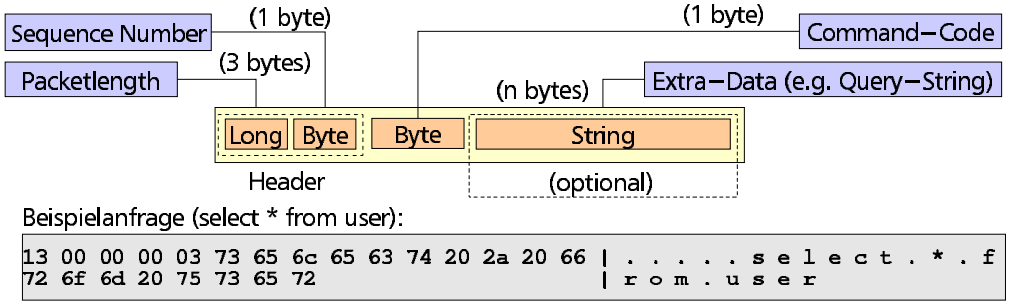


Abbildung 4.7: Aufbau eines MySQL-Kommando-Pakets

4.1.2.6 Aufbau der Resultatsetpakete

Wenn das Resultat der Anfrage an den Klienten zurückgeschickt werden soll, gibt es prinzipiell zwei sich wesentlich unterscheidende Anwendungsfälle. Einerseits gibt es Anfragen und Kommandos, die die zugrundeliegende Datenbankstruktur modifizieren und an den Klienten nur den Erfolg der Operation zurückmelden müssen (z. B. `INSERT`, `UPDATE`, `DELETE` etc.). Andererseits gibt es Abfragen, die aus der bestehenen Datenbasis eine Teilmenge auswählen und an den Klienten zurückliefern (`SELECT`-Anfragen). Dies führt zu folgenden Schwierigkeiten:

- Die Größe der Resultatmenge ist nicht von vornherein bekannt, potentiell muß der gesamte Datenbankinhalt übertragen werden können

[12] In der Praxis konnte dieses Kommando nicht beobachtet werden, da alle reconnect-Versuche von untersuchten Klienten auf die herkömmliche Weise erfolgten, d.h. den beschriebenen Login-Prozeß benutzten, um eine neue Verbindung mit dem Server aufzubauen.

- Die Struktur der zu übertragenden Datenmenge ist nicht bekannt, da er sowohl von der Datenbankstruktur selbst als auch von der einzelnen Anfrage abhängig ist

Aus diesem Grund erfolgt die Übertragung in einem mehrstufigem Verfahren, welches im folgenden beschrieben und anhand eines Beispiels verdeutlicht wird. Als Beispiel wurde folgende Anfrage an die `mysql`-Datenbank gesendet:

```
mysql> SELECT DISTINCT User, password, Select_priv
       FROM user WHERE user LIKE '%boehm';
+--------+------------------+-------------+
| User   | password         | Select_priv |
+--------+------------------+-------------+
| boehm  | 7328ac8b7bee65bd | Y           |
| Cboehm | 30d9d89d698f7357 | Y           |
+--------+------------------+-------------+
2 rows in set (0.04 sec)
```

Die Resultatmenge wird in Tabellenform mit einer festen Anzahl von Spalten und einer variablen Anzahl von Zeilen zurückgeliefert. Dabei haben alle Elemente einer Spalte denselben Typ.

Zunächst wird in einem einzelnen Paket die Anzahl der im Resultatset enthaltenen Spalten zurückgesendet.

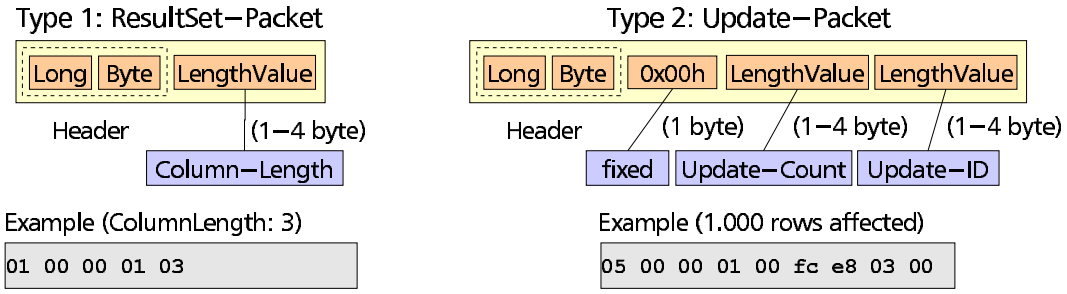
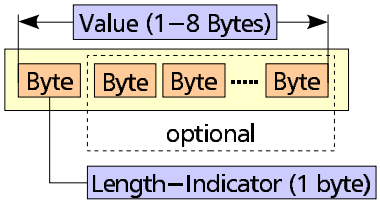


Abbildung 4.8: Resultatmenge, Schritt 1: Anzahl der Spalten

Hierfür wird der Wert *ColumnLength* in einem variablen Zahlenformat an den Klienten gesendet. Um die beiden Anfragetypen zu unterscheiden, legt das Protokoll fest, daß Update-Anfragen immer einen ColumnLength-Wert `null` zurückliefern (siehe Abbildung 4.8, Typ 2), der von der Anzahl der betroffenen Zeilen und einer Update-ID gefolgt wird. Damit ist die Rückmeldung an den Klienten abgeschlossen und der Übertragung folgen keine weiteren Pakete.

Sollte es sich bei der Abfrage, jedoch um eine `SELECT`-Abfrage handeln, so hat das Paket die in Abbildung 4.8, Typ1 dargestellte Struktur. In unserem Beispiel ist dies der Fall und die zurückgelieferte Tabellenstruktur hat drei Spalten.

Die in dem ersten Datenpaket zurückgelieferten numerischen Werte weisen eine bestimmte Struktur auf, für die die Bezeichnung *LengthValue* gewählt wurde.



Indicator	Length/Bytes	Type
0x00–0xfa	1	Byte
0xfb	1	0 (fixed)
0xfc	2	Integer
0xfd	3	LongInteger
0xfe	4/8	Long
0xff	–	(reserved)

Abbildung 4.9: Kodierung von längenvariablen Zahlen

Das Besondere an dieser Datenstruktur ist, daß sie je nach Bedarf unterschiedliche Länge und Wertebereiche besitzen kann. Erreicht wird dies, indem das erste Byte der Struktur nicht nur numerische Informationen, sondern auch Informationen über die Gesamtlänge des Wertes enthält. Hierfür wurde der Wertebereich des ersten Bytes auf 0...250 eingeschränkt und den verbleibenden Werten eine besondere Bedeutung zugewiesen, anhand derer Typ und Länge der Struktur erkannt werden kann (siehe Tabelle in Abbildung 4.9). Damit ist es möglich, je nach Bedarf große ganzzahlige Werte effizient übertragen zu können. Anzumerken bleibt, daß der Wert 0 speziell kodiert wurde und daß es seit Version 3.22.5 eine Erweiterung der Struktur auf 64-bit Long-Werte gegeben hat, durch die sich die Gesamtlänge von vier auch acht Byte erweiterte. Der spezielle Wert `0xff` wurde reserviert, um Fehlermeldungen zuverlässig erkennen zu können.

Als nächstes folgt eine Beschreibung der Struktur der Tabelle, wobei jede Spalte in einem eigenen Paket beschrieben wird.

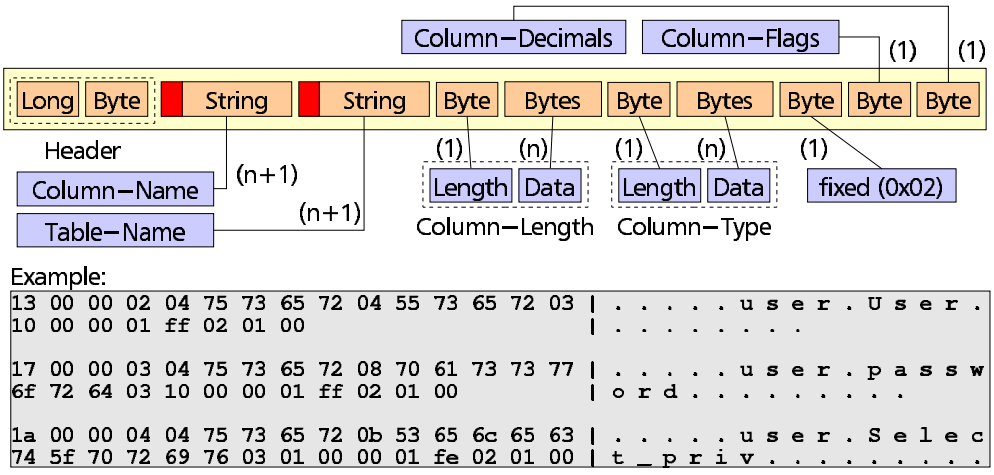


Abbildung 4.10: Resultatmenge, Schritt 2: Beschreibungsdaten für die einzelnen Spalten

Jedes Paket hat dabei die in Abbildung 4.10 dargestellte Struktur und enthält alle Informationen, die notwendig sind, um eine Spalte der Ergebnismenge zu beschreiben. Zunächst wird der Spalten- und der Tabellenname als längenkodierter String übertragen. Es folgt die Spezifikation der Länge oder Feldbreite und des Datentyps der Spalte, wobei hierfür die in Tabelle 4.3 angegebene Kodierung verwendet

wird. Da diese Informationen je nach verwendetem Zahlenformat unterschiedliche Längen haben können, wird hier eine der Längenkodierung von Zeichenketten ähnliche Darstellung verwendet.

Datentyp	Code	Datentyp	Code	Datentyp	Code
decimal	0	tiny	1	short	2
long	3	float	4	double	5
null	6	timestamp	7	long (64 bit)	8
int24	9	date	10	time	11
datetime	12	year	13	newdate	14
enum	247	set	248	tiny_blob	249
medium_blob	250	long_blob	251	blob	252
var_string	253	string	254		

Tabelle 4.3: Kodierung der unterstützten MySQL-Datentypen

Wenn die Metadaten komplett übertragen wurden, wird ein spezielles Paket geschickt, das den Beginn der Übertragung der eigentlichen Daten ankündigt[13].

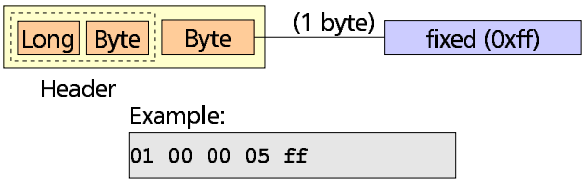


Abbildung 4.11: Resultatmenge, Schritt 3: Trennpaket zwischen Meta- und Nutzdaten

Es folgt die Übertragung der eigentlichen Ergebnismenge, wobei jedes Paket einer Zeile in der Tabellenstruktur entspricht (siehe Abbildung 4.12).

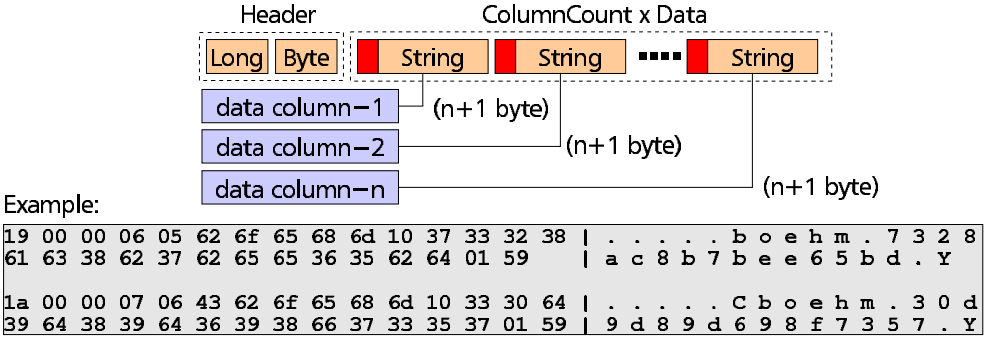


Abbildung 4.12: Resultatmenge, Schritt 4: Aufbau des Nutzdatenpakets

Hierbei fällt auf, daß die einzelnen Daten immer als längenkodierter String übertragen werden, unabhänig von dem Datentyp, den sie eigentlich besitzen. Dies hat zur

[13] eigentlich ergibt sich die Anzahl der zu übertragenden Metadatenpakete schon aus der vorher gesendeten Zeilenanzahl; dieses Paket ist also redundant

Konsequenz, daß erstens Strings nur eine maximale Länge von 256 Zeichen haben können und zweitens die String-Signatur jedes Datentyps nicht länger sein darf als die maximale Länge eines Strings[14]. Außerdem entsteht durch diese Art der Übertragung zusätzlicher Konvertierungsausfwand sowohl auf Klient- als auch auf Serverseite.

Sind alle Daten übertragen, so wird zum Abschluß ein spezielles Daten-Ende-Paket vom Server gesendet, das den Abschluß der Übertragung kennzeichnet.

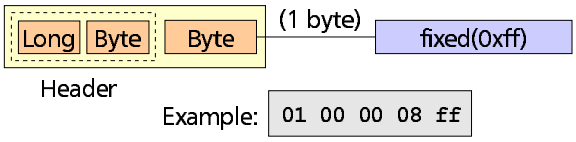


Abbildung 4.13: Resultatmenge, Schritt 5: Paket, das die Übertragung abschließt

Zu jeden Zeitpunkt dieser Übertragung kann ein Fehler auftreten, der die weitere Übertragung von Daten abbrechen läßt. In diesem Fall wird ein Fehlerpaket mit der bereits beschriebenen Struktur (siehe Abbildung 4.4 auf Seite 90) gesendet, das den Fehler kennzeichnet und beschreibt. Sollte ein Fehler gerade dann auftreten, wenn das Ende der Datenübertragung oder das Trennpaket zwischen Nutz- und Metadaten erwartet wird, so kann es zu Problemen bei der Erkennung des Fehlers kommen, da in beiden Fällen das gleiche Statusfeld verwendet wird (`0xff`).

4.1.3 Implementationsumfang und Vollständigkeit

Der implementierte MySQL-MetaServer ist in der Lage, alle Kommandos und Anfragen (siehe Tabelle 4.2 auf Seite 93) zu bearbeiten, die ein aktuelles MySQL-DBMS[15] akzeptiert. Einzige Ausnahme ist hier die Vearbeitung von BLOBs[16], die momentan innerhalb des Wortschatzprojektes nicht benötigt werden.

Teilweise war es jedoch notwendig, verschiedene Anpassungen bei den einzelnen Kommandos vorzunehmen, um ein für den Cluster adäquates Verhalten zu erreichen. Im folgenden wird das Verhalten der einzelnen Befehle beschrieben, sofern es von der bekannten Verhaltensweise abweicht.

init-DB: Dieser Befehl setzt die zu bearbeitende Datenbank für die folgenden Anfragen auf den angegebenen Datenbanknamen und propagiert diesen Befehl auf alle Knoten des Clusters, sofern die Ausführung auf einem Knoten erfolgreich war. Zusätzlich wird diese Information in den Verbindungsdaten des Klienten mitgeführt, da einige Komponenten (z. B. SynchroServer und ClientManager) die für die Verbindung aktuelle Datenbank kennen müssen.

[14] Einen Ausnahmefall bilden hier die verschiedenen BLOB-Datentypen, die Datumsgrößen von bis zu 2^{32}-Zeichen zulassen. Wie diese Daten vom DBMS übertragen werden, wurde nicht weiter untersucht, da alle gegenwärtig zur Verfügung stehenden Anwendungen des Wortschatzprojektes ohne BLOBs auskommen.

[15] zum gegenwärtigen Zeitpunkt ist die Version 3.23.14 aktuell

[16] BLOB: ein Binary Large Object ist ein Datentyp, der große unstrukturierte Datenmengen aufnimmt.

query: Dies ist das wichtigste und am häufigsten verwendete Kommando des MySQL-Protokolls. Hier werden alle Arten von Abfragen zusammengefaßt.

1. lesende Anfragen (`SELECT`-Queries)
2. MySQL-spezifische Anfragen (`SHOW`-Queries), die Informationen über das DBMS liefern (Beispiel: `show tables`)
3. DML-Statements (z. B. `INSERT`, `UPDATE`) und DDL-Statements (`CREATE`, `DROP`, `ALTER`)[17]

Die ersten beiden Kategorien sind lesende Anfragen und können in der Regel von jedem verfügbaren Knoten behandelt werden. Hierfür kommuniziert der Server mit dem Scheduler, um einen verfügbaren Knoten für die Anfrage zu reservieren, leitet dann die Anfrage an diesen Knoten weiter und schickt das Ergebnis an den Klienten zurück.

Die dritte Kategorie von Anfragen greift schreibend auf die Datenbank zu oder verändert deren Struktur. Da diese Zugriffe zum einen auf alle Knoten verteilt werden müssen und zum anderen in der Reihenfolge abgearbeitet werden, in der sie eintreffen, ist hier eine gesonderte Behandlung notwendig. Der MySQL-MetaServer leitet die Abfrage daher zunächst an ein Knoten-DBMS weiter, das ihm vom Scheduler zugewiesen wurde. Es liegt hierbei in der Verantwortung des Schedulers, eine Zuordnung zu einen Masterknoten zu realisieren, der vorrangig alle schreibenden Anfragen erhält, um sicherzustellen, daß keine Anfrage durch den unvorhergesehenen Ausfall eines Knotens verloren gehen kann. Anschließend wird der Klient über den Erfolg der Aktion informiert. Falls keine Fehler aufgetreten sind, wird die Anfrage an den SynchroServer übergeben, der sicherstellt, daß sie in der richtigen Reihenfolge auf die übrigen Knoten propagiert wird.

create/drop-DB: Dieses Kommando erzeugt bzw. löscht eine Datenbank und wird vom Server wie ein Update-Kommando behandelt. Hierbei ist anzumerken, daß beide Funktionen auch in Form eines Abfrage-Kommandos (`query`) erfolgen könnten.

reload/flush: Alle `reload`- bzw.`flush`-Kommandos werden an alle Knoten propagiert, um sie auf einem einheitlichen Stand zu halten. Lediglich inaktive Knoten sind hiervon nicht betroffen, allerdings hat dies meist keine negativen Auswirkungen, da die relevanten Informationen bei Neustart des DBMS sowieso geladen oder initialisiert werden.

Die Reload-Funktion funktioniert wie bei dem 'richtigen' DBMS, wird allerdings auf alle Knoten propagiert, um die Benutzerauthentifizierung auf dem aktuellen Stand zu halten.

[17] DDL: Data Definition Language; DML: Data Manipulation Language

shutdown: Ein `shutdown`-Kommando bewirkt das sofortige Herunterfahren aller Datenbankknoten ohne vorherige Warnung. Insbesondere wird dieses Kommando (noch) nicht mit dem SynchroServer abgeglichen, so daß es durch noch ausstehende Schreiboperationen zu einer Inkonsistenz einzelner DBMS kommen kann.

connect: Da dieses Kommando von keinem der untersuchten Klienten verwendet[18] wird, wird es in der aktuellen Implementierung des Servers auch noch nicht unterstützt, sondern generiert nur eine Fehlermeldung in Form einer Ausnahmebedingung (Exception). Insbesondere ist nicht bekannt, ob und inwiefern die Verbindungsinformationen (Name, Passwort, Datenbank) erneut übertragen werden.

kill: Das erzwungene Beenden eine Threads mit Hilfe des `kill`-Kommandos wurde nur im Ansatz für den Einsatz im Cluster implementiert, indem der Befehl vom Klient an alle Knoten weitergegeben wird, bis der Aufruf das erste Mal erfolgreich ausgeführt wurde. Dies geschieht unter der Annahme, daß die angegebene Thread-ID nur auf einem Knoten existiert – dies ist im Normalfall aufgrund der unterschiedlichen Laufzeiten und Anfragehäufigkeiten der Server auch gegeben. Sollte dies im Einzelfall nicht so sein, so wird der falsche Prozeß beendet und der eigentliche Zielprozeß arbeitet weiter.

Da als einziges Kommandoargument die Thread-ID des zu beendenden Prozesses übergeben werden kann, kann der Zielknoten nicht direkt spezifiziert werden. Eine Lösung bestünde darin, die Prozeßnummern disjunkt in den verfügbaren Wertebereich (long) abzubilden. Hierfür müßte allerdings die Ausgabe des `processlist`-Kommandos so abgeändert werden, daß auch dort die transformierten ID-Werte ausgegeben werden.

ping: Das `ping`-Kommando prüft nicht mehr die Verfügbarkeit des einzelnen MySQL-DBMS, sondern zeigt stattdessen die Bereitschaft des MetaServers an, Verbindungen anzunehmen. Es wird ein OK-Paket an den Klienten zurückgeliefert, ohne vorher die Verfügbarkeit der einzelnen DBMS explizit zu prüfen.

processlist: Die Ausgabe der Prozeßliste wurde für den Cluster angepasst, indem sie um eine weitere Spalte erweitert wurde, die den Rechnernamen des Knotens angibt, auf dem der entsprechende Prozeß läuft (siehe Abbildung 4.14).

4.1.4 Benutzerverwaltung

Ziel der Benutzerverwaltung für den MySQL-MetaServer ist es, die bereits vom DBMS verwendenten Dienste und Prinzipien soweit wie möglich beizubehalten und die bereits angelegten Benutzerdaten zu verwenden. Dafür ist es notwendig,

[18] statt die Reconnect-Funktion zu benutzen, trennen die meisten Klienten die bestehende Verbindung, bauen sie erneut auf und melden sich dann auf die gewohnte Art und Weise an.

Id	User	Host	db	Command	Time	State	Info	ClusterNode
78	synchro	woclu1...		Connect	5	Reading from net		woclu2
79	Cboehm	woclu1...		Processlist	0			woclu2
199	synchro	woclu1...		Connect	5	Reading from net		woclu1
200	metaserver	woclu1...		Sleep	4			woclu1
202	Cboehm	woclu1...		Processlist	0			woclu1
80	synchro	woclu1...		Connect	5	Reading from net		woclu3
81	Cboehm	woclu1...		Processlist	0			woclu3
17800	root	localhost	wortschatz	Sleep	26228			woclu4
18419	www	aspra9...	wortschatz	Query	8184	Sending data	selec[...]	woclu4
18494	www	aspra9...	wortschatz	Sleep	6231			woclu4
18500	www	aspra9...	wortschatz	Sleep	6122			woclu4
18504	www	aspra9...	wortschatz	Query	5703	Sending data	selec[...]	woclu4
18584	synchro	woclu1...		Connect	5	Reading from net		woclu4
18585	Cboehm	woclu1...		Processlist	0			woclu4

new column for the cluster

Abbildung 4.14: Anpassung der Prozeßliste für den Cluster

die Benutzerdaten auf jedem Knoten redundant zu speichern und die Anmeldung des Benutzers auf alle Knoten zu propagieren.

Der Realisierung dieses Ziels standen zunächst die folgenden Probleme entgegen:

- Die Benutzerverwaltung basiert zum einen auf einer Name/Paßwort-Kombination, die vom Benutzer beim Anmelden am DBMS angegeben werden muß. Diese kann jedoch nicht direkt für die Anmeldung am Cluster verwendet werden, da das Paßwort verschlüsselt übertragen wird und sich die Signatur des Paßwortes bei jedem Anmeldevorgang ändert.
- Bei der Anmeldung eines Benutzers ist der Rechner, von dem die Anmeldung erfolgt, relevant und wird vom DBMS ausgewertet. Diese Information ist im Cluster aber durch den Einsatz des MySQL-MetaServers verfälscht.

Aus diesem Grund wurde ein zweistufiges Anmeldekonzept entwickelt, das zu den einzelnen Benutzerkonten äquivalente Schattenkonten verwendet, die in Bezug auf die Rechte des Benutzers gleichwertig sind und deren Namen durch einen Präfix ergänzt wurde. Im Gegensatz zum Paßwort des richtigen Benutzers ist das Paßwort des Schattenkontos dem MetaServer bekannt. Damit läßt sich der Anmeldeprozeß für den Cluster analog zu dem üblichen Verfahren des realen MySQL-DBMS nachbilden. Einzige Ausnahme ist hierbei die Verwendung des Rechnernamens für die Auswertung der Anmeldung, die hier nicht benutzt werden kann, da alle Pakete über den MetaServer weitergeleitet werden, was zu einer Verfälschung dieser Information führt.

Die Verwendung von zusätzlichen Schattenkonten für jeden Benutzer bedeutet zunächst einen erhöhten Verwaltungsaufwand, der sich allerdings dadurch relativiert, daß die Schattenkonten automatisch aus den zugehörigen Benutzerinformatioen erzeugt werden können[19].

Die Anmeldung selbst erfolgt dann in zwei Schritten, die in Abbildung 4.15 dargestellt sind: Zunächst werden die Anmeldedaten des Benutzers zu einem Knoten weitergeleitet und dort überprüft. Dieser Schritt wird als *Authentifizierung* bezeichnet. War sie erfolgreich, so wird unter Verwendung des entsprechenden Schattenkontos die *Anmeldung* auf den Knoten des Clusters durchgeführt, die für diesen

[19] Hierfür steht das Programm `cluster.admin.CreateServerUser` zur Verfügung.

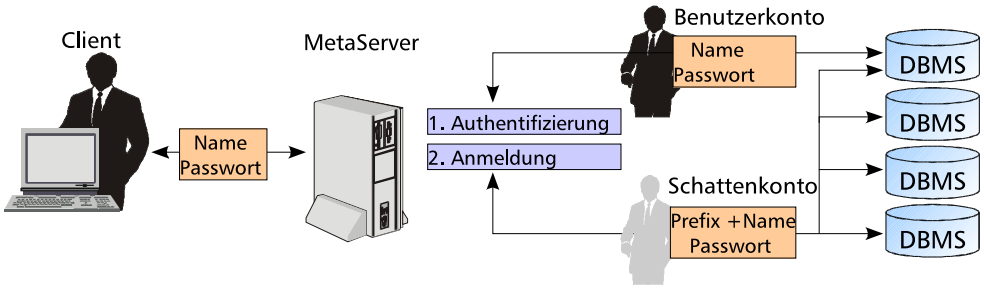


Abbildung 4.15: Anmeldung am MySQL-DBMS über den MetaServer

Nutzer zur Verfügung stehen sollen. Diese Verbindungen werden dann auch für die Abwicklung der Anfragen des Benutzers verwendet, während die für die Authentifizierung verwendete Verbindung wieder geschlossen wird.

Sollte während der Authentifizierung ein Fehler auftreten, wird dieser direkt an den Klienten zurückgeschickt und der Anmeldevorgang abgebrochen. Sollte hingegen bei der Anmeldung auf einem Knoten-DBMS ein Fehler auftreten, versucht der MetaServer, auf andere verfügbare Knoten auszuweichen. Sollte auch dies fehlschlagen, generiert der MetaServer eine spezifische Fehlermeldung, die an den Klienten weitergeleitet wird.

4.2 JDBC-Server

Der direkte Zugang zu einem DBMS unter Benutzung des entsprechenden Protokolls birgt die Gefahr einer direkten Abhängigkeit zum jeweils verwendeten Datenbanksystem in sich. Dadurch kann die Umstellung des Applikationsservers auf ein anderes DBMS erschwert bzw. durch den hohen Aufwand unmöglich gemacht werden.

Aus diesem Grund wurde nach einer alternativen Strategie gesucht, mit Hilfe derer eine Verteilung von Datenbankanfragen auf einem Cluster von replizierten DBMS gewährleistet werden kann, ohne direkte Abhängigkeiten zum verwendeten System zu schaffen.

Datenunabhängigkeit wird im allgemeinen durch eine Treiberarchitektur gewonnen, die zwischen der Datenbankanwendung und dem eigentlichen DBMS angeordnet ist. Auf Applikationsseite stellt sie den Anwendungen eine einheitliche Aufrufschnittstelle zur Verfügung und verbirgt damit die Spezifika des DBMS. Gegenüber dem Datenbanksystem hingegen tritt der Treiber wie eine Applikation auf, das heißt, er ist (in der Regel) nicht in das DBMS integriert. Leider muß die gewonnene Unabhängigkeit auch mit einigen Nachteile erkauft werden:

- Geschwindigkeitseinbußen durch die Verwendung einer weiteren Abstraktionsebene

- Beschränkung der Funktionalität auf allgemein verfügbare Funktionen, ohne auf spezielle Eigenschaften oder Funktionen eines bestimmten DBMS eingehen zu können. Andererseits müssen die Treiber unter Umständen Funktionalität nachbilden, die das DBMS nicht anbietet, um der Schnittstellendefinition genügen zu können (Beispiel: Transaktionen).

Die verbreitetsten Architekturen sind die *Open DataBase Connectivity*-API (ODBC) von Microsoft™ (beschrieben in [Gei95]) und die *Java Database Connectivity* (JDBC)[20], die von Sun Microsystems™ entwickelt wurde (beschrieben in [SW99]). Bei ODBC handelt es sich um eine Schnittstelle für die Programmiersprache C, während JDBC für den Einsatz innerhalb der Java-Plattform entworfen wurde. Beide Architekturen basieren auf dem SQL Call Level Interface (CLI) der X/Open [The95] und werden von allen wichtigen Datenbankanbietern unterstützt.

Im unserem Fall fiel die Entscheidung zugunsten der JDBC-Architektur aufgrund mehrerer Aspekte:

- in dem Wortschatzprojekt gibt es bereits einige Anwendungen, die JDBC einsetzen, während ODBC gegenwärtig nicht verwendet wird
- JDBC verspricht durch die Einbettung in die Java-Umgebung eine bessere Plattformunabhängigkeit als ODBC und ist im Gegensatz zu diesem auf vielen Betriebssystemen einsetzbar. Die JDBC-Treiber sind meist vollständig in Java geschrieben und stellen außer der Verfügbarkeit einer Java-Laufzeitumgebung (JRE) keinerlei Anforderungen an die Integration in das Betriebssystem, auf dem sie laufen (ODBC-Treiber sind plattformspezifisch, müssen in der Regel separat in dem Betriebsystem installiert sein und die zu verwendenden Datenquellen müssen für den Treiber registriert werden)
- Eine genaue und offene Spezifikation der Schnittstellen und Verfügbarkeit von Treiberquelltexten ermöglicht eine genaue Analyse der Architektur (Microsoft™ unterstützt ODBC nach wie vor, plant allerdings eine Integration in die DAO-Plattform, die nur unter MS-Windows™ verfügbar ist).
- falls notwendig, gibt es mit der von Sun™ zur Verfügung gestellten JDBC-ODBC-Bridge einen Migrationsweg, um auf Datenbanken zuzugreifen, die ausschließlich über ODBC erreichbar sind.

Ein Nachteil beim Einsatz von JDBC ist die Fixierung auf Java als Wirtssprache für die Anwendungen. Applikationen, die beispielsweise in C oder einer anderen Sprache geschrieben wurden, können bei diesem Ansatz zunächst nicht integriert werden. Um dies gewährleisten zu können, ist für jede zu verwendende Sprache eine ähnliche API zu entwerfen und zu implementieren oder ein Migrationsweg zu JDBC bereitzustellen.

[20] eigentlich ist diese Umschreibung laut dem JDBC-Guide nicht ganz korrekt: “As a point of interest, JDBC is a trademarked name and is not an acronym; nevertheless, JDBC is often thought of as standing for ’Java Database Connectivity’ ”

4.2.1 Die JDBC-API

Die JDBC-API wurde ursprünglich entwickelt, um von der Java-Plattform einen Zugriff auf SQL-basierte Datenanksysteme zu ermöglichen. Zunächst als eigenständige Lösung für den Zugriff auf relationale Datenbanken entwickelt, wurde es schon bald fest in die Version 1.1.x des Java Development Kits (JDK) integriert. Die Spezifikation der API wurde seitdem mehrfach geringfügig geändert und liegt in der aktuellen Version 1.2 vor, siehe [Sun97b].

Mit Erscheinen der Java2™-Plattform wurde dann die Funktionalität der API erheblich erweitert, um zum einen die Funktionen moderner Datenbanksysteme besser unterstützen zu können und andererseits die Integration mit anderen Java-APIs zu verbessern. Um gleichzeitig die Entwicklung entsprechender Treiber nicht zu erschweren und weiterhin eine einfache Verwendung der API sicherzustellen, wurde die Schnittstelle in zwei Teile aufgetrennt: die *Core JDBC 2.1*-API (siehe [Sun99f]), die die Funktionalität der Version 1.x enthält und hinsichtlich verschiedener Aspekte erweitert ist, und einer *JDBC 2.0 Optional Package*-API (siehe [Sun99g]). Die optionale API ist als Java-Standard-Erweiterung implementiert (Java Standard Extension) und kann bei Bedarf um weitere Funkionen ergänzt werden. Sie enthält eine Anbindung an JNDI, den zentralen Namensdienst der Java-Plattform, unterstützt die Implementierung von Connection Pooling und verteilten Transaktionen und bietet mit dem RowSet-Interface eine flexiblere Handhabung der Ergebnismengen einer Datenbankabfrage an.

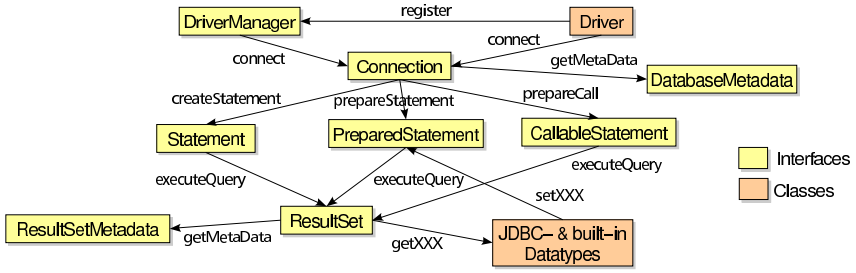


Abbildung 4.16: Klassenhierarchie der wichtigsten Core-JDBC-Klassen

Die JDBC-API besteht aus einer Reihe von Klassen, Exceptions und Interfaces, die den Zugriff auf die Datenbank durch die Java-Applikation ermöglicht (siehe Abbildung 4.16).

Die eigentliche Funktionalität zum Zugang zu einem spezifischen DMBS wird durch einen JDBC-Treiber erreicht, der die notwendigen Interfaces implementiert. Der Treiber registiert sich nach dem Laden automatisch bei dem DriverManager und wird von diesem verwaltet. Die Applikationen greifen normalerweise nicht direkt auf den Treiber zu, sondern fordern stattdessen mit Hilfe eines *Uniform Resource Locators* (URL)[21] eine Verbindung zum gewünschten DBMS an, die der DriverManager dann mit Hilfe des entsprechenden Treibers bereitstellt.

[21] URL: Universal Ressource Locator, ist die Spezifikation eines Adreßformats für Ressourcen in einem Netzwerk

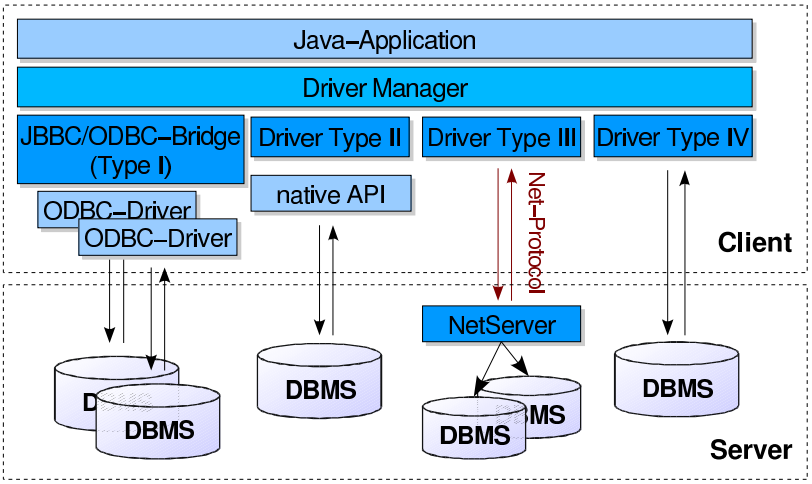


Abbildung 4.17: Aufbau der JDBC-API und der verschiedenen Treibertypen

Der Zusammenhang zwischen den einzelnen Komponenten der Architektur ist in Abbildung 4.17 dargestellt und zeigt gleichzeitig die verschiedenen Treiberarten, die von der Spezifikation unterschieden werden:

Typ I: Eine JDBC-ODBC-Bridge gewährt mittels eines oder mehrerer ODBC-Treiber Zugriff auf das DBMS. Da es sich bei den ODBC-Treibern um eine weitere Abstraktionsebene in plattformabhängigem Programmcode handelt, die auf jedem Klient installiert werden muß, ist diese Lösung langsamer als andere Treibertypen und erfordert einen höheren Installationsaufwand.

Typ II: Dieser Treibertyp verwendet eine DBMS-spezifische plattformabhängige API, um die JDBC-Aufrufe für das verwendete DBMS umzuwandeln. Auch hier ist die entsprechende plattformabhängige Bibilothek auf jedem Klientrechner zu installieren.

Typ III: Der potentiell flexibelste Treiber-Typ verwendet ein DBMS-unabhängiges Protokoll, um mit einem NetServer zu kommunizieren, der auf der Serverseite läuft und mit dem eigentlichen DBMS verbunden ist. Dies ermöglicht die Kommunikation mit verschiedenenen gleichartigen Datenbanksystemen unter Verwendung des immer gleichen Treibers auf Klientseite. Da der NetServer außerdem auf der Serverseite läuft, entsteht kaum Administrationsaufwand bei den Klienten.
Obwohl einige Hersteller mittlerweile Treiber dieses Typs anbieten, hat sich noch kein NetProtocol entwickelt, das sich universell einsetzen läßt.

Typ IV: Treiber dieses Typs verbinden sich direkt mit dem Ziel-DBMS und verwenden das entprechende Protokoll, um mit dem Datenbanksystem zu kommunizieren. Da für diese Architektur eine genaue Kentnniss des vom DBMS verwende-

ten, meist proprietären Protokolls notwendig ist, werden Treiber dieses Typs meist von den Datenbankherstellern selbst angeboten.

4.2.2 Architektur

Das Ziel der beschriebenen Architektur ist es, auf der Basis eines bereits für das zu verwendende DBMS vorhandenen JDBC-Treibers eine Clusterarchitektur umzusetzen, die in der Lage ist, die Anfragen verschiedener Klienten auf eine Reihe von Clusterknoten zu verteilen. Hierfür sind zwei wesentliche Problemkreise zu betrachten:

- **Transportproblem:** Schaffung einer geeigneten, möglichst generischen Struktur, die die API-Aufrufe der Java-Applikation auf der Klientseite zu dem JDBC-MetaServer weiterleitet und dort die entsprechenden Objekte instanziiert und Methodenaufrufe auslöst.
- **Clusterproblem:** Implementierung der clusterspezifischen Funktionalität, die notwendig ist, um eine Verteilung der Anfragen auf die verschiedenen Knoten zu ermöglichen. Hierbei ist insbesondere die Interaktion mit dem Scheduler und dem SynchroServer relevant.

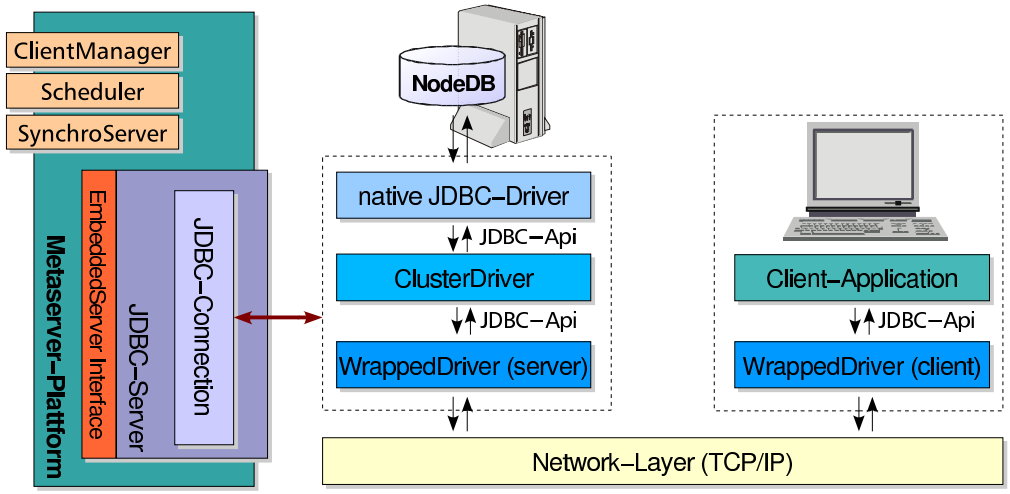


Abbildung 4.18: Architektur des JDBC-MetaServers

Die realisierte Architektur ist in Abbildung 4.18 dargestellt und gleicht von ihrem prinzipiellen Aufbau dem MySQLServer (siehe Abschnitt 4.1 auf Seite 85). Auch hier ist der JDBC-MetaServer der zentrale Bestandteil der Architektur, der innerhalb der MetaServerplattform als *EmbeddedServer* eingebunden ist. Für jeden Clienten wird ein neuer Ausführungsstrang (Thread) erzeugt, der die Anmeldung, Abfragebearbeitung und Kommunikation übernimmt. Diese Funktionalität ist in einem *JDBCConnection*-Objekt gekapselt, das mit den entsprechenden Komponenten des MetaServers (ClientManger, Scheduler) zusammenarbeitet.

Für die oben angeführten spezifischen Problemkreise wurden Lösungen entwickelt, die jeweils in einer eigenen Komponente gekapselt und in den JDBC-MetaServer integriert wurden. Neben einer klaren Aufgabentrennung hat dies den Vorteil, daß sie unabhängig voneinander verwendet und gegebenenfalls ausgetauscht werden können. Die Transportaufgabe wird mit Hilfe von server- und clientseitigen WrappedDrivers erreicht, der im nächsten Abschnitt beschrieben wird, während für die clusterspezifischen Aufgaben ein ClusterDriver entworfen wurde, der ebenfalls auf Basis der JDBC-API mit anderen Komponenten kommuniziert und auf den im Abschnitt 4.2.4 eingegangen wird.

4.2.3 WrappedDriver

Der WrappedDriver besteht aus einer klient- und einer serverseitigen Komponente und hat die Aufgabe, auf der Klientseite mit Hilfe der JDBC-API einen transparenten Zugriff auf das DBMS zu gewährleisten. Insbesondere soll der Anwendung verborgen bleiben, daß auf die Datenquelle nicht direkt von der Klientmaschine zugegriffen wird, sondern der Zugriff von einem anderen Rechner – dem JDBC-MetaServer – erfolgt. Hierfür muß auf der Klientseite die JDBC-API vollständig in Form von entsprechenden Methodenrümpfen (Stubs) implementiert sein und es muß außerdem ein Verfahren geben, um die aufgerufenen Methoden zusammen mit ihren Parametern zum Server zu übertragen. Dort müssen diese Informationen ausgewertet, die entsprechenden Objekte nach Bedarf erzeugt und die passenden Methoden aufgerufen werden. Das Ergebnis muß dann zurück an den klientseitigen Treiber geschickt werden und dort vom aufgerufenen Methodenrumpf an die Applikation zurückgegeben werden.

Die Kommunikation zwischen Server und Klient erfolgt mit Hilfe einer socketbasierten TCP-Verbindung auf einer bekannten, konfigurierbaren Portnummer (5000) oder mit Hilfe von RMI (*Remote Method Invocation*), wobei der MetaServer ein entsprechendes Interface exportiert, welches vom Klient für die Kommunikation mit dem Server verwendet wird.

Für die Übertragung der notwendigen Daten wird die Objektserialisierung der Java-Plattform verwendet. Diese ermöglicht es, ein beliebiges Java-Objekt in eine binäre Form zu wandeln (Serialisierung), die zur Übertragung in einem Datenstrom verwendet werden kann und aus dieser wieder ein neues Objekt zu rekonstruieren (Deserialisierung).

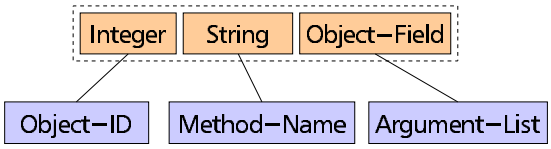


Abbildung 4.19: Aufbau eines JDBCPackets

Für die vom Klient zum Server gesendeten Daten wird hierfür die serialisierte Form eines speziellen Java-Objekts (*JDBCPacket*) verwendet, dessen Aufbau in Abbildung 4.19 dargestellt ist, während der Rückgabewert der Funktion der seriali-

sierten Form des entsprechenden Objekttyps entspricht. Die sieben nichtserialisierbaren Java-Basistypen werden von den Treibern auf beiden Seiten bei Bedarf in die äquivalenten Objekte umgewandelt (Beispiel: `int` → `Integer`). Sollte bei der Ausführung der Methode auf dem Server ein Fehler auftreten, so wird eine Ausnahmebedingung (Exception) generiert, die ebenfalls auf die beschriebene Art und Weise in serialisierter Form zum Klienten übertragen und dort ausgewertet wird.

Das Transportproblem besteht darin, daß die JDBC-Klassen auf dem Server erzeugt werden, um von dort auf die Datenbank zuzugreifen, aber auf dem Klienten von der Anwendung aufgerufen werden. Dieses Problem wird gelöst, indem der serverseitige Teil des Treibers den eigentlichen JDBC-Treiber umschließt (wrapped) und die erzeugten Objekte mit einer eindeutigen Identifikationsnummer (ID) versieht. Hierür wird der Hashwert des entsprechenden Java-Objekts verwendet, der eindeutig ist und für alle Java-Objekte zur Verfügung steht. Auf der Klientseite wird für jedes auf dem Server erzeugte Objekt ein Stub mit der entsprechenden ID erzeugt. Ruft der Klient nun eine Methode eines so erzeugten Stubs auf, so wird der Name der Methode samt Parameter und Objekt-ID in ein JDBCPacket-Objekt gekapselt und zum Server übertragen. Dort wird das Objekt anhand der ID lokalisiert und mittels *Runtime Type Identification* (RTTI) die entprechende Methode aufgerufen. Das Ergebnis des Methodenaufrufs wird dann in serialisierter Form zum Klienten zurückgesendet und bildet dort den Rückgabewerte der Stub-Funktion.

Der Klient unterstützt die Verwendung des DriverManagers und initiiert beim Laden auch das Laden des entsprechenden Treibers auf der Serverseite. Dies ist insofern ein Sonderfall, da bei diesem Ladeprozeß ein Objekt erzeugt werden muß, ohne daß eine Objekt-ID verfügbar ist. Dies wird mit Hilfe eines speziellen Schlüsselwortes (`obtainDriverID`) erreicht, das an Stelle eines Methodennamens übergeben wird.

Weiterhin läßt sich das Verhalten des JDBC-Treibers sowohl auf Klient- als auch auf Server-Seite mit Hilfe von Properties steuern. Da dieses Mittel auch von herkömmlichen JDBC-Treibern verwendet wird, um den Treiber zu parametrisieren, war es wichtig, bei der Implementierung darauf zu achten, daß es hier zu keinen Konflikten kommt und alle Einflußmöglichkeiten für den Anwender erhalten bleiben. Dies wurde erreicht, indem die Namen der Properties für die JDBC-ClusterTreiber mit dem Präfix `'cluster.'` versehen wurden. Eine Aufteilung der möglichen Parameter und ihrer Bedeutung findet sich in Tabelle 4.4. Der Treiber unterstützt das von der MetaServer-Plattform angebotene *serverseitige ConnectionCaching* und ermöglicht außerdem ein *ResultSetCaching*, das der schnelleren Übertragung großer Ergebnismengen dient.

Das beschriebene Problem des entfernten Zugriffs auf eine Datenbank mittels JDBC besteht in einer Reihe von Anwendungsfällen, in denen ein direkter Zugriff des Klients auf das DMBS aus technischen oder Sicherheitsgründen nicht möglich ist. Es gibt daher eine Reihe von freien und kommerziellen Lösungen, die auf unterschiedlichen Ansätzen basieren. Der *RmiJdbc*-Treiber von Gibello [Gib00] verwendet die RMI-Technologie, um vom Client auf die entfernten Methoden des Servers zugreifen zu können. Diese Lösung ist zwar universell und frei verfügbar, erzeugt aber durch das lastintensive RMI-Protokoll eine hohe Last auf dem Übertragungskanal, insbesondere dann, wenn viele Methodenaufrufe (z. B. bei der Auswertung

Property	Erläuterung
`cluster.connectionType`	spezifiziert die Art, die für die Verbindung zu dem MetaServer verwendet werden soll. Zugelassene Werte sind `rmi` bei Verwendung der Remote Method Invocation oder `socket` für die Verbindung mit Hilfe von TCP/IP-Sockets.
`cluster.RMI-ServerName`	Name des Servers in der RMI-Registry
`cluster.realDriver`	vollständiger Klassenname des JDBC-Treibers für den Zugrif auf die Datenbank vom JDBC-Metaserver aus
`cluster.ResultSetCaching`	ermöglicht die Verwendung des ResultSet-Caching, wenn der Wert dieser Property auf `true` gesetzt wird.
`cluster.asynchronousCaching`	Bei eingeschaltetem ResultSetCaching kann durch setzen dieser Property auf den Wert `true` erreicht werden, daß der Cache in einem asynchronen Prozeß gefüllt wird, während die Klient-Applikation noch mit der Verarbeitung von Daten beschäftigt ist.
`cluster.asynchronousCacheSize`	diese Eigenschaft gibt die Größe des ResultSet-Caches in Zeilen an
`cluster.realURL`	Hier kann der URL angegeben werden, der auf dem Server zur Herstellung der Verbindung zum serverseitigen Wrappeddriver verwendet werden soll

Tabelle 4.4: Properties für den klientseitigen WrappedDriver

eines Resultatsets) verwendet werden. Eine weitere Lösung stellt SoftGlobe mit ihrem *SocketJDBC* zur Verfügung ([Sof00]), das eigenen Aussagen zufolge erheblich schneller als RmiJdbc arbeitet und auf einer Übertragung auf der Basis von Sockets basiert.

Die vorgeschlagene Implementierung ist aufgrund ihrer modularen Architektur gegen andere Verfahren austauschbar und schränkt damit die Verwendung des JDBC-MetaServers in keiner Weise ein. Außerdem ermöglicht sie die alternative Verwendung beider Zugangswege (RMI oder Socket) und enthält bereits Möglichkeiten, die zu einer erheblichen Geschwindigkeitssteigerung beim Zugriff auf die Datenbank führen.

4.2.4 ClusterDriver

Der ClusterDriver ist dafür verantwortlich, die für den Cluster notwendige Funktionalität in einer eigenen Schicht zu kapseln und mit Hilfe eines JDBC-Treibers den Zugriff auf das DBMS sicherzustellen. Insbesondere ist hierbei die Interaktion mit den relevanten Komponenten der MetaServer-Architektur von Bedeutung: der ClientManager, der Scheduler und der SynchroServer.

Der Zugriff auf das DBMS erfolgt mit Hilfe eines *Uniform Ressource Locators* (URL), dessen Syntax von dem verwendeten Treiber abhängig ist und nur teilweise in der offiziellen Spezifikation von Sun™ in der folgenden Form festgelegt ist:

```
jdbc:<subprotocol>:<subname>
```

Hierbei sind sowohl *subprotocol* als auch *subname* notwendige Angaben und spezifizieren das verwendete Protokoll und damit im allgemeinen das zu verwendende DBMS und alle notwendigen Verbindungsparameter. Die Struktur dieser Zeichenketten unterliegt dabei keinen weiteren Regeln und wird von dem jeweils verwendeten JDBC-Treiber determiniert.

Beispielsweise verwendet der MySQL-Treiber von Mark Matthews[22] die folgende Syntax (Angaben in eckigen Klammern spezifizieren optionale Angaben, entnommen aus [Mat99]):

```
jdbc:mysql://[hostname]:[port]/dbname[?par1=val1][&par2=val2]...
```

Zusätzlich können für eine neue Verbindung in einem Properties-Objekt noch weitere Eigenschaften spezifiziert werden, die sich auch mit den im URL angegebenen Informationen überlappen können.

Für den ClusterDriver ergeben sich damit zwei Probleme: Erstens besteht eine direkte Abhängigkeit zwischen dem verwendeten Treiber und dem verwendeten DBMS[23]; der Treiber muß also in der Lage sein, die im URL enthaltenen Informationen generisch extrahieren zu können. Zweitens sind Rechnername und Portnummer direkt in den Verbindungsinformationen enthalten und müssen dynamisch gegen die Namen der Clusterknoten ausgetauscht werden. Schließlich verwendet der Clustertreiber selbst einen URL, um die Verbindung zwischen Klient und Meta-Server herzustellen.

Um diesen Problemen zu begegnen, wird festgelegt: In dem URL werden nur die für die Verbindung unbedingt notwendigen Informationen angegeben, alle anderen, insbesondere treiberspezifische, Daten sollen in dem Properties-Objekt angegeben werden. Im konkreten Fall verbleiben also nur der Hostname und die Portnumer des JDBC-MetaServers in dem URL, der damit folgende Gestalt hat:

```
jdbc:cluster://[hostname]:[port]
```

Die Angaben von *hostname* und *port* sind optional und werden mit dem Standardwert *localhost* bzw. *5000* initialisiert.

Alle optionalen Verbindungseigenschaften, ob sie nun den ClusterDriver oder den JDBC-Treiber, der auf das DBMS zugreift, betreffen, werden im Properties-Objekt

[22] Es sei angemerkt, daß die Syntax für einen JDBC-URL nur vom Treiber, nicht jedoch vom verwendeten DBMS abhängt. Im Falle von MySQL gibt es mehrere JDBC-Implementierungen, die unterschiedliche Formate für die URLs verlangen.

[23] Es kann nicht einmal davon ausgegangen werden, daß die URL-Syntax für ein konkretes DBMS einheitlich ist: Es gibt beispielsweise mehrere JDBC-Treiber für MySQL, die alle eine teilweise inkompatible Syntax verwenden.

in unterschiedlichen Namensräumen spezifiziert und mit sinnvollen Voreinstellungen belegt. Dies betrifft auch den URL der für den Zugriff auf die Datenquelle vorgesehen ist. Um diese Informationen adäquat parsen zu können, wird in dem *Driver*-Objekt des ClusterDrivers ein protokollspezifischer Parser für jedes unterstützte Protokoll vorgesehen. Die Unterscheidung der verschiedenen Protokolle erfolgt anhand der Angabe für `subprotocol`. Wenn ein neues Protokoll zu unterstützen ist, muß nur ein neuer spezifischer Parser implementiert werden, der die benötigten Informationen aus dem URL extrahiert.

Das folgende Beispiel zeigt, wie der Verbindungsaufbau zu einen JDBC-MetaServer aussehen könnte, der auf ein MySQL-DBMS zugreift, und zeigt die verschiedenen Möglichkeiten, den Verbindungsaufbau zu beeinflussen:

Beispiel für eine Verbindug zum JDBCServer:

```
// load the ClusterDriver
// (registers itself to the DriverManager)
Class.forName("cluster.jdbc.Driver").newInstance();

// specify the connection properties
Properties conProps = new Properties();

// 'standard' properties
conProps.put("user", "boehm");
conProps.put("password", "*********");

// properties for the ClusterDriver
conProps.put("cluster.realDriver","org.gjt.mm.mysql.Driver");
conProps.put("cluster.realURL","jdbc:mysql:///test");
conProps.put("cluster.ResultSetCaching", "true");
conProps.put("cluster.asynchronousCaching", "true");
conProps.put("cluster.asynchronousCacheSize", "50");

// properties for the MM-MySQL-driver
conProps.put("autoReconnect", "true");
conProps.put("maxReconnects", "10");

// obtain a connection from the DriverManager
Connection con = DriverManager.getConnection(
                   "jdbc:cluster://woclu1:5000", conProps);
```

Von der Verbindungsaufnahme abgesehen unterscheidet sich die Handhabung des ClusterDrivers in keiner Weise von der herkömmlicher JDBC-Treiber, die detailliert in [SW99] beschrieben ist.

Die Applikation auf dem Klient hat außerdem die Möglichkeit, auf die Verteilung der einzelnen Abfragen Einfluß zu nehmen, da alle Abfragen innerhalb eines *Statement*-Objekts auch auf dem gleichen Clusterknoten abgearbeitet werden. Je nach Anwendung kann somit der Benutzer den Aufwand, der durch das Scheduling der Anfragen auf dem MetaServer entsteht, minimieren, wie im untenstehenden Beispiel illustriert wird.

Beispiel für die verschiendenen Varianten der Abfrageformulierung:

```
// execute two queries on the same ClusterNode
Statement stmt = con.createStatement();
rs = stmt.executeQuery(query1);
rs = stmt.executeQuery(query1);
stmt.close();

// execute the first query on a ClusterNode
// selected by the Scheduler
Statement stmt1 = con.createStatement();
rs = stmt1.executeQuery(query1);
stmt1.close();

// execute the second query on a  potential different
// ClusterNode
Statement stmt2 = con.createStatement();
rs = stmt2.executeQuery(query1);
stmt2.close();
```

Die optimale Lösung ist hierbei stark von der jeweiligen Applikation abhängig: weniger Scheduling verringert zwar den Aufwand auf dem MetaServer, bedeutet aber auch einen geringeren Parallelisierungsgrad und damit ungleichmäßige Auslastung des Clusters, was die Effizienz des Gesamtsystems negativ beeinflußt. Andererseits erzeugt die vollständige Parallelisierung vieler einfacher Anfragen einen erheblichen Aufwand auf dem MetaServer.

4.2.5 Implementierungsumfang

Zum Zeitpunkt der Implementierung war die JDBC-API 1.2 die vorherrschende und von den meisten Treibern implementierte Spezifikation. Aus diesem Grund baut der JDBC-MetaServer hierauf auf und implementiert den Funktionsumfang bis auf wenige Ausnahmen vollständig.

Die einzige Einschränkung bildet gegenwärtig der Zugriff auf die Daten einer Ergebnismenge mit Hilfe entsprechender Streams (`getAsciiStream()`, `getUnicodeStream()` und `getBinaryStream()`), die noch nicht implementiert sind. Der Grund hierfür ist, daß diese Zugriffsart eigentlich nur bei großen Datenfeldern (BLOBs, CLOBs) verwendet wird und durchgängig vom Klient bis zum Datenbanksystem in Form eines Streams realisiert sein sollte, um den Speicherverbrauch der einzelnen Komponenten zu minimieren. Dies erfordert den Aufbau zusätzlicher Verbindungen zwischen dem MetaServer und dem Klienten. Insbesondere die Sicherstellung der Threadsicherheit der Treiber erzeugt hierbei einen hohen Implementierungsaufwand, der angesichts der Tatsache, daß diese Funktionalität innerhalb des Wortschatzprojektes nicht verwendet wird, nicht gerechtfertigt schien. Trotzdem ist eine Integration dieser Funktionalität prinzipiell möglich.

Die aktuelle Spezifikation 2.0 stellt sowohl in der Core- als auch in der erweiterten Version keine prinzipielle Änderung der in Abschnitt 4.2.1 dargestellten Architektur dar, sondern erweitert lediglich deren Funktionalität. Damit steht einer Anpassung der vorgestellten Implementierung an den neuen Standard nichts im Wege.

Bei Vorhandensein eines entsprechenden Treibers für den Zugriff auf das DBMS ist hierfür nur eine Erweiterung der Stub-Klassen für den klientseitigen WrappedDriver nötig, um ihn wie oben beschrieben einsetzen zu können.

Zu den in dem *JDBC 2.0 Optional Package*-API betrachteten Schwerpunkten zählt unter anderem Connection-Pooling und die Einbindung in den JNDI-Dienst der Java-Plattform. Mit dem ClientManager wird bereits ein serverseitiges Connection-Pooling angeboten, das allerdings nicht kompatibel zu der neuen API-Spezifikation ist. Einer Integration der MetaServer-Plattform in den Namensdienst der Java-Plattform steht nichts entgegen, jedoch ermöglichen erst die zahlreichen Verbindungsoptionen eine applikationsspezifische Einflußnahme auf die Leistungsfähigkeit des Clusters, da der URL nur die notwendigsten Informationen enthält.

Für die Verifikation der Implementierung der JDBC-API stellt Sun™ eine allgemeine Testumgebung (JDBC-Harness) zur Verfügung, die dann zusammen mit entsprechenden TestSets Informationen über die korrekte Implementierung des Treibers liefern ([Sun99h]). Für das innerhalb des Wortschatzprojekts verwendete DBMS (MySQL) steht allerdings kein entsprechendes TestSet zur Verfügung, und eine Anpassung der wenigen bereits vorhandenen Sets war kurzfristig nicht möglich. Aus diesem Grund konnte auch der auf dem MySQL-JDBC-Treiber aufbauende JDBC-MetaServer nicht mit Hilfe des JDBC-Harness verifiziert werden. Eigene Tests, die den häufigsten Anwendungsfällen entsprachen, zeigten jedoch keinerlei Fehlverhalten und bestätigten damit die Einsatzfähigkeit der Implementierung.

4.2.6 Leistungsfähigkeit

Durch die mehrschichtige Architektur des JDBC-MetaServers entsteht ein zusätzlicher Overhead beim Zugriff auf das DBMS, der dem Geschwindigkeitsgewinn durch die Clusterarchitektur entgegensteht. Das Nutzverhältnis für die einzelnen Anwendungen ist hierbei abhängig vom Verhältnis der Verarbeitungsdauer der Anfragen durch das DBMS und der Zeit, die der MetaServer für die Vermittlung der Anfragen benötigt. Komplizierte und aufwendige Queries profitieren hierbei erheblich stärker von der Effizienz der verteilten Clusterarchitektur als einfache Anfragen, die schnell von dem DBMS abgearbeitet werden können und auf die sich der Overhead der Architektur besonders stark auswirkt.

Am deutlichsten ist dieser Zusammenhang bei der Bearbeitung der Ergebnismenge einer Anfrage zu beobachten, da der Zugriff hierauf normalerweise mit Hilfe einer ganzen Reihe von Methodenaufrufen erfolgt, während für das Absetzen einer Abfrage an das DBMS nur einige wenige Aufrufe notwendig sind. Bei der Auswertung einer großen Ergebnismenge ist somit auch mit einer großen Anzahl von Methodenaufrufen des *ResultSet*-Objekts zu rechnen, und daher ist hier der zusätzliche Aufwand, der durch die MetaServer-Architektur entsteht, besonders groß. Da dies also im Hinblick auf die Geschwindigkeit des Gesamtsystems ein besonders kritischer Punkt ist, wurde die Leistungsfähigkeit des Systems an dieser Stelle untersucht.

4.2.6.1 Bearbeitung des ResultSets

Eine Abfrage an ein relationales Datenbanksystem liefert im Ergebnis eine tabellenartige Datenstruktur zurück, die in der JDBC-API in einem *ResultSet*-Objekt gekapselt wird. Mit Hilfe der Methode `next()` wird hierbei zunächst eine komplette Zeile der Ergebnismenge eingelesen, während auf das einzelne Datum innerhalb dieser Zeile mit Hilfe verschiedener `getXXX()`-Methoden zugegriffen wird[24]. Die vollständige Abarbeitung einer Resultatmenge mit m Zeilen und n Spalten erzeugt also mindestens $m + m \cdot n$ Methodenaufrufe für das *ResultSet*-Objekt.

Für die Performanzuntersuchungen wurde eine Testtabelle erzeugt, die 100 gleichwertige Attribute enthielt und mit 1.000 gleichartigen Datensätzen gefüllt wurde. Bei den enthaltenen Daten handelte es sich um variable Stringfelder mit einer Länge von 100 Zeichen. Bei jedem Test wurde die Tabelle komplett mit einer einfachen SELECT-Anfrage ausgelesen, wobei die auf dem DBMS erzeugte Last gering war. Gemessen wurde jeweils die Zeit, die für das komplette Einlesen der Daten durch die Klientanwendung benötigt wurde. Dieser Durchlauf (run) wurde 50 Mal wiederholt, wobei in einer zweiten Testreihe auch untersucht wurde, ob die Verarbeitungszeit in der Klientanwendung (processing time) einen Einfluß auf die Gesamtleistung der JDBC-Architektur hat. Zu diesem Zweck wurde die simulierte Verarbeitungszeit schrittweise um jeweils 2 ms erhöht und deckte so den Bereich von 0-100 ms ab. Die gemessenen Zeiten unterteilen sich in zwei Komponenten: die Zeit, die für den Transfer einer Datenreihe benötigt wird (`NextTime`) und die kumulierte Zeit, die für den Zugriff auf die einzelnen Elemente der Datenreihe anfällt (`FetchTime`).

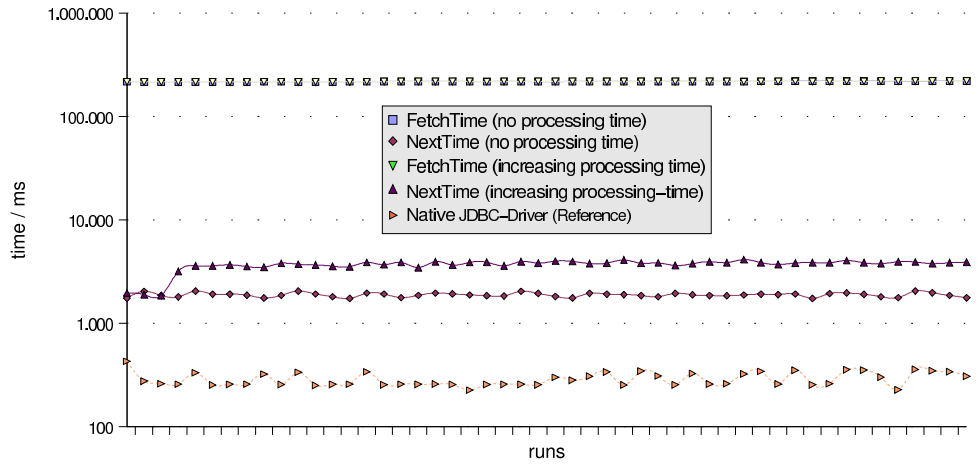


Abbildung 4.20: Auslesen des ResultSets ohne Optimierung

Ein erster Test mit dem JDBC-MetaServer zeigte, daß die Geschwindigkeitseinbußen gegenüber dem direkten Zugang über JDBC-Treiber auf die Datenbank erheblich sind und den Zugriff um ca. das 760fache verzögern (siehe Abbildung 4.20). Dies ist jedoch nicht überraschend, da jeder der 100.000 Aufrufe vom Klient auf

[24]Die neue Version der JDBC-API bietet weitere Möglichkeiten, sich in der Resultatmente zu bewegen.

den Server transportiert werden muss und die bei jedem Aufruf übertragenen Daten nur einen Bruchteil der übertragenen Information ausmachten. Der überwiegende Anteil der Verarbeitungszeit entfällt hierbei auf die `FetchTime`, während die `NextTime` kaum ins Gewicht fällt. Außerdem wird deutlich, daß die Verarbeitungszeit innerhalb der Klientanwendung keinen Einfluß auf die Zugriffsgeschwindigkeit hat.

Dieser Extremfall zeigt deutlich die architekturbedingten Schwachstellen des JDBC-MetaServers auf. In der Praxis wirken sich die beschriebenen Effekte weniger deutlich aus. Um eine Verbesserung der in diesem Test ermittelten Ergebnisses zu erreichen, wurden verschiedene Optimierungsmöglichkeiten entworfen und implementiert, die die Geschwindigkeit des Zugriffs signifikant steigern.

4.2.6.2 ResultSetCaching

Ein erster Ansatz bestand darin, bei einem `next`-Aufruf die gesamte Zeile vom Server zum Klient zu transferieren, um so zu erreichen, daß die nachfolgenden `getXXX`-Aufrufe lokal auf dem Klient erfolgen können und damit den Übertragungsoverhead zu veringern. Im konkreten Testfall bedeutet dies, daß nur noch 1.000 Aufrufe direkt vom Server bearbeitet werden müssen, und obwohl bei jedem dieser Aufrufe jetzt mehr Daten übertragen werden, da ja die gesamte 100 Zellen breite Zeile transferiert wird, zeigt die Auswertung des Benchmarks bereits eine erhebliche Geschwindigkeitssteigerung gegenüber dem nichtoptimierten Ansatz (siehe Abbildung 4.21). Die Grafik zeigt, daß sich das Verhältnis der relevanten Zeitanteile geändert hat: die `FetchTime` spielt jetzt aufgrund der lokalen Zugriffe keine Rolle mehr und kann vernachlässigt werden, während die `NextTime` jetzt der bestimmende Faktor ist, deren Schwankungen bei den einzelnen Durchläufen auf die unterschiedliche Netzbelastung zurückzuführen sind. Nach wie vor hat die Verarbeitungszeit der Klientanwendung keinen Einfluß auf die Zugriffsgeschwindigkeit.

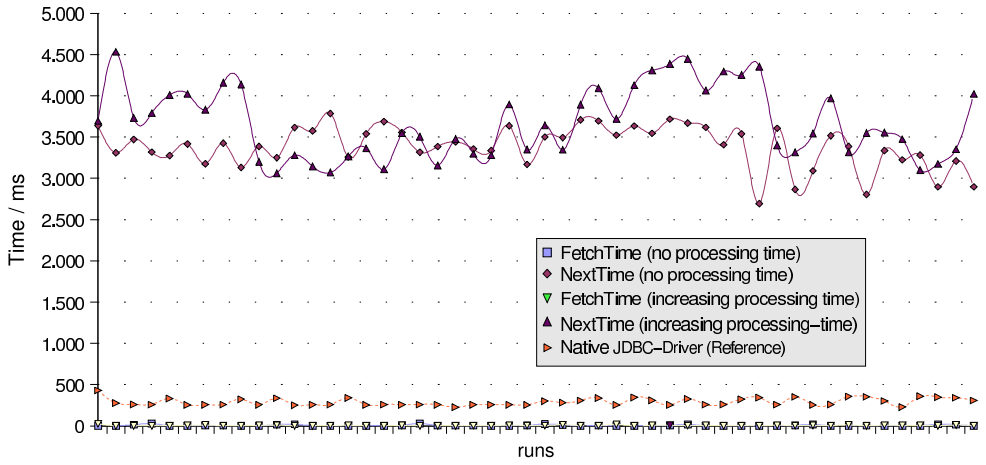


Abbildung 4.21: Auslesen des ResultSets mit ResultSetCaching

Trotzdem ist die Zugriffsgeschwindigkeit auf die Daten noch immer ca. 13 mal geringer als bei dem direkten Zugriff mit einem JDBC-Treiber und bietet Potential für weitere Verbesserungen.

4.2.6.3 Asynchrones ResultSetCaching

Ein weiterer Ansatz ergab sich aus der Idee, in dem JDBC-Treiber auf der Klientseite einen Cache für die einzelnen Zeilen der Ergebnismenge zu implementieren und diesen asynchron in einem weiteren Thread zu füllen und damit die von der Anwendung angeforderten Zeilen bereits verfügbar zu haben, ohne zuerst mit dem Server zu kommunizieren. Obwohl dieser Ansatz bei erhöhten Speichereinsatz auf dem Klient zunächst einen weiteren Geschwindigkeitsvorteil versprach, konnte dieser bei den Tests zunächst nicht beobachtet werden. Weitere Analysen zeigten, daß die Zeit, die die Applikation für die Bearbeitung der einzelnen Zeilen der Ergebnismenge benötigt, hier einen entscheidenden Einfluß hat.

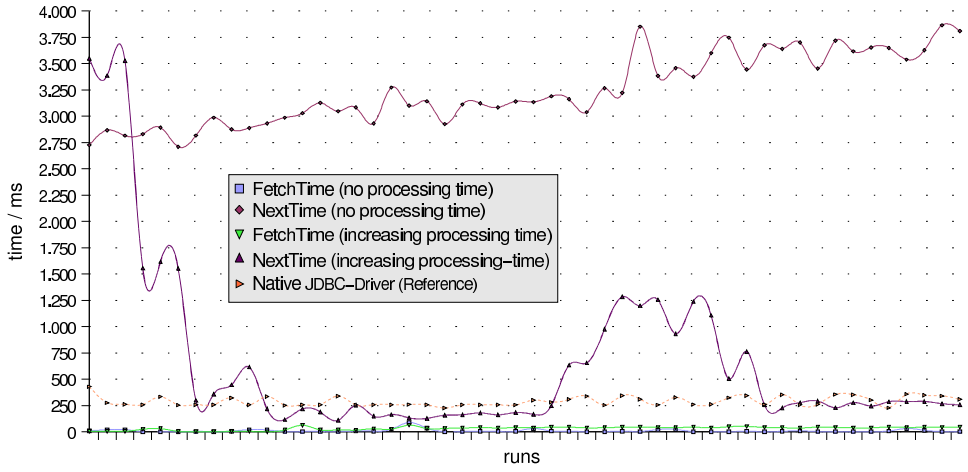


Abbildung 4.22: asynchrones ResultSetCaching

Wie in der Darstellung der Testergebnisse in Abbildung 4.22 zu erkennen ist, wird der Cache erst ab einer bestimmten Verarbeitungszeit innerhalb der Klientanwendung wirksam, die im konkreten Fall bei 12 ms liegt. Unterhalb dieser Verarbeitungszeit ist keine Verbesserung der Zugriffsgeschwindigkeit feststellbar; liest die Anwendung die Daten direkt und ohne Verzögerung aus, sind die Ergebnisse sogar noch schlechter als beim zeilenweisen ResultSetCaching. Im günstigsten Fall jedoch wird duch die Verwendung des Caches noch einmal eine deutliche Leistungssteigerung erreicht, die an die Leistung des direkten JDBC-Zugriff heranreicht bzw. diese teilweise überbietet.

Die Ursache für dieses Verhalten liegt in dem notwendigen Synchronisationsaufwand, der durch den gemeinsam genutzten Cache ensteht. Während die Klient-Applikation vermittels der `next`-Methode die einzelnen Zeilen des Caches ausliest, versucht gleichzeitig der asynchron arbeitende *FetcherThread*, den Cache wei-

ter zu füllen. Dies führt in ungünstigen Fällen zu einer gegenseitigen Behinderung, die sich in dem beobachteten Geschwindigkeitsverlust manifestiert.

Ein weiterer Faktor ist der GarbageCollector der Java-Laufzeitumgebung, der die nicht mehr benötigten Cache-Objekte beseitigt. Er arbeitet in einem eigenen Thread mit geringer Priorität und kommt zyklisch zum Einsatz. Bei einer starken Auslastung des Systems kann es dazu kommen, daß die GarbageCollection verzögert wird, was dazu führt, daß mehr Speicher angefordert wird, als eigentlich benötigt wird, was sich negativ auf die Leistung des Gesamtsystems auswirkt. Dies ist in den Testergebnissen bei den monoton ansteigenden `FetchTime`-Zeiten der Durchläufe ohne Verarbeitungszeit zu sehen, während bei der Testreihe mit simulierter Verarbeitungszeit zyklisch lokale Maxima zu beobachten sind.

Fazit

Zusammenfassend kann festgestellt werden, daß die mehrschichtige MetaServer-Architektur einen erheblichen Einfluß auf die Geschwindigkeit des Zugriffs auf das DBMS hat, der in Einzelfällen den sinnvollen Einsatz einer Applikation erschweren kann. Am Beispiel der ResultatSets wurde gezeigt, daß es eine Reihe von Möglichkeiten gibt, diese Engpässe zu umgehen. Es wurde aber ebenfalls deutlich, daß eine Optimierung nicht immer die erwünschte Wirkung zeigt und abhängig vom Verhalten der Klient-Applikation ist. Indem die verschiedenen Optimierungsmöglichkeiten vom Klienten an- und abgeschaltet werden können, wurde dieser Beobachtung Rechnung getragen und dem Entwickler der Applikation die Möglichkeit gegeben, den Zugriff auf die Cluster-DBMS für seine Bedürfnisse zu optimieren.

4.3 Der ScriptServer

Eine Aufgabe, für die der Cluster eingesetzt werden soll, ist die Bearbeitung von lang laufenden[25] Prozessen oder Jobs[26], die typischerweise während der Ausführung nicht mit dem Benutzer interagieren. Im allgemeinen sammeln und konvertieren solche Prozesse schrittweise, manchmal iterativ, Daten aus verschiedenen Quellen und speichern sie in der Datenbank ab. Eine Reihe solcher Prozesse werden gegenwärtig bereits innerhalb des Wortschatzprojekts ausgeführt, z.B. zur Indexerstellung, zur Kollokationsberechnung, zur Grundformreduktion und zur allgemeinen Textauswertung.

Diese Prozesse bestehen in der Regel aus einer Sammlung von ausführbaren Programmen und Skripten (in verschiedenen Sprachen), die in einer bestimmten Reihenfolge ausgeführt werden müssen, um aus den Datenquellen die gewünschten

[25] Die Bezeichnung 'lang laufend' ist zwar eine ungenaue Angabe, wird aber hier bewußt nicht weiter eingegrenzt, da die Klasse der in Frage kommenden Anwendungen sowohl Prozesse mit einer Laufzeit von einigen Stunden, als auch solche mit einer Laufzeit von mehreren Tagen umfaßt, ja sogar Prozesse einschließt die ständig oder zyklisch ausgeführt werden.

[26] Die Bezeichnung als *Prozeß* oder *Job* kann alternativ verwendet werden, je nachdem ob die Eigenschaft des prozesshaften (also produktiven oder erzeugenden) oder die des stapelorientierten, nicht interaktiven Verhaltens in den Vordergrund gestellt werden soll.

Resultate zu gewinnen. Während in der Anfangsphase des Wortschatzprojekts diese Abläufe meist manuell gesteuert und überwacht wurden, wird hierfür mittlerweile ein *Makefile* verwendet, das die verschiedenen Ziele modelliert und damit eine teilweise automatische Abarbeitung der Prozesse erlaubt.

Mit Hinblick auf die Architektur des Clusters ist die momentane Vorgehensweise nicht adäquat, da die Steuerung transparent durch den Cluster erfolgen und dieser die Ausführung optimieren können soll, indem er Teile des Prozeßes, soweit möglich, parallelisiert. Andererseits sollte der Migrations- und Einarbeitungsaufwand für die Benutzer der Software so gering wie möglich gehalten werden und sich daher an der bestehenden Architektur orientieren.

Die im folgenden Abschnitt beschriebene Architektur des SkriptServers basiert auf der Ausführung von Shell-Skripts, die die zu verwendenden Programme kapseln und die Schnittstelle zum Clustersystem bilden sollen. Für die eigentliche Ablaufsteuerung wurde eine einfache Skriptsprache entworfen, die es ermöglicht, den Ablauf des Prozesses in einem Skript zu formulieren. Der Begriff *Skript* wird im folgenden in zwei Bedeutungen verwendet: zum einen gibt es Shell-Skripts, die direkt auf dem Cluster-Betriebssystem ausgeführt werden und zum anderen werden die zu beschreibenden Prozesse auch in einem Skript definiert.

4.3.1 Architektur

Die Funktionalität der batchorientierten Prozeßabarbeitung ist innerhalb des Clusters als modularer *EmbeddedServer* realisiert und wie folgt aufgebaut: Der *SkriptServer* ist für die Steuerung des Prozesses zuständig, d.h. er verarbeitet das vom Benutzer erstellte Skript, welches den Prozeß beschreibt und initiiert die Ausführung der einzelnen Shell-Skripts. Dabei sorgt er auch für clusterrelevante Aufgaben wie das Scheduling und die Parallelisierung der einzelnen Aufrufe und greift dabei auf die von der MetaServer-Plattform angebotenen Dienste zurück.

Die eigentliche Ausführung der Shell-Skripte übernehmen die *NodeServer*, die auf den einzelnen Clusterknoten laufen. Sie sorgen für die Ausführung des Skripts, indem sie zunächst dem Namen in dem Prozessskript einen Skriptaufruf zuordnen, der an das Betriebssystem des Knotens absetzt wird. Nach Beendigung des Shell-Skripts liefern sie den Rückgabewert an den SkriptServer zurück, wo dieser ausgewertet und über die weitere Ausführung des Prozesses entschieden wird.

Die Architektur der Prozesssteuerung ist in Abbildung 4.23 dargestellt.

4.3.2 Administration

Die Aufgaben der Administration des ScriptServers beschränken sich auf das Starten von neuen Prozessen und die Überwachung bereits laufender Prozesse (anhalten, vorzeitig beenden, Statusabfragen). Für diese Aufgaben ist ein entsprechender *ManagementClient* notwendig, der auf einem beliebigen Klientrechner laufen kann.

Um diese Funktionalität zu unterstützen, bietet der Server ein Interface an, über das er gesteuert werden kann und das über RMI exportiert wird. Damit kann ein entsprechender Klient auf den Server zugreifen.

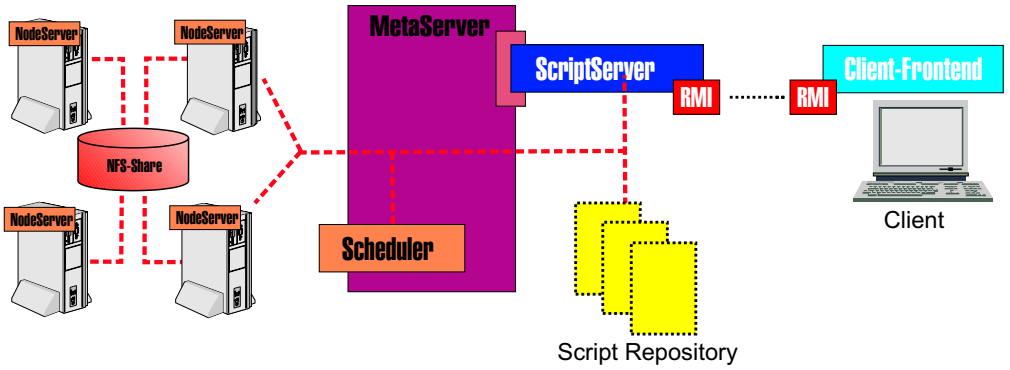


Abbildung 4.23: Architektur der Prozeßsteuerung

4.3.3 Datenaustausch

Ein wesentliches Problem dieser verteilten Architektur stellt der Datenaustausch zwischen den einzelnen Prozessen dar. Hierbei sind insbesondere folgende Punkte von Interesse:

- **Vermeidung von unnötiger Redundanz:** keine mehrfache Datenhaltung bzw. mehrfaches Kopieren von Daten zwischen den einzelnen Knoten
- **Verfügbarkeit:** Sicherstellung der Verfügbarkeit der einzelnen Daten für alle beteiligten Knoten des Cluster, auch nach Ausfall einzelner Komponenten
- **Geschwindigkeit:** effiziente Übermittlung der einzelnen Daten
- **Synchronisation:** Serialisierung konkurierender Schreib- und Lesezugriffe auf die gleichen Daten

In der gegenwärtigen Architektur wird zum Austausch der Daten zwischen den einzelnen Knoten ein NFS-Volume eingesetzt, auf das alle Knoten lesenden und schreibenden Zugriff haben.

Im Hinblick auf die oben angeführten Schwerpunkte ist NFS hinsichtlich der Redundanz recht effektiv, da die Daten nur einmal an zentraler Stelle vorgehalten werden. Dies ist allerdings im Hinblick auf die Verfügbarkeit gleichzeitig ein Nachteil, da bei Ausfall des NFS-Servers alle Knoten betroffen sind. Da NFS auf einem zustandsfreien Protokoll basiert (UDP)[27], ist es langsamer als andere Methoden, z.B. FTP, allerdings garantiert es einen transparenten Zugriff für die Klienten des Systems[28].

[27] Bei einem zustandsfreien Protokoll bleibt die Verbindung zwischen Server und Klient nicht erhalten, sondern wird für jede Datenübertragung erneut aufgebaut.

[28] Die Transparenz für den Klienten besteht darin, daß der Zugriff auf ein NFS-Verzeichnis identisch mit dem Zugriff auf ein lokales Verzeichnis ist.

4.3.4 Schnittstellen zu Shell-Skripts

Um die Steuerung des Prozesses gewährleisten zu können, war es notwendig, eine Schnittstelle zwischen den eigentlichen Programmen bzw. Skripts und dem ScriptServer zu schaffen. Realisiert wurde dies, indem eine Reihe von Rückgabewerten definiert wurde, die von dem ScriptServer nach Ausführung des Shell-Skripts erwartet werden und die die weitere Abarbeitung des Prozesses beeinflussen. Die erlaubten Werte sind in Tabelle 4.5 aufgeführt.

Rückgabewert	Bedeutung	Erläuterung
20	OK	Die Ausführung des Kommandos war erfolgreich, es kann mit der Abarbeitung der anderen Statements fortgefahren werden
21	FAILED, try again	Die Ausführung des Kommandos war nicht erfolgreich, aber eine Wiederholung hat u.U. Aussicht auf Erfolg
22	FAILED, try other	Die Ausführung des Kommandos war nicht erfolgreich, aber eine Wiederholung des Statements auf einem *anderen* Knoten kann erfolgreich sein
23	FAILED	Die Ausführung des Kommandos war nicht erfolgreich, jedoch kann mit der Abarbeitung des Prozesses fortgefahren werden
24	FATAL	Die Ausführung des Kommandos war nicht erfolgreich; die Bearbeitung des Prozesses muß abgebrochen werden (weil z. B. die folgenden Statements eine erfolgreiche Ausführung dieses Skripts erwarten)

Tabelle 4.5: erlaubte Rückgabewerte

4.3.5 Grammatik

Die Grammatik wurde auf der Basis der Anforderungen entworfen, die durch die bisher spezifizierten Prozesse vorgegeben werden und besteht derzeit nur aus wenigen Elementen. Eine Erweiterung der Funktionalität ist jedoch aufgrund der modularen Architektur des ScriptServers möglich.

Der Parser, der die Prozeßbeschreibungen entsprechend der definierten Grammatik verarbeitet, ist mit Hilfe des Parser-Compilers *JavaCC* (eine Beschreibung dieses Systems zur Erzeugung von Parsern aus einer vorgegebenen Grammatik findet sich in [Met99]) erzeugt worden und und generiert für die einzelnen Statements Java-Objekte, die das entsprechende Statement kapseln und dessen Ausführung überwachen. Die vollständige Syntax der Grammatik ist in Anhang A enthalten und wird im folgenden detailliert beschrieben.

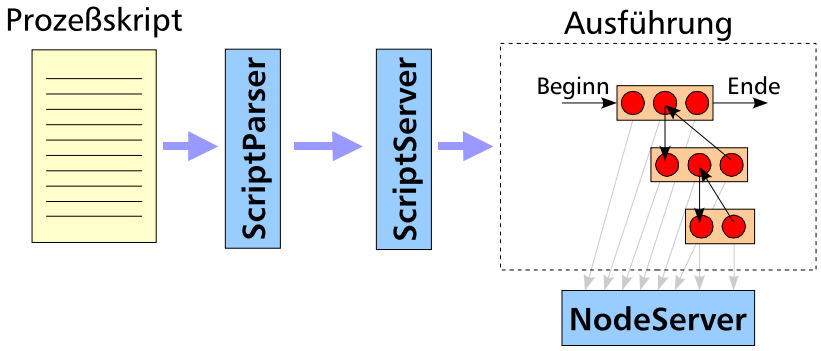


Abbildung 4.24: Ablauf der Prozeßverarbeitung

4.3.5.1 Allgemeiner Dateiaufbau

Bei den Skriptdateien, die den auszuführenden Prozeß beschreiben, handelt es sich um einfache Textdateien, die eine Reihe von *Statements* enthalten, die auf die auszuführenden Programme oder Skripte abgebildet werden. Mit den verschiedenen Varianten von Statements ergibt sich damit die folgende Struktur:

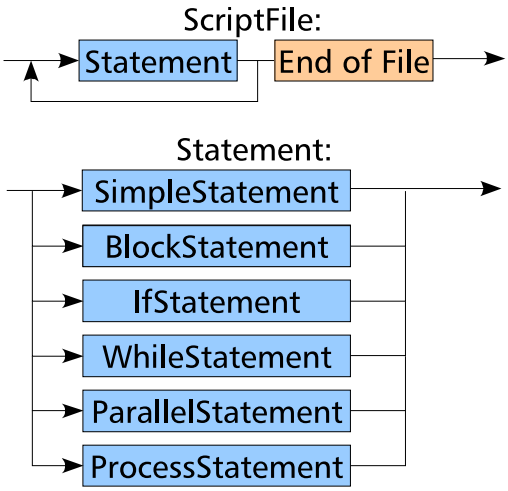


Abbildung 4.25: struktureller Aufbau des Prozess-Skripts

Jedes Statement innerhalb der Prozeßbeschreibung entspricht dabei mindestens einem Skriptaufruf. Die Zuordnung zwischen dem Namen des Betriebssystem-Skripts innerhalb des Prozessskripts und der entsprechenden Aufrufsyntax im NodeServer wird über eine entsprechde Eigenschaften (Properties) hergestellt. Die Syntax dieser Zuordnung ist dabei wie folgt definiert:

```
NodeServer.scripts=\
 <Name des Skripts> | <Aufrufsyntax für den KnotenServer> , \
 <Name des Skripts> | <Aufrufsyntax für den KnotenServer> ...
```

Hierbei dient das '|'-Zeichen als Trennzeichen zwischen Namen und Programmaufruf. In der Property können beliebig viele Zuordnungen festgelegt werden, die

untereinander durch das ','-Zeichen voneinander getrennt werden. Das '\'-Zeichen dient dabei nur dazu, den Text einer einzelnen Property auf mehrere Zeilen verteilen zu können, um eine übersichtlichere Darstellung zu erreichen[29].

Der Vorteil, der sich durch die Unterscheidung der Bezeichnung in der Prozeßbeschreibung und dem eigentlichen Skriptaufruf ergibt, ist die so gewonnene Unabhängigkeit von der verwendeten Plattform, die der Clusterknoten verwendet. Sollten sich also Pfadangaben oder die Aufrufsyntax der Shell-Skripte ändern, so ist lediglich eine Anpassung der entprechenden Properties für den betroffenen *NodeServer* erforderlich, während die Prozessbeschreibung unverändert bleiben kann. Dies ist insbesondere dann von Vorteil, wenn auf den einzelnen Knoten des Clusters unterschiedliche Betriebssysteme laufen, deren Spezifika verborgen werden müssen[30].

Das folgende Beispiel zeigt, wie eine solche Zuordnung aussehen kann. Hier wird für die Shellaufrufe die `bash`-Shell verwendet; an ihrer Stelle kann aber auch eine andere Shell (z. B. `sh` oder `csh`) oder ein beliebiges anderes ausführbares Programm eingesetzt werden, solange dies von dem Betriebssystem des Knotens unterstützt wird.

```
NodeServer.scripts=\
 KillTest|/bin/bash ./scripts/loop.sh,          \
 IfTest|/bin/bash ./scripts/if.sh,              \
 ElseTest|/bin/bash ./scripts/else.sh,          \
 ParallelTest|/bin/bash ./scripts/parallel.sh,  \
 MergerProcess|/bin/bash ./scripts/merger.sh,   \
 WhileTest|/bin/bash ./scripts/while.sh,        \
 simple|/bin/bash ./scripts/simple.sh,          \
 RetryOtherTest|/bin/bash ./scripts/other.sh,   \
 RetryTest|/bin/bash ./scripts/retry.sh
```

4.3.5.2 SimpleStatement

Die einfachste Variante eines Statements ist das *SimpleStatement*, bei dem entsprechend der in Abbildung 4.26 angegebenen Syntax ein Skriptname referenziert wird, der dann auf einem Nodeserver ausgeführt wird. Der Rückgabewert des Skripts wird wie oben beschrieben ausgewertet und mit der Ausführung des nächsten Statements fortgesetzt, sofern kein kritischer Fehler aufgetreten ist.

Abbildung 4.26: SimpleStatement

Der nachfolgende Quellcodeausschnitt zeigt ein Beispiel eines einfachen Prozeßscripts mit dem zugehörigen Betriebssystemskript. Anzumerken ist, daß die Verteilung von Statements auf mehrere Zeilen und das Voranstellen von Leer- oder

[29] Dies ist keine spezielle Eigenschaft dieser Property, sondern kann allgemein bei allen *Properties*-Objekten angewendet werden.

[30] Ein Beispiel für einen solchen Fall ist das Trennzeichen in Verzeichnisnamen, für das plattformabhänig meist \ oder / verwendet wird.

Tabulatorzeichen keine Auswirkungen hat. Damit wird die übersichtliche Erstellung von Prozeßbeschreibungen erleichtert.

Die Definitionen der gültigen Rückgabewerte erfolgt in einem weiteren Shell-Skript, das mit Hilfe des `source`-Kommandos eingebunden wird. Der Inhalt des Skripts ist in Anhang B enthalten.

einfaches Beispiel für ein SimpleStatement (simple.proc):

```
#
# checking some simple commands
#
exec("simple") exec("simple2")
  exec("simple") # with two trailing spaces
exec("simple") # and a leading tab character
```

zugehöriges Shellscript (simple.sh):

```
#/bin/bash
# ======== include the definition of the return values ==
source retvals.sh

# ================================== begin of script ==
echo "Called with value $RANDOM"
exit $RETVAL_OK
```

4.3.5.3 BlockStatement

Das BlockStatement, dessen syntaktischen Aufbau Abbildung 4.27 zeigt, faßt mehrere Einzelanweisungen zu einem Anweisungsblock zusammen und kann überall dort eingesetzt werden, wo normalerweise nur der Einsatz eines einzelnen Statements erlaubt ist, also z. B. innerhalb des Ausführungsteils einer *if*- oder *while*-Schleife.

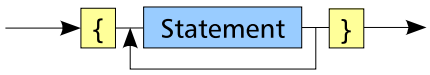


Abbildung 4.27: BlockStatement

Die Statements, die in einem BlockStatement enthalten sind, werden sequentiell in der Reihenfolge ausgeführt, in der sie angegeben wurden. Sollte ein Statement innerhalb des Blocks mit einer Fehlermeldung (`FAIL`) beendet werden, so führt dies nur zu einem Abbruch des BlockStatements. Falls es sich hierbei um einen schwerwiegenden Fehler handelt (`FATAL`), wird die Abarbeitung des gesamten Prozesses eingestellt.

Es folgt ein Beispiel für die Anwendung eines BlockStatements:

Beispiel für ein BlockStatement (block.proc):

```
#
# grouping simple commands in a block
#
{ exec("simple") exec("simple") exec("simple") exec("simple") }
{
  exec("simple")
  exec("simple")
  exec("simple")
}
```

4.3.5.4 ParallelStatement

Um die parallele Abarbeitung von Statements zu kennzeichnen, kann das *Parallel-Statement* verwendet werden.

Abbildung 4.28: ParallelStatement

Wie dem Syntaxdiagramm in Abbildung 4.28 zu entnehmen ist, kann ein einzelnes parallelisierbares Statement angegeben werden. Meist wird man an dieser Stelle ein BlockStatement einsetzen, welches dann die einzelnen parallel auszuführenden Statements enthält. Innerhalb dieses Blocks sind dann weitere Varianten denkbar. Insbesondere ist die Verwendung eines weiteren ParallelStatements möglich. Damit kann erreicht werden, daß mehrere Ausführungsstränge parallel abgearbeitet und zu einem späteren Zeitpunkt wieder zusammengeführt werden.

Sollte nach Beendigung der parallelisierten Ausführung des Statements noch eine Zusammenführung der (parallel erzeugten) Daten notwendig sein, kann dies durch das optionale *MergerScript* erfolgen, das als letztes Statement aufgerufen wird.

Das folgende Beispiel zeigt, wie ein ParallelStatement angewendet werden kann, um eine parallele Ausführung der Prozesse zu erreichen:

Beispiel für ein ParallelStatement (parallel.proc):

```
#
# executing statements in parallel and merging them at
# the end of the parallel execution thread
#
parallel("MergerProcess") {
  exec("ParallelTest")
  exec("ParallelTest")
  exec("ParallelTest")
  exec("ParallelTest")
  exec("ParallelTest")
  exec("ParallelTest")
}
```

Zunächst werden die Shellskripte, die durch den Namen "ParallelTest" identifiziert werden, parallel gestartet und auf die Beendigung jedes einzelnen Skripts gewartet. Die Skripte erzeugen jeweils ein Logfile, in das sie einige Informationen über Start- und Beendigungszeitpunkt schreiben und zwischendurch eine zufällig bestimmte Zeitspanne schlafen.

Shellskript für ParallelTest (parallel.sh):

```
#/bin/bash
# ======== include the definition of the return values ==
source retvals.sh

# ==================================== begin of script ==
# creating a logfile, writing some information in it and
# sleep for a random time
logfile=./test.log.$$
sleeptime=‘echo $(($RANDOM/ 1000))‘

echo "Started... (‘date‘) " >> $logfile

echo "Sleeping for $sleeptime seconds" >> $logfile
sleep $sleeptime

echo "... Stopped (‘date‘)" >> $logfile
exit $RETVAL_OK
```

Nach erfolgreicher Beendigung aller parallelen Statements wird das MergerSkript aufgerufen, das die einzelnen Logfiles in eine einzige Datei zusammenfasst und um eigene Informationen ergänzt (z. B. Start- und Stopzeit, Anzahl der gefundenen Logfiles). Schließlich werden die einzelnen Logfiles gelöscht.

Shellskript für MergerProcess (merger.sh):

```
#/bin/bash
# ======== include the definition of the return values ==
source retvals.sh

# ==================================== begin of script ==
# creating a logfile, writing some information in it and
# sleep for a random time
echo "Merging started... (‘date‘) " > merger.log
echo -n >> merger.log
echo "Found the following logs:" >> merger.log
ls -alh ./test.log.* >> merger.log
echo -n >> merger.log
cat ./test.log.* >> merger.log
rm -f ./test.log.*
echo -n >> merger.log
echo "Merging completed... (‘date‘) " >> merger.log
exit $RETVAL_OK
```

Die entstehende Logdatei des MergerSkripts hat dann das folgende Aussehen:

Logdatei des Mergerskripts (merger.log):

```
Merging started... (Die Aug 15 13:22:03 CEST 2000)
Found the following logs:
-rw-rw-r-- 1 boehm boehm 111 Aug 15 13:21 ./test.log.1596
-rw-rw-r-- 1 boehm boehm 112 Aug 15 13:22 ./test.log.1603
-rw-rw-r-- 1 boehm boehm 112 Aug 15 13:22 ./test.log.1620
-rw-rw-r-- 1 boehm boehm 111 Aug 15 13:22 ./test.log.1629
-rw-rw-r-- 1 boehm boehm 112 Aug 15 13:22 ./test.log.1636
-rw-rw-r-- 1 boehm boehm 112 Aug 15 13:22 ./test.log.1647
Started... (Die Aug 15 13:21:39 CEST 2000)
Sleeping for 4 seconds
... Stopped (Die Aug 15 13:21:43 CEST 2000)
Started... (Die Aug 15 13:21:39 CEST 2000)
Sleeping for 23 seconds
... Stopped (Die Aug 15 13:22:02 CEST 2000)
Started... (Die Aug 15 13:21:39 CEST 2000)
Sleeping for 15 seconds
... Stopped (Die Aug 15 13:21:54 CEST 2000)
Started... (Die Aug 15 13:21:39 CEST 2000)
Sleeping for 2 seconds
... Stopped (Die Aug 15 13:21:41 CEST 2000)
Started... (Die Aug 15 13:21:39 CEST 2000)
Sleeping for 22 seconds
... Stopped (Die Aug 15 13:22:01 CEST 2000)
Started... (Die Aug 15 13:21:39 CEST 2000)
Sleeping for 10 seconds
... Stopped (Die Aug 15 13:21:49 CEST 2000)
Merging completed... (Die Aug 15 13:22:04 CEST 2000)
```

4.3.5.5 SubProcessStatement

Das *SubProcessStatement*, in Abbildung 4.29 dargestellt, dient der Einbeziehung von weiteren Prozeßbeschreibungen zur Laufzeit der Prozeßausführung, d. h., daß die Beschreibung, die durch den angegebenen Dateinamen referenziert wird, erst bei Erreichen der entsprechenden Stelle im Prozeßskript eingelesen und ausgeführt wird. Das SubProcessStatement stellt damit eine *include*-Funktionalität bereit, wie sie von vielen anderen Sprachen verwendet wird. Im Unterschied zu den üblichen Methoden werden die einzuschließenden Teile jedoch nicht vor dem Parsen des Skripts *statisch* eingebunden, sondern *dynamisch* zu dem Zeitpunkt, wenn der Parser auf das Statement trifft.

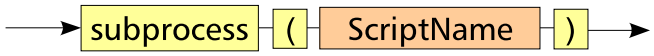


Abbildung 4.29: ProcessStatement

Damit ist es insbesondere möglich, daß der aktuelle Prozeß selbst mittels geeigneter Skripte ein neues Prozeß-Skript erzeugt, daß dann zu einem späteren Zeitpunkt innerhalb des aktuellen Prozesses ausgeführt wird. Dies ermöglicht es dem Autor des Prozeßskripts, eine variable Reaktion auf sich ändernde Situationen innerhalb

der Prozeßumgebung vorzusehen, bzw. eine Änderung existierender Skripte, um z. B. die Dateinamen neu erzeugter Daten im Skript anzupassen.

Diese Funktionalität beeinflußt zwar die Verarbeitungsgeschwindigkeit des Parsers und steigert den Implementierungsaufwand, da einerseits der Eingabestrom nicht mehr als fest angenommen werden kann und andererseits der generierte Parser nicht statisch ist, wurde aber explizit benötigt und wird für die automatische Indexerstellung innerhalb des Wortschatzprojektes genutzt.

4.3.5.6 IfStatement

Um eine alternative Skriptausführung zu ermöglichen, wurde das *IfStatement* implementiert, das in Abbildung 4.30 dargestellt ist.

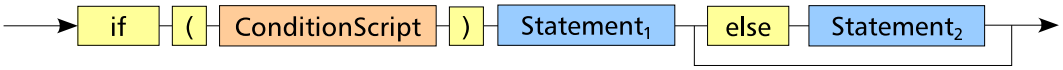


Abbildung 4.30: IfStatement

Zugunsten einer möglichst großen Flexibilität in Bezug auf die extern im Knotenbetriebssystem ausgeführten Shell-Skripts wurde bei Entwurf der ProzessSkript-Grammatik auf die direkte Implementierung der Bedingungslogik und entsprechender Operatoren verzichtet. Stattdessen wird ein *ConditionSkript* aufgerufen, welches für den gegebenen Fall entscheiden soll, ob eine Bedingung wahr oder falsch ist und dann einen entsprechenden Rückgabewert liefert. Die erlaubten Rückgabewerte sind in Tabelle 4.6 angegeben.

Rückgabewert	Bedeutung	Erläuterung
10	TRUE	Die von dem ConditionScript untersuchte Bedingung ist wahr, d. h. das IfStatement kann ausgeführt werden.
11	FALSE	Die von dem ConditionScript untersuchte Bedingung ist falsch, d. h. es wird der alternative Teil des IfStatements ausgeführt, sofern er vorhanden ist.

Tabelle 4.6: Erlaubte Rückgabewerte des ConditionScripts

Andere als die angegebenen Werte werden als nicht zulässig definiert. Sollte dennoch ein anderer Rückgabewert von dem ConditionSkript zurückgeliefert werden, so wird dies als FALSE interpretiert und in der Logdatei ein entsprechener Eintrag generiert. Dieses Verhalten wurde gewählt, um auch im Fehlerfall die Prozeßausführung so robust wie möglich zu gestalten. Alternativ könnte man diesen Fall als illegal betrachten und die Verarbeitung des Prozeßskripts abbrechen. Hierfür ist eine Änderung des Objektes, das das *IfStatement* modelliert, notwendig.

Das folgende Beispiel zeigt die Anwendung eines solchen IfStatements. Zunächst wird das ConditionStatement ausgeführt und anhand dessen Rückgabewert entschieden, welches Statement ausgeführt werden soll.

Beispiel für ein IfStatement (IfStatement.proc):

```
#/bin/bash
# ============= include the definition of the return values ==
source retvals.sh

# ========================================== begin of script ==
# demonstration of a simple IfStatement with alternating
# branches and the optional use of a BlockStatement
#
if ("ConditionTest") {
  # use a block here to encapsulate several stmts.
  exec("Simple")
  exec("IfTest")
} else
  exec("ElseTest")
```

Das ConditionScript hat folgendes Aussehen:

Conditionscript für das IfStatement-Beispiel (condition.sh):

```
#/bin/bash
# ============= include the definition of the return values ==
source retvals.sh

# ========================================== begin of script ==
# condition script that checks the existence of a certain file
# (the file is created if is does not already exist and
# removed, when it was already present to ensure that the
# script delivers different results on each invocation)

TESTFILE=./condition.dummy

if [ -a $TESTFILE ]; then
    rm $TESTFILE
    echo "Condition was TRUE"
    exit $COND_TRUE
else
    echo "testfile" > $TESTFILE
    echo "Condition was FALSE"
    exit $COND_FALSE
fi
```

4.3.5.7 WhileStatement

Für die Realiserung von Schleifen steht das *while*-Statement zur Verfügung, welches in Abbildung 4.31 dargestellt ist. In analoger Weise zum IfStatement wird

zunächst ein ConditionSkript ausgeführt, um die Schleifenbedingung zu überprüfen. Solange diese Bedingung dem logischen Wert 'wahr' entspricht, wird das Statement im Schleifenrumpf ausgeführt. Sollte während der Ausführung des Statements ein nicht behebbarer Fehler (`FAIL`) auftreten, so führt dies zum Abbruch der Schleife.

Abbildung 4.31: WhileStatement

Andere, häufig anzutreffende Schleifenvarianten, wie *for*- oder *do-while*-Schleifen, werden momentan nicht benötigt und sind daher (noch) nicht implementiert. Da es sich bei diesen Konstrukten jedoch nur um Variationen der implementierten Schleifenanweisung ohne andere Funktionalität handelt, ist eine Nachbildung dieses Verhaltens durch eine entsprechende Formulierung der Prozessbeschreibung und geeignete ConditionsSkripts möglich.

Das vorzeitige Verlassen einer Schleife durch eine Sprunganweisung (oft als *break* notiert) ist nicht vorgesehen, kann aber durch einen entsprechenden Skript-Rückgabewert simuliert werden. Im Gegensatz hierzu gibt es für den unmittelbaren Sprung zum Schleifenkopf mittels *continue* kein Äquivalent. Außerdem sind unbedingte Sprünge (*goto*-Anweisungen) generell nicht vorgesehen.

Die genannten Einschränkungen sind allerdings nicht prinzipieller Art, sondern können bei Bedarf durch Veränderung der Grammatik beseitigt werden. Die Erzeugung des neuen Parsers kann automatisiert mit Hilfe des Parser-Compilers erfolgen und die zusätzlich notwendige Ausführunglogik kann in den entsprechenden Statement-Klassen implementiert werden.

Das abschließende Beispiel zeigt die Anwendung des WhileStatements unter Verwendung des gleichen ConditionScripts wie im Beispiel für das IfStatement. Im Ergebnis wird der Block des Statements nur dann einmal ausgeführt, wenn die Testdatei, deren Existenz durch das ConditionSkript überprüft wird, bereits existiert.

Beispiel für ein WhileStatement (while.proc):

```
#/bin/bash
# ============= include the definition of the return values ==
source retvals.sh

# ========================================= begin of script ==
# demonstration of a WhileStatement with the same
# ConditionScript as in the IfStatement-example.
#
while ("ConditionTest") {
  exec("Simple")
  exec("Simple")
}
```

4.4 Der BinaryServer

Nicht alle Anwendungen des Wortschatzprojektes sind datenbankorientiert oder verfügen über eine Client-Server-Architektur, die den Einsatz auf dem Cluster in analoger Weise zu der Verteilung der DBMS auf die einzelnen Knoten ermöglichen würde. Trotzdem kann es sinnvoll sein, solche Anwendungen in die neue Umgebung einzubinden und verteilt zu nutzen.

Diese spezialisierten Programme bestehen zumeist aus einem ausführbaren Programm (*binary*) und einer Anzahl von Hilfsdateien, die die notwendigen Daten enthalten. Notwendige Parameter werden dem Programm entweder direkt beim Programmaufruf übergeben oder von dem Standard-Eingabestrom `stdin` gelesen. Die Ergebnisse des Aufrufs werden von der Anwendung direkt an den Standard-Ausgabestrom (`stdout`) gesendet, während Fehlermeldungen oder Statusmeldungen über den Fehler-Ausgabestrom (`stderr`) ausgegeben oder als Rückgabewert kodiert werden.

Eine solche Anwendung innerhalb des Wortschatzprojektes ist das Anagramm-Programm von Martin Läuter, das zu einem gegebenen Wort alle Anagramme berechnet; das sind diejenigen Worte oder Wortgruppen, die sich durch Permutation der im ursprünglichen Wort enthaltenen Buchstaben ergeben. Weiterführende Informationen zu Anagrammen finden sich in [Tho00].

Weitere Anwendungen, die in diese Klasse fallen, sind denkbar; so könnte beispielsweise die Berechnung der Kollokationsgraphen für die Web-Schnittstelle des Wortschatzprojekts auf dem Cluster mit Hilfe eines ähnlichen Verfahrens erfolgen.

Um die beschriebene Klasse von Anwendungen zu unterstützen, müssen innerhalb der MetaServer-Plattform eine Reihe von Voraussetzungen geschaffen werden:

- Realisierung einer verteilten Aufrufschnittstelle, die das entsprechende Programm (*BinaryService*) parametrisiert, dessen Ausführung auf dem Clusterknoten überwacht und die Ergebnisse des Aufrufs bereitstellt. Als Interface zu dem BinaryService dienen die verschiedenen Ein- und Ausgabeströme, sowie der Rückgabewert des Programms.

- Schaffung einer Schnittstelle zu den eigentlichen Anwendungsprogrammen oder Klienten außerhalb der Cluster-Umgebung. Hierbei ist es wichtig, daß die Schnittstelle verschiedene Verbindungsarten unterstützt und die Übermittlung der Daten (Aufrufparameter und Ergebnisse) in dem vom Klient geforderten Format erfolgt.

Bei der Implementierung kommen die in Abbildung 4.32 dargestellten Komponenten zum Einsatz, wobei dem *BinaryServer* die Anbindung der externen Anwendungsprogramme und – mit Hilfe des Schedulers – die Verteilung der Anfragen auf die verschiedenen Clusterknoten obliegt, während die Ausführung und Überwachung der *BinaryServices* von einer Komponente des NodeServers durchgeführt wird. Die Kommunikation zwischen dem *BinaryServer* und der *ServiceComponent* des NodeServers erfolgt dabei über die exportierte NodeServer-Schnittstelle unter Verwendung des RMI-Protokolls.

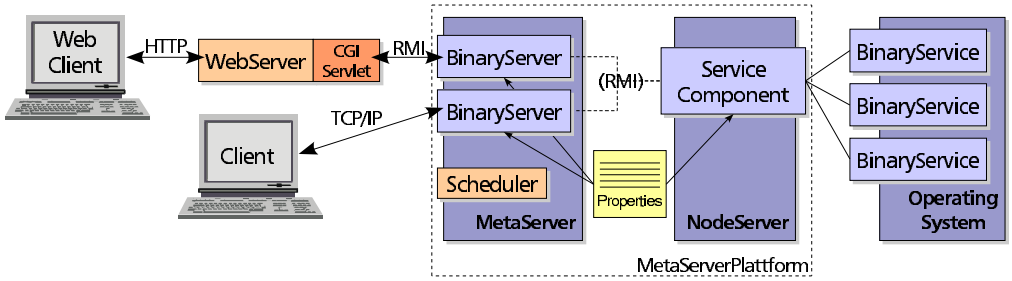


Abbildung 4.32: Architektur des Binary Servers

Alle Daten, die an die *BinaryServices* übergeben, bzw. von diesen geliefert werden, gelangen als Zeichenketten vom Datentyp `String` zum *BinaryServer* und werden von dort an den Klienten weitergeschickt. Damit wird dem breiten Anwendungsspektrum Rechnung getragen, daß die Übertragung nahezu aller Arten von Informationen erlaubt. Bei der Übertragung großer Datenmengen wäre allerdings die Verwendung von Datenströmen (Streams) an Stelle von Zeichenketten effektiver. Da diese Datentypen jedoch nicht serialisierbar sind und damit die Grundvoraussetzung für die Verwendung im Zusammenhang mit dem RMI-Protokoll nicht erfüllen, ist hierfür ein alternativer Übertragungsweg (z. B. über TCP-Sockets) zu implementieren. Da dieser Fall bei den gegenwärtigen Anwendungen nicht auftritt, wird die Verwendung von Streams derzeit nicht unterstützt.

Welche *BinaryServices* von der ServiceComponent des NodeServers angeboten werden, sowie deren Aufrufsyntax wird durch entsprechende Eigenschaften (Properties) für jeden Cluster-Knoten separat bestimmt.

Für den BinaryServer gibt es zwei Möglichkeiten der Konfiguration. Die variablen Parameter sind einerseits die Art der Verbindung (z. B. RMI oder TCP/IP) zu dem Klienten mitsamt der entsprechenden Konfigurationsparameter (z. B. Portnummer oder Name des zu exportierenden Interfaces) und andererseits die zu verwendenden BinaryServices sowie die Art und Weise, mit der diese Informationen an den Klienten übermittelt werden.

Während sich die Verbindungsoptionen über entsprechende Eigenschaften festlegen lassen, ist dies bei den Aspekten, die den BinaryService an sich betreffen, nicht so einfach möglich. Wahlweise kann man eine allgemeine Schnittstelle definieren, die die verschiedenen Programme einheitlich behandelt und eine generische Parametrisierung mit dem Namen des zu verwendenden Dienstes und der notwendigen Parameter erlaubt oder für die einzelnen BinaryServices entsprechende BinaryServer implementieren, die die spezifische Funktionalität eines Dienstes kapseln und nur noch mit den entsprechenden Parametern versehen werden. Die Art des Dienstes wird über den Zugangsweg zu dem BinaryServer bestimmt.

Die generische Lösung bietet den Vorteil des gleichbleibend geringen Implementierungsaufwandes auch bei einer hohen Anzahl verschiedener BinaryServices. Außerdem wird dem Client der Zugang zu verschiedenen Services über eine einzige Verbindung ermöglicht. Nachteilig ist zum einen, daß durch die generische Verarbeitung der Daten nicht auf spezifische Ansprüche der Klient-Anwendung an

bestimmte Datenformate oder Protokolle eingegangen werden kann und andererseits die feste Kodierung des Dienstnamens in der Klientanwendung, durch die sich eine direkte Abhängigkeit zu der Konfiguration des Clusters ergibt.

Der spezialisierte Ansatz hingegen erfordert einen höheren Implementierungsaufwand, der mit der Anzahl der zur Verfügung stehenden Dienste zunimmt. Andererseits kann man durch die individuell angepaßten BinaryServer besser auf die Anforderungen verschiedener Klienten eingehen. Dies ermöglicht beispielsweise ein differenzierteres Sicherheitskonzept oder die Bereitstellung verschiedener Datenformate für den gleichen BinaryService.

Eine solche Anwendung könnte beispielsweise die Generierung von Kollokationsgraphen sein, die unter Verwendung des HTTP-Protokolls[31] mit einen Web-Browser kommuniziert, wobei die notwendigen Parameter in einem URL übergeben werden und die erzeugte Grafik mit dem korrekten MIME-Typ[32] zurückgeliefert wird. Damit könnte der sonst notwendige Umweg über CGI oder Servlets[33] für die Integration in den Web-Server, wie in Abbildung 4.32 auf der vorherigen Seite, umgangen werden.

Die gegenwärtige Implementierung verfügt über einen BinaryServer für die verteilte Berechnung von Anagrammen, der mit dem NodeServer auf die beschriebene Art und Weise zusammenarbeitet und auf den Klienten sowohl über RMI als auch TCP/IP zugreifen können. Für die Integration in die Web-Schnittstelle des Wortschatz-Projekts ist eine Klient-Anwendung – z. B. als Servlet – zu entwerfen, die den Datenaustausch zwischen Web-Server und BinaryServer organisiert und in HTML-Seiten integriert ist, die für die Präsentation der Daten zuständig sind. Damit ist die direkte Anwendbarkeit des Konzepts des BinaryServers gezeigt und kann als Vorlage für ähnliche Dienste dienen. Wie oben dargestellt, bieten sich dafür zwei verschiedene Strategien an, die ihre jeweiligen Vorzüge und Nachteile haben.

[31] HTTP: Hypertext Transfer Protocol, wird für die Kommunikation zwischen Web-Server und -Browser verwendet.

[32] MIME: Multipurpose Internet Mail Extension ist ein Nachrichtenformat, daß den Transport zusammengesetzter getypter Daten ermöglicht. Ursprünglich wurde dieses Format für den E-Mail-Transport entwickelt; es wird aber heute auch innerhalb anderer Protokolle eingesetzt, beispielsweise im HTTP-Protokoll, um den Typ der gesendeten Daten auszuzeichnen.

[33] Das *Common Gateway Interface* (CGI) ist eine Schnitstelle zu extenen Programmen, die dazu benutzt werden können, die Funktionalität des Webservers zu erweitern. *Servlets* verfolgen das gleiche Ziel, implementieren jedoch die benötigte Funktionalität in einem Modul, das vom Webserver bei Bedarf geladen wird.

Kapitel 5

Zusammenfassung

Der zunehmende Erfolg des Wortschatzprojektes hat dazu geführt, daß die bis dato eingesetzte Infrastruktur an ihre systembedingten Grenzen stößt. Steigende Datenmengen, die Zunahme der Zahl der verschiedenen Analyseverfahren und wachsende Benutzerzahlen gaben Anlaß zur Suche nach einem alternativen Konzept, das das Potential hat, die gestellten Anforderungen auch langfristig erfüllen zu können.

Ausgehend von der Grundlage eines erweiterbaren Clustersystems, das aus Standard-PCs aufgebaut ist, wurde ein Konzept für eine das Wortschatzprojekt unterstützende Infrastruktur aufgebaut, wobei sich das zur Verfügung gestellte Funktionsspektrum stetig erweiterte und das urspünglich sehr eng gesteckte Entwicklungsziel zugunsten einer allgemeineren Struktur aufgegeben werden mußte. So entstand im Ergebnis ein Applikationsserver, der auf die Bedürfnisse linguistischer und speziell datengetriebener korpuslinguistischer Software ausgerichtet ist, zu dessen Vertretern das Projekt “Deutscher Wortschatz” gehört.

Die Untersuchungen von Applikationsservern als neue, eigenständige Softwaregattung hat ergeben, daß es sich hierbei um ein junges, noch in Entwicklung befindliches Teilgebiet der Systemsoftware handelt, für das sich noch keine allgemeine Definition durchgesetzt hat. Der objektive Vergleich verschiedener Applikationsserver gestaltet sich schwierig, da die Systeme meist mit einem breiten Funktionsspektrum ausgestattet sind, die in den seltensten Fällen direkt vergleichbar sind. Applikationsserver sind zwar prinzipiell anwendungsunabhängig, wurden aber im Einzelfall immer für ein bestimmtes Anwendungsspektrum entworfen. Zudem sind sowohl der Einarbeitungsaufwand als auch die finanziellen Aufwendungen für kommerzielle Produkte üblicherweise hoch. Die Auswahl eines Applikationsservers für ein konkretes Projekt muß daher aufgrund der oben angeführten Gründe sehr sorgfältig abgewogen werden. Innerhalb der Arbeit wurde versucht eine möglichst generische Definition für den Begriff Applikationsserver herzuleiten und ein Eignungskoeffizient vorgeschlagen, der den Vergleich verschiedener Produkte unabhängig von deren Funktionsumfang aber bezogen auf die zu unterstützenden Anwendungen erleichtern soll.

Die Analyse der Eigenschaften von linguistischer Software führte – verglichen mit Standardanwendungen – zu einer sehr differnzierten Bild in Bezug auf die benötig-

te Funktionalität. Da die Verarbeitung sprachlicher Daten momentan noch immer einen Nischenmarkt besetzt, existiert keine mit einem Applikationsserver vergleichbare Software, die alle benötigten Funktionen unterstützt. Insbesondere aufgrund der Verwendung einer Cluster-Architektur und dem hohen Facettenreichtum der im Wortschatzprojekt eingesetzten Software, überstreichen die benötigten Funktionen mehrere Teilgebiete und würden bei Verwendung kommerzieller Software den Einsatz mehrerer Lösungen erfordern (ein paralleles DBMS, ein Applikationsserver und Software für die Parallelisierung von Prozessen), von denen jeweils nur ein Bruchteil der Funktionalität genutzt werden würde. Außerdem sind eventuell entstehende Ineroperabilitätsprobleme durch den Anwender zu lösen. Diese besondere Situation unterstreicht die Individualität datengetriebener linguistischer Software und rechtfertigt die Entwicklung einer eigenen spezialisierten Lösung.

5.1 Entwicklungsstand der Implementierung

Im Rahmen der Diplomarbeit wurde ein funktionsfähiger, ausbaubarer Prototyp entwickelt, der ca. 30.000 Zeilen Java-Programmkode umfaßt, der sich auf ca. 200 Klassen verteilt, sowie verschiedene Skripts und andere Hilfsdateien, die für die Ausführung der verschiedenen Server notwendig sind.

Zusammen mit der installierten und konfigurierten Cluster-Hardware ist die Implementierung innnerhalb des Wortschatzprojektes bereits einsetzbar und kann Teilaufgaben innerhalb des Projekts übernehmen. Die verschiedenen implementierten Dienste unterstützen die Aufgaben, die innerhalb des Projekts anfallen und können bei Bedarf erweitert werden. Außerdem können neue Plattformanwendungen entworfen und in die MetaServer-Plattform integriert werden, um Funktionen zur Verfügung zu stellen, die zur Zeit noch nicht benötigt werden.

Während der Implementierung wurde darauf geachtet, die entsprechenden Funktionsgruppen vollständig und transparent zu implementieren, um eine möglichst einfache Einbindung der Klientanwendungen zu ermöglichen. Dies hat dazu geführt, daß die meisten Klientanwendungen ohne spezielle Veränderungen oder Anpassungen auf die Clusterarchitektur zugreifen und deren Vorteile nutzen können. Damit wird die Einsatzfähigkeit von bestehenden Anwendungen, die nicht eigens für die Clusterarchitektur entwickelt wurden, gewährleistet und die Akzeptanz der neuen Infrastruktur erhöht. Um die Anwendungen auf die speziellen Eigenschaften des Clusters anpassen zu können, sind relevante Parameter konfigurierbar (z. B. die JDBC-Clientparameter) und unterliegen damit dem Einfluß der Klientanwendung.

Die Funktionalität und Korrektheit der Implementierung wurde mit den üblichen Klientanwendungen und einigen Testprogrammen überprüft und erfüllt die an sie gestellten Erwartungen.

5.2 Ausblick

Hinsichtlich der Leistungsfähigkeit der Gesamtarchitektur stehen umfangreichere Tests noch aus. Insbesondere ist hierbei der Vergleich mit der momentan eingesetzen Architektur von Interesse. Da der Prototyp noch nicht optimiert wurde, sind hier noch Leistungsreserven zu erwarten, die bislang nicht genutzt werden.

Die vorliegende Implementierung stellt bereits ein breites auf die Bedürfnisse des Wortschatzprojektes angepaßtes Funktionsspektrum bereit, dessen Einsatzfähigkeit wurde bereits an einigen Beispielanwendungen demonstriert wurde.

Für den erfolgreichen Einsatz und die Weiterentwicklung des Prototypen in eine Software für den produktiven Einsatz ist vor allem die Akzeptanz durch die Benutzer entscheidend. Eine aktive Mitarbeit interessierter Entwickler bei der Ausgestaltung des Applikationsservers und dessen Funktionsumfangs wäre hierbei wünschenswert, um den notwendigen Reifegrad zu erreichen, der von einem Entwicklungswerkzeug dieser Art erwartet wird.

Da das Wortschatzprojekt von mehreren Entwicklern mit unterschiedlichsten Aufgabengebieten und Zielstellungen betrieben wird, erfordert dies vor allem in der Einführungsphase eine über die eigene Anwendung hinausgehende Zusammenarbeit.

Anhang A

Skript-Grammatik

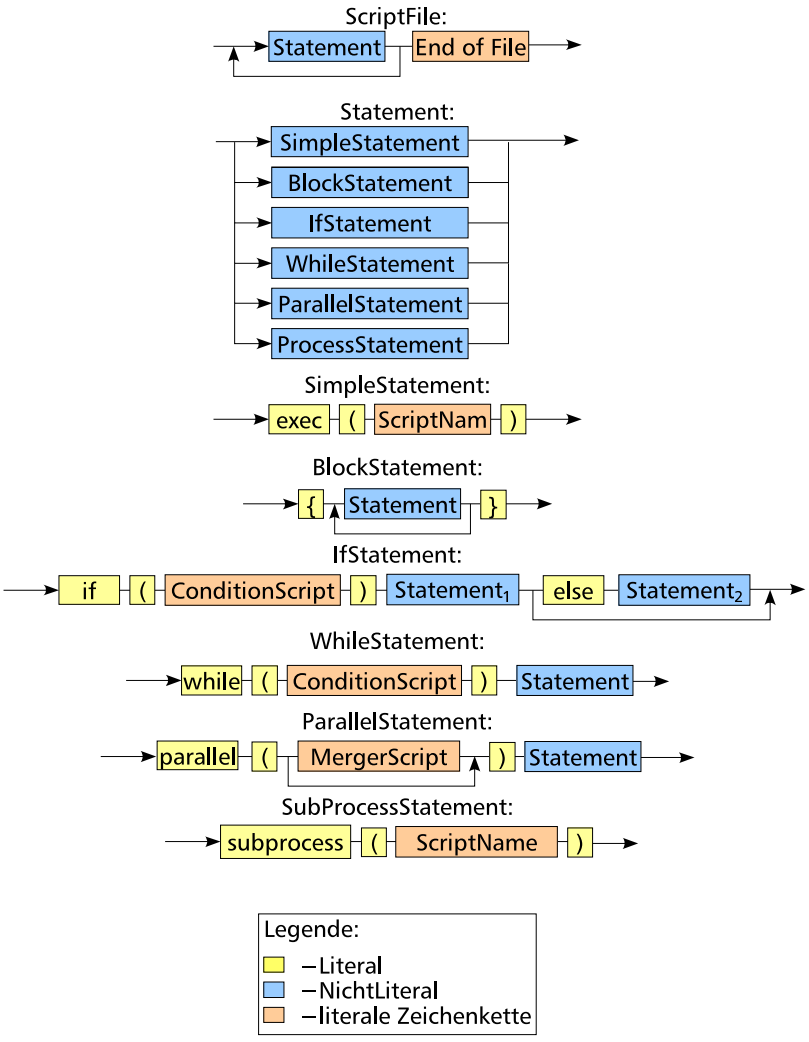


Abbildung A.1: Skriptgrammatik

Anhang B

Skript-Template für die Definition der Rückgabewerte

Skript-Template zur Definition der entsprechenden Werte für den ScriptServer (retvals.sh):

```
#
# RETURN-values
# -------------
# definition of the return values used for the communication
# with the cluster
#
RETVAL_OK=20
RETVAL_RETRY=21
RETVAL_OTHER_NODE=22
RETVAL_FAIL=24
RETVAL_FATAL=25
#
# CONDITION-values
# ----------------
# used for the evaluation of conditions in the scripts
COND_TRUE=10
COND_FALSE=11
```

Anhang C

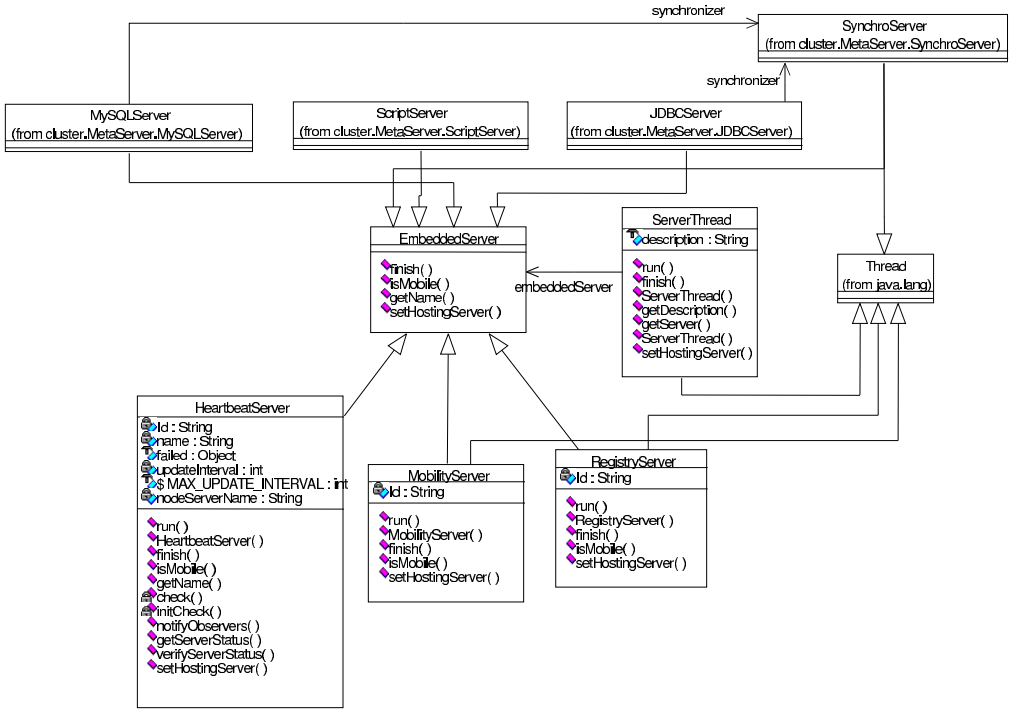
Klassendiagramme

Die folgenden Klassendiagramme wurden mit Rational Rose erstellt und stellen die wichtigsten Klassen des implementierten MetaServers und deren Beziehungen untereinander in UMT-Notation dar. Aus Gründen der Übersichtlichkeit sind nicht immer alle Attribute und Methoden vollständig dargestellt.

C.1 Der MetaServer

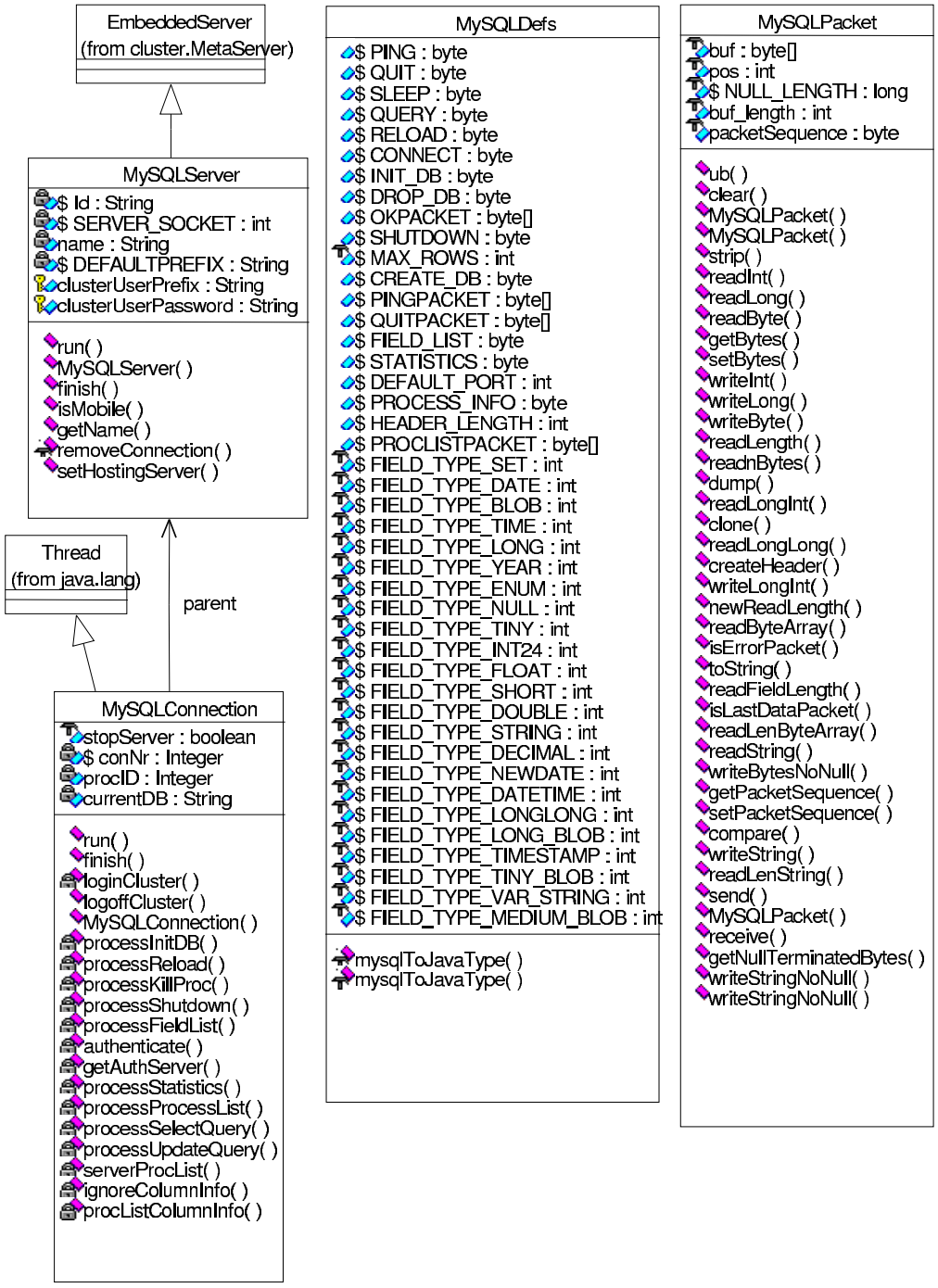
Eingebette Server

Dieses Diagramm stellt die in der Plattform eingebetteten Server dar, die zu diesem Zweck das *EmbeddedServer*-Interface implementieren.



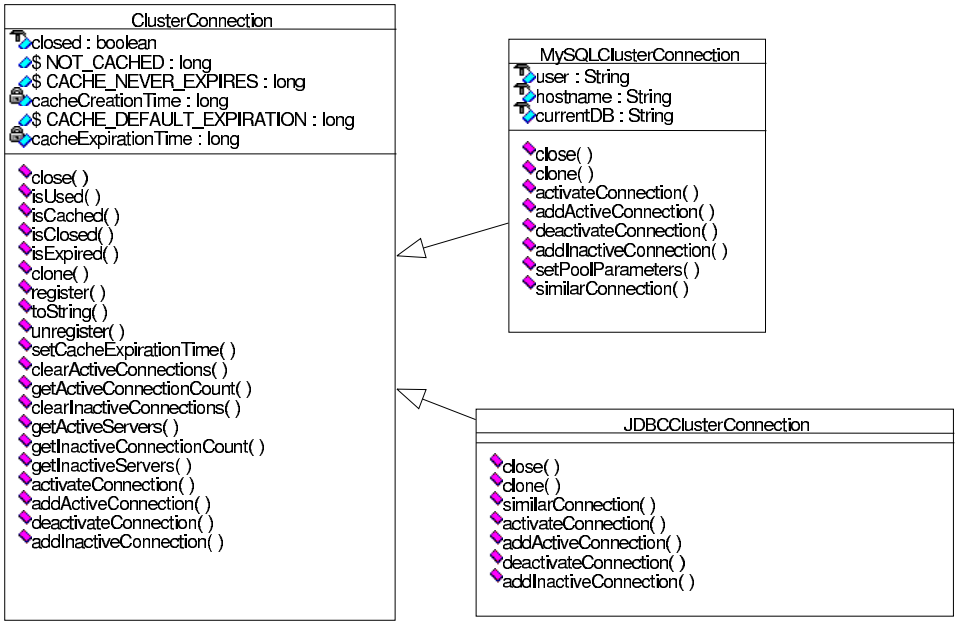
MetaServer-Hauptklassen

Die Implementierung des Plattformkerns erfolgt durch die im folgenden Diagramm angegebenen Klassen und die in dem folgenden Abschnitt angegebenen Hifsklassen.

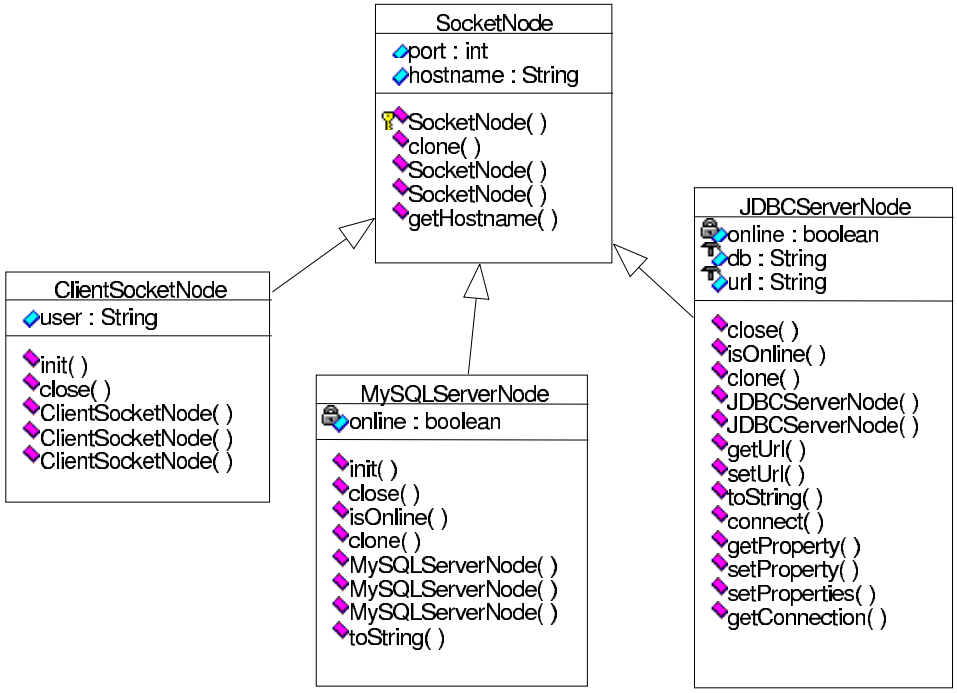


Hilfsklassen

Das folgende Diagramm zeigt die Klassen, die zur einheitlichen Abbildung verschiedener Verbindungstypen benutzt werden.

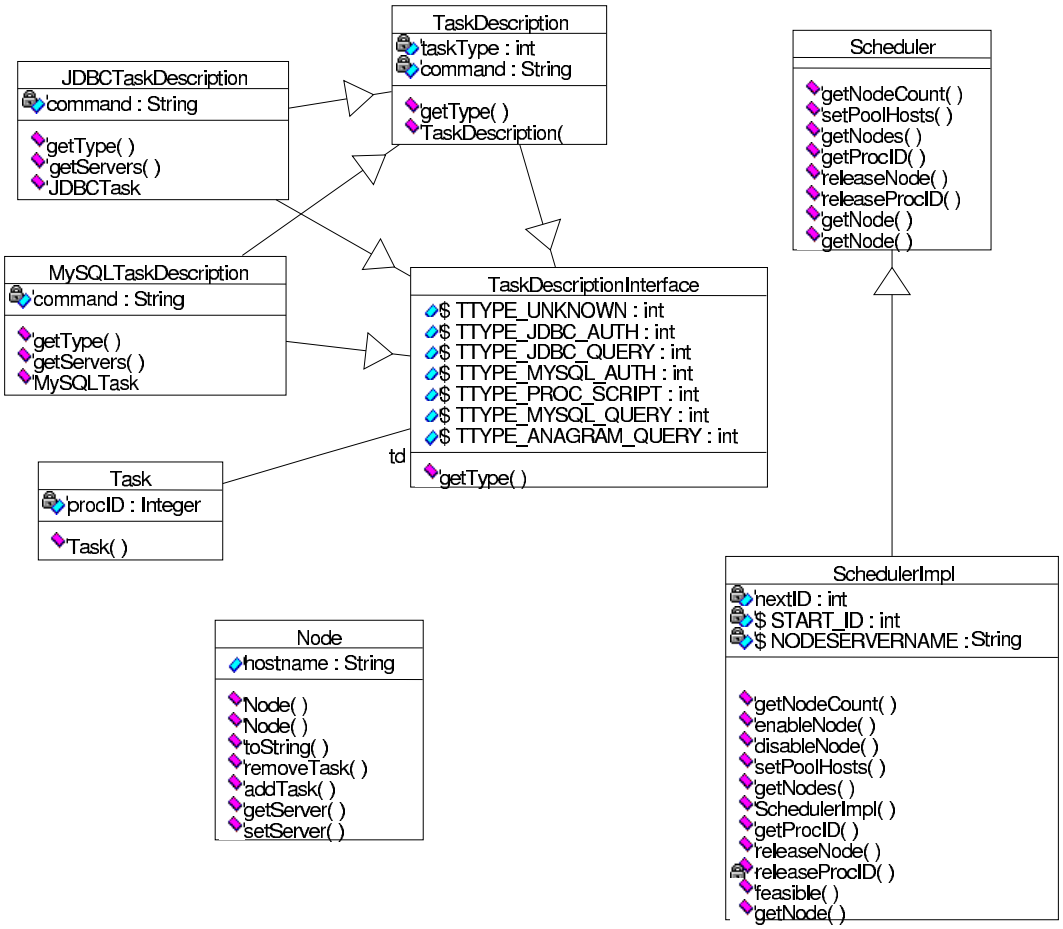


Die Modellierung der verschiedenen Kommunikationsendpunkte (Klientenverbindungen, Serviceanbieter auf dem einzelnen Knoten etc.) erfolgt über eine Reihe von Klassen, die in der unten dargestellten Hierarchie angeordnet sind.



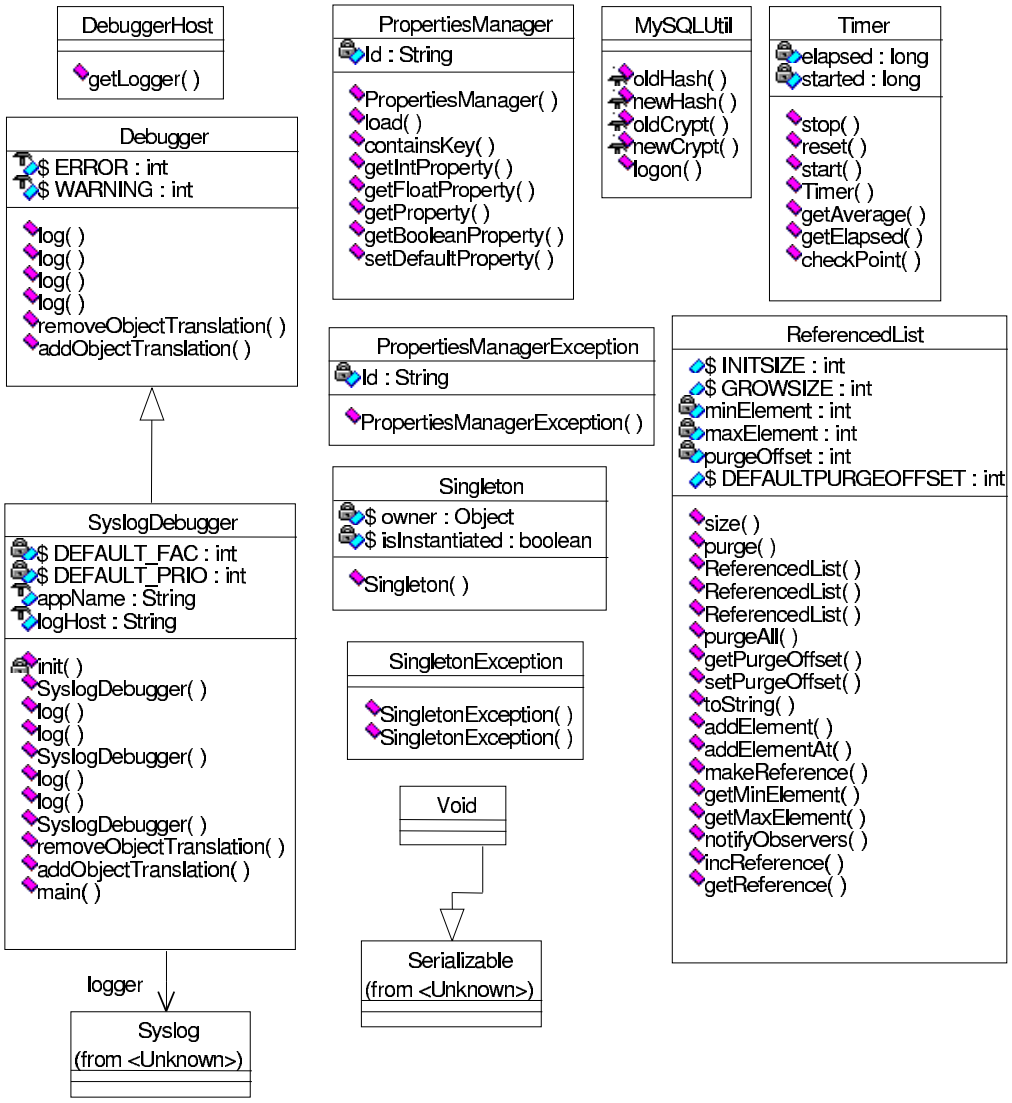
C.2 Der Scheduler

Die Scheduling-Umgebung wird mit den unten dargestellten Klassen realisiert, abstrahiert von der verteilten Struktur des Clusters und bettet die zu verwendenden Scheduling-Algorithmen ein.



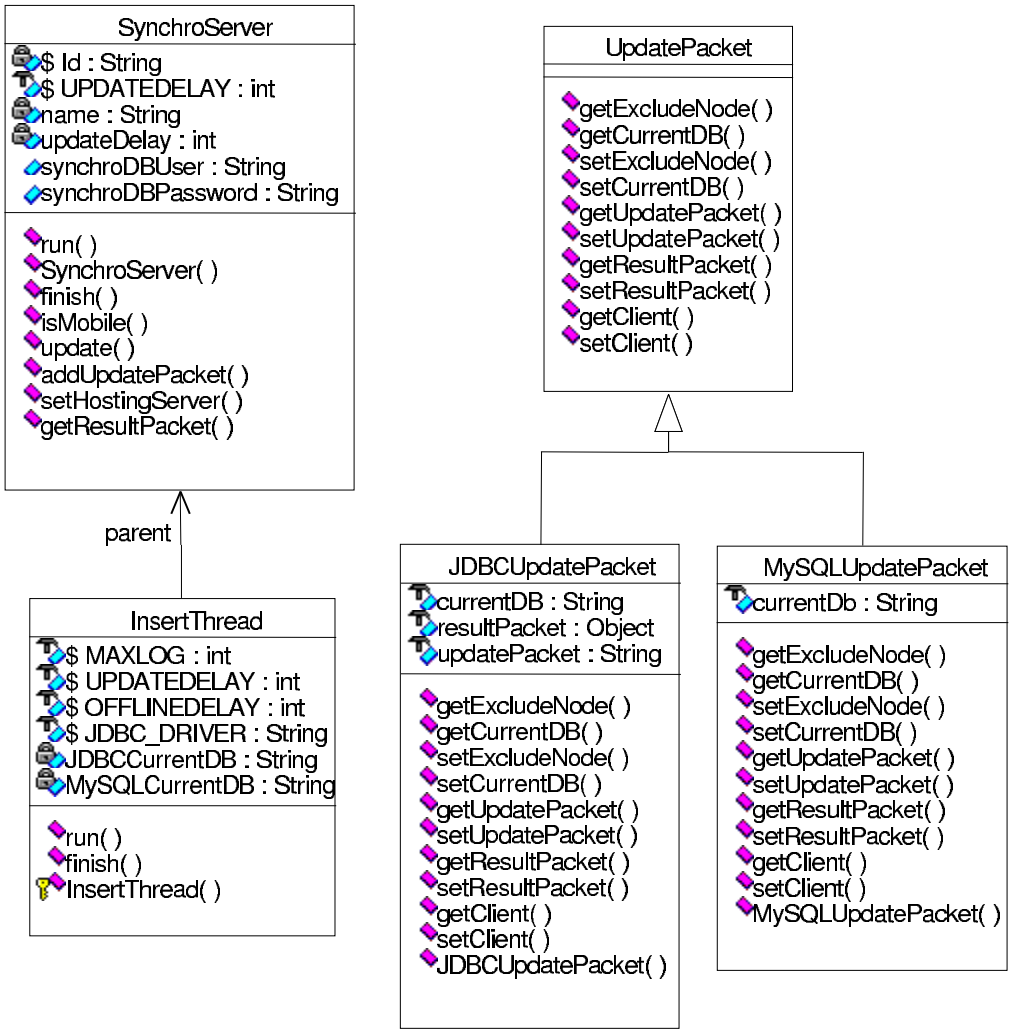
C.3 Allgemeine Hilfsklassen

Die unten dargestellten Hilfsklassen werden in dem Paket `cluster.util` zusammengefaßt und realisieren verschiedene Funktionen, die von unterschiedlichen Komponenten des MetaServers verwendet werden.



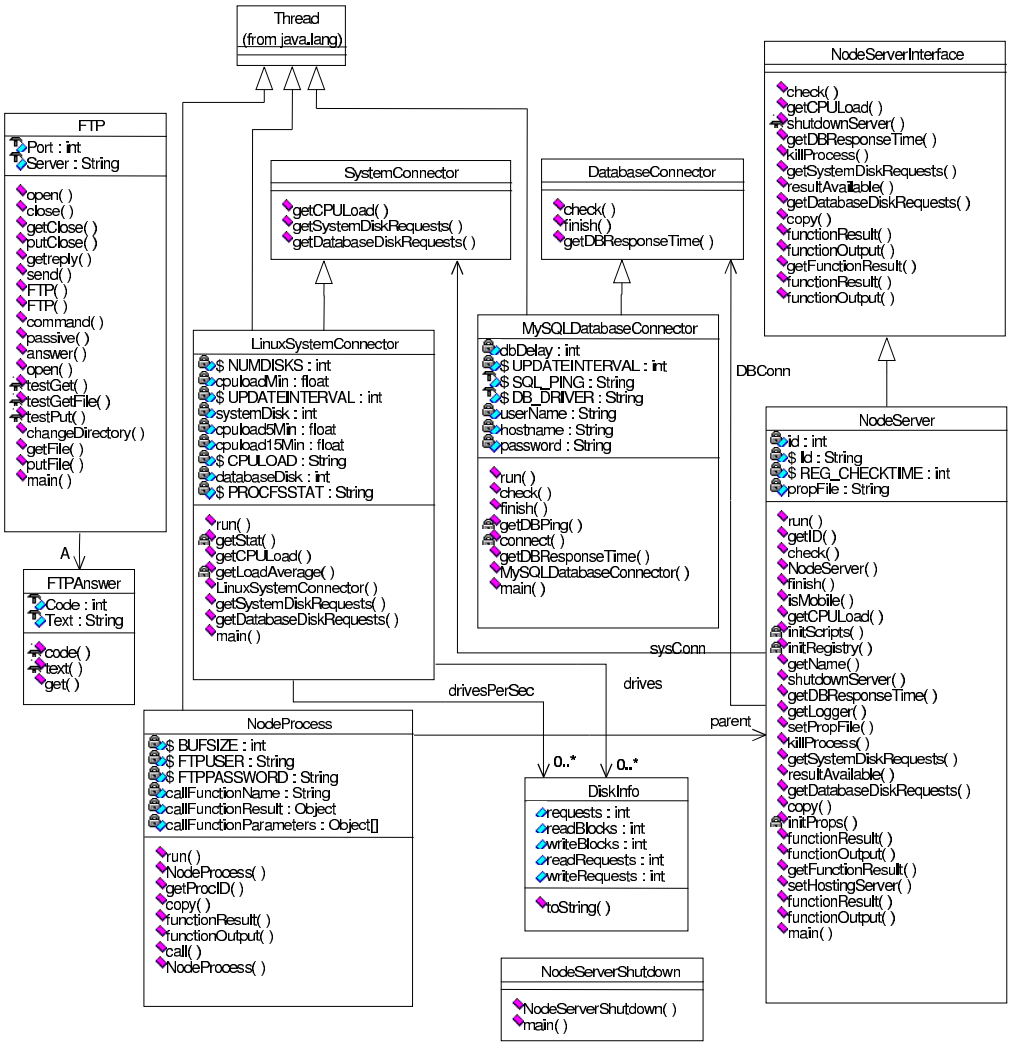
C.4 Der SynchroServer

Der SynchroServer realisiert mit den unten dargestellten Klassen die Synchonisation der Daten der Knoten-DBMS und ist als EmbeddedServer realisiert.



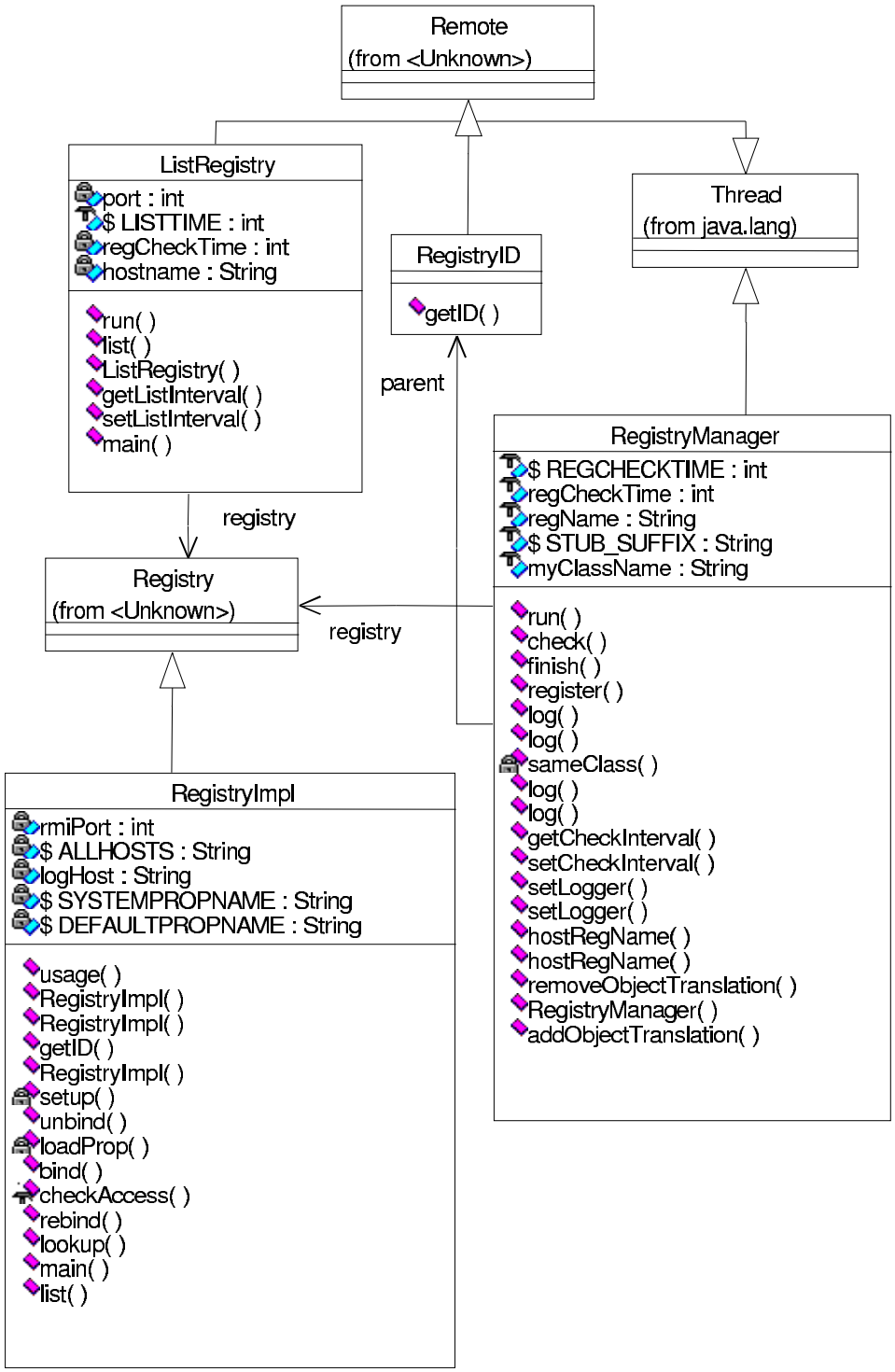
C.5 Der NodeServer

Der NodeServer ist ein interner Dienst der MetaServer-Plattform und realisiert als EmbeddedServer oder als eigenstängige Applikation den Zugriff auf die einzelnen Clusterknoten.



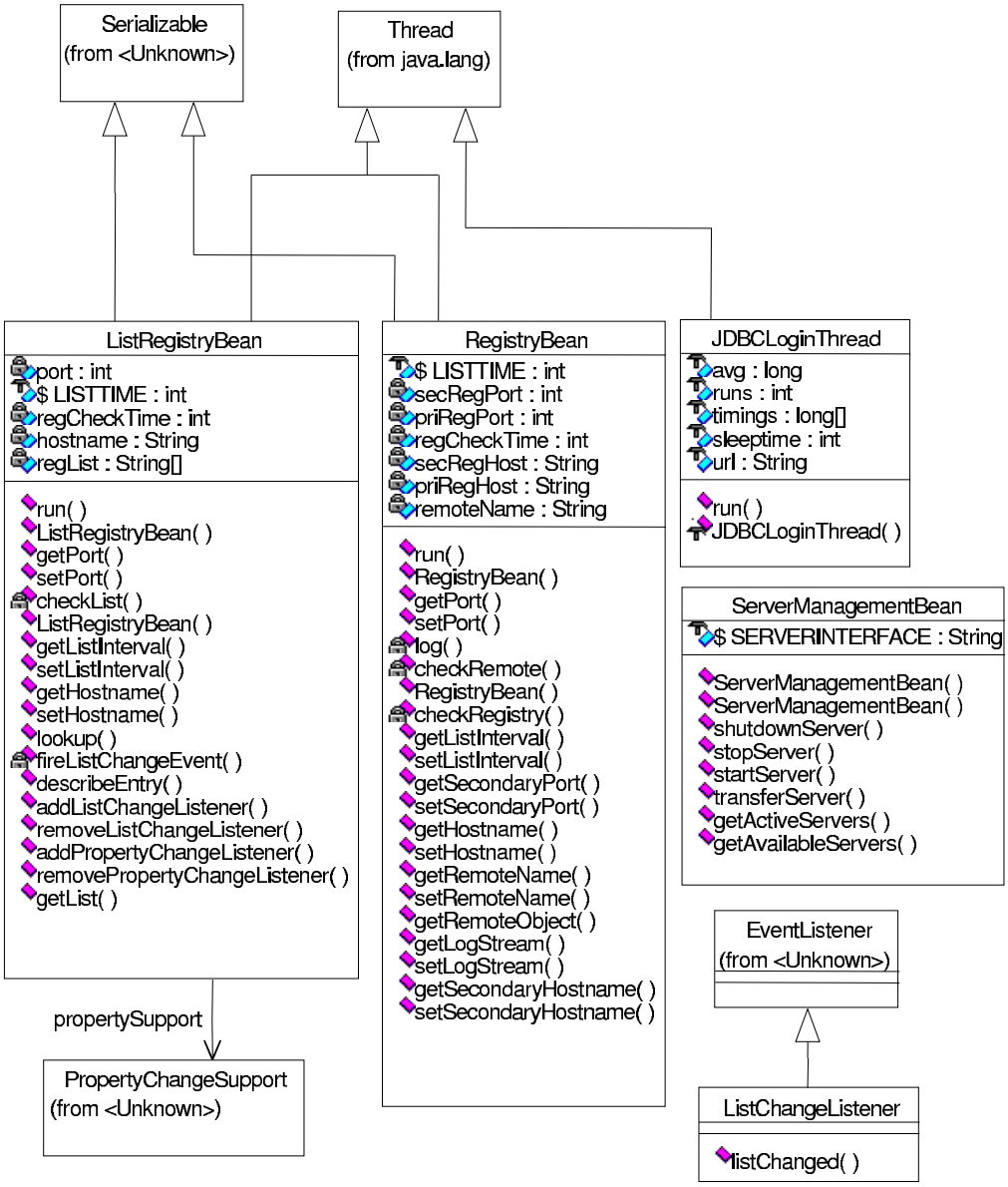
C.6 Die Registry

Die Registry realisiert den Verzeichnisdienst, der von der Plattform und den auf ihr laufenden Anwendungen benutzt wird und stellt eine Reimplementierung der von Sun bereitgestellten Version dar.



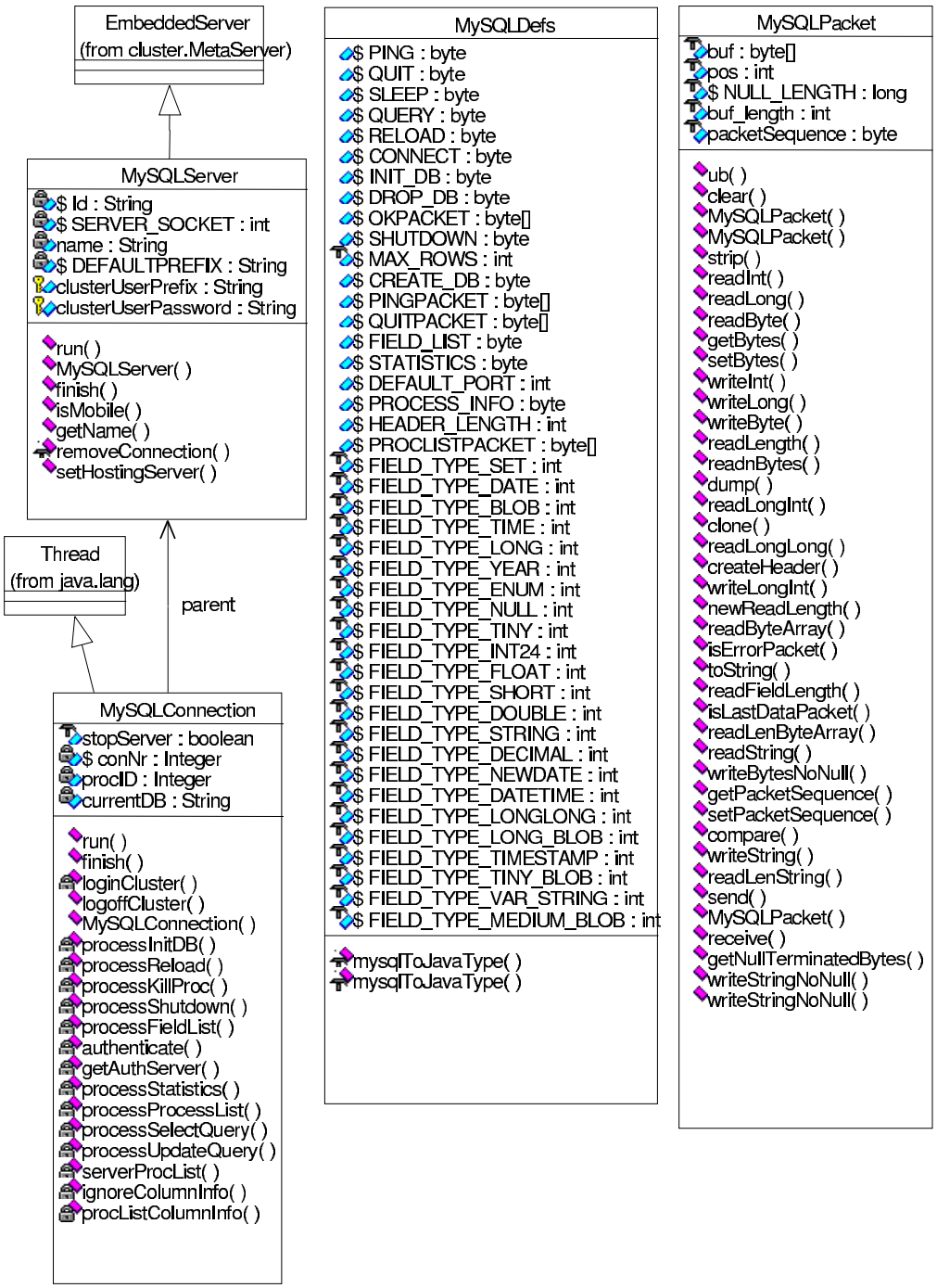
C.7 Frontend-Komponenten

Die in dem folgenden Klassendiagramm dargestellten Java-Beans realisieren das vorgestellte Frontend-Konzept und gewähren Zugriff auf die zu verwaltenden Server-Komponenten.



C.8 Der MySQL-Server

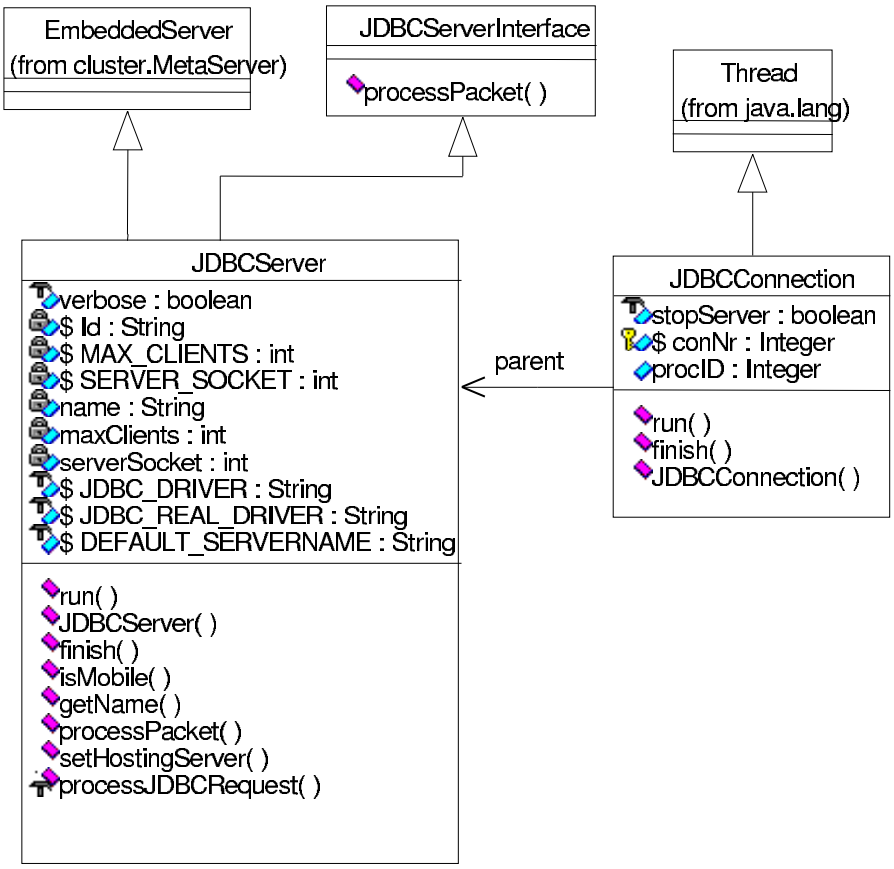
Der MySQL-Server realisiert den Zugriff auf die Knoten-DBMS unter Verwendung des proprietären Datenbankprotokolls und ist als EmbeddedServer realisiert.



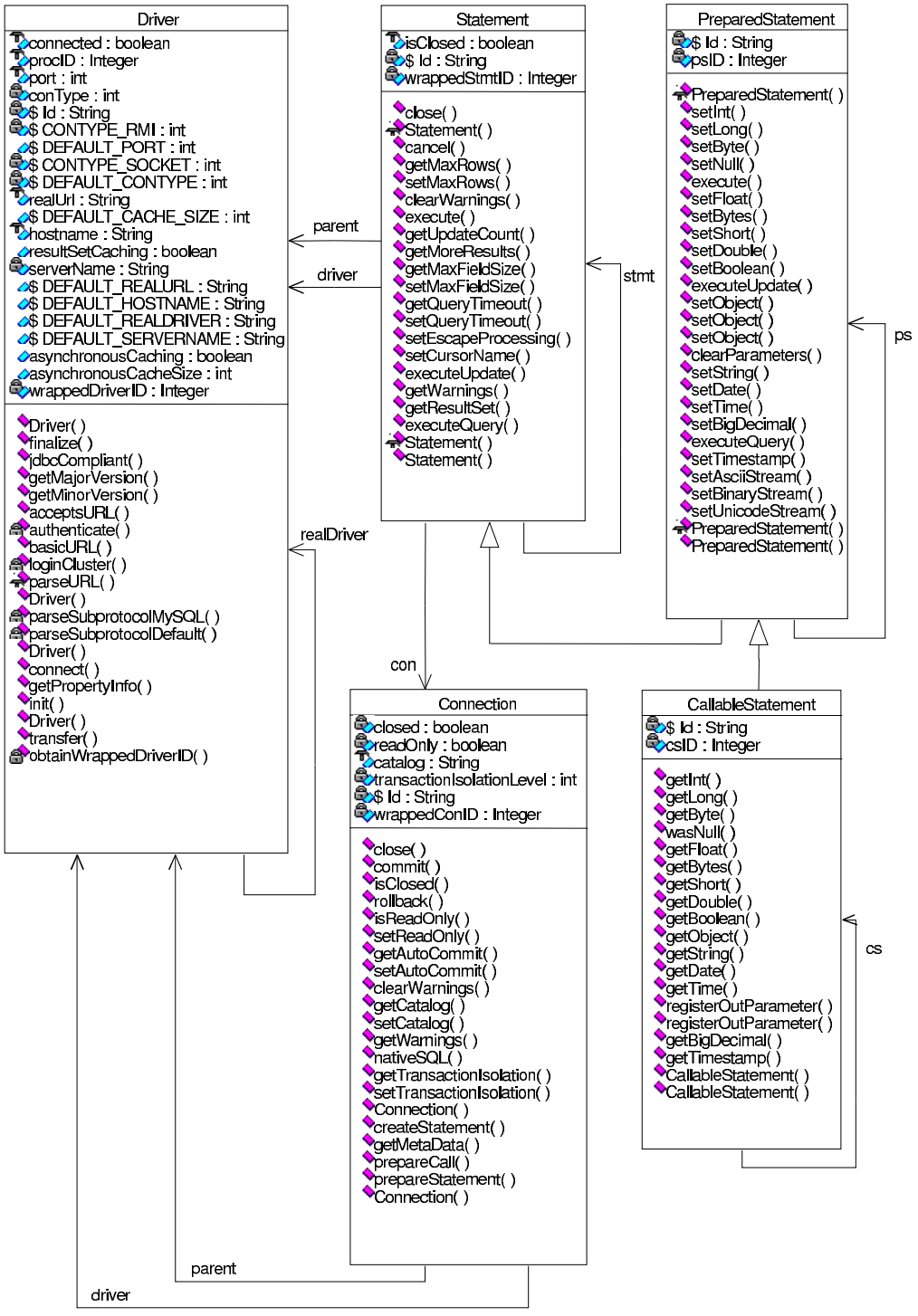
C.9 Der JDBC-Server

Der JDBC-Server benutzt sowohl für die Kommunikation mit dem Knoten-DBMS als auch für die Interaktion mit den Klienten die JDBC-API, indem entsprechende Treiber verwendet werden. Zugangsvoraussetzung für die Datenbank ist ein vorhandener JDBC-Treiber.

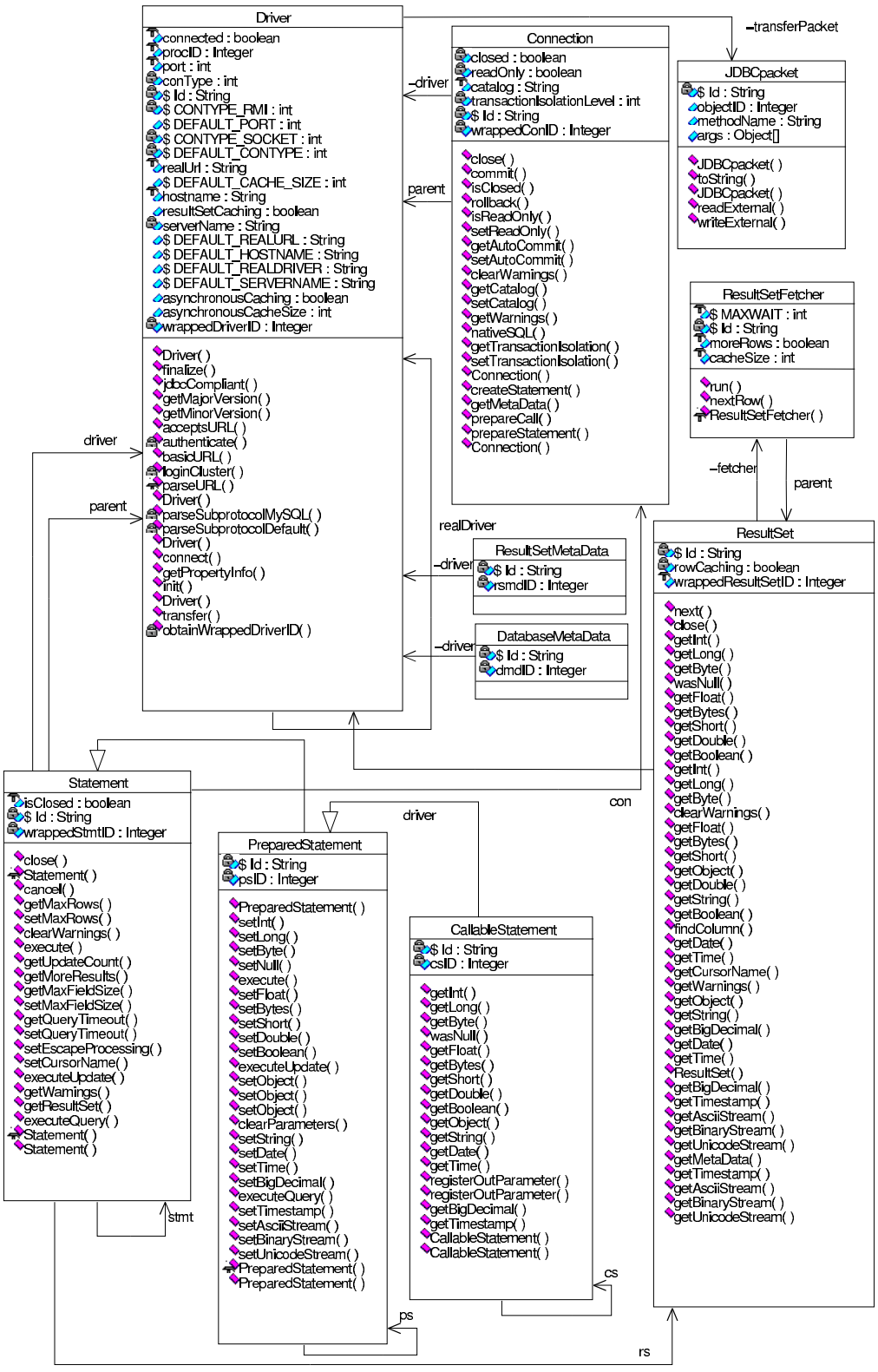
Die clusterspezifische Funktionalität ist in dem Paket *ClusterDriver* implementiert, für die Kommunikation zwischen Klientanwendung und MetaServer sind jeweils ein server- und ein klientseitiger Treiber (`cluster.jdbc.WrappedDriver` bzw. `cluster.jdbc.Driver`) zuständig.



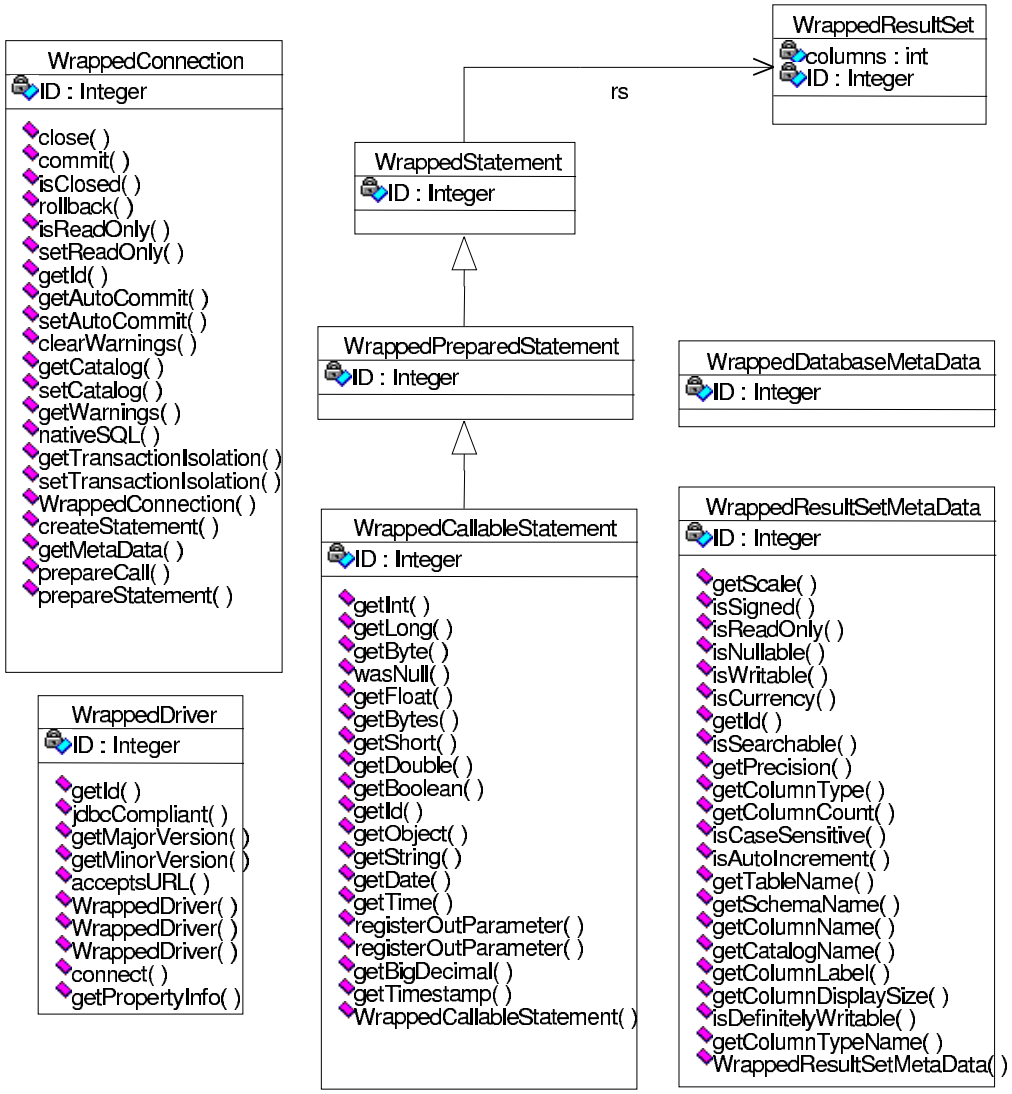
Der ClusterDriver



Der JDBC-Driver

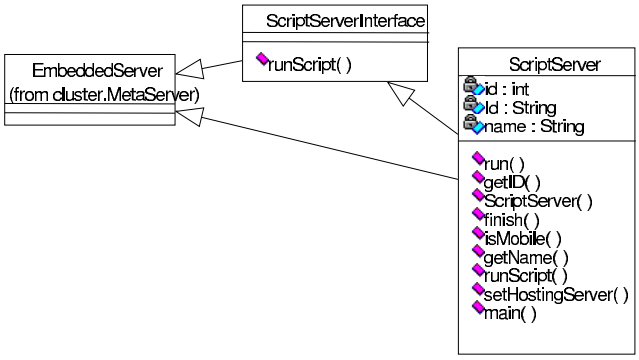


Der WrappedDriver

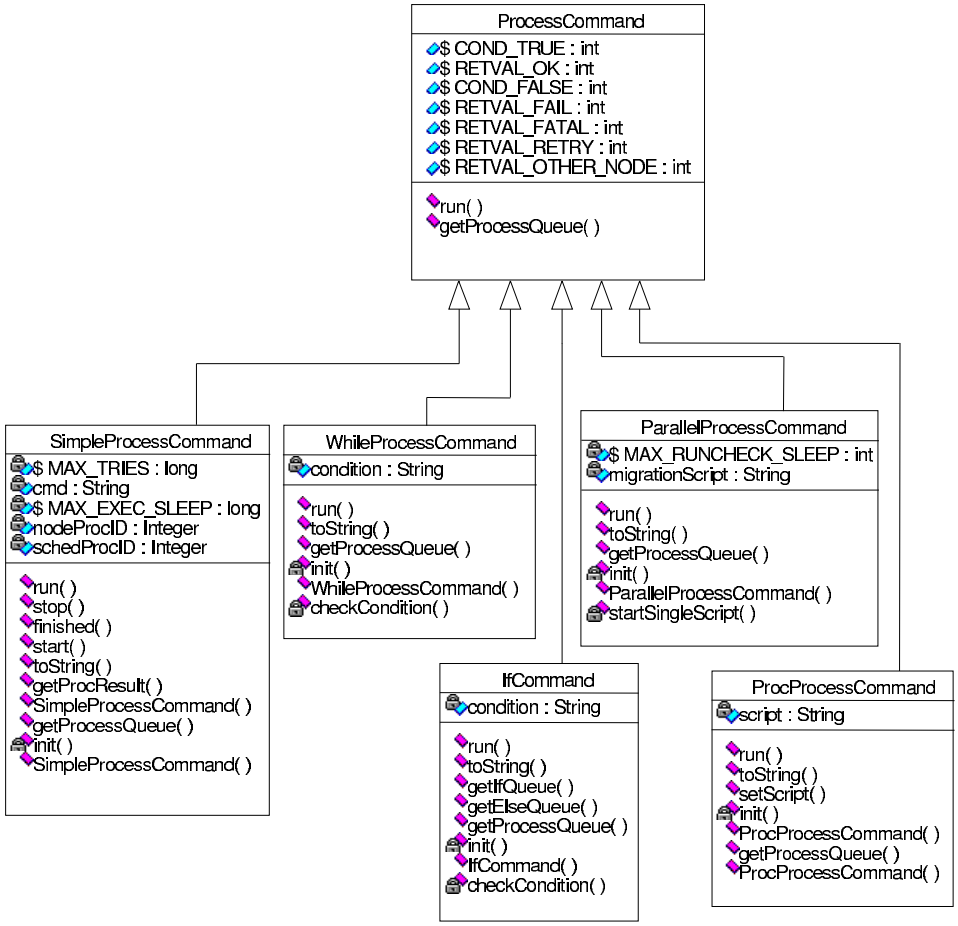


C.10 Der ScriptServer

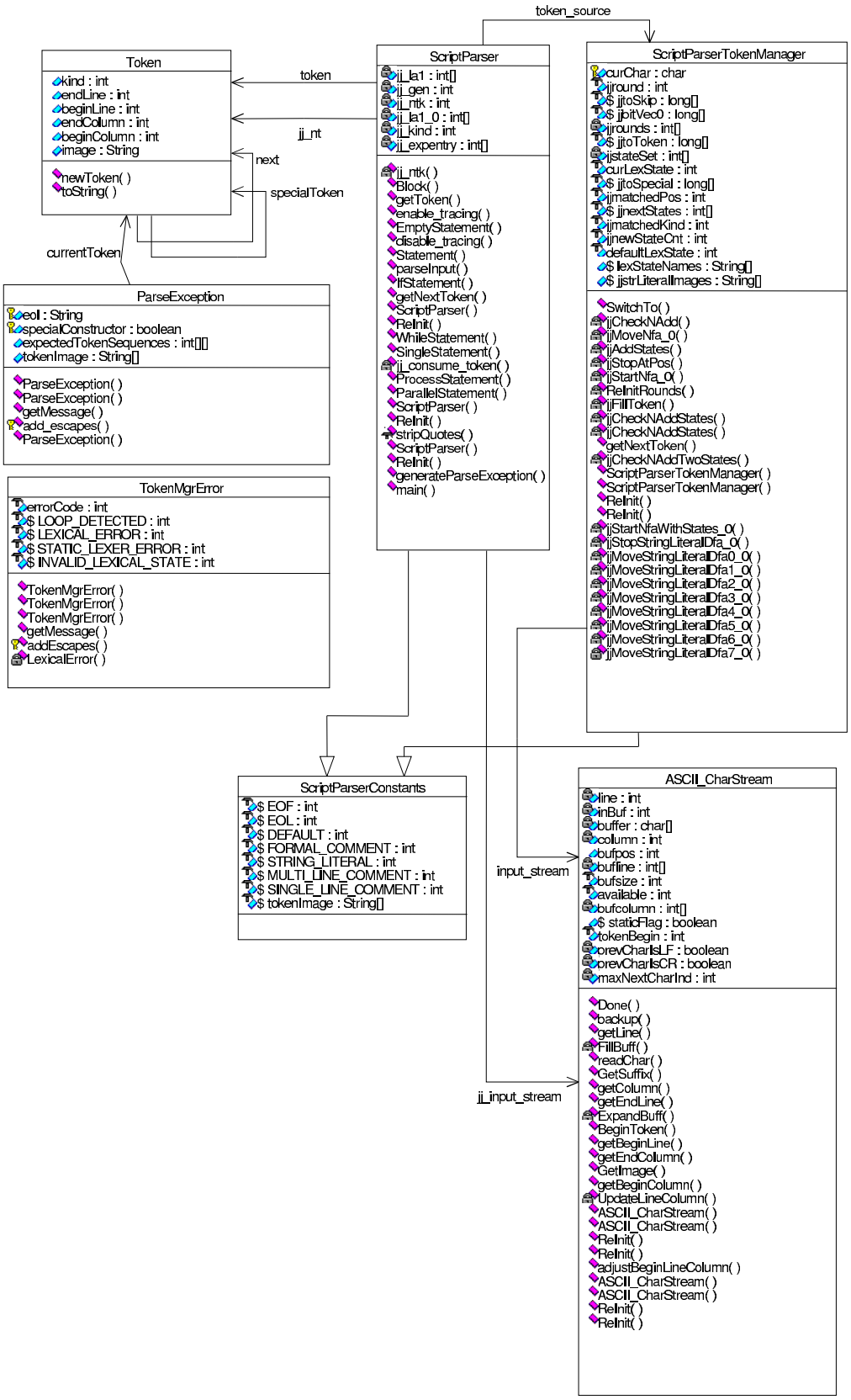
Der ScriptServer sorgt für die verteilte Ausführung nichtinteraktiver, lang-laufender Prozesse und implementiert zu diesem Zweck einen Parser, der automatisch aus einer für diese Zwecke entworfenen Grammatik erzeugt wurde und eine Reihe von Objekten, die die einzelnen Ausführungsschritte repräsentieren.



Klassen zur Prozeßbeschreibung



Der Skriptparser



Literaturverzeichnis

[Bal96] Helmut Balzert. *Lehrbuch der Software-Technik, Software-Entwicklung, Band I*. Spektrum Akademischer Verlag Heidelberg, 1996.

[Beh99] Henning Behme. Objektgeschäft: Java jenseits von Applets. *iX-Magazin für professionelle Informationstechnik*, September 1999.

[Ber99] Jürgen Diercks Bernhard Merkle. Java für Geschäftsanwendungen. *iX-Magazin für professionelle Informationstechnik*, Mai 1999.

[Böh97] Karsten Böhm. Java — Architektur und Konzepte einer neuen Programmiersprache. Technical report, Universtität Leipzig, Institut für Informatik, Abteilung ASV, 1997.

[Böh99] Karsten Böhm. The Management of a Distributed Linguistic Database using Cooperative Agents, 1999.

[Bla00] Blackdown. Java-Linux. http://www.blackdown.org, 2000.

[Chr00] Chris Barlock, IBM. The Logging Toolkit for Java. http://www.alphaworks.ibm.com/tech/loggingtoolkit4j, 2000.

[Con97] Stefan Conrad. *Förderative Datenbankssteme: Konzepte der Datenintegration*. Springer Verlag, Berlin Heidelberg, 1997.

[D. 96] S. Tripathi D. Saha, S. Mukherjee. Carry-Over Round Robin: A Simple Cell Scheduling Mechanism for ATM Networks, 1996.

[Dad96] Peter Dadam. *Verteilte Datenbanken und Client/Server-Systeme: Grundlagen, Konzepte und Realisierungsformen*. Springer Verlag, Berlin Heidelberg, 1996.

[Die99] Jürgen Diercks. Das Web: Plattform für Firmenanwendungen. *iX-Magazin für professionelle Informationstechnik*, Januar 1999.

[Dou98] Randi Reppen Douglas Biber, Susan Conrad. *Coprus Linguistics: Investigating Language Structure and Use*. Cambridge University Press, 1998.

[DuB99] Paul DuBois. *MySQL*. New Riders, 1999.

[DuB00] Paul DuBois. *MySQL – Entwicklung, Implementierung und Referenz*. Markt und Technik, München, 2000.

[Fis00] Frank Fischer. ISAAK: Intelligente Suchagenten für die Aquisiton neuer Korpora. Technical report, Universtität Leipzig, Institut für Informatik, Abteilung ASV, 2000.

[Gam94] Erich Gamma. *Design Patterns*. Addison Wesley Longman Publishing Co, 1994.

[Gei95] Kyle Geiger. *Inside ODBC*. Microsoft Press Deutschland, 1995.

[Geo94] Tim Kindberg George Coulouris, Jena Dollimore. *Distributed Systems Conecpts and Design*. Addison Wesley, 1994.

[Gib00] Pierre-Yves Gibello. RmiJdbc: A client/server JDBC Driver based on Java RMI. http://www.objectweb.org/RmiJdbc/RmiJdbcHomePage.htm, 2000.

[Gün94] Wolfgang Sternefeld Günther Grewendorf, Fritz Hamm. *Sprachliches Wissen: eine Einführung in modener Theorien der grammatischen Beschreibung*. Suhrkamp, 1994.

[Gra00] Grace Software. JavaLog. http://www.homestead.com/javalog/, 2000.

[Hem99] Upamanyu Madhow Hemant Chaskar. Fair scheduling with tunable latency: A round robin approach, 1999.

[IBM00] IBM Inc. IBM Developer Kit for Linux, Java 2 Technology Edition, Version 1.3. http://www.ibm.com/java/jdk/linux130/, 2000.

[Kel94] Kevin Kelly. *Out of control, The new biology of machines*. Fourh Estate, London, 1994.

[Lut00] Lutris Technologies. Getting started with Lutris Enhydra. http://www.lutris.com/documentation, 2000.

[Maa99] Jason Maassen. An Efficient Implementation of Java's Remote Method Invocation. Technical report, Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science, 1999.

[Mat99] Mark Matthews. Documentation for MM.MySQL. http://www.worldserver.com/mm.mysql, 1999.

[Mel99] Steve Meloan. The Java HotSpotTM Performance Engine: An In-Depth Look. Technical report, Sun Microsystems, Inc., 1999.

[Mer00a] Bernhard Merkle. Im Vergleich: Acht EBJ-Application Server. *iX-Magazin für professionelle Informationstechnik*, Dezember 2000.

[Mer00b] Bernhard Merkle. Vergleich: sieben EBJ-Application Server. *iX-Magazin für professionelle Informationstechnik*, April 2000.

[Met99] Metamata Inc. JavaCC — A Parser Generator for Java. http://www.metamata.com, 1999.

[Mik98] Mike Ricciuti, CNet News. Application server eludes definition. http://news.cnet.com/category/0-1003-200-332521.html, 1998.

[MyS] MySQL AB. MySQL Reference Manual. http://www.mysql.com/documentation.

[Pet00] Nikolai Bauer Peter Mandl. Chancen und Risiken der EJB-Komponententechnik. *iX-Magazin für professionelle Informationstechnik*, Januar 2000.

[Qua98a] Uwe Quasthoff. Deutscher Wortschatz im Internet. *LDV-Forum*, 2, 1998.

[Qua98b] Uwe Quasthoff. Projekt Deutscher Wortschatz. In Christian Wolff Gerhardt Heyer, editor, *Linguistik und neue Medien*. Deutscher Universitätsverlag, 1998.

[Qua98c] Uwe Quasthoff. Tools for automatic lexicon maintenance: Acquisition, error correction, and the generation of missing values. In *Proceedings of the first International Conference on Language Resources & Evaluation, ELRA*, pages 853–856, 1998.

[Rah94] Erhard Rahm. *Mehrrechnerdatenbanksysteme*. Addison Wesley Publishing Co, 1994.

[Ran99] Tim King Randy Jay Yarger, George Reese. *MySQL & mSQL*. O'Reilly, 1999.

[Red00] Redhat Inc. The GNU Compiler for the Java Programming Language. http://sources.redhat.com/java, 2000.

[Rey87] Craig Reynolds. Flocks, Herds and Schools: A distributed Behavioral Model. *Computer Graphics*, July 1987.

[Sch99] Fabian Schmidt. Automatische Ermittlung semantischer Zusammenhänge lexikalischer Einheiten und deren graphische Darstellung. Master's thesis, Univerität Leipzig, Institut für Informatik, 1999.

[Sof00] Softglobe. Socket-based client/server JDBC Driver. http://www.softglobe.com/socketjdbc/index.htm, 2000.

[Ste00] Bernd Steppan. Enterprise Java Beans: Welche Infrastruktur braucht man? *iX-Magazin für professionelle Informationstechnik*, Juni 2000.

[Sun97a] Sun Microsystems, Inc. JavaBeans Specification, Version 1.01. http://www.java.sun.com/products/javabeans/docs/spec.html, 1997.

[Sun97b] Sun Microsystems, Inc. JDBC 1.2. API Specification. http://www.java.sun.com/j2se/1.3/docs/guide/jdbc/spec/jdbc-spec.frame.html, 1997.

[Sun98] Sun Microsystems, Inc. Java Remote Method Invocation Specification, Revision 1.5. http://www.java.sun.com/products/rmi, 1998.

[Sun99a] Sun Microsystems, Inc. Enterprise Java Beans Specification, Version 1.1. http://java.sun.com/products/ejb/docs.html, 1999.

[Sun99b] Sun Microsystems, Inc. Java 2 Plattform Enterprise Edition Specification, Version 1.2. http://java.sun.com/j2ee/download.html, 1999.

[Sun99c] Sun Microsystems, Inc. Java Language to IDL Mapping Specification. http://www.omg.org/cgi-bin/doc/?formal/99-07-59, 1999.

[Sun99d] Sun Microsystems, Inc. Java Naming and Directory Interface Application Programming Interface (JNDI API) Version 2.1. http://www.java.sun.com/j2se1.3/jndi.ps, 1999.

[Sun99e] Sun Microsystems, Inc. Java Tranaction API (JTA). http://java.sun.com/products/jta, 1999.

[Sun99f] Sun Microsystems, Inc. JDBC 2.1 Core API Specification. http://www.java.sun.com/j2se/1.3/docs/guide/jdbc/spec2/jdbc2.1.frame.html, 1999.

[Sun99g] Sun Microsystems, Inc. JDBC 2.1 Standard Extensions API Specification. http://www.java.sun.com/j2se/1.3/docs/guide/jdbc/jdbc20.stdext.ps, 1999.

[Sun99h] Sun Microsystems, Inc. The JDBC Driver Test Suite. http://java.sun.com/products/jdbc/download2.html, 1999.

[Sun00a] Sun Microsystems, Inc. Enterprise Java Beans Specification, Version 2.0. http://java.sun.com/products/ejb/docs.html, 2000.

[Sun00b] Sun Microsystems, Inc. Java HotSpotTM Technology. http://java.sun.com/products/hotspot/, 2000.

[SW99] Maydene Fischer Seth White. *JDBC API Tutorial and Reference*. Addison Wesley Longman Publishing Co, 1999.

[Tan92] Andrew S. Tanenbaum. *Computer-Netzwerke*. Wolframs Fachverlag, 1992.

[Teg99] Frank Tegtmeyer. Open-Source-Web-Publishing mit Zope. *iX-Magazin für professionelle Informationstechnik*, August 1999.

[The95] The Open Group. SQL Call Level Interface (CLI). http://www.opengroup.org/public/pubs/catalog/c451.htm, 1995.

[The99] Erhard Rahm Theo Härder. *Datenbanksysteme, Konzepte und Techniken der Implementierung*. Springer Verlag, 1999.

[Tho00] Carmen Thomas. *Das Anagramm-Geheimnis. Vom Sinn und Hintersinn im Namen.* Droemer, München, 2000.

[Tim00] Tim Endres, ICE Engineering. Java Syslog Package. http://www.trustice.com/java/syslog, 2000.

[Tiv00] Tivoli Sys-
tems Inc. Tivoli Enterprise Concepts, Architecture and Services reference. http://www.tivoli.com/products/documents/whitepapers/enterprise36_wp.html, 2000.

[Tra00] Transmeta Inc. Kaffe – a complete, fully compliant Open Source Java environment. http://www.transvirtual.com/kaffe-architecture.htm, 2000.

[Uma97] A. Umar. *Application (Re)Engineering: Building Web-Based Applications and Dealing With Legacies*. Prentice Hall, 1997.

[Uwe99] Christian Wolff Uwe Quasthoff. Korpuslinguistik und große einsprachige wörterbücher. *Linguistik online*, 2, 1999.

[Ves99] Frederic Vester. *Die Kunst vernetzt zu denken, Ideen und Werkzeuge für den Umgang mit Komplexität*. Deutsche Verlags-Anstalt GmbH, Stuttgart, 1999.

[Wit00a] Thomas Wittig. Computerlinguitische Homonymie- und Polysemieanalyse, Möglichkeiten der Clusteranalyse. Technical report, Universtität Leipzig, Graduiertenkoleg 'Universalität und Diversität', 2000.

[Wit00b] Thomas Wittig. Ermittlung emantischer Relationen aus Korpora. Technical report, Universtität Leipzig, Graduiertenkoleg 'Universalität und Diversität', 2000.

Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig
22. Dezember 2000

(Karsten Böhm)

Copyright-Vermerk