Messung, Überwachung und Bewertung einer C++ - basierten Server Software

Diplomarbeit

vorgelegt von

cand. inform. Mehmet-Oktay Tugan

Arbeitsbereich Technische Informatik am

Wilhelm-Schickard-Institut der Universität Tübingen

in Zusammenarbeit mit

IBM Entwicklung GmbH

Betreuer:
Prof.Dr.-ing.Wilhelm G. Spruth
Dipl.Ing.(BA/FH) Marek Szermutzky (IBM Entwicklung GmbH)

Tübingen, 02.Februar 2009

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Tübingen, 02.Februar 2009 _____

Unterschrift

Kurzfassung

Ein wichtiger Gesichtspunkt in der Entwicklung von Serversoftware ist die Performance-Qualitätssicherung. Dabei sollte die Qualitätssicherung eine möglichst genaue Differenzierung zwischen systemorientierten Performancegrößen und benutzerorientierten Performancegrößen beinhalten. Jedoch entzieht sie sich einer allgemein gültigen Beschreibung, da die objektive Bewertung von Performance-Qualität als äußerst schwierig empfunden wird.

In der vorliegenden Arbeit wird anhand des OpenPegasus CIM Servers definiert, wie man Serverleistung objektiv erfassen und automatisiert überwachen kann. Zu diesem Zweck wird ein Verfahren vorgestellt, mit dem ausgewählte Messgrößen automatisiert ermittelt und zu einer reproduzierbaren, signifikanten Bewertungsgröße zum Durchsatz und Verbrauch von Systemressourcen konsolidiert werden. Somit kann man die Bewertungsgrößen zwischen verschiedenen Softwareständen vergleichen, so dass es möglich wird, Änderungen an der Codebasis hinsichtlich ihrer Auswirkung auf die Gesamtleistung zu klassifizieren. Die Anforderungen für die Auswahl der verwendeten Messgrößen belegen hierbei die Signifikanz der konsolidierten Bewertungsgröße.

Das Ergebnis der Implementierung des vorgestellten Verfahrens ist eine Kennzahl, die basierend auf dem Vergleich der normierten Bewertungsgrößen zwischen zwei Softwareständen ausgegeben wird und Aufschluss darüber gibt, inwiefern die Änderung der Gesamtleistung zu interpretieren ist.

Inhaltsverzeichnis

1 Einleitung

Die Performance-Qualitätssicherung entzieht sich im Vergleich zu anderen Qualitätssicherungsdisziplinen einer detaillierten und allgemein gültigen Prozess- und insbesondere Methodenbeschreibung, was unter anderem daran liegt, dass in ihr die Entwicklung und Handhabung von objektiven Qualitätskriterien als äußerst schwierig empfunden werden. Hinzu kommt, dass diese Kriterien notwendigerweise einen starken technischen Bezug zu den betreffenden IT-Systemen aufweisen müssen.
Es gibt immer noch viele Unternehmen, die die Performance-Qualitätssicherung als eine Ad-Hoc-Disziplin begreifen, die allenfalls dann zum Einsatz kommt, wenn akute Performance-Probleme auftreten. Angesichts der Komplexität und Heterogenität heutiger IT-Systeme, die ein entsprechend langes Assessment zur Erforschung von Performance-Engpässen erfordern, ist diese Einstellung als höchst kurzsichtig zu bezeichnen.

Eine diesbezüglich weitsichtige Haltung zeichnet sich aus durch das Bestreben, eine kontinuierlich betriebene Performance-Qualitätssicherung zu etablieren, die sich unter anderem mit folgender Fragestellung befasst:

Durch welche Kennzahlen ist die Performance des IT-Systems bestimmt?

1.1 Motivation

In der Entwicklung von Serversoftware ist die Performance-Qualitätssicherung eines der wichtigsten Grundbausteine. Hierfür werden am Beispiel des OpenPegasus CIM Servers im IBM Labor Böblingen regelmäßige, nächtliche Tests absolviert, die man Nightly Test and Builds nennt, um die Qualitätssicherung zu überwachen.

In diesem Zusammenhang wird eine automatisierte Performance-Überwachung benötigt, die Aufschluss darüber gibt, inwiefern sich Codeänderungen auf die Performance auswirken. Somit wird gewährleistet, dass in einem hochdynamischen OpenSource-Umfeld die Performance sehr regelmäßig überwacht wird und ein sehr wichtiger Schritt für die Qualitätssicherung getan wird. Das ist das Ziel der vorliegenden Arbeit.

1.2 Aufbau der Arbeit

In Kapitel 2 wird zunächst auf die technologischen Grundlagen eingegangen, die das Fundament für diese Arbeit darstellen.

Die bereits verfügbaren Methoden zur Messung der Performance werden in Kapitel 3 vorgestellt und eine Erklärung darüber gegeben, aus welchem Grund diese für die vorliegende Arbeit nicht von Nutzen sind.

In Kapitel 4 wird die eigentliche Motivation dieser Arbeit erklärt und eine Diskussion über die Problemstellung geführt sowie der Lösungsansatz erläutert.

Kapitel 5 stellt die Analyse dieser Arbeit dar, in der die Kenngrößen bezüglich der Serverleistung vorgestellt und nach zuvor definierten Kriterien ausgewählt werden. Diese Analyse stellt auch den ersten Schritt für das Verfahren zur automatisierten Performanceüberwachung dar.

Das Konzept und die Implementierung in der Programmiersprache C++ werden detailliert in Kapitel 6 vorgestellt und die Einordnung dieses Verfahrens in das vorhandene OpenPegasus-Konzept erläutert.

Abschließend wird in Kapitel 7 eine Zusammenfassung über die Arbeit gegeben und ein Ausblick auf mögliche Weiterentwicklungen geworfen.

2 Grundlagen

In diesem Kapitel wird eine Einführung über die technologischen Grundlagen gegeben und diesem Zusammenhang das Common Information Model und dessen Beispielimplementierung OpenPegasus vorgestellt.

2.1 Common Information Model

Da es in heterogenen Umgebungen von Haus aus keine Vereinbarungen über die Information gibt, die zu Managementzwecken ausgetauscht werden muss, spielt das Informationsmodell eine zentrale Rolle innerhalb einer Managementarchitektur. Es spezifiziert die Methoden zur Modellierung und Beschreibung von Managementobjekten und legt damit fest, was überhaupt verwaltet werden soll, welche Ressourcenmerkmale und Vorgänge relevant sind.

Das Common Information Model (CIM) bildet deshalb den wichtigsten Bestandteil der WBEM-Architektur. Es verfolgt bei der Strukturierung der Managementinformation durchweg einen objektorientierten Ansatz. Bei seiner Entwicklung wurde Wert darauf gelegt, dass sich vorhandene Informationsmodelle (wie z.B. SNMP-SMI) möglichst verlustfrei auf CIM abbilden lassen.

2.1.1 Anforderungen an CIM

Zum Zeitpunkt des Entwurfs von CIM existierten bereits einige Standards für das Management von Systemen und Anwendungen. Da CIM erst später entstand, besteht natürlich der Anspruch, ein Konzept zu bieten, das ausgehend von vorausgegangenen Erfahrungen Verbesserungen mit sich bringt. Somit lassen sich einige Anforderungen an CIM formulieren:

- Es soll mit CIM möglich sein, ein Management verteilter Informationssysteme zu betreiben. Dies impliziert, dass CIM in der Lage ist, durch ein Agent-to-Manager-Konzept in einem Netzwerk verteilte Systeme zu verwalten. Damit CIM auch in großen Umgebungen einsetzbar ist, sollte eine mehrstufige Architektur mit einer Manager-to-Manager-Anbindung vorgesehen sein.

- Mit CIM soll ein umfassendes Systemmanagement möglich sein. D.h., CIM soll sämtliche Aspekte des Betriebs eines Informationssystems abdecken, so dass der Einsatz anderer Standards nicht nötig ist, und somit eine Vereinfachung zustande kommt. Dies impliziert, dass das CIM zugrunde liegende Datenmodell sehr mächtig und umfassend ist. Außerdem muss ein Normierungsgremium dafür sorgen, dass das Modell alle Hersteller gleichermaßen zufrieden stellt und dass Aktualisierungen eingepflegt werden.

- Das Systems Management mit CIM soll unabhängig von der Implementierung des zu verwaltenden Systems sein. Das bedeutet beispielsweise im Falle des Managements zweiter verschiedener Datenbankserver, dass es ein einheitliches Datenmodell geben muss, so dass Implementierungsdetails für eine Managementanwendung abstrahiert werden. Oder anders ausgedrückt: Das Modell darf keine herstellerspezifischen Objekte aufweisen.

- Zusätzlich zur Implementierungsunabhängigkeit kommt für CIM noch eine Architekturunabhängigkeit. D.h., es darf für das Datenmodell keine Rolle spielen, welche Rechnerarchitektur oder welches Betriebssystem verwaltet wird. Zusätzlich müssen für die Kommunikation innerhalb von CIM offene Standards zum Einsatz kommen, um eine architekturunabhängige Implementierung zu ermöglichen.

- Die Integration von älteren Standards wie SNMP und DMI in CIM soll möglich sein.

2.1.2 Organisationen

CIM ist ein offener Standard, welches wie jeder Standard eine Organisation braucht, die ihn spezifiziert. Diese werden im Folgenden im Falle CIM vorgestellt.

Distributed Management Task Force (DMTF)

Die Distributed Management Task Force (DMTF) wurde im Jahre 1992 zunächst als Desktop Management Task Force gegründet. Dabei handelte es sich um ein Industriekonsortium, was u.a aus folgenden Firmen bestand:

- Dell
- HP
- IBM
- intel
- Microsoft
- Novell
- SCO
- SunSoft
- Symantec

Ziel dieses Konsortiums war die Bereitstellung eines gemeinsamen Standards für das Management verteilter Anwendungen. Aufbauend auf diesem Wunsch wurde CIM entworfen. Dadurch wurde die damalige DMI in DMTF umbenannt und setze seitdem das Hauptaugenmerk auf das Management verteilter heterogener Systeme.

Zwischenzeitlich verfügt die DMTF über ca. 30 volle Mitgliedsfirmen und ca. 100 Partnerfirmen, die im Rahmen des Standardisierungsprozesses eine beratende Funktion besitzen.

Im Frühjahr 1996 wurde das erste Release von CIM vorgestellt, seitdem zeichnet sich die DMTF für die weitere Pflege des Datenmodells sowie für die Klärung von Fragen der Softwareinteroperabilität verantwortlich.

WBEMsource

Die WBEMsource-Initiative ist eine Arbeitsgruppe der OpenGroup, die selbst ein Industriekonsortium der meisten großen Hersteller von IT-Systemen ist. Sie hat sich zum Ziel gesetzt, die Interoperabilität von IT-Systemen zu fördern.

Das Aufgabenfeld von WBEMsource umfasst:

- die Förderung von OpenSource-Implementierungen für CIM
- Interoperabilitätstests von CIM-Implementierungen
- Standardisierung von APIs im CIM-Umfeld

Die OpenGroup hat CIM zwischenzeitlich als eigenen Standard übernommen, darüber hinaus wurde er als publicly available specification (PAS) bei der ISO eingereicht.

2.1.3 CIM Datenmodell

Das CIM-Datenmodell - die von der DMTF verwaltet wird und auf der Website der Organisation eingesehen werden kann – ist ein sehr mächtiges Datenmodell und unterteilt CIM in drei Schichten:

- Das Core Model ist ein „Kern“ von einigen wenigen Klassen, Assoziationen und Eigenschaften, die eine Basis für Verfeinerungen und ein grundlegendes Vokabular für die Analyse und Beschreibung zu verwaltender Systeme bilden.

  So steht die allgemeinste Klasse CIM_ManagedElement für jegliche Art von (evtl. verschachtelten) Managementobjekten; davon abgeleitet ist die Klasse CIM_ManagedSystemElement, welche allgemein gültige Merkmale wie den Namen oder das Installationsdatum eines Elements besitzt. Wiederum hiervon abgeleitet sind die Klassen CIM_PhysicalElement und CIM_LogicalElement, welche physische Elemente (also Hardware) bzw. logische Elemente wie Prozesse, Dateien oder Aggregationen physischer Elemente beschreiben.

- Das Common Model besteht aus einer Menge von Klassen die konkreter gefasst sind als im Core Model und bereits als Basis für eine Reihe von Managementanwendungen dienen können. Sie sind aber immer noch unabhängig von bestimmten Technologien und Implementierungen. Abgedeckt werden hierbei Bereiche wie Systeme, Applikationen, Netze, Endgeräte und Benutzerverwaltung, wobei mit fortschreitender Entwicklung des Modells neue Anwendungsfelder hinzukommen können.

- Extension Schemas stellen technologiespezifische Erweiterungen des Common Model dar und sind auf spezielle Umgebungen (insbesondere Betriebssysteme) zugeschnitten. Als Beispiel sei Win32 Schema genannt, das bei Microsofts WBEM-Implementierung zum Einsatz kommt.

Core und Common Model gemeinsam nennt man CIM Standard Schema oder einfach CIM Schema (nicht zu verwechseln mit dem CIM Meta Schema im folgenden Abschnitt). Es wird von der DMTF ständig erweitert und in seinen verschiedenen Entwicklungsstufen veröffentlicht.

2.1.4 CIM Meta Schema

Das Meta Schema ist eine formale Definition des Modells. Es legt die im CIM verwendeten Begriffe sowie deren korrekten Gebrauch und die Semantik fest. Das Meta Schema selbst wird mit Hilfe der Unified Modeling Language (UML) definiert. Seine Struktur zeigt Abbildung 2.1.

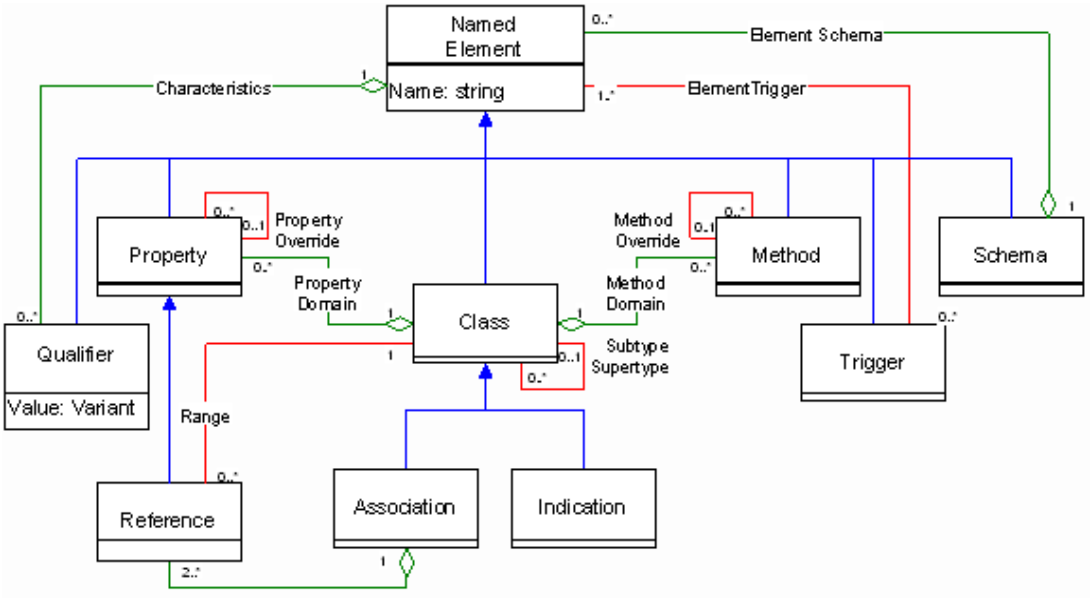

Abbildung 2.1: CIM Meta Schema

Die Elemente des Modells sind Schemata, Klassen (mit Ereignisklassen und Assoziationen als Untertypen), Eigenschaften (mit Referenzen als Untertyp) und Methoden.

Ein Schema

ist eine Gruppe von Klassen, die aus Verwaltungsgründen zusammengefasst wurden. Schemata definieren Namensräume; ein voll qualifizierter Klassenname hat die Form "`<Schema>_<Klasse>'". Innerhalb eines Schemas müssen die Klassennamen eindeutig sein.

Eine Klasse ("`Class"')

ist eine Menge von Objekten desselben Typs. Von einer Klasse abgeleitete Unterklassen erben deren Eigenschaften und Methoden, können diese jedoch überschreiben; Mehrfachvererbung ist nicht möglich. Ein konkretes Objekt als Ausprägung einer Klasse nennt man Instanz.

Eine Eigenschaft ("`Property"')

ist ein Wert, mit dem Instanzen einer Klasse charakterisiert werden. Man kann sie sich als ein Paar von get - und set -Funktionen vorstellen die, angewandt auf ein Objekt, den Zustand zurückliefern bzw. verändern.

Eine Methode ("`Method"')

definiert die Signatur (Methodenname, Rückgabetyp und Parameter) einer Operation, die auf Instanzen der Klasse anwendbar ist. Methoden repräsentieren das "`Verhalten"' von Klassen.

Ein Trigger

liegt vor, wenn eine Zustandsänderung einer Instanz erkannt wird. Dabei kann es sich um Zugriffe auf Eigenschaften handeln sowie um Erzeugung, Löschung oder Update der Instanz selbst.

Eine Ereignisklasse ("`Indication"')

repräsentiert einen Typ von Ereignismeldungen. Eine Instanz davon stellt eine konkrete Ereignismeldung dieses Typs dar. Sie wird als Reaktion auf einen Trigger erzeugt.

Eine Assoziation ("`Association"')

ist eine Klasse, die mindestens zwei Referenzen enthält. Sie repräsentiert eine Beziehung zwischen Objekten. Die referenzierten Klassen werden dabei nicht beeinflusst; d.h. insbesondere, dass ihre Schnittstellen nicht modifiziert werden müssen, wenn sich an der Assoziation etwas ändert.

Eine Referenz ("`Reference"')

ist eine spezielle Eigenschaft, die einen Verweis auf eine Klasse darstellt. Nur Assoziationen können über Referenzen verfügen.

Ein Qualifikator ("`Qualifier"')

ist ein Mittel, mit dem man eine Klasse, Eigenschaft oder Methode näher charakterisieren kann. Beispielsweise lässt sich eine Eigenschaft mit den Qualifikatoren Read und Write als les- und schreibbar auszeichnen. Neben einer Reihe von Standard-Qualifikatoren sind auch benutzerdefinierte Qualifikatoren erlaubt. Auf diese Weise lässt sich das Meta Schema in beschränktem Umfang erweitern.

2.1.5 Managed Object Format (MOF)

Klassendiagramme sind ein probates Mittel, um Managementinformation für Menschen übersichtlich darzustellen. Das Managed Object Format (MOF) dient dagegen primär dem Zweck, die Informationen maschinenlesbar zu formatieren. Es handelt sich dabei um eine Sprache zur textuellen Beschreibung CIM-konformer Information. Die Hauptkomponenten einer MOF-Spezifikation sind Klartextbeschreibungen von Klassen, Assoziationen, Eigenschaften, Referenzen, Methoden, Instanzdeklarationen sowie die zugehörigen Qualifikatoren. Eine MOF-Textdatei kann mittels eines Compilers verarbeitet werden. Auf diese Weise lassen sich statische Daten wie das CIM Schema in ein Repository einlesen.

Die vollständige MOF-Syntax ist in der CIM-Spezifikation beschrieben.

2.1.6 CIM Core Schema

Der Grundstein für das Management von Informationssystemen mit Hilfe von CIM wird durch das Common Schema gelegt. Es definiert einige grundlegende Klassen, die soweit generalisiert sind, dass sie für alle Aspekte des Systems Management verwendet werden können. Andererseits kann man an den Klassen bereits konkrete Verwendungsmöglichkeiten erkennen. Die Aktualisierung des Core Schemas erfolgt jeweils gemeinsam mit dem Common Schema.

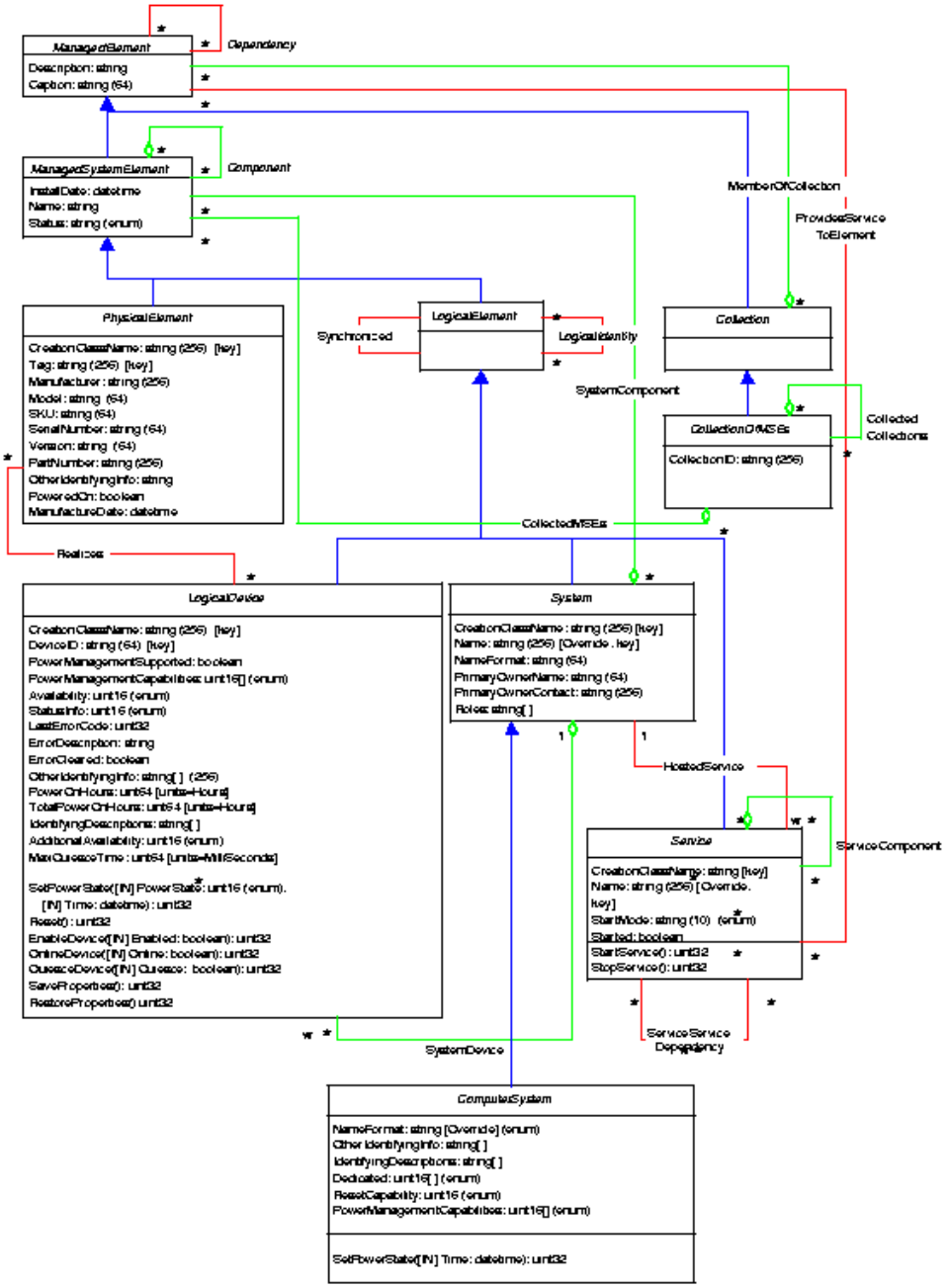

Abbildung 2.2: Auszug aus dem CIM Core Schema

Basis des Core Schemas und somit der kompletten CIM-Klassenhierarchie ist die Klasse ManagedElement, von der alle CIM-Klassen (außer Assoziationen) direkt oder indirekt abgeleitet werden. Sie enthält lediglich die Attribute Caption für eine Kurzbeschreibung und Description für eine detailliertere Beschreibung.

Eine generische Abhängigkeit zwischen Managed Elements und somit zwischen beliebigen Klassen kann durch die Assoziation Dependency dargestellt werden.

Von ManagedElement abgeleitet ist die Klasse ManagedSystemElement. Sie bildet die Oberklasse für alle logischen und physikalischen Komponenten eines Computersystems und kennt daher als zusätzliche Attribute InstallDate (Datum der Inbetriebnahme), Name und Status. Bei Status handelt es sich um eine Aufzählung verschiedener vordefinierter Zustände wie OK, Stressed (Komponente ist überlastet) oder NonRecover (Komponente hat einen nicht behebbaren Fehler).

Zur Abbildung einer Ist-Teil-von-Beziehung von ManagedSystemElements wurde die generische Aggregation Component definiert. Über sie ist beispielsweise die Aufgliederung eines Computersystems in seine Hardwarekomponenten und weiter in Subkomponenten möglich oder eine Zergliederung einer Softwareinstallation in Pakete mit einzelnen Dateien.

Ein PhysicalElement ist jede beliebige Hardwarekomponente. Sie verfügt über Attribute wie SerialNumber (Seriennummer) oder den boolschen Wert PoweredOn (eingeschaltet).

Bei den meisten Komponenten eines IT-Systems handelt es sich jedoch um LogicalElements -- dies ist die von ManagedSystemElement abgeleitete Klasse, die alle abstrakten Objekte wie Dateien, Benutzeraccounts oder aber auch Service Level Agreements oder Konfigurationseinstellungen umfasst.

Allen LogicalElements sind zwei mögliche Assoziationen gemeinsam: Syncronized (synchronisiert) und LogicalIdentity (logische Gleichheit). Die Erstere dient dazu, zu beschrieben, dass beliebige Objekte miteinander synchronisiert sind (z.B. Filesysteme oder Timer). Die Zweitere legt fest, dass zwei Objekte identisch sind. Da die CIM-Spezifikation keine Mehrfachvererbung zulässt, können auf diesem Weg zwei verschiedene Klassen instanziiert und über LogicalIdentity verknüpft werden.

Von LogicalElement abgeleitet sind LogicalDevice, Service und System. Dabei übernimmt ein LogicalDevice die logische Sicht auf eine Hardwarekomponente (Assoziation Realizes). System stellt eine beliebige Art von System dar, das selbst wieder aus einzelnen

Komponenten besteht (Aggregation SystemComponent). Dabei ist SystemDevice eine spezielle SystemComponent, die Geräte mit Systemen verknüpft.

Klassen, die etwas repräsentieren, was eine Dienstleistung zur Verfügung stellt, sind unter Service zusammengefasst. Dienste können voneinander abhängen (ServiceServiceDependency) und Unterdienste beinhalten (Aggregation ServiceComponent). Service stellt auch Methoden zum Starten (StartService()) und Anhalten zur Verfügung (StopService()). Auf welchem System ein Service abläuft, wird durch HostedService gezeigt. Dabei handelt es sich um eine Weak Association, das System erscheint also im Dienst als Fremdschlüssel.

Eine spezielle und die wichtigste Art von System ist ein ComputerSystem, unabhängig davon, wie es implementiert ist (z.B. Einzelsystem oder Cluster).

Zusammenstellungen von Instanzen können in eine Collection aufgenommen werden. Dabei wird über die Aggregation MemberOfCollection auf die Mitgliederobjekte verwiesen. Speziell für ManagedSystemElements wurde dafür die Klasse CollectionOfMSEs mit CollectedMSEs geschaffen. Damit soll es beispielsweise möglich sein, Konfigurationen zu gruppieren, um diese global bearbeiten zu können. Eine CollectionID dient dabei der Identifikation.

Allen bisher vorgestellten Klassen ist gemeinsam, dass sie abstrakt sind. Das bedeutet, dass eine Instanziierung erst im Rahmen einer Unterklasse möglich ist. Konkret äußert sich dies bei den bisher vorgestellten Klassen dadurch, dass Schlüsselattribute nicht oder unvollständig definiert wurden. Eine Methode zur Erzeugung eindeutiger Schlüssel ist jedoch bereits erkennbar: Ein Attribut CreationClassName zeigt an, welche Unterklasse konkret instanziiert wurde. Dies vermeidet einerseits Kollisionen bei gleichen Attributen in verschiedenen Unterklassen, ermöglicht andererseits eine einheitliche Identifikation von Instanzen auf einem hohen Abstraktionsniveau. Für die Klasse ComputerSystem, für deren Unterklassen eine einheitliche Identifikation gefordert wird, wurde zusätzlich das Attribut NameFormat eingeführt, über das festgestellt werden, auf welcher Basis die Benennung des Systems erfolgt, wie z.B. über einen DNS-Name oder eine X.25-Adresse.

Aspekte des Core Schema

Insgesamt behandelt das Core Schema damit folgende Aspekte, die für das Systemmanagement relevant sind:

- Repräsentation managebarer Komponenten
- Trennung zwischen Hard- und Softwarekomponenten
- Erste Ansätze für eine Vertiefung in Anwendungsgebiete (Produkte, Statistiken, Konfigurationen)

2.1.7 CIM Common Schema

Während die 26 Klassen des Core Schema generelle Klassen für das Systemmanagement umfassen, repräsentiert das Common Schema einen vollständigen Rahmen für das Management von Systemen.

Von den allgemeinen Anforderungen an CIM (siehe Kapitel 2.1.1) lassen sich damit folgende ableiten:

- Das Common Schema muss alle Bereiche des Systems Management abdecken, also Fehler-, Performance-, Accounting-, Konfigurations- und Sicherheitsmanagement.
- Es muss *generisch* sein, muss also die Modellierung diverser Architekturen ermöglichen oder diese weitestgehend abstrahieren
- Das Schema muss andere Modelle wie DMI, SNMP oder LDAP integrieren.

Im Gegensatz zu einem OO-Modell für ein Softwareprojekt ist das Common Model weder an eine unveränderliche Welt gebunden (da sich die Computertechnik in einem ständigen Wandel befindet), noch gilt das Modell für ein bereits vorher definiertes Computerprogramm (da es für alle Managementanwendungen einsetzbar sein soll).

Aus diesem Grund erfolgt eine regelmäßige Aktualisierung des Common Schema durch die DMTF. Notwendige Anpassungen am Core Schema, die möglichst vermieden werden, erscheinen ebenfalls in diesem Rahmen.

Da das gesamte Common Schema zu groß ist, befasst sich jeweils eine Arbeitsgruppe der DMTF mit der Fortschreibung eines Bereichs. Resultat ist jeweils eine MOF-Datei, in der

dieses Teilschema festgelegt ist. An dieser Stelle werden die aktuellen 12 Klassen kurz erläutert:

Physical

Das Physical Schema definiert Klassen für die Beschreibung von Hardware. Beschrieben werden Gehäuse, Schränke, Stecker, Kabel, Karten, Chips, Medien und Aufstellungsorte.

System

Das System Schema behandelt alle Klassen, die von System abgeleitet werden, also insbesondere Computersysteme, und zwar aus logischer Sicht. Dazu gehören *zusätzlich Prozesse, Dienste, Zeitzonen, Hard- und Softwareressourcen, Protokolle, Diagnoseroutinen und Spezialisierungen für UNIX-Betriebssysteme.*

Event

Das Event Schema definiert die Behandlung von Indications und weiteren Events, die bei der Modifikation von Klassen ausgelöst werden können.

Interoperability

Das Interoperability Schema dient der Beschreibung von CIM-Implementierungen selbst. Dies ermöglicht das Management von Software, die für eine Implementierung clientseitig eingesetzt wird und die Beschreibung von CIM-Infrastrukturen. Bereitgestellt wird beispielsweise die Klasse Namespace und weitere, die den Komponenten eines WBEM-Systems (siehe Kapitel 1.3) entsprechen. Auf das Interoperability Schema wird in den folgenden Kapiteln nicht weiter eingegangen.

User-Security

Das User-Security Schema definiert alle Klassen, die in Bezug zum Security- und User-Management stehen. Es werden Klassen für Organisationen, Personen, Gruppen, Rollen, Autorisierungsdienste, Kerberos, Public Key-Verfahren, Accounts und Zutrittskontrollsysteme eingeführt.

Policy

Mittels des Policy Schemas können Regeln, bestehend aus Bedingungen und Aktionen, definiert werden. Dies ermöglicht die Repräsentation von Service Level Agreements.

Application

Die Klassen des Application Schema erlauben es insbesondere, Softwareprodukte in Teilsysteme aufzugliedern, Pakete zu definieren und Software verteilt zu installieren. Auch können Abhängigkeiten von Softwarepaketen untereinander und zu Betriebssystemen modelliert werden. Das Application Schema entspricht damit dem *Konfigurationsmanagement* -- die *System- und Anwendungskonfiguration* wird hier nicht abgedeckt.

Metrics

Das Metrics Schema definiert Klassen zur Repräsentation von Einheiten, insbesondere von Arbeitseinheiten für die Abrechnung von Geleistetem (sowohl von Personen als auch von EDV), was für das Performance Management von Bedeutung ist. Auf das Metrics Schema wird in den folgenden Kapiteln nicht weiter eingegangen.

Network

Das Network Schema beinhaltet Klassen für die Repräsentation von Netzwerkgeräten und -strukturen. Es bietet Klassen wie RangeOfIPAddresses (IP-Adressbereich), FilterList (Netzwerkfilterung), Routingbeschreibungen, SNMP-Integration und die Darstellung von Routingprotokollen.

Database

Das Database Schema dient der Modellierung von Datenbanken. Dargestellt werden können Datenbanksoftware, -instanzen und -dienste. Auf das Database Schema wird in den folgenden Kapiteln nicht weiter eingegangen.

Device

Mittels des Device Schema wird die Klasse LogicalDevice zur Darstellung von Hardwarekonfigurationen verfeinert. Dazu gehören Klassen für Spannungsversorgung, Prozessoren, Controller, Bussysteme, Netzwerkkarten, FibreChannel, Medien, PC-Peripheriegeräte und Drucker. Auch Konzepte wie Redundanz und Load Balancing wurden hier berücksichtigt.

Support

Dieser Bereich bildet den Solution Exchange Standard (SES) ab. Dabei handelt sich um ein Modell für die Repräsentation von Ticketsystemen und Knowledgebases.

Eine detaillierte Vorstellung aller 12 Common Schemata würde sicherlich den Rahmen dieser Arbeit sprengen. Tiefgreifendere Informationen über die Schemata für Interessierte befinden sich auf der Homepage der DMTF.

2.1.8 CIM Extension Schema

Ein CIM Extension Schema (Erweiterungsschema) ist ein Schema, das ein spezifisches Hard- oder Softwareprodukt repräsentiert und im Idealfall mit diesem ausgeliefert wird. Da ein Extension Schema nicht mehr generisch ist, wird es auch nicht durch die DMTF normiert. Stattdessen werden in einem Extension Schema Klassen des Common Schema durch Vererbung für ein konkretes Produkt angepasst.

Durch den objektorientierten Ansatz von CIM ist es auf diese Weise jederzeit möglich, auch proprietäre Systeme einheitlich zu verwalten, solange das Common Schema als Modellierungsgrundlage verwendet wird.

Klassen, die in einer CIM-Umgebung instanziiert werden, sind immer Klassen des Erweiterungsschemas, da nur sie Objekten der realen Welt entsprechen.

2.1.9 Interoperabilität

In den ursprünglichen WBEM-Spezifikationen war keine Transportschicht vorgesehen. Aus diesem Grund ist derzeitige WBEM-Software trotz Standardkonformität nicht zwangsläufig interoperabel. Erst 1999 wurden Mappings für die Darstellung von CIM-Informationen im XML-Format und der Transfer über das *Hypertext Transfer Protocol (HTTP)* definiert.

2.1.10 CIM-Darstellung in XML

XML (Extensible Markup Language) ist eine universelle Auszeichnungssprache und bildet als solche ein mächtiges Werkzeug zur Datenmodellierung. Als plattformunabhängiger und vergleichsweise einfach zu implementierender Standard, insbesondere aber im Hinblick auf die Verwendung von HTTP als Transportprotokoll, bietet sich das XML-Format für die Einhüllung von CIM-Daten an.

2.1.11 CIM-Operationen via HTTP

Die in spezifizierte Abbildung ordnet WBEM-Operationen den Requests von HTTP zu und schafft so Interoperabilität zwischen den Transportprotokollen. Per POST oder M-POST werden XML-Seiten verschickt, die Methodenaufrufe und Datenaustausch beinhalten. Neben den Vorteilen eines gut verstandenen Protokolls, für das performante Implementierungen verfügbar sind, bringt HTTP quasi automatisch Mechanismen zur verschlüsselten Übertragung mit sich. Bei erhöhten Anforderungen an die Sicherheit wird einfach auf eine SSL-Verbindung zurückgegriffen.

Das folgende Beispiel zeigt, wie so ein Request aussehen kann. Mit einer Anfrage dieser Art würde ein Client die Klassendefinition von CIM_PhysicalElement beim CIMOM anfordern:

```
M-POST /cimom HTTP/1.1
HOST: http://www.myhost.com/
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx
Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73
73-CIMOperation: MethodCall
73-CIMMethod: GetClass
73-CIMObject: root/cimv2
<?xml version="1.0" encoding="utf-8" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
 <MESSAGE ID="4711" PROTOCOLVERSION="1.0">
 <SIMPLEREQ>
  <IMETHODCALL NAME="GetClass">
   <LOCALNAMESPACEPATH>
    <NAMESPACE NAME="root"/>
    <NAMESPACE NAME="cimv2"/>
   </LOCALNAMESPACEPATH>
   <IPARAMVALUE NAME="ClassName"><CLASSNAME NAME="CIM_PhysicalElement"/>
     </IPARAMVALUE>
   <IPARAMVALUE NAME="LocalOnly"><VALUE>FALSE</VALUE></IPARAMVALUE>
  </IMETHODCALL>
 </SIMPLEREQ>
 </MESSAGE>
</CIM>
```

Die Antwort des CIMOM enthielte im Erfolgsfall unter anderem die vollständige Klassendefinition im XML-Format.

2.1.12 Web Based Enterprise Management (WBEM)

Nachdem die oberste Ebene von CIM in Form seiner Spezifikation und seines Datenmodells vorgestellt wurde, soll nun auf seine Komponenten eingegangen werden. Diese werden für die Codierung von Klassen, Instanzen und Operationen darauf benötigt und für deren Transport. Damit wird eine Kommunikation zwischen Manager und Ressource innerhalb einer CIM-Umgebung ermöglicht.

Die Erweiterung von CIM zu einem Gesamtframework wurde 1998 auf Initiative der DMTF vollzogen, um eine Interoperabilität auf allen Netzwerkschichten zu erreichen. Diese Architektur wurde auf den Namen Web Based Enterprise Management getauft, auch wenn dieser Name ein wenig irreführend ist, da es keinen Zusammenhang mit dem World Wide Web gibt.

Bei der Implementierung entschied man sich für die folgenden etablierten Standards:

Codierung:

Die Repräsentation von Klassen und Instanzen und möglichen Operationen darauf erfolgt über XML.

Transport:

Die Übermittlung von XML-Nachrichten erfolgt über HTTP.

Eine Veranschaulichung der Gesamtarchitektur WBEM bietet Abbildung 2.3.

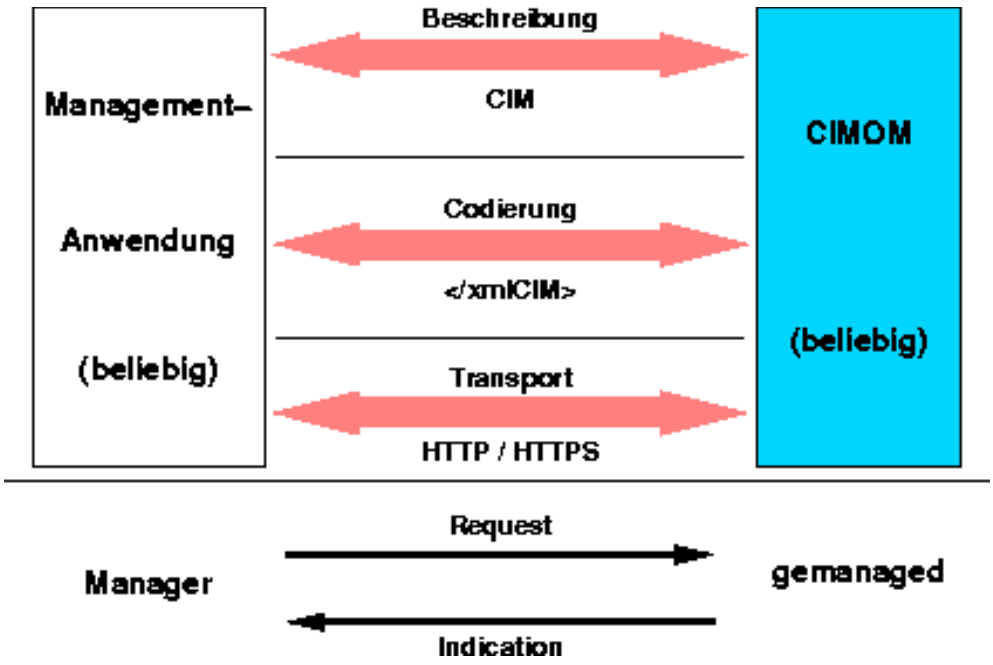

Abbildung 2.3: Architektur von WBEM

Komponenten einer WBEM-Architektur

Nachdem CIM und WBEM im Rahmen ihrer theoretischen Grundlagen und ihrer Spezifikation in den vorausgegangenen Kapiteln ausführlich beschrieben wurden, soll nun auf ihre Implementierung eingegangen werden.

Da WBEM auf Basis eines Client-Server-Konzepts arbeitet, gibt es in einer solchen Architektur zwei Seiten:

- Bei der **Manager**-Seite handelt es sich um den Client innerhalb einer CIM-Umgebung.

- Die **Ressource** ist das System, das über CIM und WBEM verwaltet wird. Damit stellt es im Rahmen der Client-Server-Umgebung die Server-Seite dar.

Während für das Management beliebige Applikationen zum Einsatz kommen können, die gegebenenfalls spezielle Verwaltungsaufgaben erfüllen, wie beispielsweise die Überwachung von kritischen Systemzuständen oder die Konfiguration von Anwendungen, bedarf es serverseitig einer verwobeneren Infrastruktur, damit ein System WBEM-managebar ist:

- Ein zentrales Softwareprogramm muss alle WBEM-Anfragen entgegennehmen und bearbeiten. Es fungiert als ein zentrales Schaltwerk, da es weiß, welche Klassen existieren und mit welchen gemanagten Systemkomponenten sie korrelieren. In Anlehnung an CORBA wird dieser Teil als **CIM Object Manager (CIMOM)** bezeichnet.

- Ein CIMOM muss Kenntnis darüber besitzen, welche Klassen, Assoziationen und Qualifier überhaupt existieren. Dazu bedarf es eines **Repositories**, in dem alle statischen CIM-bezogenen Informationen hinterlegt sind.

- Ein CIMOM, der direkt auf gemanagte Systemressourcen zugreift, ist sehr unflexibel, da es dann nicht möglich wäre, dynamisch neue Ressourcen hinzuzufügen. Daher wird für WBEM ein modulares Konzept vorgezogen: Für Instanziierungen einer bestimmten Klasse kommt genau ein **Provider** zum Einsatz.

- Ein **Modell** der Welt, die ein Provider ver- und bearbeiten kann, muss dem CIMOM bekannt gemacht werden.

- Damit eine Systemkomponente überhaupt durch einen Provider managebar wird, muss sie zunächst einmal selbst managebar sein, das heißt, sie muss eine Schnittstelle bieten, über die Verwaltungsinformationen mit der Umgebung ausgetauscht werden können. Im Rahmen von WBEM wird diese Schnittstelle als **Instrumentierung** bezeichnet.

Eine Übersicht über diese Gesamtarchitektur zeigt Abbildung 2.5. Der Fluss von Managementinformationen erfolgt dabei zwischen Manager und Ressource auf der linken Seite von oben nach unten über die Schichten Management-Anwendung -- CIMOM -- Provider -- Instrumentierung -- Zielressource. Dem korrespondiert auf der rechten Seite die Beschreibung dieser Informationen: Im Repository ist das CIM-Modell der lokal managebaren Ressourcen hinterlegt.

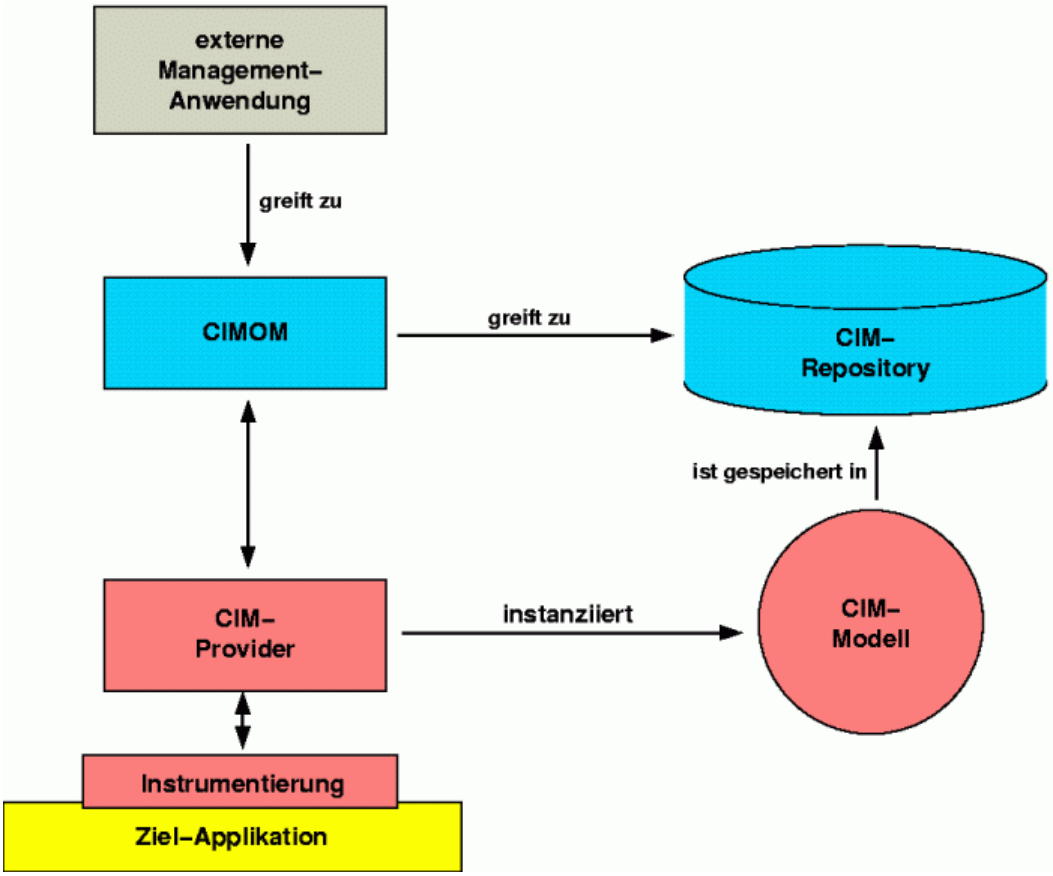

Abbildung 2.4: Korrespondenz von Modell und Implementierung in WBEM

2.1.13 CIM Object Manager (CIMOM)

Ein CIMOM fungiert als zentrale Schaltstelle für alle WBEM-Anfragen, deren Ziel das lokale System ist, auf dem er läuft. Ist eine Anfrage auf eine Klasse bezogen, wird sie über das Repository behandelt. Bezieht sich die Anfrage dagegen auf eine Instanz, wird sie an den entsprechenden Provider weitergeleitet. Auch für den umgekehrten Weg in Form einer Indication ist der CIMOM zuständig. Er weiß, welches Ereignis -- das von einem Provider gemeldet wird -- an welchen Manager zuzustellen ist.

Der Aufbau eines CIMOMs geht im Detail aus Abbildung 2.5 hervor.

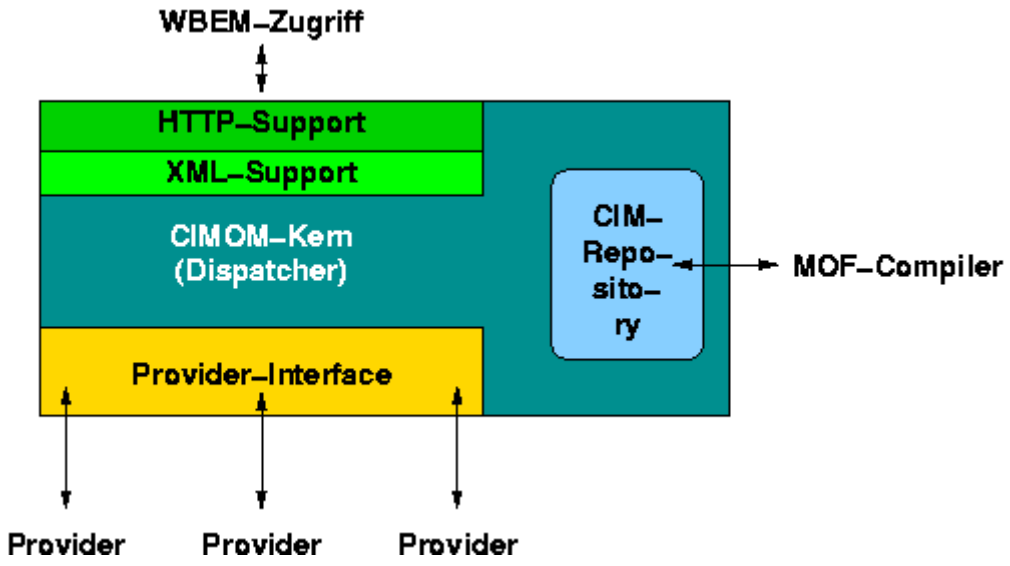

Abbildung 2.5: Komponenten eines CIMOM

- Der **Kern** des CIMOM ist für die Koordination von WBEM-Requests und Indications verantwortlich. Alle WBEM-Operationen auf Klassen kann er selbst verarbeiten, indem es die entsprechenden Informationen aus dem Repository liest (z.B. GetClass) oder in selbigem ablegt (z.B. CreateClass).

- Instanzbezogene Operationen (z.B. CreateInstance) werden direkt an den zuständigen Provider weitergereicht, die anschließende Antwort wird an den Manager weitergeleitet. Der Kern muss auch berücksichtigen, dass bestimmte Operationen (z.B. EnumerateInstances) an mehrere Provider weiterzuleiten sind, wenn die Operation auf eine Oberklasse durchgeführt wird, die in verschiedenen Unterklassen durch Provider implementiert wird.

- Bei dem **Repository** handelt es sich um eine interne Datenbank des CIMOM. In ihr sind -- getrennt nach Namespaces -- Klasseninformationen hinterlegt. Für Klassen, die durch Provider implementiert werden, wird zusätzlich angegeben, welcher Provider aufzurufen ist.

- Eine Methode, um Klasseninformationen im Repository zu bearbeiten, ist -- wie bereits beschrieben -- eine WBEM-Operation. Bei der initialen Einrichtung des CIMOM beispielsweise ist es jedoch sinnvoll, solche Daten direkt in das Repository zu laden. Daher stellen viele CIMOMs einen **MOF-Compiler** zur Verfügung, über den MOF-Dateien direkt in das Repository geladen werden können.

- Für Entgegennahme und Decodierung bzw. die Codierung und den Versand von WBEM-Anfragen benötigt ein CIMOM entsprechende Schichten für **HTTP-Support** und **XML-Support**. Aufgabe der XML-Schicht ist es auch, die Integrität der Anfrage anhand der DTD zu validieren, während die HTTP-Schicht für die Authentifizierung zuständig ist.

- Das **Provider-Interface** schließlich kommuniziert direkt mit den entsprechenden Providern. Da Provideraufrufe grundsätzlich lokal erfolgen, werden für die Implementierung normalerweise *Shared Libraries* eingesetzt. Aufgabe des Providerinterfaces ist es daher, die Bibliothek zu laden und eine Funktion aufzurufen, die der WBEM-Operation entspricht.

Provider

Aufgabe eines WBEM-Providers ist es, Instanzen einer einzelnen Klasse zu verarbeiten. Neben extrinsischen Methodenaufrufen sind die möglichen Operationen im Folgenden aufgeführt.

GetInstance	Umsetzung des Objektschlüssels in eine reale Entität, Rückgabe als CIM-Instanz
EnumerateInstances	Finden aller Instanzen, die der gesuchten Klasse entsprechen, Rückgabe als Liste
EnumerateInstanceNames	Finden aller Instanzen, die der gesuchten Klasse entsprechen und Rückgabe einer Liste von Schlüsseln
CreateInstance	Anlegen einer Instanz der entsprechenden Klasse undErzeugen des Schlüssels und Rückgabe desselben.
ModifyInstance	Verändern eines oder mehrerer Parameter einer Instanz
DeleteInstance	Umsetzung des Objektschlüssels in eine Instanz, Löschen derselbigen.
Associators	(nur für Assoziationen) Umsetzung des Objektschlüssels in eine Instanz, Finden aller damit assoziierten Entitäten, Rückgabe als Liste von CIM-Instanzen
AssociatorNames	(nur für Assoziationen) Umsetzung des Objektschlüssels in eine Instanz, Finden aller damit assoziierten Entitäten, Rückgabe als Liste von Instanz-Schlüsseln.

Da die einzelnen Operationen, die durch einen Provider durchgeführt werden müssen, sehr rechenintensiv sein können, kann ein Caching gesammelter Informationen sinnvoll sein. Da Informationen über eine Instanz zeitlich veränderlich sind, darf ein CIMOM sie nicht zwischenspeichern oder gar in seinem Repository ablegen. Stattdessen wird empfohlen, bei CPU-intensiven Instanzen, die relativ statisch sind, ein Caching auf Providerebene durchzuführen.

Zusätzlich zur Implementierung der WBEM-Operationen muss ein Provider, der Indications implementiert, das instrumentierte Subsystem überwachen und im Fehlerfall dem CIMOM das Ereignis übergeben.

Da der Aufruf des Providers ausschließlich durch den CIMOM erfolgt, der lokal läuft, bietet sich meist eine Shared Library als Basis für eine Implementierung an. Die entsprechenden WBEM-Aufrufe müssen dann als Funktionen ansprechbar sein. Die Details ergeben sich aus der jeweiligen CIMOM-Implementierung.

2.1.14 Instrumentierung

Damit eine Systemkomponente überhaupt durch einen Provider managebar wird, muss sie grundsätzlich managebar sein, das heißt, sie muss eine Schnittstelle bieten, über die Verwaltungsinformationen mit der Umgebung ausgetauscht werden können. Diese Schnittstelle ist die *Instrumentierung*.

Die Beschaffenheit der Schnittstelle selbst spielt für WBEM keine Rolle: Eine entsprechende Umsetzung wird schließlich durch den Provider vorgenommen. Stellt jedoch die gemanagte Komponente kein oder nur ein unzureichendes Interface zur Verfügung, macht dies die Implementierung von WBEM-Providern deutlich aufwändiger.

2.2 OpenPegasus

The Open Group ist ein internationales firmen- und technologieneutrales Konsortium, das sich als Vermittler zwischen Nutzern und Herstellern von Informationstechnologie versteht und dessen erklärtes Ziel die Förderung von Interoperabilität in allen Bereichen der IT ist. Sponsoren der Open Group sind die Firmen Fujitsu, Hewlett-Packard, Hitachi, IBM und Sun Microsystems.

Mit OpenPegasus hat die Open Group ein WBEM-Projekt gestartet, das vollständig in C++ realisiert ist. Es läuft unter den meisten UNIX-Arten, unter Linux und Microsoft Windows. Pegasus wird unter der *MIT open source license* vertrieben, die aktuelle Releasefassung hat die Versionsnummer 2.9.0. Das Projekt-Portal befindet sich auf der Homepage.

2.2.1 Eigenschaften von OpenPegasus

Das OpenPegasus Paket enthält über den eigentlichen CIMOM hinaus eine Reihe nützlicher Tools, die dem Entwickler bei seiner Arbeit behilflich sind. Das Kernstück stellt allerdings der *cimserver* dar, der standardmäßig als Daemon startet und im Hintergrund auf TCP Port 5988 (bzw. 5989) lauscht und CIM-over-HTTP (bzw. CIM-over-HTTPS) Anfragen entgegennimmt. Die Datenhaltung erfolgt in einem sogenannten Repository, das auf einem sekundären Speichermedium in XML Darstellung abgelegt wird. Das Repository enthält auf unterster Ebene mehrere *Namespaces*, zur parallelen Verwaltung zahlreicher Schemata. Standardmäßig wird unter Pegasus das aktuelle CIM-Schema in dem *Namespace* root/cimv2 verwaltet. Frühere CIM Versionen verwendeten dabei den *Namespace* root, seit Einführung der zweiten Version hat sich dies jedoch geändert. Jeder *Namespace* enthält zur Datenhaltung drei Verzeichnisse, classes, instances und qualifiers.

Der MOF-Compiler *cimmof* bzw. *cimmofl* ist ein weiteres nützliches Hilfsmittel und sollte hier keinesfalls unerwähnt bleiben. Er formt MOF-Files in XML um und lädt sie ins Repository, im Normalfall also in das Verzeichnis classes; hier wird zentral die Klassenhierarchie verwaltet.

OpenPegasus wird regelmäßig auf einer Vielfalt von Plattformen von der Entwicklungsgruppe geprüft.

Die Ergebnisse dieser „Nightly Test and Build Status“ können auf der Homepage von OpenPegasus [Open] gefunden werden.

2.2.2 Installation des OpenPegasus CIMOM

Benötigt werden die Quelldateien des OpenPegasus Projekts. Zwei Möglichkeiten stehen hier zur Wahl, zum einen kann ein fertiges Release heruntergeladen werden, zum anderen ein aktueller Snapshot der CVS Entwicklungsumgebung. Aufgrund der recht aktiven Entwicklergemeinde, welche dieses OpenSource Projekt besitzt, wurde auf die zweite Möglichkeit zurückgegriffen.

Der OpenPegasus-Opengroup CIMOM wurde mit C++ entwickelt, in dem betrachteten Testlauf kam die GNU Compiler Collection zur Verwendung.

Im Besitz der Quelldateien lässt sich mit der eigentlichen Erstellung fortfahren. Hierzu muss entschieden werden, auf welcher Plattform OpenPegasus verwendet wird. Mit diesem Wissen lassen sich die Umgebungsvariablen PEGASUS_HOME und PEGASUS_PLATFORM auf dem Testrechner mit korrekten Werten belegen.

OpenPegasus unterstützt dabei folgende Plattformen:

- WIN32_IX86_MSVC
- LINUX_IX86_GNU
- AIX_RS_IBMCXX
- HPUX_PARISC_ACC
- ZOS_ZSERIES_IBM

In dem vorliegenden Beispiel sind demnach folgende Anweisungen nötig:

- export PEGASUS_HOME=/mnt/disk/_cim/pegasus
- export PEGASUS_PLATFORM=LINUX_IX86_GNU

Des Weiteren sollte das Binary-Verzeichnis (bin, befindlich im Pegasus-Wurzelverzeichnis) und das Library-Verzeichnis (lib, ebenfalls im Pegasus-Wurzelverzeichnis) den entsprechenden Pfaden angefügt werden (PATH und LD_LIBRARY_PATH).

Mittels *make* im Pegasus-Wurzelverzeichnis wird der OpenPegasus CIMOM erstellt.

Nach Erstellung des OpenPegasus ist des Weiteren ein anfänglich erstelltes Repository nötig, welche das gültige CIM-Schema enthält und in welchem später alle Instanzen als XML gespeichert werden. Der zugehörige Befehl lautet *make repository* .

3 Stand der Technik

In diesem Kapitel wird ein Überblick über bereits vorhandene Techniken für die Performancemessung gegeben.

3.1 Key Performance Indicator

Der **Key Performance Indicator (KPI)** ist eine Kennzahl aus der Betriebswirtschaftslehre, anhand derer der Fortschritt oder der Erfüllungsgrad hinsichtlich wichtiger Zielsetzungen oder kritischer Erfolgsfaktoren innerhalb einer Organisation ermittelt werden kann. Als Beispiel kann man hier die Overall Equipment Effectiveness nennen, das die tatsächliche Auslastung einer Maschine oder Systems gegenüber ihrer theoretisch möglichen untersucht. [KPI]

Verwendung findet dieses Werkzeug auch im ITIL-Framework. Jedoch ist sie auf die Dienstleistungsbranche zugeschnitten und kann bei dieser Arbeit nicht berücksichtigt werden.

3.2 TPC

Das **Transaction Processing Performance Council** (**TPC**) ist ein 1988 gegründetes Non-Profit-Konsortium aus verschiedenen IT-Unternehmen, das mithilfe standardisierter Benchmarks Angaben über die Leistungsfähigkeit von Transaktionssystemen und Datenbankmanagementsystemen machen möchte. Das Ziel hierbei ist die Leistungsbewertung von Transaktionssystemen, die geschäftliche Transaktionen wie das Überweisen eines Geldbetrags oder den Bestellvorgang von Waren/Dienstleistungen abwickeln und im Allgemeinen den Zugriff auf eine Datenbank einschließen. Verschiedene Anwendungen weisen dabei verschiedene Nutzungsmuster auf. Zur Evaluation der wichtigsten Leistungsmerkmale solcher Systeme wurden Benchmarks entwickelt, welche für diese Systeme eine realistische Last nachbilden und so einen Vergleich der einzelnen Produkte zulassen sollen. Problematisch bei früheren, nicht standardisierten Benchmarks war, dass sie häufig unvollständig spezifiziert waren, dass deshalb solche Systeme nur schwer zu vergleichen und Aussagen der Hersteller die Leistungsfähigkeit ihrer Systeme betreffend nur schwer nachvollziehbar waren. Das TPC wurde deshalb zum Zwecke der Definition objektiver, standardisierter Benchmarks ins Leben gerufen, um einen möglichst guten

Vergleich von Transaktionssystemen und Datenbankmanagementsystemen (DBMS) verschiedener Hersteller zu ermöglichen. [TPC]

Es gibt verschiedene Versionen von TPC, von denen die aktuellsten an dieser Stelle beschrieben werden:

3.2.1 TPC-App

Ist ein Benchmark zur Leistungsbewertung eines Applikationsservers und für Web Services; der Benchmark testet erhältliche Applikationsserver-Produkte und deren Kommunikation mit Datenbanksystemen als Ganzes durch das Nachbilden von Lasten, wie sie bei typischen B2B-Geschäftsvorgängen entstehen.

3.2.2 TPC-C

Ist ein Benchmark, der zum Testen erhältlicher DBMS aus dem OLTP-Bereich die Auftragsbearbeitung in einem Handelsunternehmen modelliert und eine realistische Transaktionslast aus verschiedenen Transaktionstypen generiert.

3.2.3 TPC-E

Ist ein Benchmark, der wie der TPC-C-Benchmark ein DBMS aus dem OLTP-Bereich testet; er simuliert, wie Broker und Trader an der Börse handeln. Im Vergleich zum TPC-C-Benchmark wird hier mehr die Interdependenz einzelner Transaktionen betrachtet, dass also Transaktionen bestimmte weitere Transaktionen zur Folge haben können.

3.2.4 TPC-H

Ist ein Benchmark, der die Leistungsfähigkeit von Datenbankmanagementsystemen aus dem Decision-Support-Bereich testet. Das System wird mit Ad-hoc-Anfragen, also Anfragen, die kein Vorwissen nutzen, getestet; es handelt sich im Allgemeinen um Anfragen, die eine relativ lange Ausführungszeit zur Folge haben.

Auch hier besteht das Problem weiterhin darin, dass diese Benchmarks auf spezielle Anwendungsfälle, in diesem Fall Transaktions- und Datenbankmanagementsysteme zugeschnitten sind und nicht benutzt werden kann.

3.3 SPECjbb2005

SPECjbb2005 ist ein Java Server Benchmark, der von der Standard Performance Evaluation Corporation (SPEC) entwickelt wurde. Sie misst die Leistung von serverseitigem Java durch Emulation eines 3-tier-Client/Server Systems.

Dieser Benchmark testet die Implementationen der Java Virtual Machine (JVM), Just-in-Time-Compiler, Garbage Collection, Threads und einige Bereiche des Betriebssystems. Ausserdem wird die Leistung der CPU, Caches, Memory Hierarchie und die Skalierbarkeit der Shared Memory Prozessoren (SMP) gemessen. Dieses Konzept erlaubt auch eine herstellerübergreifende Vergleichsmöglichkeit und wird von der Industrie durchweg akzeptiert.[SPEC05]

Sie ist ein sehr guter funktionaler Leistungstest für Java-Plattformen.

Da der OpenPegasus CIM Server in C++ implementiert ist und dieser Benchmark für serverseitiges Java entwickelt wurde, kann es für diese Arbeit nicht benutzt werden.

3.4 Nagios

Die Software Nagios dient der Überwachung komplexer IT-Infrastrukturen.

Dafür bietet Nagios eine Sammlung von Modulen zur Überwachung von Netzwerken, Hosts und speziell Diensten sowie einer Web-Schnittstelle zum Abfragen der gesammelten Daten. Nagios steht unter der GPL, ist also Open Source, und läuft unter zahlreichen Unix-ähnlichen Betriebssystemen.

Die zu kontrollierenden Hosts und Dienste werden mittels Konfigurationsdateien konfiguriert und Nagios bekannt gemacht. Die Überwachung kann aber erst erfolgen, wenn dementsprechend Kommandos definiert wurden. Das Zusammenfassen in Gruppen für einzelne Hosts, Dienste und Kontakte ist ebenfalls möglich.

Nagios kann den Status verschiedener Dienste (z. B. SSH, FTP, HTTP) sowie den Festplattenplatz, Speicher- und CPU-Auslastung usw. über diverse Plug-ins abfragen und auswerten. Da einige Testmethoden auf Protokollebene arbeiten (TCP, SNMP, …), ist es möglich, verschiedene Betriebssysteme zu überwachen.

Sobald ein Dienst oder ein Host einen (teilweise einstellbaren) kritischen Wert erreicht oder gar nicht mehr verfügbar bzw. erreichbar ist, alarmiert Nagios die Kontaktpersonen über beliebige Kanäle (z. B. E-Mail, SMS, Pager, IM-Messages, Telefonanrufe, …). Dabei besteht auch die Möglichkeit, festzulegen, in welcher Reihenfolge Meldungen an weitere Kontaktpersonen erfolgen sollen, wenn eine Störung nach den ersten Meldungen nicht behoben wurde (Eskalationsmanagement). Ebenfalls können bei der Überwachung der Dienste untereinander bestehende Abhängigkeiten berücksichtigt werden. Wird etwa die Erreichbarkeit eines Rechners und auf ihm laufende Programme überwacht, so werden bei einem Ausfall des gesamten Rechners die Meldungen über die einzelnen nicht mehr laufenden Programme unterdrückt. [NAG]

Nagios ist ein Monitoring-System für ein Gesamtnetzwerk und eignet sich nicht für die Performance-Qualitätssicherung bei der Softwareentwicklung. Es ist vielmehr ein Tool für die Netzwerküberwachung und der Früherkennung von Netzwerkproblemen.

4 Motivation

In diesem Kapitel wird auf die Motivation dieser Arbeit eingegangen und der Lösungsansatz aus Kapitel 6 in einem ersten Schritt vorgestellt.

4.1 Problemstellung

Die Performance-Qualitätssicherung bei der Entwicklung von Serversoftware birgt ein großes Problemfeld, die mit der Dynamik der Entwicklung stark zunimmt.

Dies kristallisiert sich besonders bei einem hochdynamischen Umfeld wie bei der OpenSource-Entwicklung heraus. Innerhalb einer OpenSource-Entwicklung können kleinere Änderungen der Codebasis zu drastischen Performanceeinbußen führen. Dies kann allerdings nicht allein mit einer Bewertungsgröße abgefangen werden und dient lediglich als Werkzeug für einen Prozess der Qualitätssicherung. Dieser Prozess könnte in Form von regelmäßigen TestBuilds der aktuellen Entwicklung stattfinden, desweiteren müsste eine regelmäßige Bewertung anhand einer Kenngröße stattfinden um den Einfluss der Änderung auf die Performance zu überwachen und zu protokollieren.

Diese setzt in einem ersten Schritt voraus, dass man die Gesamtleistung eines Systems *objektiv* definieren kann, um darauf aufbauend die Performance zu bewerten. Jedoch ist das nicht ohne Weiteres möglich, denn die menschliche Sicht in Bezug auf die Gesamtleistung ist oft subjektiv und emotional, folglich schwierig zu bestimmen. Das bedeutet, dass die Definition der Gesamtleistung in einem Anwendungsfall A eine vollkommen andere ist als im Anwendungsfall B. Somit wird ersichtlich, dass man die Gesamtleistung ausschließlich *spezifisch* definieren kann, die abhängig ist vom vorhandenen System und der Aufgabenstellung.

Die Auseinandersetzung mit dieser Problematik deckt ein weiteres Problem auf: Leistungskenngrößen.
Es gibt viele Leistungskenngrößen, mit denen man die Performance eines Servers messen kann. Das Problem hierbei ist nunmehr, welche Kenngrößen extrahiert werden müssen, um die Bewertungsgröße zu erstellen. Dafür müssen in einem zweiten Schritt Anforderungen

festgelegt werden, mit deren Hilfe einzelne Kenngrößen ausgewählt und zu einer Bewertungsgröße konsolidiert werden, die gleichermaßen reproduzierbar als auch signifikant ist. Dadurch wird es erst möglich, Vergleiche zwischen verschiedenen Softwareständen zu vollziehen und Änderungen an der Codebasis bezogen auf die Gesamtleistung zu klassifizieren.

Wenn man diese Schritte vollzogen hat, bleibt ein letztes Problem übrig: Konzept des Verfahrens zur Konsolidierung.

Das Ziel des Konzeptes sollte es sein, eine benutzerfreundliche Ausgabe zu beinhalten, die eine leichte Interpretation der Gesamtleistungsänderung sicherstellt. Das bedeutet im engeren Sinne, dass das Konzept nicht von der Qualifizierung des Benutzers abhängig sein darf. Unter diesem Aspekt muss in einem letzten Schritt ein solider Grundgedanke erarbeitet werden, dass diese beiden Voraussetzungen erfüllt, gleichzeitig aber auf die Aufgabenstellung dieser Arbeit zugeschnitten ist.

4.2 Lösungsansatz

Im Rahmen dieser Arbeit wird zunächst definiert, was Gesamtleistung spezifisch für diesen Anwendungsfall ist. Basierend auf dieser Definition werden verschiedene Kenngrößen vorgestellt und hierarchisch aufgeteilt. Dabei wird unterschieden zwischen systemorientierten Kenngrößen und benutzerorientierten Kenngrößen. Dies stellt die Grundvoraussetzung für die geforderte Objektivität und die Signifikanz der späteren Bewertungsgröße dar. Anschließend werden die Kenngrößen auf ihre Schwankungs-/Fehlerrate untersucht und demzufolge ausgewählt.

Das im darauffolgenden Kapitel vorgestellte Konzept zielt darauf ab, die Gesamtleistung als eine geometrische Fläche darzustellen, so dass man für den Vergleich zwischen verschiedenen Softwareständen die Flächeninhalte in Relation setzen kann und eine Kennzahl als Bewertungsgröße erhält. Dadurch wird das Ergebnis allgemeinverständlich und die Interpretation der Gesamtleistungsänderung simpel.

5 Analyse

In diesem Kapitel wird die Definition des Begriffs der Gesamtleistung aufgebaut und anschließend die Analyse für die Auswahl der Kenngrößen zur Bildung der Bewertungsgröße für die Gesamtleistung diskutiert.

5.1 Qualitätsbegriff in der Softwareentwicklung

Qualität ist allgemein definiert als „*die Gesamtheit von Eigenschaften und Merkmalen eines Produkts oder einer Tätigkeit, die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht*“ [Ba98].

Wenn man das auf die Entwicklung von Software projiziert, kann man die Qualität in zwei Cluster unterteilen:

- Anforderungsmanagement
- Performance.

5.1.1 Anforderungsmanagement

Das Anforderungsmanagement ist ein Managementinstrument zur Steuerung von Entwicklungen, welches dem Management ermöglicht, rechtzeitig auf äußerliche Einwirkungen und Veränderungen in der Gesamtanforderung schnell und effizient zu reagieren.

Dafür stellt es drei Werkzeuge bereit:

- Risikomanagement
- Änderungsmanagement
- Umsetzungsmanagement.

Risikomanagement ist ein systematisches Verfahren, wodurch man potenzielle Risiken erfassen und bewerten kann. Am einfachsten kann man es sich an folgendem Beispiel klarmachen:

Firma A entwickelt für Kunde B Software. Kunde B stellt Anforderungen bezüglich der Funktionalität der zu entwickelnden Software. Die Firma A hat durch Neueinstellungen einen Zuwachs von 20% erfahren. Nun ermittelt das Projektmanagement, welches aus Firma A und Kunde B besteht, die Risiken für das Projekt. Kunde B sieht als Risiko die Unerfahrenheit der neuen Mitarbeiter. Firma A muss nun auf dieses Risiko reagieren und schickt alle neuen Mitarbeiter zu geeigneten Schulungen und Seminaren.

Dieses einfache Szenario beschreibt, wie wichtig das Risikomanagement in jedem Projektstatus ist.

Das *Änderungsmanagement* beschreibt Vorgehensweisen für Produktmodifikationen und Plananpassungen in Projekten. Sie wird als ein Prozess realisiert. Der Kern des Änderungsmanagements bildet die Änderungsanforderung, die den Wunsch einer Änderung oder Modifikation eines Produkts oder Projekts formal beschreibt. Ein Beispiel aus der Softwareentwicklung ist das Change Management (ITIL).

Als letzter Punkt bleibt das *Umsetzungsmanagement*. Sie hält dem Projektmanagement Maßnahmenformulierungen bereit und kontrolliert den Umsetzungsfortschritt.

Erweitern wir zum Verständnis das Beispiel mit Firma A und Kunde B:

Kunde B verlangt von Firma A einen transparenten Terminplan, der alle Status des laufenden Projekts aufzeigt. Desweiteren bespricht Firma A regelmäßig den Entwicklungsstand mit Kunde B.

5.1.2 Performance

Ein weiterer Aspekt der Qualität bei der Softwareentwicklung ist die Performance.

In der Informatik wird Performance als allgemeiner Ausdruck für die Leistung eines Systems, welches Hardware, Software oder auch nur ein einzelner Algorithmus sein kann, benutzt. So unterschiedlich die Komponenten sein können, so unterschiedlich sind auch die möglichen Parameter die zur Bestimmung von Performance herangezogen werden.

Demzufolge ergibt sich die Performance aus zwei Hauptfaktoren:

- Leistungsfähigkeit
- Zuverlässigkeit.

5.1.2.1 Leistungsfähigkeit

Die Leistungsfähigkeit einer Software muss aus zwei Blickwinkeln betrachtet werden.

Erstens, die systemorientierte Sicht. Diese zeigt die Limitationen des eingesetzten Systems auf und stellt die Obergrenze der Leistungsfähigkeit der Software dar.

Beispiel:

Software A ist für Rechnersystem B konzipiert. Software A läuft ziemlich nah an den Grenzen des Rechnersystems B. Portiert man Software A zu einem Rechnersystem C, dessen Leistungswerte (Hardware) weitaus höher liegen als Rechnersystem B. So lässt sich folgende Aussage formulieren: Software A läuft auf Rechnersystem B performant, allerdings bedeutet dies nicht, dass Software A auf Rechnersystem C performant läuft.

Dieses Beispiel verdeutlicht eine alltägliche Problemstellung in der Informatik. Eine Realitätsnahe Umsetzung dieses Beispiels zeigt die Umstellung von Embedded Systems von einer Hardwaregeneration zur Nächsten.

Zweitens, die benutzerorientierte Sicht. Sie beinhaltet die Effektivität und Effizienz des gelieferten Codes. Diese Sicht ist gleichzeitig die am schwierigsten beschreibbare Größe, da sie ein gewisses Potenzial an Subjektivität mit sich bringt. Das hat den einfachen Grund, dass jeder Entwickler nur diejenigen Kenngrößen betrachtet, die für seine Arbeit relevant sind.

Wenn man eine Software aus benutzerorientierter Sicht analysieren möchte, so kann dies nur in Relation zur systemorientierten Sicht geschehen.

5.1.2.2 Zuverlässigkeit

Unter Zuverlässigkeit versteht man die Gewährleitung der Funktionalität einer Software und des zugrundeliegenden Systems. Es ist somit ein zusammenhängendes Merkmal von Software und Hardware.

5.1.3 Leistungsbewertung

Unter Leistungsbewertung versteht man den Versuch, eine objektive Aussage über ein Rechnersystem durch messbare Größen zu erhalten.

Der Grund für diese Bewertung ist es, eine qualitative Aussage über einzelne Komponenten eines Systems oder Software zu treffen, so dass es möglich wird, potenzielle Fehlerquellen bzw. kritische Zonen (Flaschenhälse) rechtzeitig zu erfassen. Dadurch wird eine Optimierung der Qualitätssicherung angestrebt.

Für eine Leistungsbewertung gibt es Voraussetzungen, die die Signifikanz gewährleisten müssen. Das ist die Analyse und Auswahl von Hard- und Software eines Gesamtsystems, die man mittels Messmethoden und –programmen ermitteln kann.

Wie bereits erwähnt, ist es schwierig, die Leistungsfähigkeit einer Software zu bestimmen, da man das nicht isoliert machen kann. Hinzu kommen noch zum einen, dass es keine objektiven Vergleichskriterien gibt und zum anderen, dass viele Einflüsse darauf einwirken. Der größte Einfluss bei der Entwicklung von Serversoftware ist die Qualität des geschriebenen Codes.

Damit man die Performance bewerten kann, braucht man den Begriff der Gesamtleistung. Sie bildet die Grundlage für das objektive Vergleichskonstrukt, so dass man eventuelle Performanceeinbußen auf die Codebasis deuten kann. Zu diesem Zweck wird als nächstes die Gesamtleistung definiert.

5.2 Definition der Gesamtleistung

Wie in Kap. 5.1 beschrieben ist, bildet die systemorientierte Sicht die Obergrenze der maximal erreichbaren Performance. Das bedeutet, dass keine Software schneller sein kann, als das vorhandene System zulässt.

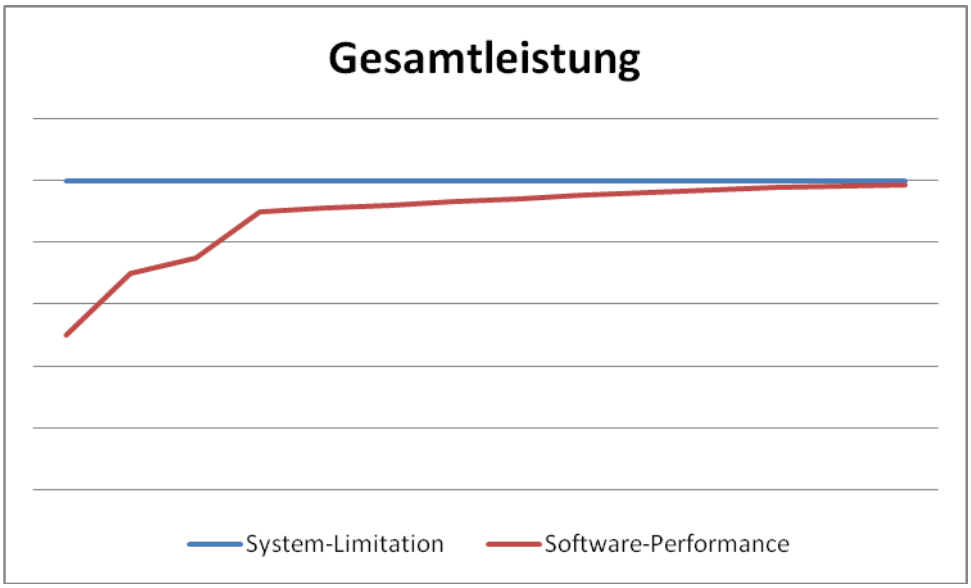

Abbildung 5.1 Gesamtleistungsgraph

Abbildung 5.1 zeigt, dass die Performance einer Software sich asymptotisch an die Limitation des vorhandenen Systems annähert. Hierbei ist zu beachten, dass die Limitation unterschiedlich zu verstehen ist. So können sie im Falle von Embedded Systems die gesamte Leistung der Hardware und der Mikrokernels sein, und in einem anderen Fall kann es nur ein reservierter Teil der vorhandenen Ressourcen sein.

Ist dabei der Unterschied zwischen der Kurve der Software (s. Abbildung 5.1) groß in Relation zur Limitation des Systems, so lastet die Software die vorhandenen Ressourcen nicht vollständig aus. Dies bedeutet, dass die Software weniger performant ist. Ebenfalls lässt dies den Rückschluss zu, dass die Performance der Software mit absteigendem Abstand größer wird.

Wie im Beispiel in Kap. 5.1 beschrieben ist, stellt allerdings die Portierung von Software auf Systeme mit anderen Leistungsmerkmalen eine große Herausforderung der Informatik dar. Dies kann man sich einfach verdeutlichen, indem man die Limitationskurve aus Abbildung 5.1 in der X-Achse nach oben bzw. nach unten verschiebt (s. Abbildung 5.2).

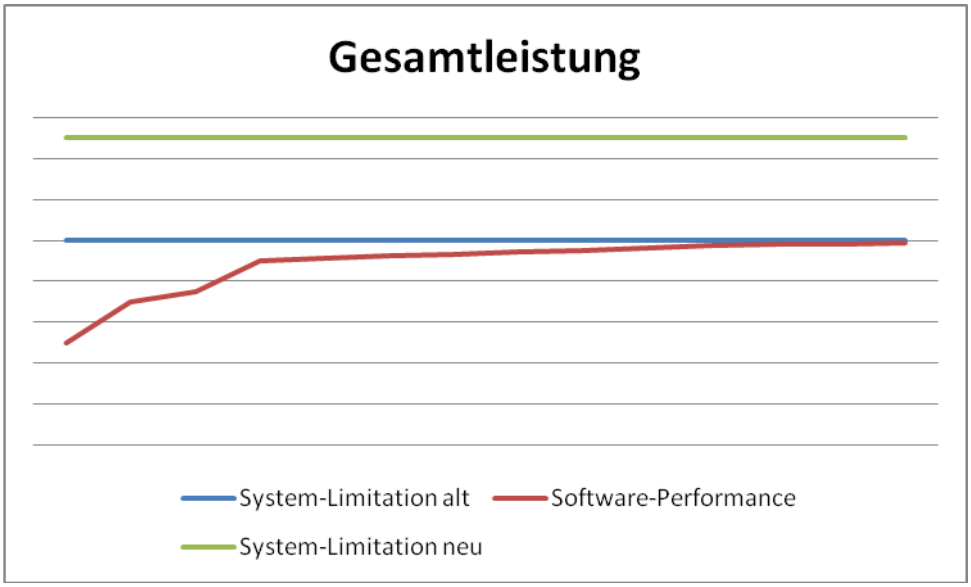

Abbildung 5.2 Veränderung der System-Limitation

Unter diesen Gesichtspunkten ist die Gesamtleistung die bestmögliche Annäherung der Software-Performance zur Limitation des Systems, bei dem der Abstand der Software-Performance am geringsten zur Limitation des vorhandenen Systems.

Innerhalb dieser Arbeit wird diese Definition als Gesamtleistung verstanden.

Für die Bildung der Gesamtleistung werden Leistungskenngrößen benötigt, wodurch die Gesamtleistung resultieren kann und sowohl die Ressourcen des vorhandenen Systems beschreiben als auch Aufschluss über die Qualität der Software geben können.

5.2.1 Leistungskenngrößen

Damit eine quantitative Leistungsanalyse vollzogen werden kann, müssen die Leistungskenngrößen in mess- und bewertbare Metriken untergliedert werden [LZ99].
An dieser Stelle werden Leistungskenngrößen zur Performancemessung vorgestellt.

Antwortzeit

Die Antwortzeit ist eine zeitbezogene Größe, die die Zeit misst, wie lange ein System oder Teilsystem zur Erledigung einer Aufgabe/Anforderung benötigt. Ein Beispiel hierfür ist die Zugriffzeit bei DDR-RAMs.

Die Antwortzeit kann man wiederum in zwei Größen unterteilen: Bearbeitungszeit und Wartezeit. Die Bearbeitungszeit beschreibt lediglich die Bearbeitungsdauer einer Aufgabe/Anforderung an einem Rechnersystem oder an einer Softwarestelle. Die Wartezeit hingegen ist die Dauer, die ein Auftrag an einer Stelle des Systems oder Software warten muss, bevor es bearbeitet wird. Die meistbenutzten Zeiteinheiten hierbei sind Sekunden, Millisekunden oder CPU-Takte.

Die Antwortzeit resultiert aus der Summe dieser beiden Größen.

Durchsatz

Die Größe Durchsatz gibt an, wieviele Aufgaben/Anforderungen pro Zeiteinheit abgearbeitet werden. Einfacher ausgedrückt gibt sie das Datenvolumen an, welches eine Software oder Rechnersystem handhaben und verarbeiten kann. Sie stellt gleichzeitig auch den physikalischen Leistungsbegriff dar. Beispiel sind die Übertragungsraten von Festplatten wie z.B. 100 MByte/Sek oder Seiten/Minute eines Druckes.

Die benutzten Maßeinheiten bei dieser Größe sind Million of Instructions Per Second (MIPS) oder Million of Floating-Point-Operations Per Second (MFLOPS) bei CPU-Durchsätzen, Jobs pro Sekunde bei Batch-Systemen sowie Bit pro Sekunde bei allgemeiner Datenübertragung.

Zusätzlich kann man auch hier Größen wie Grenzdurchsatz und Verlustrate nennen.

Der Grenzdurchsatz beschreibt den maximal erreichbaren Durchsatz, d.h. die Anzahl an Aufträgen, für die die Antwortzeit nicht eine bestimmte Schranke überschreitet.

Die Verlustrate hingegen beschreibt die pro Zeiteinheit verlorengegangenen oder abgewiesenen Aufträge.

Für diese Arbeit wird zwischen External Throughput Rate (ETR) und Internal Throughput Rate (ITR) unterschieden, da diese Größen Performancemetriken für den Durchsatz darstellen [IBM05].

Auslastung

Die Auslastung gibt an, welchen Prozentsatz der Zeit ein System oder Komponente aktiv bzw. produktiv ist. Beispiele sind Floating Point Unit in einer CPU, CPU allgemein und Netzwerkverbindungen. Sie wird auf das Gesamtsystem bezogen und unterteilt sich u.a. in folgende Zweige: CPU-Auslastung, Speicherauslastung und Netzwerkauslastung sowie Instruktionslänge.

Die CPU-Auslastung ist hierbei die relative Aktivzeit der CPU, die zur Bearbeitung von Aufträgen bzw. Datenmengen benötigt wird. Eine hohe Auslastung spiegelt sich daran wieder, dass die Antwortzeit stark zunimmt.

Die Speicherauslastung ist eine Größe für den Speicherverbrauch, das ein Auftrag/Anforderung zur Ausführung benötigt.

Die Netzwerkauslastung spiegelt die Bandbreitenausnutzung eines Netzwerkes dar.

Instruktionslänge gibt die CPU-Taktrate eines Auftrages/Anforderung an.

5.2.2 Auswahl der Kenngrößen

Das Ziel dieser Arbeit ist es, dass man Änderungen der Gesamtleistung auf die Codebasis deuten kann. Die Gesamtleistung ist nach Kap.5.2 eine Annäherung der Software-Performance zur Systemlimitation. Sie muss auf der einen Seite den Ressourcenverbrauch des Systems durch die Software beschreiben und auf der anderen Seite die tatsächlichen Ressourcen aufzeigen. Somit muss die Auswahl der Kenngrößen zur Bildung der Gesamtleistung folgende Punkte enthalten:

- Qualifizierung der Kenngrößen zur Bildung der Gesamtleistung
- Fehlerbetrachtung
- Schwankungsrate
- Signifikanz der ermittelten Bewertungsgröße

Für die Beschreibung der Systemressourcen wird die Referenzierung auf drei Aspekte des Systems benötigt: CPU, Speicher und Durchsatz.

Durch diese Größen ist man in der Lage, das maßgebende System zu beschreiben. Gleichzeitig kann man durch Analyse dieser Größen aber auch auf die Qualität der Software deuten. Deshalb eignen sich besonders gut zur Bildung der Gesamtleistung, um daraus die Bewertungsgröße zu ermitteln.

5.2.2.1 Qualifizierung der Kenngrößen

Für die CPU-Referenz gibt es zwei Größen zur Auswahl; CPU-Auslastung und Instruktionslänge.

Die CPU-Auslastung kommt hierbei nicht in Frage, da es stark von der Belastung des Gesamtsystems abhängt. Somit würde die Objektivität verlorengehen und die Signifikanz nicht bestätigt.

Die Instruktionslänge hingegen ist eine relativ feste Größe, da sie die verbrauchte CPU-Taktrate bei der Ausführung eines Auftrages widergibt und ist in der Lage, Hinweise auf die Codebasis bei einer Performanceveränderung zu geben.

Beispiel:

Eine Transaktion verbraucht eine gewisse Anzahl an Instruktionen. Nun ändert man die Codebasis. Jetzt verbraucht dieselbe Transaktion mehr Instruktionen. Somit kann man festhalten, dass die Performanceänderung direkt mit dem Code zusammenhängt.

Für die Speicher-Referenz gibt es im Grunde nur eine Größe; Speicherauslastung.
Die Speicherauslastung gibt den aktuellen Speicherverbrauch eines Prozesses in Bytes wieder und ist somit eine Größe bezüglich der Speichereffizienz, durch sie man auch in der Lage ist, Codeoptimierungen deuten zu können. Durch Variationen in der Codebasis kann man feststellen, ob die Änderungen positiver oder negativer Natur sind.

Beispiel:

Eine Transaktion hat einen gewissen Speicherverbrauch. Wenn man nun die Codebasis ändert, so dass der Speicherverbrauch weniger wird, kann man erkennen, dass durch die Änderung eine Optimierung erlangt wurde.
Somit ist die Forderung, dass man die Gesamtleistung bilden und gleichzeitig ein Indiz für die Codebasis geben kann, gegeben.

Die letzte Größe ist der Durchsatz. Für diese Größe stehen zwei Performancemetriken zur Auswahl: die External Throughput Rate (ETR)und die Internal Throughput Rate (ITR).
Sie ist die einzige Größe, die auch die benutzerorientierte Sicht miteinbezieht, da der Durchsatz die Antwortzeiten bestimmt und diese direkt den Benutzer beeinflussen.

Die ETR ist die Transaktionsrate eines Systems. Sie beschreibt somit die beendeten Transaktionen innerhalb einer Zeitspanne. Sie ist relativ unabhängig von der Hardware und assoziiert mit dem Durchsatz des Systems.

Die ITR hingegen beschreibt die Anzahl der beendeten Transaktionen innerhalb der verbrauchten CPU-Zeit und ist eine Funktion der CPU-Schnelligkeit. Das bedeutet, je schneller die CPU ist, umso höher fällt die ITR aus. Zudem ist sie abhängig vom Betriebssystem und der Transaktionscharakteristika. Kurze, triviale Transaktionen haben eine höhere ITR zur Folge.

Da für eine Bewertung aber eine Objektivität verlangt wird, qualifiziert sich die ITR aufgrund dieser Gegebenheiten nicht für eine Auswahl.

Durch diese Analyse der Qualifizierung der vorhandenen Kenngrößen fällt die Entscheidung auf Instruktionslänge, Speicherauslastung und ETR. Diese stellen die Größen dar, die eine objektive Betrachtung des Systems erlauben, aber auch auf die Codebasis deuten können.

In einem nächsten Schritt werden nun die Fehlerraten dieser Größen betrachtet, um die Signifikanz der Bewertungsgröße zu erreichen.

5.2.2.2 Fehlerbetrachtung

Nachdem die Größen auf ihre Qualifizierung untersucht und ausgewählt wurden, wird an dieser Stelle auf ihre Fehlerrate eingegangen. Diese ist wichtig für die später geforderte Signifikanz der Bewertungsgröße, da diese nur im Falle einer geringen Fehlerrate gegeben ist. Wenn die Fehlerrate zu hoch, kann man die betroffene Größe nicht in die Bewertung einfließen lassen, da das Nutzen der Bewertungsgröße enorm abnehmen würde.

ETR

Die ETR hat einen absoluten Messfehler: die Round-trip-Time.

Unter Round-trip-Time (RTT) versteht man die Reaktionszeit eines kompletten Netzwerks. Es ist die Zeitspanne, die erforderlich ist, um ein Signal von einer Quelle über das Netzwerk zum Empfänger zu senden und die Antwort des Empfängers wiederum über das Netzwerk zurück zum Sender zu transportieren.

Um diesem Messfehler vorzubeugen, kann man zwei Timer benutzen. Der erste Timer (Tt) liest die Zeit bis zum Absenden der n-Transaktionen aus und der zweite Timer (Tr) liest die Zeit aus nach Erhalt der Antwort.

Nachdem man diese beiden Werte erhält, kann man durch [(Tt)+(Tr)/n] diesen absoluten Messfehler bestimmen. Im diesem Fall kann man die ETR in die Bewertung hinzufügen, da zwar ein Messfehler vorhanden ist, dieser aber absolut ist und durch Berechnung relativiert werden kann. Somit ist eine Genauigkeit bei der Ermittlung dieser Größe gegeben.

Instruktionslänge

Bei der Instruktionslänge gibt es den Fehler, dass die Instruktionslänge für einen Auftrag nicht die tatsächliche Instruktionslänge darstellen kann. Das hat den einfachen Grund, dass bei der Ermittlung der Instruktionslänge die benötigten Instruktionen zum Hochfahren des Systems oder der Leerlaufprozess miteinbezogen werden. Um diesen Fehler auszuschließen, wendet man ein zweistufiges Messverfahren an, bei dem im ersten Anlauf die benötigten Instruktionen zum Hochfahren und Leerlauf ermittelt werden und anschließend beim Absetzen des Auftrages dieser Schritt wiederholt wird. Darauf wird im folgenden Kapitel

näher eingegangen. Aufgrund dieser Vorgehensweise ist eine Genauigkeit auch in diesem Fall gegeben.

Speicherauslastung

Hier ist die Fehlerbetrachtung analog zur Instruktionslänge. Auch hier besteht die Gefahr, dass in die Ermittlung dieser Größe die Speicherauslastung im Leerlauf miteinfließt. Die Vorgehensweise in diesem Fall ist ebenfalls analog zur Instruktionslänge und es wird ein zweistufiges Messverfahren angewendet, um auch hier den Messfehler zu relativieren.

5.2.2.3 Schwankungsrate

Beim ETR ist die Schwankungsrate relativ gering, da es durch den Benutzer selbst berechnet werden kann. Dieser beläuft sich auf 20% bei Remoteanfragen und geht runter auf 10% bei Lokalanfragen.

Bei der Speicherauslastung ist die Bestimmung der Schwankung etwas schwieriger. Es kann vorkommen, dass sich die Speicherauslastung parallel laufenden Programmen anpasst oder durch die Benutzeraktivitäten während der Messung verändert wird. Dem kann man nicht vorbeugen. Deshalb wird der Idealfall betrachtet, welches in unserem Fall auch der Realitätsfall ist. Da die Test Builds nächtlich ablaufen, bestehen diese Gefahren nicht. In diesem Fall liegt dann die Schwankung der Speicherauslastung bei etwa 20%.

Die Schwankung der Instruktionslänge ist analog zu der Speicherauslastung. Auch hier liegt die Schwankung bei etwa 15%. Das liegt daran, dass redundante Instruktionen vorkommen können aufgrund von „Death-Code“.

5.3 Signifikanz der Bewertungsgröße

Aus der Diskussion aus Kap. 5.2 folgt heraus, dass für die Bildung der Gesamtleistung diese Größen die erforderliche Signifikanz belegen.

Sie sind qualifizierte Größen, die eine qualitative Zuordnung der erlangten Werte in Bezug auf das System erlauben und dessen Leistung beschreiben.

Sie besitzen zusätzlich eine bestimmbare Fehlerrate, die man durch Berechnung relativieren kann und weisen eine relativ niedrige Schwankungsrate auf.

Die Zusammenführung dieser Aspekte unter der Betrachtung der Definition der Gesamtleistung rechtfertigt die signifikante Bewertungsgröße für die Gesamtleistung

6 Konzept und Implementierung

In diesem Kapitel werden die in Kapitel 4 vorgestellten Aussagen zuerst zu einem Konzept zusammengefasst und dann auf eine Implementierung instanziiert.

6.1 Konzept

Die Bildung der Gesamtleistung durch die ausgewählten Größen aus Kapitel 5 wird durch ein **Performancedreieck** realisiert. Dabei werden die Größen Speicherauslastung und Instruktionslänge auf der X-Achse angelegt, da diese Größen sich entgegenwirken: Je mehr Instruktionen gebraucht werden, umso höher steigt der Speicherverbrauch.

Der Durchsatz hingegen, in diesem Fall in Form der ETR, wirkt auf beide Größen analog und wird auf der Y-Achse angelegt. Wenn man die Werte für diese Größen besitzt, kann man nunmehr den Flächeninhalt berechnen und diese bei verschiedenen Softwareständen miteinander vergleichen.

Diese Vorgehensweise stellt einen objektiven Vergleich dar und kann auf einfache Weise reproduziert werden.

6.1.1 Grundlage für Flächenberechnung

Die Grundlage für die Flächenberechnung liefert der Satz von Heron.

Die bekannteste Formel für die Flächenberechnung eines Dreiecks lautet [A=gh/2]. Dabei stehen die Bezeichnungen g und h für die Länge einer Seite (Grundseite) und die zu dieser Seite senkrechte Höhe. Die Formel setzt voraus, dass man wenigstens eine der drei Höhen des Dreiecks kennt – eine Bedingung, die längst nicht immer erfüllt ist.

Sind etwa die Seitenlängen a, b und c gegeben, so würde die Anwendung von [A = g*h/2] erst die mühsame Berechnung einer Dreieckshöhe erfordern. Wesentlich schneller führt in diesem Fall die so genannte heronische Formel

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

zum Ziel.

Sie ist benannt nach dem griechischen Mathematiker und Physiker Heron von Alexandria (etwa 50 bis 100 n. Chr.).

6.2 Implementierung

Innerhalb dieses Abschnitts wird die Implementierung des vorgestellten Konzepts aus Kapitel 6.1 erläutert.

6.2.1 Overall-Design der Implementierung

6.2.1.1 Bedeutung der Implementierung als Ganzes

Die vorgestellte Implementierung stellt einen Algorithmus dar, der aus den ermittelten Messwerten eine Kennzahl resultieren lässt, mit der zwei Softwarestände miteinander verglichen werden können und eine Interpretation bei einer möglichen Performanceveränderung vollzogen wird. Das Ergebnis stellt somit eine automatisierte Performanceanalyse dar.

Bezogen auf die Nightly Test Builds des OpenPegasus bedeutet dies, dass man die Performance des CIM Servers für den Tag X mit der Performance des Tages X-1 vergleichen kann und an der erhaltenen Kennzahl erkennt, ob und in welchem Maße die Performance sich verändert hat. Bei einer negativen Performanceänderung, die einer Codebasisänderung folgt, werden die Konsequenzen dieser Codeänderung sofort erkannt und der Grundstein für die zeitnahe Reaktion gelegt.

Somit erhält man ein automatisiertes und regelmäßiges Vergleichswerkzeug für die Performance, um die Qualitätssicherung zu sichern.

6.2.1.2 Einbettung in Konzept von OpenPegasus

Die vorgestellte Implementierung ist momentan als eine eigenständige Applikation designed. Dies steht allerdings im Kontrast zu dem Prinzip der Nightly Test Builds.

Um dieser Anforderung gerecht zu werden sind weitere Schritte nötig. Dies kann auf 2 Arten erfolgen:

- Einbettung in OpenPegasus-Code
- Skriptgesteuerter Ablauf

Nachdem die Sprache der Implementierung als C++ gewählt worden ist und die Implementierung von OpenPegasus auch in derselben Programmiersprache geschrieben ist, wäre es möglich, die Implementierung dieser Arbeit in die Codebasis von OpenPegasus zu integrieren und durch Parameter die Messung der Gesamtleistung „einzuschalten“. Damit wäre es dem Benutzer überlassen die Performance zu überwachen.

Ein skriptgesteuerter Ablauf stellt die zweite Möglichkeit der Automatisierung dar. Mit Nightly Test Builds wird der aktuelle Entwicklungsstand kompiliert und zu einer Applikation zusammengefasst. Es wäre möglich ein Skript zu schreiben, welches den Ablauf steuert. Es müsste zuerst die Kompilierung angestoßen werden, danach könnte die Performanceüberwachung gestartet werden.

Nachdem die Nightly Test Builds von OpenPegasus bereits verfügbar sind, kann die Implementierung das aktuelle Build bewerten. Scheduled man nun dieses Skript, würde die Automatisierung erreicht werden.

Beide der vorgestellten Möglichkeiten würden den Anforderungen einer Automatisierung für die Nightly Test Builds genügen und stellen eine subjektive Entscheidung des Benutzers dar.

6.2.2 Ermittlung der Kenngrößen

In diesem Abschnitt wird erläutert wie die ausgewählten Kenngrößen aus Kapitel 5 innerhalb der Implementierung ermittelt werden.

6.2.2.1 Instruktionslänge

Die Instruktionslänge beschreibt die Anzahl der ausführenden Anweisungen pro Anfrage oder Programm.

Hierfür wird aufgrund seiner Simulationsfähigkeit das Plugin Callgrind von **Valgrind** benutzt.

Valgrind

Die Software-Suite Valgrind wurde zum Debuggen und zur Messung von Laufzeitverhalten von Programmen auf Basis von GNU/Linux entwickelt. Mithilfe von Valgrind ist es möglich Speicheranalysen, Codeanalysen und Simulationen bezüglich des Verhaltens zur Laufzeit zu ermitteln.

Durch die Speicheranalyse lassen sich Fehler bei der Initialisierung von Speicherzellen und vor allem Speicherlecks und noch viel wichtiger Pufferüberläufe feststellen.

Valgrind ist nicht als monolithisches Tool zu verstehen, sondern als ein Kern mit dazugehörigen Plug-Ins. Innerhalb des Kerns ist die Basisfunktionalität realisiert, während in den Plug-Ins die eigentliche Funktionalität implementiert ist.

Der Kern von Valgrind ist hierbei als virtuelle Maschine realisiert. Das bedeutet, dass kein Programm, welches Valgrind zur Überprüfung gegeben wurde, direkt auf der Host-CPU läuft, sondern auf der CPU der virtuellen Maschine von Valgrind. Dabei bedient sich Valgrind einer Technik die besonders durch die Java Virtual Machine an Beliebtheit gewonnen hat, dem Bytecode. Der Bytecode, der bei Valgrind UCODE heißt, wird durch eine JIT (Just-In-Time Kompilierung) erzeugt.

Mithilfe des Bytecodes können die einzelnen Plug-Ins Veränderungen bezüglich des Speichers und des Ablaufes festlegen. Erst nachdem die Plug-Ins fertig sind und ihre Veränderungen abgeschlossen haben, übersetzt Valgrind das zu überwachende Programm in Maschinencode und führt diesen dann aus.

Dies stellt einen kleinen Overhead dar zum „normalen“ Debuggprozess, allerdings ermöglicht dies, dass ein Debuggen von Programmen stattfinden kann dessen Quellcodes closed-Source sind oder dessen Quellcodes einfach nicht verfügbar oder relevant sind.

Die wichtigsten, oder am meisten benutzten Plug-Ins für Valgrind sind:

- Memcheck
- Callgrind
- Helgrind

Memcheck erlaubt die Überwachung auf Speicherebene, so dass man Probleme durch die Beobachtung der Speicherausnutzung feststellen kann.

Callgrind erstellt einen ausführlichen Report über die Laufzeit eines Programms, das bedeutet im einfachsten Fall, dass alle Anweisungen des Programms mit ihrer jeweiligen Laufzeit angegeben werden. Diese Aussage über die Laufzeit kann allgemein gedeutet werden, da nicht die Ausführungszeit, sondern die CPU-Takte ermittelt werden, wodurch es möglich wird, die Ergebnisse des gleichen Programms zu Aussagen auf verschiedenen Systemen zu formulieren.

Helgrind ermöglicht kritische Blöcke innerhalb eines Programms zu finden und zu bewerten. Dies ist ein großes Hilfsmittel bei Thread-Programmierung.

Im Zuge dieser Arbeit wird das Plug-In Callgrind verwendet, um Aussagen über die Instruktionslänge einer CIM-Anfrage auf den OpenPegasus CIM-Servers zu ermöglichen.

Listing 1 zeigt ein Auszug eines Callgrind-Reports.

```
……
cfn=(118)
calls=1 0
0 20
0 4


fn=(64)
0 2645
cfn=(66)
calls=529 0
0 20328
0 1058


totals: 11868208
```

Listing 1 Beispiel eines Callgrind Reports

Der in Listing 1 gezeigte Ausschnitt zeigt mit der Zeile „totals: 11868208“ die gesamte Instruktionslänge für das getestete Programm.

Die Ermittlung der Instruktionslänge einer Anfrage muss zweistufig erfolgen. Die erste Stufe misst die Instruktionen die nötig sind, um den OpenPegasus CIM-Server hochzufahren und im Leerlauf zu betreiben, und stellt die Basiszahl für die Instruktionen dar. Im zweiten Schritt wird mit jeder Anfrage die Instruktionslänge erneut gemessen und von der Basiszahl subtrahiert, um die Instruktionslänge der jeweiligen Anfrage zu ermitteln.

6.2.2.2 Speicherauslastung

MemInfo

Meminfo ist ein Python-Skript dass entwickelt worden ist um die Speicherauslastung bei Multi-User-Systemen zu ermitteln.

Listing 2 zeigt eine Ausgabe von meminfo.py

```
……
Memory usage by processes with same names:
-----------------------------------------
            CMD COUNT USER-TIME SYS-TIME MEM-TOTAL
      konqueror     4    48m24s    5m49s 140.75 MiB
           Xorg     1  7h35m27s 1h15m42s 124.61 MiB
      amarokapp     2     9m36s      48s 96.12 MiB
mozilla-thunder     2     5m16s      19s 58.08 MiB
        konsole     7     1m07s       8s 19.57 MiB
           gaim     1    40m11s    1m45s 10.22 MiB
         kspread     1        7s       0s 6.48 MiB
            bash     9        4s       4s 5.87 MiB
            kate     1        9s       0s 5.73 MiB
       kdesktop     2     1m58s      45s 5.00 MiB
          kicker     2    35m17s    7m51s 4.74 MiB
           xchat     1    15m45s    2m02s 4.70 MiB
            kwin     2     8m17s    1m44s 4.28 MiB
 vmware-serverd     1     1m07s      15s 3.75 MiB
            kded     1     2m30s      15s 2.86 MiB
   kio_uiserver     1       37s       3s 2.00 MiB
         mcedit     2        5s       0s 1.93 MiB
         python     2        4s       0s 1.89 MiB
 adept_notifier     1       59s       3s 1.83 MiB
       khotkeys     2     1m53s      30s 1.66 MiB
        knotify     1        5s       1s 1.37 MiB
            ssh     2       48s      22s 1.05 MiB
           Rest    56     2m30s    4m34s 12.29 MiB
```

Listing 2 eine Beispielausgabe von meminfo.py

Hier ist leicht zu erkennen welcher Prozess wie viel Speicher benutzt. Diese Auswertung muss allerdings genauso wie der Auswertung der Valgrind-Ergebnisse zweistufig erfolgen.

In der ersten Stufe muss die Speicherauslastung im Leerlauf ermittelt werden, dieser Wert gilt wieder als Basiszahl. Im zweiten Schritt wird die Speicherauslastung nach jeder Anfrage gemessen, und nochmals an Ende der Testanfragen. Damit soll gewährleistet werden, dass etwaige Speicherentladungen von OpenPegasus abgefangen werden.

6.2.2.3 External Throughput Rate (ETR)

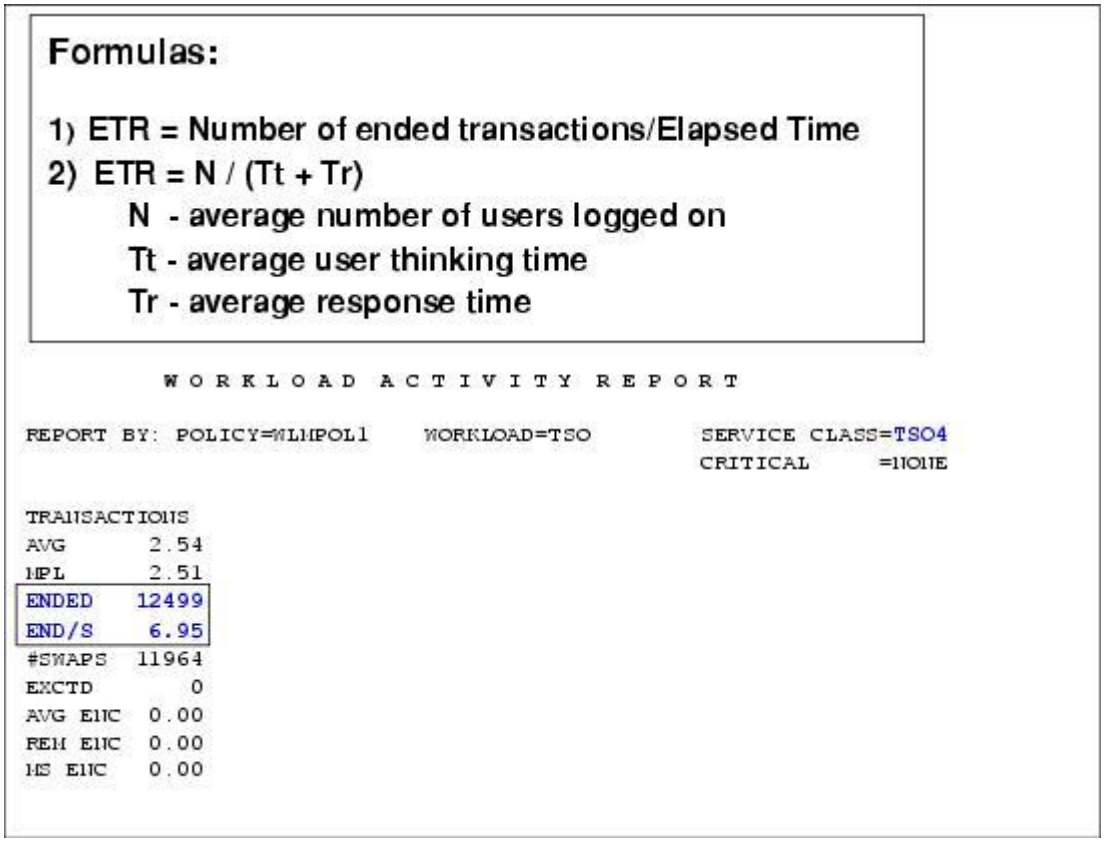

Abbildung 6.1 Herleitung der ETR-Funktion

Die ETR beschreibt die Anzahl der beendeten Transaktionen innerhalb einer bestimmten Zeitscheibe, die ETR wird auch als Transaktionsrate bezeichnet.

Allerdings muss die Formel angepasst werden damit sie dieser Arbeit genügt.

$$ETR=N/(Tt+Tr)$$

Formel 1 Die ETR-Funktion

Innerhalb dieser Implementierung wird angenommen dass nur ein User eingeloggt ist, daher gilt N=1. Desweiteren werden die Anfragen an den CIM-Server per Automatismus gestellt daher existiert keine Bedenkzeit für den User darum gilt Tt=0. Mit diesen Angaben ergibt sich folgende Formel für die ETR-Funktion innerhalb dieser Implementierung (s. Formel 2).

$$ETR=1/Tr$$

Formel 2 Die ETR-Funktion für die Implementierung

Die Ermittlung der Tr wird dadurch realisiert dass die Zeit gemessen wird zwischen Anfrage und dem Erhalt des Ergebnisses der Anfrage. Um etwaige Netzwerklatenzen ausschließen zu können, wird die Implementierung ebenfalls auf dem gleichen System ausgeführt wie der eigentliche CIM-Server.

6.2.3 Flussdiagramm der Implementierung

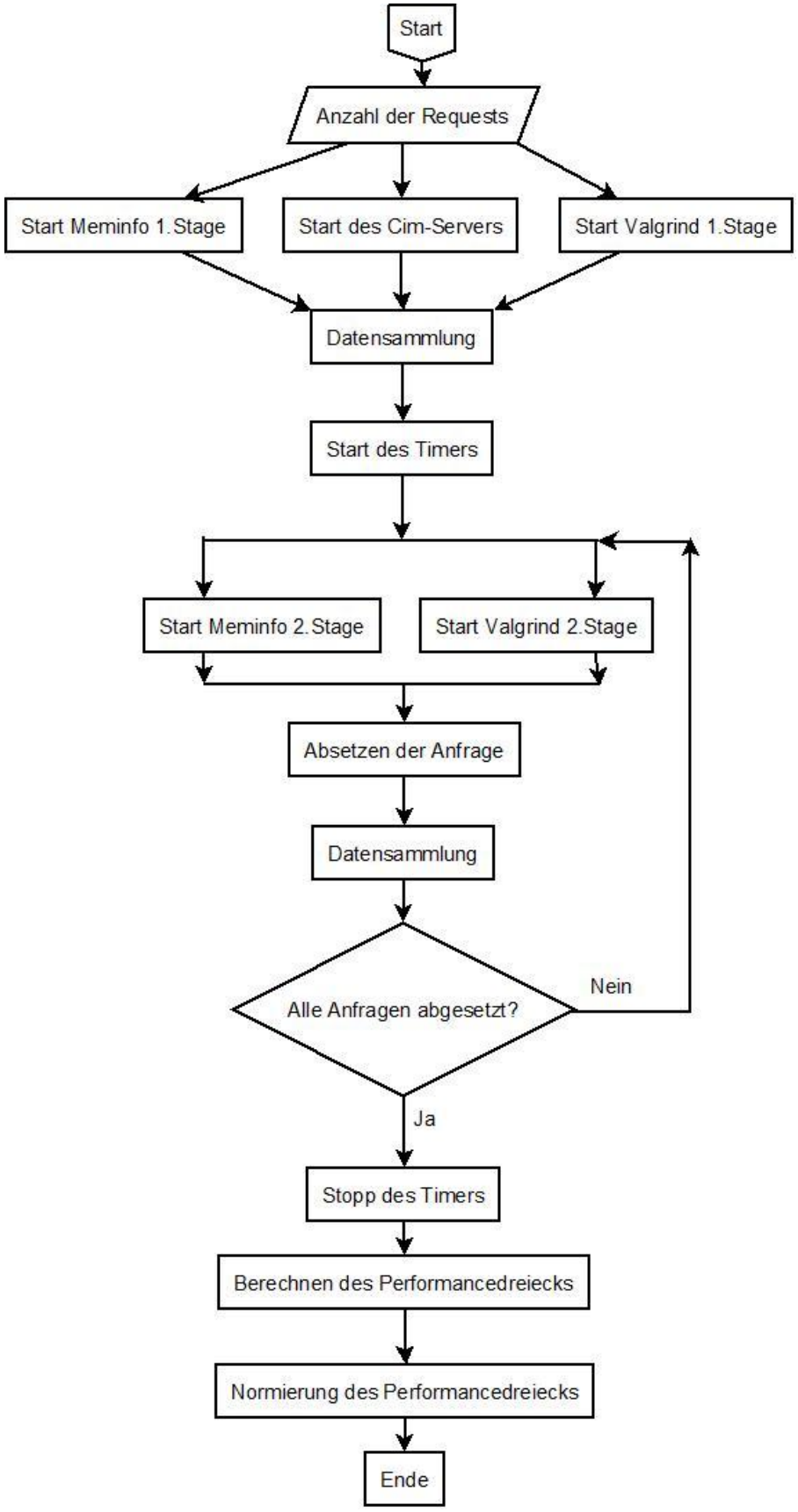

Abbildung 6.2 Flussdiagramm der Implementierung

Für die Berechnung des Performancedreiecks wird eine neue Datenstruktur „param“ angelegt (s. Listing 3).

```
struct param{
        int pos1;
        int pos2;
        int identifier;
        string marker;
        string fname;
        string stdfile;
        string ValgrindInfofile;
        double Triangle;
        double normTriangle;
        double ETR;
        double MemInfoValue;
        double ValgrindInfo;
        int countRequests;
        int comparingStates;
        int Step;
};
```

Listing 3 Die Datenstruktur „param“

Innerhalb der Datenstruktur werden mehrer Zwischenwerte sowie Ergebnisse abgelegt. So wird in der Variable „Triangle“ die Fläche des Dreiecks abgelegt, desweiteren werden die Werte von Valgrind und Meminfo in dieser Datenstruktur abgespeichert.

Somit bringt die Struktur „param“ stets alle Werte mit sich, die es benötigt.

Der Timer

Für die Zeitmessung wird die C++Funktion clock() angewandt, diese Messung ist notwendig für die Errechnung des ETR (s. Kapitel 6.2.2.3).

```
…
clock_t start, finish;
…
start=clock();
…
finish=clock();
…
double timediff = (static_cast<double>(finish - start)) / CLOCKS_PER_SEC * 1000.0;
…
```

Listing 4 eingesetzte Timer-Funktionen

Listing 4 beschreibt die eingesetzte Timer-Funktion. Die Variable „timediff“ wird noch über die Anzahl der Anfragen normiert, woraus die ETR-Berechnung resultiert.

Datensammlung

In dem Prozess Datensammlung werden alle Zwischenergebnisse in der Struktur „param“ abgelegt.

Starten des CIM-Servers und das Absetzen einer Anfrage

Der Cim-Server wird mit dem Aufruf „start cimserver“ innerhalb der Konsole. Damit der Aufruf direkt unter C++ erfolgen kann muss man ihn mit einem „system()“ kapseln.

```
System(„./start cimserver“);
```

Listing 5 Start des CIM-Servers

```
#include <Pegasus/Common/Config.h>

#include <Pegasus/Client/CIMClient.h>


PEGASUS_USING_PEGASUS;

PEGASUS_USING_STD;

…

Array<CIMInstance>      cimInstances;

 CIMClient              client;

…

client.connectLocal();

…

 cimInstances = client.enumerateInstances(

                        NAMESPACE,

                        CLASSNAME,

                        deepInheritance,

                        localOnly,

                        includeQualifiers,

                        includeClassOrigin);
```

Listing 6 Absetzen einer CIM-Anfrage

Mit dem Aufruf „client.connectLocal();“ verbindet sich der CIM-Client an den lokal laufenden CIM-Server. Die Authentifizierung wird mithilfe der Logindaten durchgeführt.

6.2.4 Errechnung des Performancedreiecks

6.2.4.1 Satz des Heron

Wie in Kapitel 6.1 beschrieben wird für die Flächenberechnung des **Performancedreiecks** der Satz von Heron angewendet.

Dazu muss allerdings noch über den Satz von Pythagoras die Länge der individuellen Seiten errechnen, Listing 7 zeigt den Code für den Satz des Pythagoras.

```
double CalcPyth(double a, double b){
        double d=sqrt(a*a + b*b);
        return d;
}
```

Listing 7 Satz des Pythagoras in C++

Damit können die Seitenlängen nun bestimmt werden und der Funktion für den Satz des Heron übergeben werden.

```
double CalcTriangle(param p){
        double a, b, c;
        a=CalcPyth(p.ETR,p.MemInfoValue);
        b=CalcPyth(p.ETR,p.ValgrindInfo);
        c=CalcPyth(p.MemInfoValue,p.ValgrindInfo);
        double d= sqrt((a+b+c)*(a+b-c)*(b+c-a)*(a+c-b))/4;
        return d;
}
```

Listing 8 Satz des Heron in C++

6.2.5 Normierung des Dreiecks

Die Fläche des Dreiecks steht allerdings für eine Anzahl von N Anfragen, dieses Ergebnis muss noch auf eine 1 Anfrage normiert werden. Dies geschieht mit der in Listing 9 gezeigten Funktion.

```
double CalcNormTriangle(param p){
        double d = CalcTriangle(p)/p.countRequests;
        return d;
}
```

Listing 9 Normierung des Performancedreiecks

6.2.6 Ausgabe der Implementierung und Deutung

Aufgrund der Ergebnisse der Berechnungen für das Performancedreiecks lassen sich Bewertungen für einzelne Performancedreiecke erstellen (s. Listing 10).

```
Uebersicht
Step    ||Heute    ||Vortag  ||Veraenderungsgrad    ||Bewertung    |
1       ||3        ||6       ||0.5                  ||+            |
```

Listing 10 Ausgabe der Implementierung

Die Ausgabe vergleicht ein Performancedreieck mit einem vorherigen und errechnet den Veränderungsgrad. Dabei ist zu berücksichtigen, dass je kleiner die Bewertungsgröße für den jeweiligen Stand ist, desto besser die Performanceausnutzung ist. Analog bedeutet dies, dass ein kleiner Veränderungsgrad eine verbesserte Performanceausnutzung symbolisiert.

Desweiteren gibt sie eine Bewertung aufgrund dieser Ergebnisse ab, so bedeutet ein „+“ eine Verbesserung und analog ein „-“ eine Verschlechterung der Performance, eine „0“ bedeutet schlicht dass sich keine Veränderung in der Gesamtperformance ergeben hat. Dies kann bedeuten dass sich Verbesserungen einer Kenngröße einerseits durch Verschlechterung einer anderen ausgeglichen haben.

Wie bereits in Kapitel 4 beschrieben, ist es wichtig in einem hochdynamischen Feld wie beispielsweise beim Open-Source-Coding die Weiterentwicklung bezüglich der Performance stets im Auge zu behalten. Dies wird durch den Vergleich zweier Stände ermöglicht, wodurch die Performance von heute mit der Performance von gestern verglichen werden können. Damit ist es möglich, eine ständige Überwachung der Gesamtperformance zu gewährleisten.

Dies genügt den Anforderungen, die in der Motivation dieser Arbeit erwähnt worden sind, da es möglich ist, Performance zu überwachen und objektiv zu vergleichen, um die Qualitätssicherung zu gewährleisten.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Die Problematik bei dieser Arbeit war es, die Performance-Qualitätssicherung bei der Entwicklung von Serversoftware in einem hochdynamischen Feld zu überwachen, um Performanceeinbußen interpretieren zu können.

Zu diesem Zweck wurde ein Verfahren vorgestellt, mit dessen Hilfe eine Bewertung der Performance automatisiert möglich ist und Änderungen der Performance hinsichtlich der Codebasis interpretierbar sind.

Für diesen Zweck wurde im ersten Schritt der Begriff Gesamtleistung definiert. Diese stellt die Relation zwischen der Limitation des vorhandenen Systems und der Software-Performance her. Und in einem zweiten Schritt diese Gesamtleistung zu ermitteln, wurden Leistungskenngrößen bezüglich Serverleistung analysiert und nach zuvor beschriebenen Kriterien ausgesucht. Diese Kriterien waren sehr wichtig, da dadurch die Signifikanz der Bewertungsgröße rechtfertigt wurde. Nachdem die Messgrößen ausgewählt wurden, wurde ein geometrisches Dreieck erstellt, dessen Flächeninhalt die Gesamtleistung darstellt. Somit wurde ermöglicht, dass die Gesamtleistung zwischen verschiedenen Softwareständen verglichen werden kann und eine Änderung unabhängig von der Qualifizierung des Benutzers zu interpretieren ist.

Bei einer ermittelten Verschlechterung der Performance ist man in der Lage, durch Analyse der Kenngröße, die schlechter wurde, auf die Codebasis zu deuten.

Das ist ein sehr wichtiger, wenn auch nicht weit verbreiteter, Schritt zur Qualitätssicherung von Software.

7.2 Ausblick

Eine Erweiterungsmöglichkeit aufbauend auf dieser Arbeit wäre die Entwicklung eines modularen Testbeds, bei dem der Benutzer in der Lage wäre, sich sämtliche Kenngrößen zusammenzustellen. Der Zugriff könnte auf jede Kenngröße vorhanden sein und durch eine grafische Oberfläche könnte man sich die Kenngrößen aussuchen, die man momentan braucht. Und zu einem anderen Zeitpunkt, bei dem man andere Parameter braucht, könnte man die einbezogenen Kenngrößen ändern.

Dies hätte den großen Vorteil, dass die Performanceüberwachung nicht an einen bestimmten Anwendungsfall gebunden wäre, sondern dynamisch angepasst werden könnte.

Ein weiterer Vorteil in diesem Zusammenhang wäre die Möglichkeit der Portierung, weil man sich die Definition der Gesamtlösung bzw. Bewertungsgröße jederzeit neu zusammenstellen könnte.

Abkürzungsverzeichnis

CIM…….. Common Information Model

DMTF……..... Distributed Management Task Force

WBEM……..... Web Based Enterprise Management

PAS……..... publicly available specification

MOF……..... Managed Object Format

XML……. Extensible Markup Language

CIMOM…… CIM Object Manager

KPI……Key Performance Indicator

TPC……...Transaction Processing Performance Council

ETR…….. External Throughput Rate

ITR……..... Internal Throughput Rate

SNMP……... Simple Network Management Protocol

HTTP…… Hypertext Transfer Protocol

DMI…….. Desktop Management Interface

UML……. Unified Modeling Language

JIT……Just-in-Time

ITIL……...IT Infrastructure Library

LDAP……Lightweight Directory Access Protocol

SPEC…….Standard Performance Evaluation Corporation

Abbildungsverzeichnis

Literaturverzeichnis

[Al94] Alper, M.: *Professionelle Softwaretests. Praxis der Qualitätsoptimierung kommerzieller Software*. Vieweg+Teubner 1994. ISBN: 978-3528054540

[Ba98] Balzert, H.: *Lehrbuch der Software-Technik*. Band 2. Spektrum 1998. ISBN: 978-3827411617

[Bi04] Bisson, S.: Saving software in distress. http://www.appdevadvisor.co.uk/Downloads/ADA7_1/Bisson7_1.pdf. Abrufdatum: 20.08.2008

[BR03] Bennicke, M.; Rust, H.: *Software-Produktanalyse*. 2003. http://wwwsst.informatik.tu-cottbus.de/~db/doc/SoftwareEngineering/SoftwareProduktanalyse_Metriken_QA.pdf. Abrufdatum: 20.08.2008

[Ce03] Cecchet, E. et. al.: *Performance Comparison of Middleware Architecturesfor Generating Dynamic Web Content*. 2003.http://rubis.objectweb.org/download/Middleware-2003.pdf. Abrufdatum: 20.08.2008

[Fe03] Fey, D.: *Leistungsbewertung*. 2003. http://www2.informatik.uni-jena.de/~fey/ra1ws03/folien/ra1_kap5_1bis36.pdf. Abrufdatum: 20.08.2008.

[Fr97] Frühauf, K.; Ludewig, J.; Sandmayr, H.: *Software-Prüfung. Eine Anleitung zum Test und zur Inspektion*. Vdf Hochschulverlag 1997. ISBN: 978-3728130594

[LZ99] o. V.: *Bewertung der Leistung und Zuverlässigkeit*. 1999. http://www3.informatik.uni-erlangen.de/Lehre/OTRS_I/WS1999/folien/OTR16H.pdf. Abrufdatum: 20.08.2008

[MF03] Müller-Clostermann, B.; Flüs, C.: *Kapazitätsplanung und Leistungsbewertung*. Kovac 2003. ISBN: 978-3830036029

[My91] Myers, G.: *Methodisches Testen von Programmen*. 4. Aufl., Oldenbourg 1991. ISBN: 978-3486256345

[Nf03] o. V.: *Nicht-funktionale Testmethoden*. 2003. http://qse.ifs.tuwien.ac.at/courses/QS/ws0304/10S_Nonfunkt_wid_20031126.pdf. Abrufdatum: 20.08.2008

[Pt04] o. V.: *Performance test tools* http://www.opensourcetesting.org/performance.php Abrufdatum: 20.08.2008

[Ta02] Thaller, E.: *Software-Test. Verifikation und Validation*. Heise 2002. ISBN: 978-3882291988

[Tr93] Trauboth, H.: *Software-Qualitätssicherung. Konstruktive und analytische Maßnahmen*. Hanser 1993. ISBN: 978-3446213678

[IBM05] Rogers P., Bari P., Munchen S., Oliveira J.N., Salla A., Sokal V., IBM Redbook: *ABCs of z/OS System Programming Volume 11*, 2005

[CIM22] *Common Information Model (CIM) Version 2.2*. Specification, Distributed Management Task Force, June 1999. http://www.dmtf.org/standards/cim_spec_v22/. Abrufdatum: 20.08.2008

[WBEM05] Hobbs, Chris: *A Practical Approach to WBEM/CIM Management*, CRC Press Inc 2004, ISBN: 978-0849323065

[CIM11] *Specification for CIM Operations over HTTP, Version 1.1*. Specification, Distributed Management Task Force, May 2002. http://www.dmtf.org/standards/documents/WBEM/DSP200.html. Abrufdatum: 20.08.2008

[TPC94] Transaction Processing Performance Council (TPC): *TPC Benchmark A, Standard Specification*. Revision 2.0, 1994.

[LIG02] Liggesmeyer P.: *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. Spektrum 2002. ISBN: 978-3827411181

[TPC] *Transaction Processing Performance Council*, http://www.tpc.org. Abrufdatum: 20.08.2008

[KPI] *Key Performance Indicator Library*, http://kpilibrary.com/. Abrufdatum: 20.08.2008

[NAG] *Nagios Homepage*, http://www.nagios.org. Abrufdatum: 20.08.2008

[SPEC05] *SPECjbb2005 Whitepaper*, www.spec.org/jbb2005/docs/WhitePaper.html, Abrufdatum: 20.08.2008

[Open] *OpenPegasus Projekt Homepage*, http://www.openpegasus.org. Abrufdatum: 20.08.2008

Anhang A

Inhalt der beigefügten CD

Auf der beiliegenden CD gibt es folgende Verzeichnisstruktur:

Performance.cpp: Das ist die vorgestellte Implementierung zur Performanceüberwachnung in der Programmiersprache C++

Literaturreferenzen: Alle Literaturreferenzen sind an dieser Stelle elektronisch nochmal verfügbar, inklusive Internetseiten.

Tugan.pdf: Die vorliegende Diplomarbeit in pdf-Format. Zum Öffnen wird Acrobat Reader benötigt, welches unter www.acrobat.com kostenlos zum Download bereitsteht.