

WIRTSCHAFTSUNIVERSITÄT WIEN

BAKKALAUREATSARBEIT

Titel der Bakkalaureatsarbeit:
Datenbank-Monitoring

Englischer Titel der Bakkalaureatsarbeit:
Database-Monitoring

VerfasserIn: Heinz-Peter Lang
Matrikel-Nr.: 0450631
Studienrichtung: Wirtschaftsinformatik
Kurs: 0996 IT-Praktikum mit Bachelorarbeit
Textsprache: Deutsch
BetreuerIn: Dipl.-Ing. Dr. Albert Weichselbraun
Unternehmen/Betreuer: IDIOM

Ich versichere:

dass ich die Bakkalaureatsarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe. dass ich die

Ausarbeitung zu dem obigen Thema bisher weder im In- noch im Ausland (einer Beurteilerin/ einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe. dass diese Arbeit mit der vom Betreuer beurteilten Arbeit übereinstimmt.

Datum

Unterschrift

Abstract

Die Anforderungen des IDIOM Projekts und dem damit verbundenen weblyzard, ein Instrument zur Analysierung von Webseiten, und die dahinterstehenden Projekte erfordern ein Mindestmaß an Datenqualität.

Um fehlerhafte oder nicht vorhandene Daten aufzuspüren, sind die Nutzer gezwungen, manuell nach diesen Fehlern zu suchen. Die Lösung dieses Problems wird durch die Erstellung eines Datenbank-Monitors gelöst, welcher die vorhandenen Daten analysiert. Aus der Feststellung der Qualität im Vergleich zu den bereits bestehenden Daten ergibt sich der Ansatzpunkt des Datenbank-Monitors.

Inhaltsverzeichnis

1	Projektplanung	7
1.1	Das Unternehmen [Id06]	7
1.2	Motivation	8
1.3	Anforderungen an das Projekt	10
1.3.1	Hierarchie des Projektes	10
1.3.2	Metriken	10
1.3.3	Visualisierung der Kennzahlen	11
1.3.4	Notifikationen	11
1.4	Liste der Meilensteine	12
1.5	Projektstrukturplan	12
1.6	Gantt-Chart	12
1.7	Analyse	15
1.8	Systemdesign	16
2	Theoretischer Hintergrund	17
2.1	Anforderungen an Monitoringsysteme	17
2.2	Probleme beim Monitoring	18
2.2.1	Vier-Ebenen Architektur für die MCS Entwicklung	19
2.2.2	Ein allgemeines Model für das Design von Korrelation Agenten: die ST-Logik	21
2.3	Verbreitete kommerzielle Systeme	21
2.3.1	Simple Network Traffic Protocol	21
2.3.2	Multi Router Traffic Grapher	23
3	Praktische Umsetzung	24
3.1	Datenbankdesign	24

3.2	Aufbereitung der Daten	25
3.3	Bereitstellung von grundlegenden Funktionen	27
3.3.1	Klasse Database	29
3.3.2	getFileName und getPathFileName	30
3.3.3	Error-logging	30
3.4	Visualisierung der Metriken	30
3.4.1	Klasse Metrics	31
3.4.2	Speicherung der Bilder	34
3.4.3	Ausgabe für das Webinterface	36
3.4.4	Statistiken	39
3.5	Umsetzung der Notifications	40
3.5.1	Monitoring in den PL/SQL-Funktionen	40
3.5.2	Monitoring bei der Erstellung der Bilder	41
3.5.3	Notification-Mailer	41
3.6	Administration	42
3.7	Installation des Database-Monitors	42
3.7.1	Vorbereitung des Servers	42
3.7.2	Vorbereitung der Datenbank	46
3.7.3	Dateien	48
3.7.4	Anpassung bei PHP	48
3.7.5	Erstellung eines Cronjobs	49
3.8	Manuelle Ausführung	49

4 Zusammenfassung 51

Abbildungsverzeichnis

1.1	Projektstrukturplan	13
1.2	Gantt-Chart	14
1.3	Use-Case Diagramm	15
1.4	Komponenten Diagramm	16
2.1	4-Ebenen-Architektur des Monitor Control Systems [Bandini05]	20
3.1	Relationales Datenmodell	26
3.2	Aktivitätsdiagramm - PL/SQL	28
3.3	Übersicht der Klassen	29
3.4	Aktivitätsdiagramm createImages()	33
3.5	Ausgabe des Balkendiagramms im Webinterface	35
3.6	Größenunterschied JPG und PNG	36
3.7	Webinterface: Ausgabe der Balkendiagrammen	37
3.8	Ausgabe der Statistiken im Webinterface	39
3.9	Verzeichnisstruktur	43
3.10	Startseite Webinterface	44

Tabellenverzeichnis

1.1	Liste der Meilensteine	12
3.1	Konfiguration	45
3.2	Parameter measureData.php und createImages.php	50
3.3	Parameter createImages.php	50

1 Projektplanung

In diesem Abschnitt werden das Unternehmen, die Rahmenbedingungen und die ersten Überlegungen zu diesem Projekt genauer vorgestellt.

1.1 Das Unternehmen [Id06]

Der Datenbankmonitor wird für das IDIOM (Information Diffusion across Interactive Online Media) Projekt entwickelt. Finanziert wird es von den FIT-IT Semantic Systems [FIT]. Dabei handelt es sich um ein Programm des Bundesministeriums für Verkehr, Innovation und Technologie, zur Förderung der Forschung anspruchsvoller Innovationen und Technologieentwicklung auf dem Gebiet der Informationstechnologie.

Linguisten definieren das Wort 'Idiom' als einen Ausdruck, dessen Bedeutung sich von derjenigen einzelner Wörter unterscheidet. Ähnlich verhält es sich bei der Untersuchung von Informationsdiffusion, deren Erkenntnisse sich nicht von einzelnen Netzwerkelementen ableiten lassen.

Trotz des wachsenden Forschungsinteresses wird das 'Web 2.0' noch immer von Prototypen und Mash-ups dominiert. Zur gleichen Zeit treten bei derzeitigen Media Monitoring und Corporate Knowledge Management Projekten noch verschiedene Mängeln auf, wie das Fehlen eines analytischen Frameworks, der Fokussierung auf ein spezielles Medium oder der Nichtbeachtung der dualen Nutzerrolle, i.e. als Informationsproduzent und -nutzer. IDIOM versucht diese Lücken zu schließen, indem fundamentale Mechanismen der Informationsverteilung über interaktive Medien aufgedeckt werden sollen

und verschiedene Formen von Services die Schaffung und Konsumation von elektronischem Content analysieren soll.

Folgende Ziele werden von IDIOM verfolgt:

- Die Erforschung von neuen Interfaces um elektronischen Content zu erstellen, zu verwenden und zu analysieren.
- Die Aufdeckung der grundsätzlichen Informationsdiffusion über elektronische Medien.

Neueste Fortschritte in gemeinschaftlichen Webtechnologien wurden durch positive Netzwerkeffekte und der Bündelung von kollektivem Wissen durch die Konsumenten selbst und algorithmischem Datenmanagement andererseits hervorgebracht. Dadurch ist es möglich, rascher Informationen über Wikis, Blogs, Webseiten und andere direkte Kommunikationskanäle zwischen den Mitgliedern der Online Communities auszubreiten. IDIOM unterstützt und erforscht die elektronische Interaktivität anhand einer generischen, service-orientierten Architektur. Diese Architektur enthält ontologisch-basierte Tools, um zusammenhängende Informationsräume, ein Framework um die Contentdiffusion und Interaktions-Patterns zwischen mehreren Räumen zu analysieren, zu bauen und zu erhalten. Außerdem enthält die Architektur eine Interfacetechnologie, die es den Nutzern ermöglicht zwischen semantischen und geographischen Topologien zu wechseln. IDIOM führt eine radikale neue Interface-Metapher namens Knowledge Planet ein. Diese soll der neuen Generation von geobrowsing Plattformen, wie NASA World Wind oder Google Earth, den Weg bereiten um als Front-End für Semantic Services zu dienen.

1.2 Motivation

Ein zentraler Bestandteil des IDIOM Projektes ist weblyzard[WL08]. Wie der Name vielleicht erahnen lässt, handelt es sich dabei um einen Webseiten-

Analysen, welche automatisch Struktur und Inhalt von Webseiten analysiert.

Auf Basis dieser Daten sind einige weitere Projekte hervorgegangen, die auf EcoResearch [ER08] zu finden sind. Zwei davon sollen kurz näher vorgestellt werden, um ein Beispiel für den Einsatzzweck und Funktionsweise zu geben.

- Der *US Election 2008 Web Monitor* [EM08] überwacht internationale Nachrichtenseiten von den USA, Kanada, dem Vereinigten Königreich, Australien und Neuseeland, Umweltorganisationen, Fortune 1000 und auch 100 bekannte Blogs, die sich mit Politik beschäftigen. Auf Basis dieser Daten wird beispielsweise die Aufmerksamkeit für die Präsidentschaftskandidaten in den Medien gemessen.
- *Media Watch on Climate Change* [CC08] beschäftigt sich mit der Medienberichterstattung über Klimawandel und verwandten Themen. Das besondere ist hierbei, dass die Daten auch in einem Geobrowser dargestellt werden und somit ein dreidimensionaler Wissensplanet entsteht.

Diese beiden Projekte zeigen: Es ist für IDIOM von großem Interesse, dass die Daten korrekt vorliegen und die Datenbasis und damit die Ergebnisse dieser Projekte nicht durch fehlerhafte Daten verfälscht werden. Die Fehlerquellen sind vielfältig. Im Artikel [Raz02] wird zwischen Verbindungs-, Software-, Syntax- und semantischen Fehlern unterschieden. Verbindungsfehler treten meist durch unzureichende Bandbreite, Serverfehler oder Fehler in anderen Netzwerkelementen auf. Syntax- oder Formfehler können auftreten, wenn Änderungen in den Datenquellen durchgeführt wurden und die Daten dadurch nicht korrekt extrahiert werden können. Eine solche Änderung könnte auftreten, wenn beispielsweise die Strukturierung der Informationen geändert wird oder indem Services neu angeboten werden. Diese Arbeit wird sich nicht mit der Aufspürung von semantischen Fehler beschäftigen, wo zwar Daten extrahiert wurden, diese aber unbrauchbare Werte enthalten, sondern mit der Überprüfung der Quantität der gespiegelten Daten.

1.3 Anforderungen an das Projekt

In diesem Kapitel werden die Anforderungen aus dem Workpackage [Wo07] genauer analysiert und erste Überlegungen angestellt, wie diese zu lösen sind.

Ziel dieser Arbeit ist es ein Tool zu erstellen, welches die Parameter des Mirroringprozesses visualisiert und auf potentiell ungültige Samples hinweist.

1.3.1 Hierarchie des Projektes

Bevor näher auf die Details des Projektes eingegangen wird, ist ein kurzer Einblick in die Datenstruktur erforderlich.

Die Daten werden wie gefolgt gespeichert:

Webseiten, die überwacht und gespeichert werden, setzen sich aus mehreren Mirrordaten zusammen, welche die jeweiligen Daten darstellen. Diese Webseitendaten werden zu Samples zusammengefügt, die dann Webseiten eines gleichen Themas enthalten. Ein Beispiel dafür wäre *AT Medien*, welches Webpräsenzen aller österreichischen Tageszeitungen enthält.

1.3.2 Metriken

Für jede gespeicherte Webseite wird in der Datenbank eine Vielzahl von Parametern zusätzlich zum Content gespeichert. Diese können zum Beispiel Daten sein, wie gespeicherte Dokumente, Größe der gespeicherten Daten oder durchgeführte Analyseschritte. Aus diesen Felder werden Metriken erstellt, die es ermöglichen sollen fehlerhafte Daten zu finden.

Ein wichtiger Betrachtungspunkt bei der Lösung dieser Aufgabe wird mit Sicherheit die Performance sein. Mit über 7.000 überwachten Seiten ist der Datenbestand enorm, und durchgeführte Berechnungen benötigen dadurch eine höhere Rechenleistung. Aus diesem Grund werden die aggregierten

Daten gleich nach deren Berechnung in der Datenbank gespeichert und performance-dienliche Maßnahmen, wie beispielsweise der Verwendung der Procedural Language von PostgreSQL [PL08] oder Prepared Statements [PS08], angewandt.

Im Rahmen dieser Berechnung bzw. Aggregation der Daten soll auch gleich evaluiert werden, ob die Daten in der gewünschten Form vorliegen, ob gar keine Daten existieren oder die Daten nicht einem gewünschten Anforderungen (Minimum) entsprechen.

1.3.3 Visualisierung der Kennzahlen

Einen weiteren Punkt stellt die Visualisierung der Kennzahlen dar. Der Nutzer soll die Möglichkeit erhalten, über ein Webinterface sich die aufbereiteten Daten in Form von Balkendiagrammen anzeigen zu lassen. Dabei soll die Möglichkeit bestehen, Daten in unterschiedlichen Ebenen anzeigen zu lassen. Diese Ebenen entsprechen der unter Abschnitt 1.3.1 vorgestellten Datenstruktur. Die oberste Ebene bildet demnach alle Samples, die nächste Ebene die zugehörigen Webseiten zu den jeweiligen Samples und schließlich detaillierte Statistiken, wie beispielsweise Durchschnitt oder Standardabweichung der gespeicherten Dokumente, zu den Webseiten selbst.

Um einen besseren Überblick über die Daten zu erhalten, soll die Möglichkeit bestehen, bestimmte Parameter wie Jahr oder Samplemirkorhäufigkeit einzustellen, um nur ausgewählte Daten anzuzeigen.

1.3.4 Notifikationen

Zusätzlich zu dem Webinterface soll der Nutzer auch die Möglichkeit erhalten Informationen über fehlerhafte Daten per Mail zu erhalten. Dies wird dadurch erreicht, dass für Samples ein Minimum der vorhandenen Daten festgelegt werden kann. Hier wird besonders die Zuverlässigkeit der versandten E-Mails eine Rolle spielen, und dass nicht durch Fehlalarme eine Art

Tabelle 1.1: Liste der Meilensteine

Projektstart
1. Meilenstein Voruntersuchung abgeschlossen und Ergebnisse analysiert
2. Meilenstein Modellierung abgeschlossen
3. Meilenstein Implementierung abgeschlossen
4. Meilenstein Testphase abgeschlossen
voraussichtliches Projektende Dokumentation ist vollständig, System ist fertig und getestet

Spambot entsteht, und dadurch die Notifikationen keine Beachtung mehr finden.

1.4 Liste der Meilensteine

Die Liste der Meilensteine in Tabelle 1.1 hebt die wichtigsten Schritte des Projekts hervor.

1.5 Projektstrukturplan

Die Abbildung 1.1 zeigt den Projektstrukturplan, welcher die einzelnen Phasen des Projekts übersichtlich darstellt.

1.6 Gantt-Chart

Das in Abbildung 1.2 gezeigte Gantt-Chart beschreibt den zeitlichen Ablauf des Projekts.

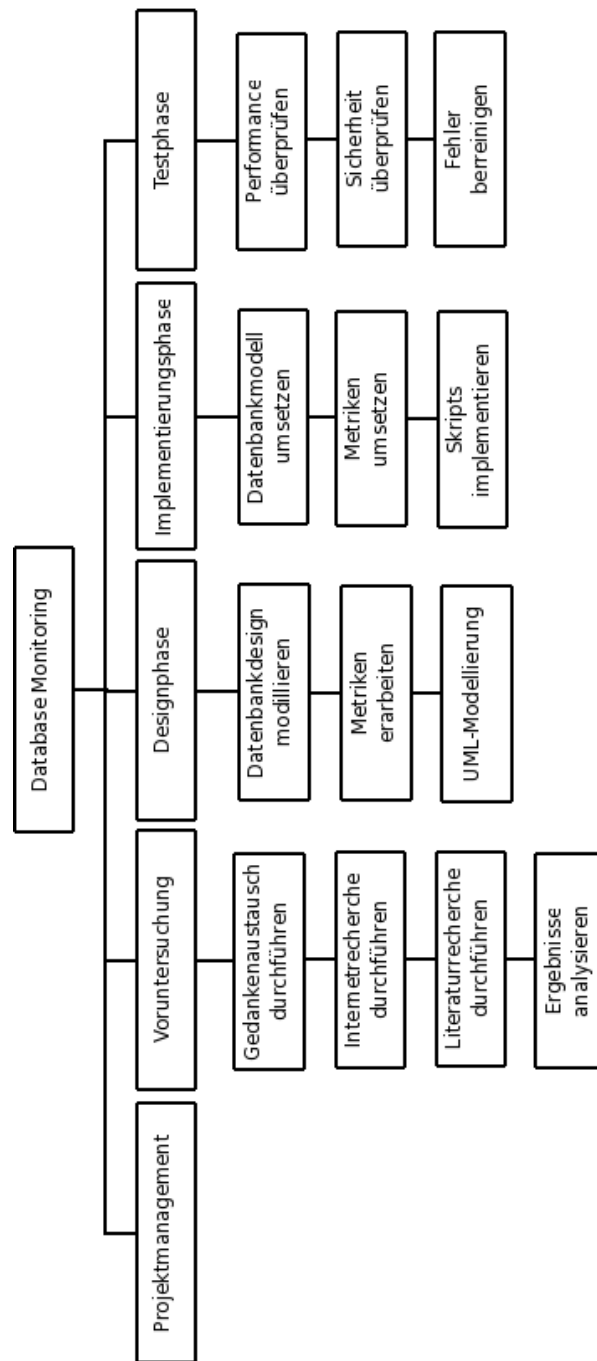


Abbildung 1.1: Projektstrukturplan

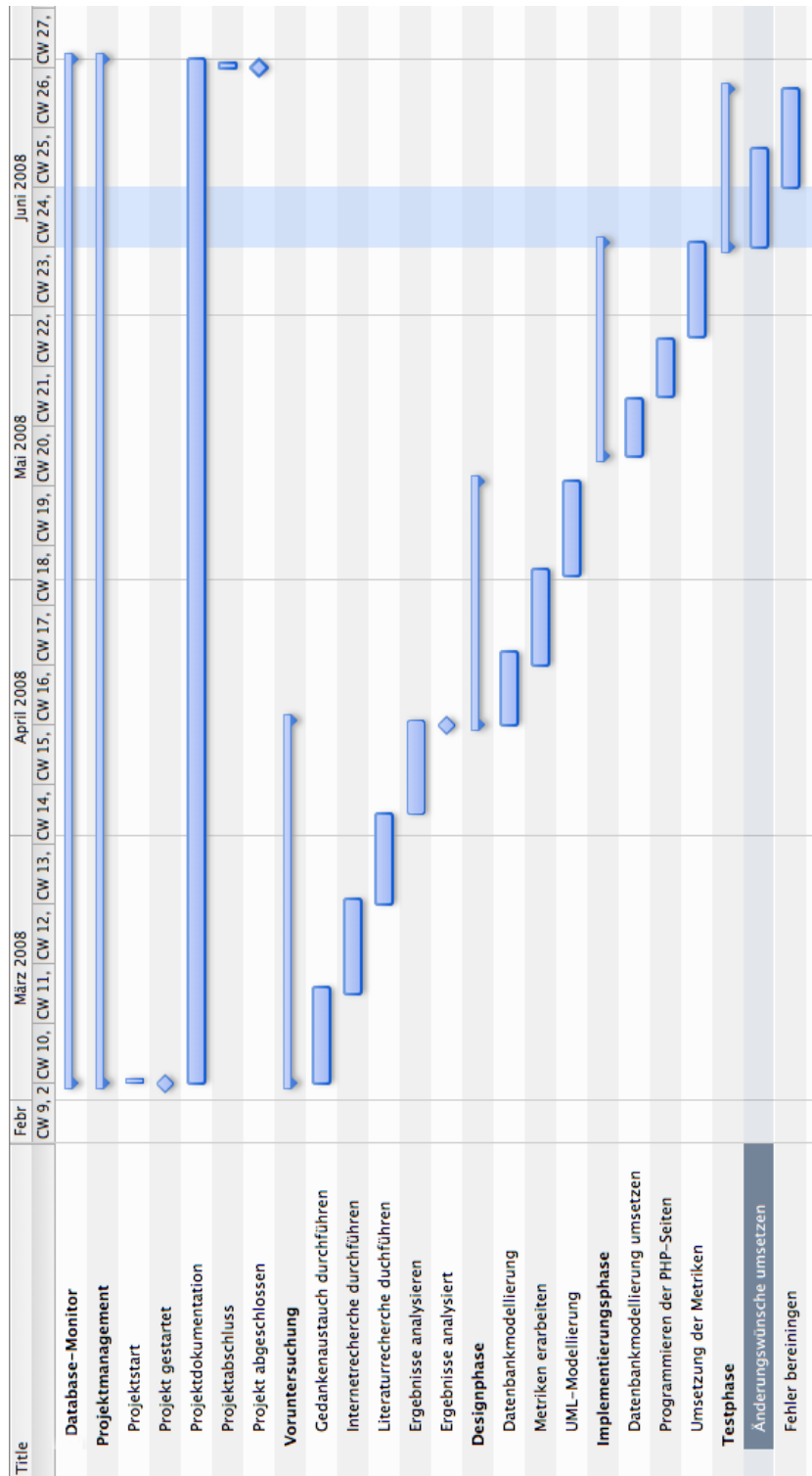


Abbildung 1.2: Gantt-Chart

1.7 Analyse

Zur Veranschaulichung der Analyse wird ein Use-Case Diagramm 1.3 erstellt.

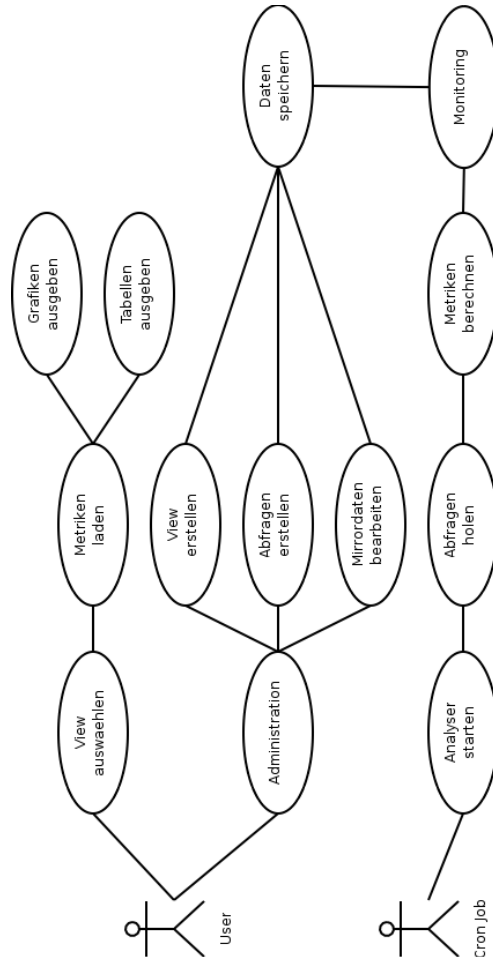


Abbildung 1.3: Use-Case Diagramm

1.8 Systemdesign

Das Systemdesign wird mit einem Komponenten Diagramm 1.4 dargestellt.

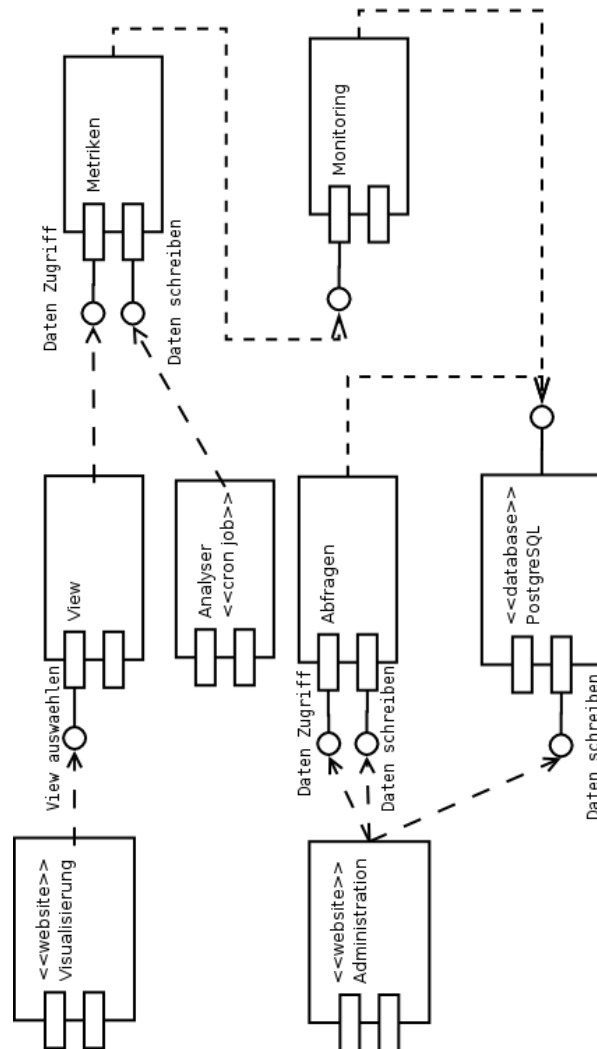


Abbildung 1.4: Komponenten Diagramm

2 Theoretischer Hintergrund

In diesem Abschnitt werden einige theoretische Hintergründe von Monitoringsystemen genauer analysiert. Im Abschnitt 2.1 werden die Anforderungen an ein Monitoringsystem beschrieben. Abschnitt 2.2 analysiert das Problem von fehlerhaften Sensoren genauer und empfiehlt eine Lösung. Im letzten Abschnitt 2.3 werden die Konzepte hinter den verbreiteten Lösungen Simple Network Messaging Protocol (SNMP) und Multi Router Traffic Graph [MRTG08] vorgestellt.

2.1 Anforderungen an Monitoringsysteme

Der Artikel [Nebel06] stellt die Anforderung an ein Monitoringsystem dar und bietet sich daher als Startpunkt der Betrachtung an.

Vor der Einführung eines Monitoringsystems müssen, wie bei jeder Software, die Anforderungen detailliert festgelegt werden. Dabei ergeben sich Fragestellungen, ob nur an einer Stelle oder an mehreren gemessen wird, ob und wie andere Monitoringprogramme zu unterstützen sind. Support für SNMP, Sinn und Zweck des Monitorings (aktueller Stand versus Kapazitätsplanung), Komplexität des Benutzerfrontends und die Formen von Alarmierungen. Solche Systeme können zwar dank Open Sourcelösungen, von denen einige Konzepte der beiden Vertreter SNMP und MRTG im Abschnitt 2.3 vorgestellt werden, preiswert beschafft werden können, internes Know-How ist aber besonders wichtig, da die Komplexität dieser Systeme erst bei der Konfiguration entsteht.

Zunächst sind die Basiskennzahlen festzulegen, wobei es sich um Parameter handelt, welche direkt vom System zurückgeliefert werden, wie beispielsweise RAM-Auslastung, CPU-Last oder die Festplattenauslastung eines Servers. Weiters ist auch die Nutzerwahrnehmung vom System zu erfassen: Die externen Kennzahlen. Bei diesen Daten ist zu überlegen, wie diese gespeichert werden, denn nur bei einem ausreichend großen Datenbestand können Abnormalitäten entdeckt werden.

Eine weitere Überlegung betrifft, die jeweiligen Ansichten der Datenzusammenfassung. Mit einer verständlichen Beschreibung wird dadurch die Akzeptanz des Monitoringsystems bei den Nutzern gesteigert.

Ist ein solches System erfolgreich produktiv eingesetzt worden, ist aufgrund der zahlreichen Schritte, die zwischen der Datenerfassung und graphisch aufbereiteter Ausgabe liegen, ein Plausibilitätstest durchzuführen. Dabei wird überprüft, ob die ermittelten Werte die richtigen Einheiten bzw. Größenordnungen aufweisen und im erwartenden Bereich liegen.

Letztendlich ist der Nutzer auch vom Ergebnis des Monitorings zu informiert. Dabei ist es wichtig, Fehler- und auch Erfolgsmeldungen zu geben. Für Fehlermeldungen sind im Vorfeld Schwellenwerte festzulegen, wodurch bei abweichendem Verhalten weitere vordefinierte Aktionen gestartet werden können. Erfolgsmeldungen sind ebenso wichtig wie Fehlermeldungen, da sie dem Nutzer ein Gefühl für die Verlässlichkeit bzw für die Häufigkeit und Dauer von Störungen geben.

2.2 Probleme beim Monitoring

Um die Nutzerakzeptanz für ein Monitoringsystem aufzubauen, ist es notwendig die Zuverlässigkeit des Systems zu steigern und die Häufigkeit von Fehlalarmen zu minimieren.

Häufig entstehen Fehlalarme durch fehlerhafte Sensoren. Eine Lösungsmöglichkeit wird im Artikel [\[Bandini05\]](#) vorgestellt, welcher die Einführung eines

Monitoring Control Systems (MCS) beschreibt. Das Hauptziel des MCS besteht darin, den Nutzer bei seinen Entscheidungen in Problemsituationen zu unterstützen.

Monitoringsysteme werden derzeit mit folgender Funktionalität bzw. folgenden Schichten realisiert:

- Überwachung (Observation)
Über verschiedene Sensoren werden Daten über die überwachte Situation gesammelt.
- Interpretation
Die gesammelten Daten werden ausgewertet, um zu entscheiden, ob es sich bei den Werten um eine anormale Situation handelt.
- Ausführung (Actuation)
Sollten Fehler aufgetreten sein, werden in dieser Schicht entsprechende Aktionen zur Problemlösung gestartet.

Bereits durch einen fehlerhaften Sensor kann es zu Falschmeldungen im gesamten System kommen. Um dies verhindern wurde diese Architektur erweitert, welche im nächsten Abschnitt genauer beschrieben wird.

2.2.1 Vier-Ebenen Architektur für die MCS Entwicklung

Um die Interpretation der gesammelten Daten zu verbessern, wird die zuvor vorgestellte Architektur um eine weitere Schicht, zwischen der Interpretationsschicht und der Ausführungsschicht, erweitert. Abbildung 2.1 zeigt die Architektur dieser vier Ebenen. Die neue Ebene setzt die beiden Ebenen in Beziehung zueinander. Diese Verbindung wird dadurch erreicht, dass alle Evaluierungen der Interpretationsschicht gesammelt werden und darauf basierend eine globale Sicht entsteht. Erst auf Basis aller Daten entscheidet der Korrelationsagent, welche Aktionen gestartet werden. Damit werden falsche Interpretationen der Fehler und in weiterer Folge falsche Entscheidungen vermieden.

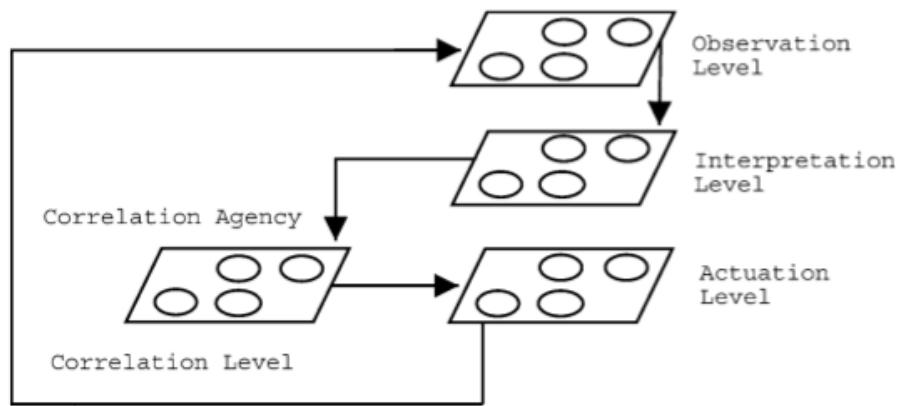


Abbildung 2.1: 4-Ebenen-Architektur des Monitor Control Systems [Bandini05]

Im Vorfeld ist es dazu notwendig ein Regelwerk festzulegen, welche Aktionen auf Basis der globalen Ansicht auszuführen und wie die einzelnen Sensoren miteinander zu verbinden sind.

Zur Lösung bietet sich ein auf Agenten basierendes Modell an. Dazu gibt es zwei Lösungsansätze. Die erste Möglichkeit besteht darin, die Probleme mittels gleichartiger Sensoren zu erkennen. Daraus ergibt sich, dass die Funktion der Architektur abhängig von ihren Agenten ist und vice versa. Als letztes könnten die Agencies auch durch Interaktion zu einem Ergebnis kommen.

Je nach Anforderung können die Agenten als reaktive Agenten oder *utility driven Agents* realisiert werden. Bei ersteren erhalten die Agenten immer einen Input, woraus eine Aktion folgt. Der Monitoringagent erhält Daten des Sensors und leitet diese an den Interpretationsagenten weiter. Dieser trifft auf Basis dieser Daten eine Entscheidung, durch deren Weiterleitung an den Korrelationsagenten über die auszuführende Aktion entschieden und zur Durchführung dem Ausführungsagenten übertragen wird. Die andere Lösungsmöglichkeit sind *Utility Driven Agents*. Bei diesen sammelt der Korrelationsagent alle Entscheidungen der Interpretationsebene, um auf Basis aller Daten eine Aktion an den Ausführungsagenten weiterzuleiten.

2.2.2 Ein allgemeines Model für das Design von Korrelation Agenten: die ST-Logik

Anormale Situationen werden meist bei ihrem Auftreten weiter gegeben, doch wenn diese an unterschiedlichen Orten und Zeiten auftreten, bedarf es einer neuen Vorgehensweise, welche im Artikel [Bandini05] als *Space/Time Logic* vorgestellt wird. Die im vorhergehenden Abschnitt beschriebene Architektur stellt deren Umsetzung dar.

Dafür ist es notwendig, dass der Korrelationsagent, dessen Ziel es ist, alle Daten zu sammeln, bestimmte Anforderungen erfüllt. Er muss den gesamten Bereich repräsentieren und auch das Verhalten der Agenten zueinander.

Um die Sensoren über Zeit und Raum zu verknüpfen, wird die Formalisierung der ST-Logik eingeführt. Dadurch wird Alarm ausgelöst, wenn beispielsweise an unterschiedlichen Orten zu gleicher Zeiten ein Fehler auftritt. Sich nur auf einen Sensor zu verlassen, kann schwerwiegende Folgen haben.

2.3 Verbreitete kommerzielle Systeme

Neben kommerziellen System gibt es auch eine Zahl von Opensource-Lösungen, um Monitoringaufgaben zu bewältigen. In diesem Abschnitt wird die Funktionsweise von SNMP und dem darauf aufbauenden MRTG analysiert, um ein Verständnis dafür zu erhalten, wie solche Systeme arbeiten.

2.3.1 Simple Network Traffic Protocol

Im Laufe der Zeit haben sich mehrere Versionen von SNMP entwickelt, wenn auch alle die selben Basiskonzepte verfolgen, welche bereits im Artikel [Stallings98] genauer vorgestellt wurden.

- Netzwerk Management Architektur

- Trap-Directed Polling
- Proxies

Die Netzwerk Management Architektur besteht aus Management Stationen, Management Agenten und Management Informationsbasen. Bei der Management Station handelt es sich um eine Einrichtung, welche ein Protokoll zum Austausch und eine Datenbank zum Sammeln der Daten von ihren überwachten Agenten zur Verfügung stellt. Management-Agenten sind beispielsweise Router oder Hubs, ausgestattet mit einer SNMP Agenten Software. Dadurch ist es möglich, dass die Management-Agenten von der Station gesteuert werden und mit ihr Daten austauschen können. Die Ressourcen der Agenten werden als Objekte bezeichnet, aus deren Sammlung die Management Information Basis (MIB) entsteht, welche der Station eine Anzahl von Accesspoints liefert. Durch Verarbeitung der MIBs wird schließlich das Monitoring durchgeführt.

Schlußendlich werden Agenten und Station durch die Netzwerk Management Protokoll Architektur verknüpft. Diese ist grundsätzlich als Application-Level Protokoll entworfen worden und daher oberhalb von UDP angesiedelt, wodurch eine Implementierung von SNMP, UDP und IP für jeden Agenten erforderlich wird.

Das Konzept des *Trap-Directed Polling* beseitigt die Ineffizienz der Administrationsabfrage vieler Agenten. Anstelle einer laufenden Agentenabfrage, werden diese während der Initialisierung der Management Station und in der Folge in unregelmäßigen Intervallen abgefragt. Unerwartete Ereignisse werden von den Agenten direkt gemeldet. Diese SNMP Nachrichten sind Auslöser für die Durchführung verschiedene Aktionen durch die Station, um das aufgetretene Problem genauer zu analysieren.

Die Notwendigkeit aller Agenten UDP und IP zu unterstützen, würde zu einer Limitierung der verwendeten Systeme führen. Um dies zu vermeiden wurde das Konzept der Proxies eingeführt. Für Geräte, welche aufgrund fehlender Unterstützung von UDP und IP nicht verwendet werden könnten, werden SNMP Agenten als Vermittler zur Verfügung gestellt. Aufgabe dieser

Proxies ist es, Anfragen der Geräte so zu konvertieren, dass die Geräte diese untereinander verarbeiten können.

2.3.2 Multi Router Traffic Grapher

Multi Router Traffic Grapher (MRTG) ist eine wichtige Monitoring Applikation um Netzwerkverkehr zu überwachen. Das Konzept wird in dem Artikel [Oetiker01] genauer erläutert. Die Idee zu diesem Programm entstand aus dem Interesse für die Vorgänge in den Netzwerkleitungen und potentielle Problemauslöser.

Ein wichtiges Konzept des MRTG verfolgt ist das sogenannte *Lossy data storage*. Darunter ist zu verstehen, dass vor allem im Netzwerkbereich nur aktuelle Daten relevant sind und den alten dadurch weniger Beachtung geschenkt werden muss. Dies wird dadurch gelöst, dass die Daten der letzten 24 Stunden in einem Intervall von fünf Minuten gespeichert werden, während die Daten der letzten zwei Wochen nur noch in einem Intervall von 30 Minuten gespeichert werden, bis schließlich noch ältere Daten aggregiert werden. Dadurch können Speicherplatzprobleme oder Abstürze, welche durch zu große Log-Dateien entstehen können, vermieden werden.

So hat sich MRTG zu einem sehr nützlichen Tool nicht nur für das Auffinden von Problemen in Netzwerken entwickelt, sondern auch zu einer verbesserten Kommunikation der Defizite in bestehenden Netzwerkinfrastrukturen mit den höheren Managementschichten.

3 Praktische Umsetzung

In folgendem Abschnitt ist die praktische Umsetzung des Datenbankmonitors zu finden.

3.1 Datenbankdesign

Als Datenbanksystem dient PostgreSQL. Abbildung 3.1 zeigt das entsprechende Datenbankmodell. Hier werden jene Tabellen gezeigt, welche für den Datenbankmonitor neu erstellt wurden. Im folgenden werden diese Tabellen, deren Verwendungszweck und deren Einbindung in die bestehende Datenbank genauer beschrieben.

Die Daten stammen aus dem View *vw_heinz*, welcher einen Ausschnitt der Mirror-Statistiken aus der webyzard Datenbank darstellt. Der View enthält Spalten aus den Tabellen *Company*, *Datasource-Sample*, *Sample*, *Mirror*, *Sample-Control* und *Sample-Control-Mirror-Types*.

In der Tabelle *Views* werden die Ansichten definiert, welche Felder als ID-Feld gelten und aus welcher Tabelle diese Daten stammen, beispielsweise *Sample* oder *Company*. Abfragen zu den Daten, genauer gesagt, welche Felder entsprechend ihren Typen aggregiert werden, werden in der Tabelle *Queries* eingetragen. Die Zuordnung der Queries zu einem View wird in der Tabelle *view_has_query* gespeichert.

Die Definition der Schwellenwerte wird in der Tabelle *Limits* vorgenommen. Dabei muss das zu überwachende Sample, ein Minimum in Prozent, wie viele Daten im Vergleich zu den bestehenden Daten vorliegen müssen, und eine

E-Mailadresse angegeben werden, an welche eine Notifikation zu schicken ist. Für Abweichungen von den Limitwerten im Laufe des Monitorings eines bestimmten Sample wird eine Zeile in der Tabelle *Notifications* hinzugefügt, um diese später zu verarbeiten.

Zwecks Vereinfachung der Anlage von neuen Abfragen und der Zuordnung an Views, besteht im Webinterface die Möglichkeit einer Dateneintragung. Details dazu sind im Kapitel 3.6 beschrieben.

Die wichtigste Relation stellt die Tabelle *Metrics*. Hier werden nach der Verarbeitung durch die PL-SQL-Funktionen 3.2 die aggregierten Daten gespeichert. Anders als bei MRTG werden die Daten nach einer gewissen Periode nicht gelöscht. Für den Datenbankmonitor kann es durchaus interessant sein, wenn die Daten länger im vollen Detailgrad gespeichert werden.

3.2 Aufbereitung der Daten

Für Aufbereitung und Aggregation der Daten werden PL/SQL-Funktionen [PL08] verwendet, welche in Abbildung 3.2 als Aktivitätsdiagramm zu finden sind. Bei diesem Diagramm wurde bewußt auf die Darstellung des Startpunktes und der Akteure verzichtet, da alle Funktionen direkt aufgerufen werden können und deswegen die Übersichtlichkeit darunter leiden würde.

Die Parameter zur Aggregation können frei gesetzt werden, um damit nur tatsächlich benötigte Ausschnitte neu zu berechnen und damit die Ausführungsgeschwindigkeit zu reduzieren. Es ist möglich die Variablen *Sample-ID*, *View*, *Query* und das Datum des letzten Updates zu definieren. Je nach Vorhandensein der Parameter wird die entsprechende Funktion in der Datenbank aufgerufen, um weitere benötigte Parameter abzufragen. Als Unterstützung zu diesen Funktionen können Methoden eingesetzt werden, um das letzte Update aller Daten oder für eine ausgewählte Query, View oder Sample-ID, zu erhalten. Die Unterscheidung ist notwendig, da ältere Daten sonst, beispielsweise bei der Berechnung eines bestimmten Views, keine Beachtung fänden. Die Funktion *measure_sample* mit den Parametern

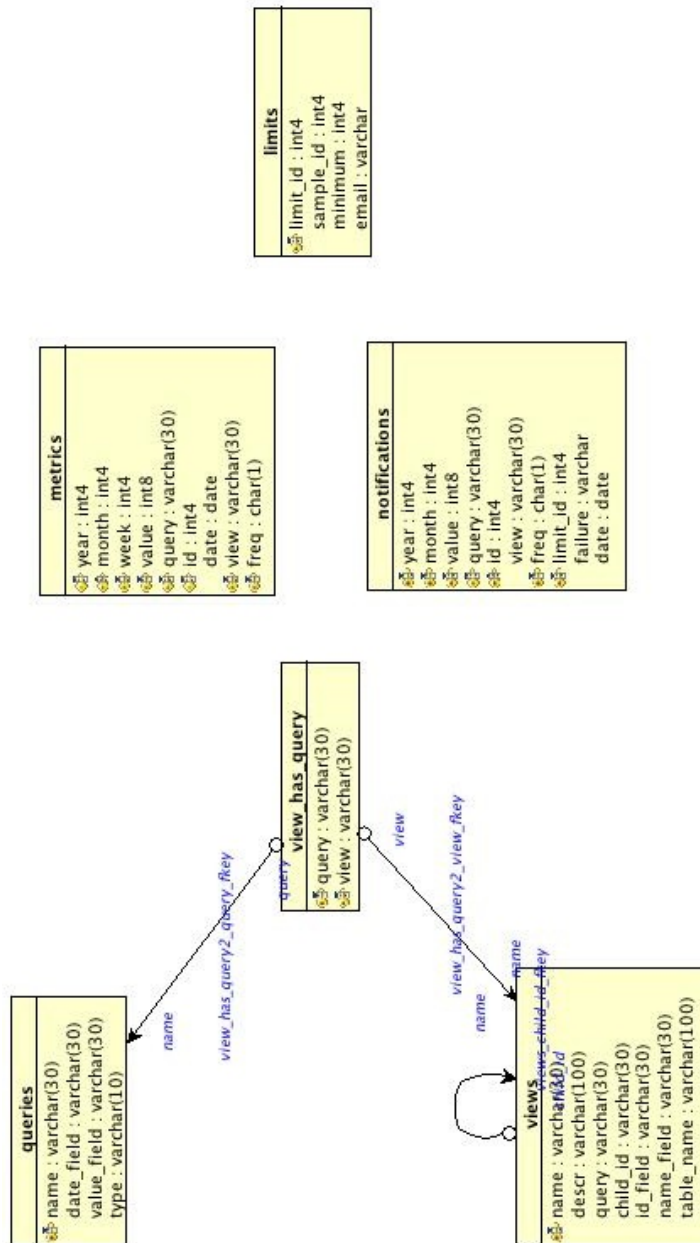


Abbildung 3.1: Relationales Datenmodell

Sample-Id, dem Datum der letzten Abfrage und der Abfrage der Metriken stellt den Endpunkt dar.

Diese Funktion erfasst zunächst die zum View gehörenden Daten, wie die Abfragen, um daraus, entsprechend dem Typus der Abfrage, ein SQL-Statement für die Aggregation zu erstellen.

Dieses Statement wird in der Funktion *insert_metrics* weiter verarbeitet. Hier findet sich auch der erste Monitoringsensor.

Zunächst wird ein weiteres SQL-Statement erstellt, um alle Minimumwerte für das gegebene Sample abzufragen. Es können beliebig viele Minima pro Sample und Nutzer definiert werden. Danach wird der Maximumwert mit jedem aktuellen Wert verglichen. Ist der aktuelle Wert höher als das aktuelle Maximum, wird dieser zum neuen Maximumwert. Ist der Wert kleiner, wird überprüft, ob der Wert unter dem Minimumwert liegt und gegebenenfalls eine Benachrichtigung in der Tabelle *Notifications* gespeichert.

Der abgefragte Wert wird in jedem Fall in der Tabelle *Metrics* gespeichert, wobei Verletzungen des Primärschlüssels mit Hilfe einer Exception abgefangen werden.

3.3 Bereitstellung von grundlegenden Funktionen

Für häufig aufzurufende Funktionen und der Einbindung von häufig verwendeter Pakete wurde die Klasse *DB-Monitor* geschaffen. Wie im Klassendiagramm in Abbildung 3.3 ersichtlich ist, verwendet diese Klasse die Funktionen von *Database* und stellt sie ihren eigenen Subklassen zur Verfügung.

Im folgenden werden die Klasse *DB-Monitor*, wie auch die Klasse *Database* genauer beschrieben.

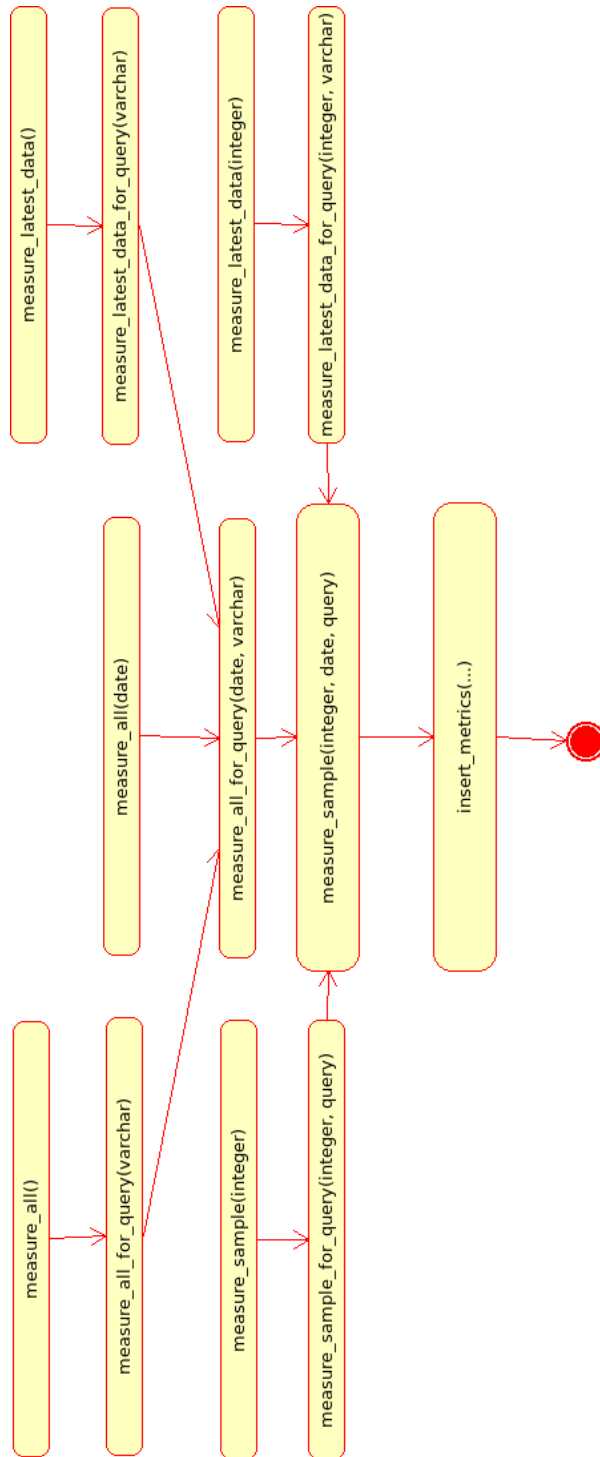


Abbildung 3.2: Aktivitätsdiagramm - PL/SQL

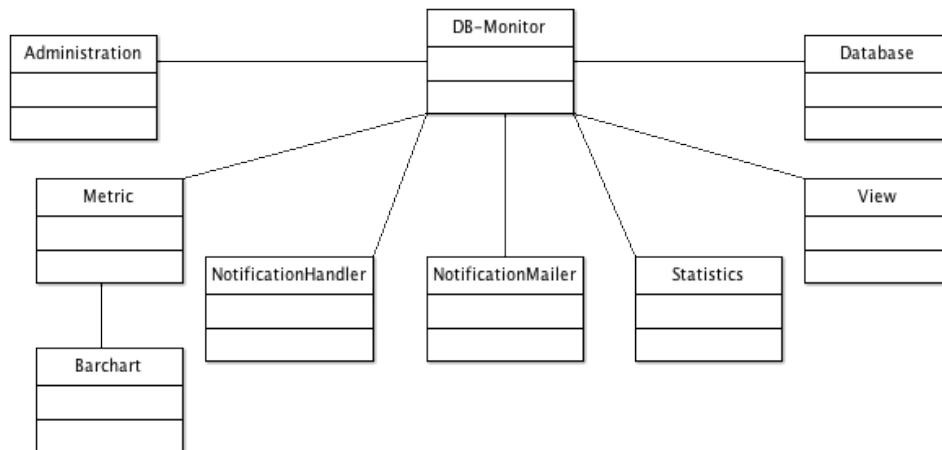


Abbildung 3.3: Übersicht der Klassen

3.3.1 Klasse Database

Wie sich aus dem Namen schließen lässt, stellt die Klasse *Database* die Schnittstelle zur Datenbank zur Verfügung. Da für sehr viele Funktionen eine Datenbank-Verbindung benötigt wird, wurde diese Klasse implementiert.

Der Zugriff auf die Datenbank erfolgt mit dem PHP Data Object [PDO08]. Zwar gäbe es auch die Möglichkeit mittels eigener von PHP bereitgestellten PostgreSQL-Funktionen auf die Datenbank zuzugreifen, allerdings bietet das PDO viel flexiblere Zugriffsfunktionen. So kann etwa die zugrundeliegende Datenbank mit nur einer Variable leichter geändert werden. Der größte Vorteil besteht in der Möglichkeit, *Prepared Statements* zu verwenden. Das PDO wird im Konstruktor erstellt und bietet mittels der Funktion *getConnection* eine Zugriffsmöglichkeit. Abgesehen von der Funktion eine Verbindung zu beenden *closeConnection*, existiert auch die Funktion *getTableFields*. Für die übergebene Tabelle werden alle Spaltennamen, welche diese Tabelle enthält zurückgeliefert.

3.3.2 getFileName und getPathFileName

Die Bilder für die Grafiken werden mittels eines bestimmten zusammengesetzten Dateinamens gespeichert. Zwar besteht die Möglichkeit in die Konfiguration einzugreifen (siehe dazu den Abschnitt 3.6), aber sinnvoller Weise nur bei Werten die fix als Konstante gespeichert werden können. Da diese Funktionen von unterschiedlichen Klassen zum Speichern und Abrufen verwendet werden, wurden sie auch in diese Klasse aufgenommen.

Den korrekt zusammengesetzten Dateinamen:

Jahr_ID_View_Query_Freq.Bild-Typ

stellt die Funktion *getFileName* zusammen. Bei der Funktion *getPathFileName* wird der Dateiname der vorherigen Funktion noch um den Bilderpfad, der in der Konstante *IMAGE_PATH* definiert wird, erweitert.

3.3.3 Error-logging

Die Klasse DB-Monitor stellt eine Error-Logging Funktion zur Verfügung. Subklassen können damit auftretende Fehler, wie beispielsweise, dass ein Bild nicht existiert oder ein Fehler mit der Datenbank aufgetreten ist, im Array *Errors* speichern, wodurch eine Speichermöglichkeit von nicht kritischen Fällen gegeben ist. Gleichzeitig ist dadurch eine problemlose Umleitung in eine Logdatei oder in die Datenbank durchzuführen.

Neue Fehler lassen sich mit der Funktion *addError* hinzufügen und mit *printAllErrors* können diese wieder ausgegeben werden.

3.4 Visualisierung der Metriken

Die Bilder werden mit der Klasse *Metrics* erstellt, welche eine Erweiterung der DB-Monitor-Klasse darstellt. Aufgabe dieser Klasse ist es, die Funktionen zur Verarbeitung der gespeicherten Daten aus der Tabelle *Metrics* zur

Verfügung zu stellen, um in der Folge mit Hilfe der Klasse *Barchart* die Grafiken zu erstellen.

3.4.1 Klasse *Metrics*

Im Konstruktor wird als zunächst die Datenbankverbindung gestartet, um anschließend die Prepared Statements für die verwendeten Funktionen zu erstellen. Da sich während der Entwicklung zeigte, dass es für den Speicherbedarf nicht dienlich ist, die Statements bei jedem Funktionsaufruf neu erstellt werden, werden diese gleich hier erstellt und anschließend in Klassenvariablen gespeichert. Außerdem würde sich der Vorteil der Prepared Statments aufheben, wenn diese erst wieder in jeder Funktion neu erstellt werden müssten. Für die Ausführung werden noch zusätzliche Daten benötigt, welche während der Laufzeit unverändert bleiben und somit gleich am Anfang abgefragt werden können. Parameter *Freq*, *Sample-Id*, *Query* und *Year*, die mittels den Set-Funktion gesetzt wurden, werden ebenfalls an dieser Stelle verarbeitet. Zuletzt wird noch eine Instanz von der Klasse *NotificationHandler*, welche sich um die Speicherung der Notifikationen kümmert, erstellt.

Mit den Set-Funktionen ist es möglich, die Instanzvariablen *overwriteExistingImages*, *freq*, *query*, *sampleId*, *view* und *year* zu setzen. Dadurch kann genau bestimmt werden, welche Bilder erstellt werden sollen. Unter Umständen kann es praktikabler sein kann, beispielsweise nur die Bilder für eine neuerstellte Abfrage zu generieren. Mit der Funktion *overwriteExistingImages* ist es möglich alle bestehenden Bilder zu überschreiben.

Die Funktion *createImages* und ihre Subfunktionen, wie zum Beispiel *createImagesForAllSamples*, verarbeiten die gesetzten Parameter. Den genauen Ausführungsweg stellt das Aktivitätsdiagramm in Abbildung 3.4 dar. Zuerst wird die Funktion *createImages* aufgerufen. Sie überprüft mit der Funktion *checkDataExistance*, ob Daten für die gesetzten Parameter vorliegen, um die Ausführungszeit zu verkürzen. Weitere Schritte werden nur unternommen wenn Daten existieren. Anschließend werden, die weiteren Funktionen

abgearbeitet und entsprechen den gesetzten Variablen, werden entweder weitere Daten, wie zum Beispiel alle Sample-IDs abgefragt, oder mit dem gesetzten Wert für die Variable fortgesetzt, bis die Ausführung bei der Funktion *createImage* angekommen ist.

Sind die vorher beschriebenen Funktionen abgearbeitet, beginnt *createImage* ihren Dienst. Zunächst wird überprüft, ob in der Variable *query* einen Punkt enthält, was darauf hindeutet, dass es sich bei dieser um eine Nichtinteger-Abfrage handelt. Der Name der Abfrage muss aufgesplittet werden, da sonst der Dateiname nicht korrekt zusammengesetzt werden würde, was als nächstes mit Hilfe der Funktion *getFileName* geschieht. Mit der Funktion *checkFile* wird überprüft, ob die Datei schon besteht bzw. ob das Erstellungsdatum einer bereits bestehenden Datei älter ist als das Datum der letzten Datenaktualisierung. Diese Überprüfung kann umgangen werden, indem die Instanzvariable *overwriteImage* auf true gesetzt wird und somit alle bestehenden Bilder neu erstellt werden. Erst wenn *checkFile* das OK zum Erstellen der Datei gegeben hat, werden die Daten mit Hilfe der Funktion *queryData* abgefragt. Wenn Daten vorliegen, wird die Grafik mit der Klasse *MyBarchart* erstellt.

Mit der Funktion *getMetricDataQuery* wird ein SQL-Statement entsprechend der Sampleirrhäufigkeit zusammengestellt. Sollte ein Daily-Sample vorliegen, sind weitere Schritte notwendig, da diese Daten nicht in der Tabelle *Metrics* gespeichert und direkt aus dem View *vw_heinz* abzufragen sind. Zurück in der Funktion *queryData* werden die Parameter an dieses Statement übergeben und die Metriken abgefragt. Beim Abspeichern der Werte in einem Array und werden diese wieder mit den Schwellenwerten aus der Tabelle *Limits* verglichen. Die Werte werden mit Hilfe der Funktion *getSampleMinimum*, die für eine Sample-ID alle vorhandenen Minimums mit deren Limit-ID zurückliefert, abgefragt. Abweichungen werden mit dem *Notification-Handler* verarbeitet, der dazu die Funktion *saveNotification* zur Verfügung stellt. In dieser wird überprüft, ob bereits eine Meldung gespeichert ist, gegebenenfalls aktualisiert oder eine neue angefügt.

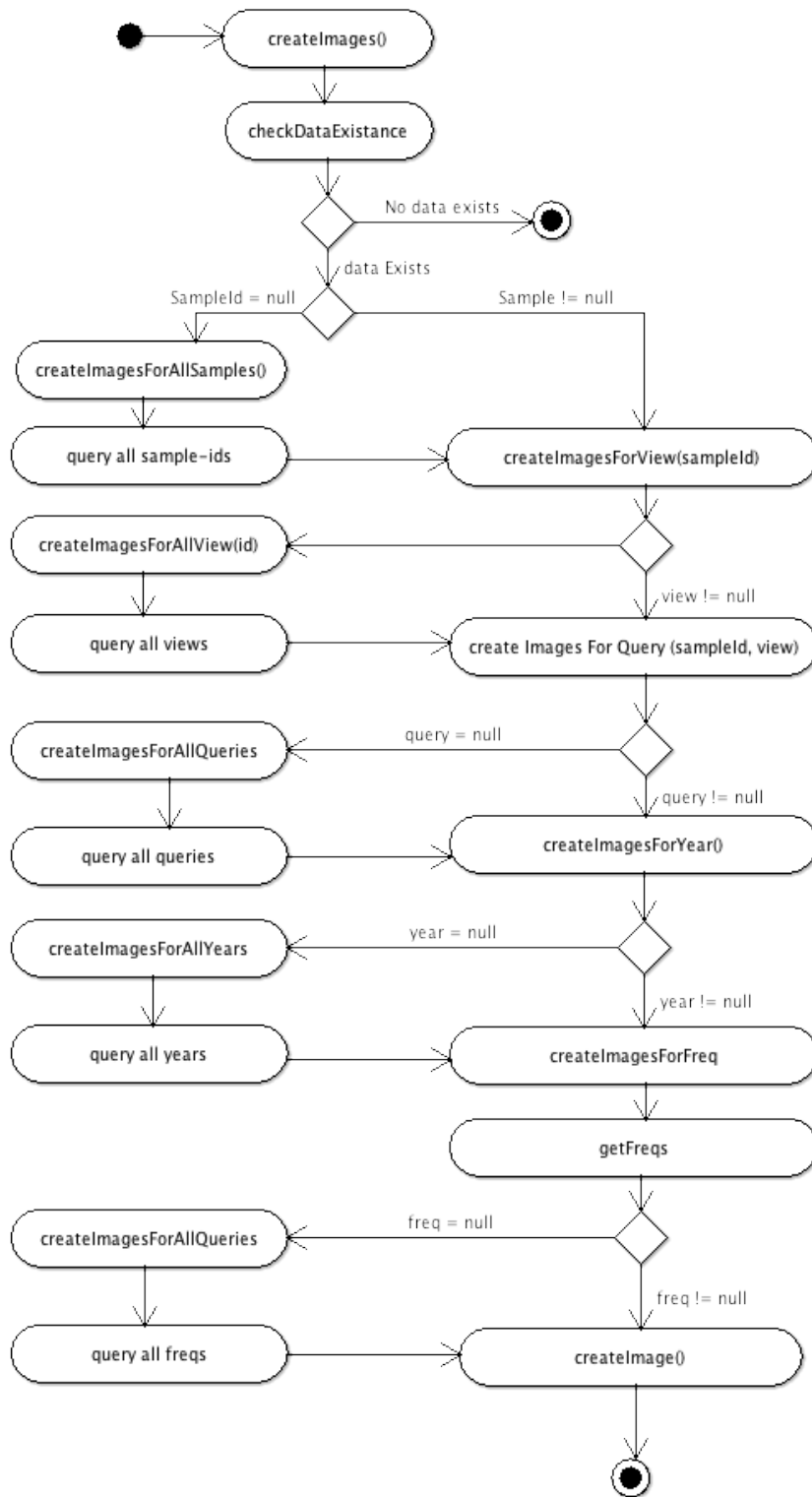


Abbildung 3.4: Aktivitätsdiagramm createImages()

3.4.2 Speicherung der Bilder

Aufgrund der Ausführungsdauer wäre es zu langsam die Bilder erst bei Aufruf des Webinterfaces zu generieren und deswegen müssen diese auf dem Server gespeichert werden. Es wurde zuerst erwogen, diese in der Datenbank zu speichern. Allerdings müssten die Bilder dann auch erst wieder aus der Datenbank geladen und zwischengespeichert werden, wodurch sich kaum Vorteile ergeben würden. Da die Funktion für die Speicherung in der Datenbank erstellt wurde, besteht dennoch diese Möglichkeit.

Anfangs wurde versucht, die Bilder mit dem PEAR Package Image Graph [IG08] zu erstellen. Während der Implementierungsphase zeigte sich, warum es sich bei diesem Paket noch um eine Alphaversion handelt: Da bei der Erstellung mehrerer Bilder ein Speicherleck besteht, wodurch an eine sinnvolle Ausführung nicht mehr zu denken war, weil sehr schnell das Speicherlimit von PHP erreicht wurde und das Skript abgebrochen wurde. Daher musste nach Alternativen gesucht werden, wie dem Tutorial auf Partdigital [PD08]. Auf dieser Seite wird ein Skript vorgestellt, das Balkendiagramme mit Hilfe der PHP GD Bibliothek [GD08] erstellt. Um den Anforderungen zu genügen, wurde es an einigen Stellen modifiziert. Im folgenden wird auf eine genauere Erklärung der Funktionsweise verzichtet, da diese auf der Webseite des Autors [PD08] im Detail erklärt ist. Einzig durchgeführte und relevante Änderungen gegenüber dem Original werden beschrieben. Abbildung 3.5 zeigt ein fertiges Balkendiagramm. Die X-Achse zeigt die Monate bzw. Wochen und die Y-Achse die jeweiligen Werte.

Eine Änderung betrifft die Höhe und Breite des Bildes. Diese Werte werden in *include/constants.php*, siehe Abbildung 3.9, als Konstante definiert und können damit rasch und problemlos an die Bedürfnisse der Nutzer angepasst werden.

Die größte Änderung gegenüber dem Originalskript betrifft den Abschnitt zur Erzeugung von horizontalen Linien. In der Ursprünglichen Version durchlief eine Schleife alle Werte von null bis zum darzustellenden Wert. Dies

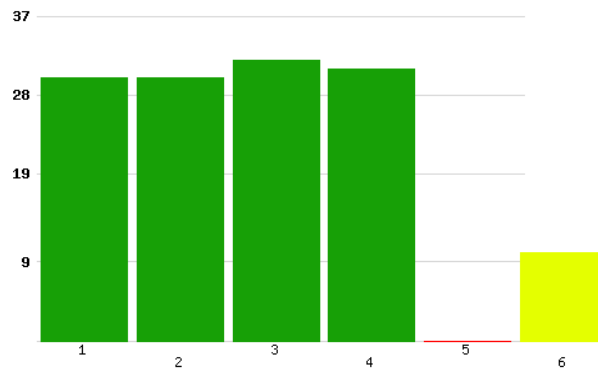
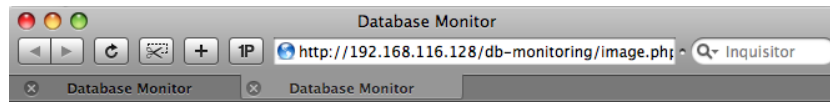


Abbildung 3.5: Ausgabe des Balkendiagramms im Webinterface

erwies sich als äußerst unpraktikabel, da zum Beispiel die Werte von Bytes-Abfragen mit ihren Biginteger-Werten die Ausführungsgeschwindigkeit extrem verlangsamten. Deswegen wurde eine neue Schleife geschrieben, die jetzt nur fünf mal durchläuft um die Grundlinie und horizontale Linien in 25% Schritten bis zum Maximalwert einzeichnet, wie es Abbildung 3.5 zeigt.

Bei der Formatierung wurden weitere Änderungen vorgenommen. Die Balkenfarben werden entsprechend den Prozentangaben in der Konfigurationsdatei *constants.php* gefärbt, die Beschriftung der X-Achse wurde durch eine Anpassung lesbarer, womit die Werte auch bei vielen Balken noch ersichtlich sind. Die Ausrichtung aller Bestandteile des Balkendiagramms wurde geändert, um auch den Y-Werten mehr Platz zu bieten.

Ursprünglich wurden die Bilder nach dem Durchlauf direkt ausgegeben. Da jedoch die Ausführung für alle Dateien zu komplex wäre, wird der Graph unter dem übergebenen Dateinamen gespeichert. Weiters werden diese nun

nicht mehr als JPG abgespeichert sondern als PNG, wodurch der Speicherbedarf spürbar minimiert werden konnte. Abbildung 3.6 zeigt einen Screenshot des Image-Folders, welcher deutlich der Größenunterschied zwischen JPG und PNG erkennen lässt.

Index of /db-monitoring/images










Name	Last modified	Size	Description
 Parent Directory		-	
 2005 4 Samples bytes a.jpg	07-Mar-2008 03:16	17K	
 2005 4 Samples bytes a.png	07-Mar-2008 00:53	2.9K	
 2005 4 Samples bytes m.jpg	07-Mar-2008 03:16	17K	
 2005 4 Samples bytes m.png	07-Mar-2008 00:53	2.9K	
 2005 4 Samples mirrored docs a.jpg	07-Mar-2008 03:16	15K	
 2005 4 Samples mirrored docs a.png	07-Mar-2008 00:53	2.7K	
 2005 4 Samples mirrored docs m.jpg	07-Mar-2008 03:16	15K	
 2005 4 Samples mirrored docs m.png	07-Mar-2008 00:53	2.7K	

Abbildung 3.6: Größenunterschied JPG und PNG

3.4.3 Ausgabe für das Webinterface

Die Ausgabe, wie Abbildung 3.7 zeigt, für das Frontend erfolgt mit Hilfe der Klasse *View*. Diese stellt Funktionen bereit, um den HTML-Code zu erzeugen, welcher in der Datei *view.php*, siehe Abbildung 3.9, ausgegeben wird.

Wie bei vielen anderen Klassen, wird im Konstruktor eine Datenbankverbindung aufgebaut. Die Instanzvariablen werden mit den Standardwerten, wie dem aktuellen Jahr, initialisiert. Die Metriken für den ausgewählten View werden mit Hilfe der Funktion *getQueriesForView* abgefragt, und das SQL-Statement anhand der gesetzten Parameter mit der Funktion *parseViewSqlStmnt* zusammengesetzt. In dieser Funktion wird entsprechend dem View über die Zusammensetzung des Statements entschieden, da dies einen

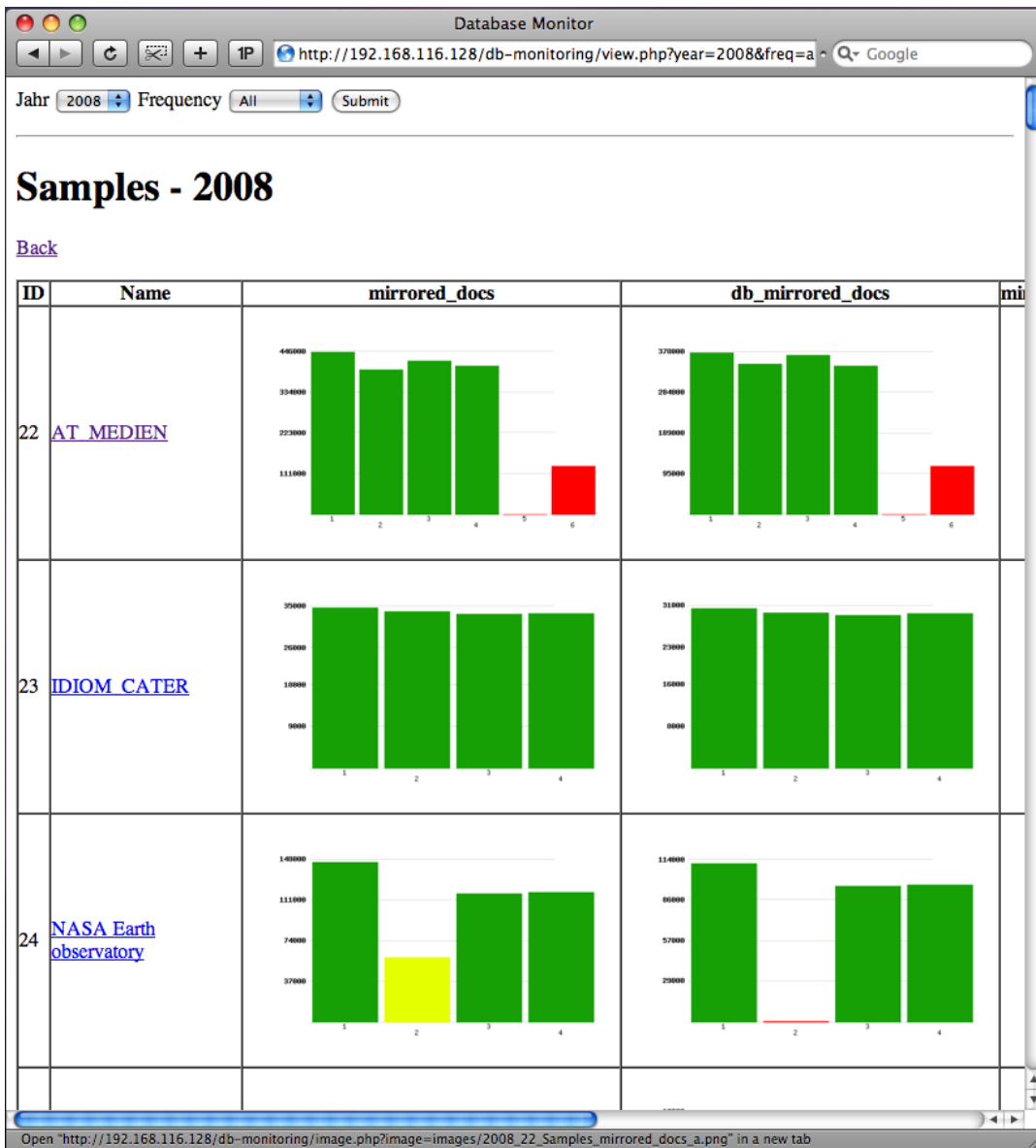


Abbildung 3.7: Webinterface: Ausgabe der Balkendiagrammen

enormen Effekt auf die Ausführungsgeschwindigkeit bei der Anzeige der Webseite hat. IDs und Namen von Samples oder Websites werden direkt aus deren Ursprungstabellen abfragt. Mit der Funktion *getParent* wird das ID-Feld des übergeordneten Views abgefragt, um die Anzeige der Webseiten entsprechend der Samplezugehörigkeit zu limitieren.

Auf der Ausgabeseite besteht die Möglichkeit, Parameter zur Anzeige bestimmter Daten, wie das Jahr und die Sample-Häufigkeit (Frequency), zu bestimmen. Die Klasse View stellt deswegen Funktionen bereit, wie *getFormYears* und *getFormFreqs*, die die möglichen Parameter als HTML-Formular-Liste zurückliefern.

Die Konfigurationsdatei bietet die Möglichkeit, die Zahl der angezeigten Datensätze zu limitieren. Dies geschieht durch Setzen der Konstanten *ROWS-PER-VIEW*. Standardmässig ist dieser Wert auf *ALL* gesetzt ist, um alle Datensätze auf einer Seite anzuzeigen oder auf einen Ganzzahlenwert gesetzt werden kann, um nur die angegebene Zahl an Reihen auszugeben. Die Limitierung der angezeigten Daten erfolgt durch Anpassung des SQL-Statements, indem dieses um *LIMIT* und *OFFSET* erweitert wird. *LIMIT* entspricht dem Wert der Konstanten und *OFFSET* gibt die jeweilige Position auf den Daten an. Dieser Wert wird mit der POST-Variable *offset* an andere Seiten übergeben.

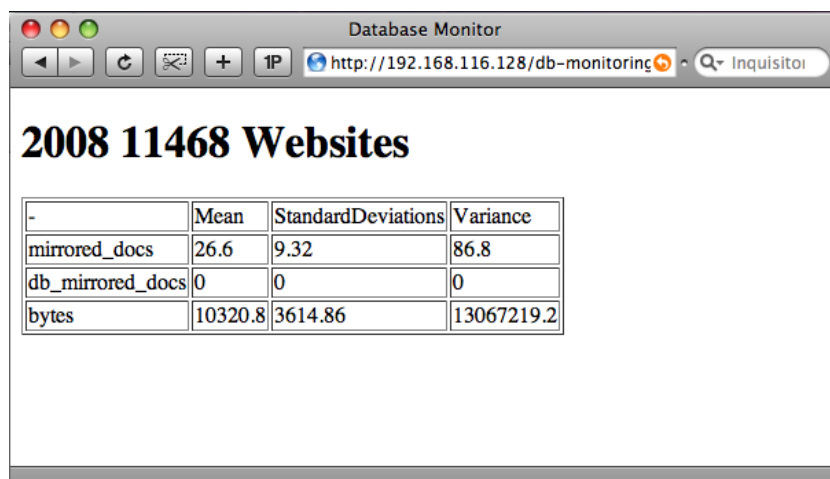
Zur Navigation durch die Daten werden die Funktionen *getNextLink*, *getPreviousLink* und *getLink* zur Verfügung gestellt. Bei *getLink* handelt es sich um eine Funktion, welche die beiden anderen durch die korrekte Link-Tag Zusammensetzung unterstützt. *getNextLink* und *getPreviousLink* berechnen den Offset, um in den Daten vor bzw. zurück zu navigieren und erstellt dann mit der Hilfe von *getLink* den Link.

Die Ausgabe der gespeicherten Bilder erfolgt in einer Tabelle, welche mit der Funktion *printTable* ausgegeben. Zunächst wird die Kopfzeile erstellt, der nächste Schritt ist der Aufruf der Funktion *getIdsNames*. Hier werden die gesetzten Variablen als Parameter für das im Konstruktor erstellte SQL-Statement übergeben. Durch die Ausführung werden die einzelnen Zeilen

der Tabelle ausgegeben, beginnend mit einer Spalte für die abgefragte ID und den Namen, der gleichzeitig einen Link zu dem untergeordneten View erhält beziehungsweise zur Statistikseite. Die weiteren Spalten werden mit den Bildern befüllt. Mittels der Funktion *getAllQueries* werden alle gespeicherten Abfragen durchlaufen, und für jede eine Spalte erstellt, in welche mittels der Funktion *printImage* das Bild ausgegeben wird. Vor Ausgabe eines Bildes prüft *printImage*, ob ein Bild entsprechend den gesetzten Parameter zusammengesetzten Dateinamen besteht, um dieses dann auszugeben oder einfach zu schreiben, dass keine Daten vorliegen.

3.4.4 Statistiken

Die unterste Ansichtsebene sind die Statistiken für die Webseiten, die Abbildung 3.8 zeigt.



	Mean	StandardDeviations	Variance
mirrored_docs	26.6	9.32	86.8
db_mirrored_docs	0	0	0
bytes	10320.8	3614.86	13067219.2

Abbildung 3.8: Ausgabe der Statistiken im Webinterface

Auf dieser Seite des Webinterfaces lassen sich zu einem ausgewählten View, Website-ID und Jahr verschiedene Statistiken anzeigen. In Abstimmung mit den bestehenden Benutzern wurden einige Statistikfunktionen ausgewählt, welche von Anfang an implementiert wurden. Details dazu werden in diesem Abschnitt vorgestellt.

Für PHP gibt es zwar die PHP Extension Community Library [PECL08], die einige Statistikfunktionen zur Verfügung stellt. Allerdings hält sich der Aufwand diese selbst umzusetzen in Grenzen.

Von Beginn an wurden Durchschnittswert, Varianz und Standardabweichung umgesetzt. Für jedes einzelne dieser Maße bestehen zwei Funktionen, eine Print-Funktion, um das Maß als HTML-Absatz auszugeben, und eine Funktion zur Berechnung des Maßes, die ein Float zurückliefert.

Bevor diese Funktionen ausgeführt werden können, müssen noch die Daten mit der Funktion *getDataForAllQueries* zu den übergebenen Variablen abgefragt und in einem Array gespeichert werden. Für eine neue Maßzahl sind diese beiden Funktionen zu implementieren und der Funktion *printAllStatistics*, welche alle Statistiken berechnet und als Tabelle ausgibt, eine Zeile hinzuzufügen, um die neu erstellte Funktion einzubinden.

3.5 Umsetzung der Notifications

Die Funktion der Notifikationen verteilt sich über mehrere Funktionen, welche teilweise bereits kurz vorgestellt wurden. Sie sollen aber hier nochmals zusammen angeführt werden:

3.5.1 Monitoring in den PL/SQL-Funktionen

Bei der Aggregation der Daten in der PL/SQL-Funktion, werden die prozentuellen Abweichungen mit den bereitsvorhandenen Einträgen in der Tabelle *Limits* verglichen und bei einer Unterschreitung dieser Werte werden diese in der Tabelle *Notifications* mitprotokolliert.

3.5.2 Monitoring bei der Erstellung der Bilder

Bei der Erstellung der Bilder liegt das Hauptaugenmerk des Monitors im Auffinden von Perioden, in welchen keine Daten vorliegen, da diese bei der Überprüfung in den PL/SQL-Funktionen nicht berücksichtigt werden.

3.5.3 Notification-Mailer

Die Klasse *Notification-Mailer* ermöglicht es, Fehler, welche während vorheriger Verarbeitungsschritte in Verbindung mit den Sampledaten aufgetreten sind, per Mail an die Nutzer zu versenden.

Aufgetretene Dateninkonsistenzen werden in der Tabelle *Notifications* gespeichert, welche in Kapitel 3.1 schon vorgestellt wurde. Beim Aufruf dieser Klasse wird zuerst das Datum der zuletzt versendeten Mails mittels der Funktion *setLastNotification* aus der Logdatei *notification_mailer.log* im Verzeichnis *log*, siehe Abbildung 3.9, ausgelesen und in der Klassenvariablen *last_notification* gespeichert. Weiters wird im Konstruktor wieder eine Datenbankverbindung eröffnet, welche später für den Abruf der E-Mailadressen aus der Datenbank benötigt wird.

In *sendLastestNotifications* wird zuerst das Datum des letzten Updates aus der Variablen *last_notification* ausgelesen, um die Funktion *sendNotifications* mit diesem Datum aufzurufen. In dieser wird ein SQL-Statement zur Abfrage aller E-Mail-Adressen von der Tabelle *Limits* erstellt, um damit mittels *buildNotificationMessage* den Nachrichtentext zu erstellen. Dafür benötigt die Funktion den Rückgabewert von *getNotificationByUser*, welche ein SQL-Statement liefert, um alle Daten für eine bestimmte Mail-Adresse abzufragen. Schließlich wird diese Nachricht mit der Funktion *sendMail*, das die PHP-Funktion *mail* nutzt, versendet.

3.6 Administration

Wie die Verzeichnisstruktur in Abbildung 3.9 zeigt, enthält das Verzeichnis *include* die Datei *constants.php*. In ihr können Parameter angepasst werden, welche später als Konstante zur Verfügung stehen, um das Verhalten des DB-Monitors an die eigenen Verhältnisse anzupassen.

Die Verwendung der wichtigsten Parameter und deren Standardwerte werden in Tabelle 3.1 genauer vorgestellt.

Zur Verwaltung der wichtigsten Aufgaben besteht im Webinterface eine Möglichkeit, wie dies Abbildung 3.10 mit einem Screenshot zeigt. Metriken lassen sich über den Link *Create new query* erstellen. Um die korrekte Verarbeitung sicherzustellen, ist ein Name zu vergeben und für das Datumfeld und Wertfeld eines der Felder aus den Listenfeldern auszuwählen. Zuletzt muss noch der Typ der Abfrage selektiert werden,

Im nächsten Link *Add query to view* lassen sich neu erstellte Metriken einem bestehenden View zuweisen. Dazu müssen aus den jeweiligen Listenfeldern die entsprechenden Werte ausgewählt werden.

Unter dem Link *Create limit* lassen sich Minimumwerte zu bestimmten Samples zuordnen. Dabei ist die Angabe einer E-Mailadresse erforderlich, um Notifikationen von abweichenden Werten zustellen zu können.

3.7 Installation des Database-Monitors

Im folgenden werden die erforderlichen Schritte erläutert, welche notwendig sind, um den Database-Monitor auf einem Server lauffähig zu machen.

3.7.1 Vorbereitung des Servers

Damit die Applikation lauffähig ist sind folgende Komponenten zu installieren:

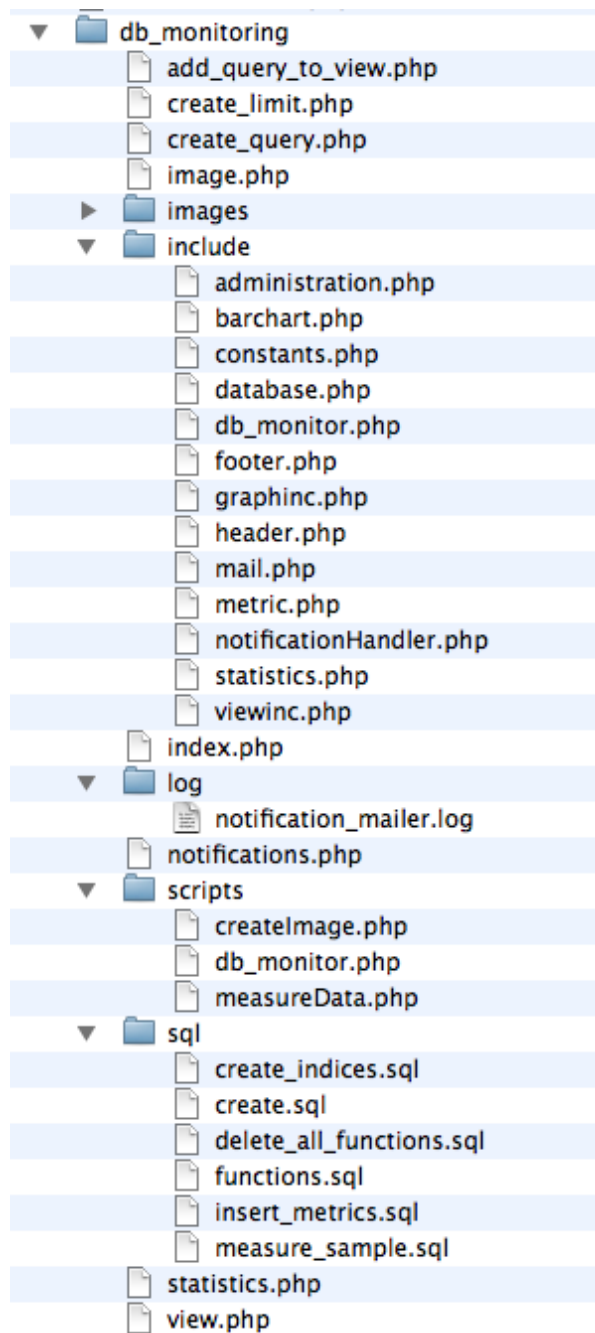


Abbildung 3.9: Verzeichnisstruktur

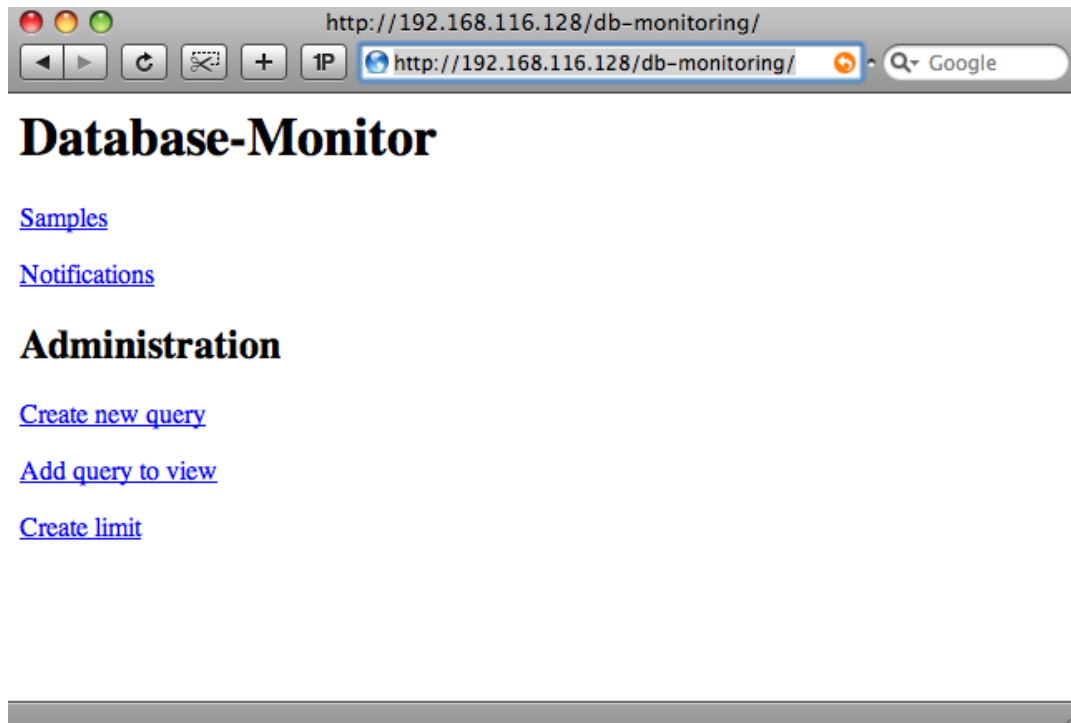


Abbildung 3.10: Startseite Webinterface

Tabelle 3.1: Konfiguration

Konstante	Funktion	Standardwert
DEFAULT_VIEW	View der standardmässig aufgerufen wird	Samples
ROWS_PER_VIEW	Bietet die Möglichkeit, wenn ein Integer-Wert zugewiesen wurde, die Zeilen pro Seite auf diese Zahl zu limitieren und Links einzubauen, um durch die Daten zu navigieren. Um alle Zeilen anzuzeigen kann 'All' übergeben werden.	ALL
FIRST_YEAR	Jahr ab welchem Daten vorliegen, bzw. die Auswahlmöglichkeit dafür bestehen soll	2005
IMAGE_PATH	relativer Speicherpfad für Bilder. Kann jeder Ordner sein, für den der Benutzer eine Schreibberechtigung besitzt.	images/
IMAGE_TYPE	Typ der Bilder. Es wird nur png unterstützt, für weitere Formate ist <i>bar-chart.php</i> anzupassen	png
DB_USER	Datenbankbenutzer	heinz
DB_NAME	Name der Datenbank	weblyzard
DB_PASSWORD	Passwort für Datenbank	
DB_TYPE	Type der Datenbank. Alle möglichen Parameter finden sich in der Dokumentation des PHP Persistent Data Objects [PDODriver08]. Es kann auch mysql verwendet werden, jedoch wurde diese Funktion nicht getestet.	pgsql
DB_LOCATION	Host der Datenbank	gecko3.wu-wien.ac.at

Verwendete Debian-Pakete

Die Entwicklung erfolgte unter Debian Etch 4.0. Dabei wurden folgende Programmpakete in den jeweils angegebenen Versionen verwendet. Diese sollten auch auf dem Server installiert werden:

- apache2.2-common 2.2.3-4
Web-Server, der für die Ausgabe verwendet wird
- libapache2-mod-php5
Modul für Apache, damit dieser PHP-Dateien verarbeiten kann
- php5 5.2.0-8
- php5-cli 5.2.0-8
Command Line Interface
- php5-common 5.2.0-8
- php5-gd 5.2.0-8
Bibliothek, damit PHP auch Bilder verarbeiten kann
- php5-pgsql 5.2.0-8
Modul, damit PHP mit PostgreSQL-Datenbanken kommunizieren kann
- postgresql 8.2.5

3.7.2 Vorbereitung der Datenbank

In diesem Abschnitt werden die Schritte gezeigt, die notwendig sind, um die Datenbank auf das Tool vorzubereiten. Alle notwendigen Dateien, welche die Datenbank betreffen, befinden sich, wie die Abbildung 3.9 zeigt, im Verzeichnis *sql*.

Import der Tabellen

Die Definition der Tabellen befindet sich im Ordner *sql*, in der Datei *create_tables.sql*. Nachdem man in das Verzeichnis gewechselt ist und in PostgreSQL

als Superuser angemeldet ist, kann diese mit dem Befehl

```
\i create_tables.sql
```

importiert werden. In dieser Datei sind auch Insert-Statements definiert, welche definierte Ansichten, Abfragen und Minimum-Werte enthalten.

Import der PL/SQL-Funktionen

Die Definition der PL/SQL-Funktionen befinden sich ebenfalls im Verzeichnis *sql* in der Datei *functions.sql*. Diese wird mit dem Befehl

```
\i functions.sql
```

eingebunden.

Daten in der Datenbank

Um von Anfang an die Lauffähigkeit der Applikation sicherzustellen, sind Views und Queries zu definieren. Die Standardwerte können mit dem Befehl

```
\i data_insert.sql
```

eingefügt werden.

Verbindungsdaten

Die Verbindungsdaten zur Datenbank, sind in der Datei *constants.php* im Verzeichnis *include* anzupassen. Im Abschnitt *Database* besteht die Möglichkeit Datenbanknutzer, Namen der Datenbank, Passwort, Datenbanktype und Datenbankhost anzugeben.

Lese- und Schreibrechte

Um die korrekte Funktion zu gewährleisten, muss der Datenbanknutzer im Besitz der Leserechte für die Tabellen *vw_heinz*, *sample* und *company* sein. Für die Ausführung der PL/SQL-Funktionen muss der Nutzer Schreibrechte für die neu erstellten Tabellen *images*, *limits*, *metrics*, *notifications*, *queries*, *view_has_query* und *views* besitzen.

3.7.3 Dateien

Die Dateien und Ordner müssen in das passende Verzeichnis kopiert werden, so dass sie vom Webserver erreichbar sind. Bei Apache ist das unter Debian standardmäßig das Verzeichnis */var/www*, kann allerdings in der Konfigurationsdatei */etc/apache2/site-available* angepasst werden.

Der wichtigste Punkt betrifft die Lese- und Schreibrechte. Standardmäßig heißt der Nutzer des Webservers Apache *www-data* und gehört der gleichnamigen Gruppe an. Für die korrekte Ausführung der Monitoringaufgaben benötigt der Nutzer Lese- und Schreibrechte, um die Bilder speichern zu dürfen. Dies geschieht mit dem Befehl:

```
chown -r www-data:www-data images
```

3.7.4 Anpassung bei PHP

Die ausführbaren Dateien liegen im Verzeichnis *scripts*, diese verwenden auch die Klassen aus dem Verzeichnis *includes*. Wird eines der Skripts nun nicht aus seinem Verzeichnis aufgerufen, wie zum Beispiel

```
php opt/weblyzard/lib/core/db\_monitoring/scripts/db\_monitor  
-date current
```


erscheint die Fehlermeldung, dass die einzubindenden Dateien nicht gefunden werden können. Deswegen ist es notwendig in der Konfigurationsdatei des PHP Command Line Interfaces */etc/php5/cli/php.ini* den *include_path* anzupassen, so dass dieser auch das Include-Verzeichnis des DB-Monitors enthält. Die Zeile des *include_path* sieht dann wie folgt aus:

```
include_path = " ./usr/share/php:/var/www/db_monitoring/include"
```

Mit dieser Einstellung sucht PHP zuerst im aktuellen Verzeichnis, dann im eigenen PHP-Verzeichnis und schließlich im DB-Monitorverzeichnis nach den einzubindenden Dateien.

3.7.5 Erstellung eines Cronjobs

Um laufend die Metriken und Bilder zu berechnen, sollte ein Cronjob erstellt werden, der regelmäßig die Skripte ausführt.

Um automatisch alle Schritte auszuführen besteht das Skript *db_monitor.php*. Dieses und die Subskripte werden genauer im Abschnitt 3.8 beschrieben. Vorerst reicht es zu wissen, dass alle Metriken und Bilder mit dem Aufruf

```
./db\_monitor -date current
```

erstellt werden können. Dieser Befehl ist daher in die Datei */etc/cron.daily* einzutragen , um täglich ausgeführt zu werden.

3.8 Manuelle Ausführung

Zur Vereinfachung der Ausführung für den Cronjob beziehungsweise des Nutzers wurden drei Skripte geschrieben.

Mit dem Skript *db_monitor.php* können alle Funktionen des Monitors aufgerufen werden, da er die beiden anderen Skripte einbindet und zuletzt

Tabelle 3.2: Parameter `measureData.php` und `createImages.php`

Parameter	Funktion
<code>-date</code>	Nur Daten nach dem festgelegten Datum werden berechnet
<code>-sample</code>	Integer für ein Sample, um nur dieses zu berechnen
<code>-query</code>	String, um nur Daten für die angegebene Query zu berechnen

Tabelle 3.3: Parameter `createImages.php`

Parameter	Funktion
<code>-year</code>	Die Bilder werden nur für das angegebene Jahr erstellt
<code>-freq</code>	Nur die Bilder für diese Häufigkeit werden erstellt

die Notifikationen per Mail versendet. Zuerst müssen die Daten aggregiert werden, dies geschieht mit Hilfe des Skripts `measureData.php`, welches die PL-SQL-Funktionen aufruft. Die möglichen Parameter sind in Tabelle 3.2 angeführt. Zusätzlich zu diesen bestehen für das Skript `createImage.php`, das die Bilder mit Hilfe der Klasse `Metrics` erstellt, noch weitere Parameter, die in Tabelle 3.3 nachgelesen werden können.

4 Zusammenfassung

Ein zentraler Bestandteil des IDIOM Forschungsprojekts ist der *weblyzard*. Dieser spiegelt Webseiten, auf welchen verschiedene Analysen durchgeführt werden. Fehler beim Mirroring waren in der Vergangenheit nur mit größtem Zeitaufwand durch die Nutzer selbst zu finden. Die Lösung dieses Problems war der erste Ansatzpunkt für den Datenbankmonitor.

Eine genaue Analyse theoretischer Konzepte zu den Anforderungen an Monitoringsysteme, möglicher auftretender Probleme von fehlerhaften Sensoren und der Konzepte hinter SNMP und MRTG stellte die Ausgangsbasis dar. Zur Sicherstellung verlässlicher Datenqualität für das IDIOM Forschungsprojekt wurden folgende Schritte gesetzt:

- Die Aggregation der Daten mittels PL-SQL-Funktionen, um die Daten aufzubereiten und Abweichungen von einem gesetzten Minimum mitzuprotokollieren.
- Im nächsten Schritt werden diese Daten weiter verarbeitet, um daraus Bilder zu erstellen, um den Endnutzern einen Überblick über die vorhandenen Daten liefern. Die Grafiken werden als Balkendiagramme dargestellt.
- Schließlich werden Notifikationen über die Abweichungen versendet, wobei hier besonders auf die Zuverlässigkeit geachtet wurde, um Falschmeldungen zu vermeiden.

Basierend auf dieser Arbeit ist über Erweiterungsmöglichkeiten nachzudenken, wie eine Automatisierung weiterer Analyseschritte oder der Ausführung benutzerspezifischer Skripte bei abweichenden Werten. Dadurch werden der

Zeitaufwand der Nutzer, die Dateninkonsistenzen besser verfolgen und rascher korrigieren zu können.

Durch eine Unterstützung von SNMP, im besonderen die Unterstützung des *Trap-Directed Polling*, und der Bereitstellung einer weiteren Form der Notifikation, realisiert durch Newsfeeds, wären Fehlermeldungen besser zu verarbeiten.

Literaturverzeichnis

- [Bandini05] Stefania Bandini and Fabio Sartori. Improving the effectiveness of monitoring and control systems exploiting knowledge-based approaches. *Personal and Ubiquitous Computing*, 9(5):301 – 311, Sep 2005.
- [CC08] Ecoresearch.net. Ecoresearch | media watch on climate change. <http://www.ecoresearch.net/climate/>.
- [EM08] Ecoresearch.net. Ecoresearch | us election 2008 web monitor. <http://www.ecoresearch.net/election2008/>.
- [ER08] Ecoresearch.net. Ecoresearch network | environmental online communication. <http://www.ecoresearch.net/>.
- [FIT] FIT-IT.at. <http://www.fit-it.at/>.
- [GD08] PHP.net. Php: Gd functions - manual. <http://us3.php.net/manual/en/ref.image.php>.
- [IG08] PHP.net. Pear image graph. http://pear.php.net/package/Image_Graph/.
- [Id06] Idiom. Project description. <http://www.idiom.at/project-description/>.
- [MRTG08] T Oetiker. Tobi oetiker's mrtg - the multi router traffic grapher. <http://oss.oetiker.ch/mrtg/>.
- [Nebel06] M Nebel. Tipps und tools zum monitoring präventivanalyse. *iX Magazin für Magazine für professionelle Informationstechnik*, 05:118 – 122, May 2006.

- [Oetiker01] T Oetiker. Monitoring your it gear: the mrtg story. *IT Professional*, 3(6):44 – 48, Nov 2001.
- [PD08] Partdigital.com. Gd library bar chart. <http://www.partdigital.com/tutorials/bar-chart/>.
- [PDO08] PHP.net. Php: Pdo-manual. <http://at.php.net/manual/en/book.pdo.php>.
- [PDODriver08] PHP.net. Php: Pdo drivers - manual. <http://at.php.net/manual/en/pdo.drivers.php>.
- [PECL08] PHP.net. Pecl :: The php extension community library. <http://pecl.php.net>.
- [PL08] PostgreSQL.org. PostgreSQL: Documentation: Manuals: PostgreSQL 8.1: Pl/pgsql - sql procedural languag. <http://www.postgresql.org/docs/8.1/static/plpgsql.html>.
- [PS08] PHP.net. Php: Pdo prepared statemants - manual. <http://at.php.net/manual/en/pdo.prepared-statements.php>.
- [Raz02] O Raz, P Koopman, and M Shaw. Semantic anomaly detection in online data sources. *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pages 302 – 312, Jan 2002.
- [Stallings98] W Stallings. Snmp and snmpv2: the infrastructure for network management. *Communications Magazine*, pages 37 – 43, Jan 1998.
- [WL08] Weblyzard.com. Web and media monitoring. <http://www.weblyzard.com/>.
- [Wo07] Gerhard Wohlgenannt. Work package: Database monitoring - semanticlab wiki. https://wiki.semanticlab.net/index.php/Workpackage:_Database_Monitoring.