Christian-Albrechts-University Kiel
Department of Computer Science
Software Engineering Group

Bachelorarbeit

Multidimensionale Performance-Analyse komponentenbasierter Softwaresysteme

Björn Weißenfels
bjw@informatik.uni-kiel.de

30. September 2011

Advised by: Prof. Dr. Wilhelm Hasselbring
Dipl.-Wirt.-Inf. Jens Ehlers

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

DB	Datenbank
DBMS	Datenbankmanagementsystem
DOLAP	Desktop OLAP
E/R	Entity/Relationship
FASMI	Fast Analysis of Shared Multidimensional Information
HOLAP	Hybrid OLAP
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity
MDX	Multidimensional Expression
MOLAP	Multidimensional OLAP
OLAP	Online Analytical Processing
OLAP4J	OLAP for Java
ROLAP	Relational OLAP
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
XML	Extensible Markup Language
XMLA	XML for Analysis

1 Einleitung

Heutzutage stützen viele Unternehmen ihre Geschäftsprozesse auf Softwaresysteme. Deshalb kann sich ein Unternehmen einen Ausfall wichtiger Software-Komponenten über einen längeren Zeitraum nicht leisten. Gleichzeitig wächst die Komplexität dieser Softwaresysteme ständig. Manche Dienste werden nur den Mitarbeitern des Unternehmens bereitgestellt, andere für wesentlich größere Benutzergruppen, zum Beispiel über die Website des Unternehmens. Manchmal werden Websites nur zu Informations- und Werbezwecken genutzt, oft aber auch für Kundenkontakt und Vertrieb.

Während die Ressourcen für Software, die nur Mitarbeitern eines Unternehmens vorbehalten bleibt, möglicherweise gut abzuschätzen sind, ist dies bei Webshops wesentlich schwieriger, da der mögliche Benutzerkreis sehr groß ist und die Auslastung des Systems starken Schwankungen unterzogen sein kann.

Trotzdem wird von jedem Dienst und der Softwarekomponente, die diesen Dienst bereitstellt, hohe Benutzbarkeit gefordert. Hier ist neben der Verfügbarkeit das Antwortzeitverhalten maßgeblich. Diese Eigenschaften stellen die Leistungsfähigkeit (engl. Performance) des Softwaresystems dar. Das Antwortzeitverhalten wird durch die Datenübertragungsrate, die Anzahl der aktuellen Anfragen und die Bearbeitungszeit des Systems bestimmt.

Den Kontrollfluss und die Antwortzeiten von Methodenaufrufen zu überwachen bietet die Möglichkeit Probleme frühzeitig zu erkennen, Engpässe aufzudecken und Ressourcen effizient zu nutzen.

Bei diesem sogenannten Monitoring entstehen schon nach kurzer Zeit viele Daten. Damit der Anwender den Überblick behalten kann, müssen diese Daten aufbereitet und graphisch ansprechend dargestellt werden. Online Analytical Processing (OLAP) bietet die Möglichkeit, dem Analysten zu überlassen, welche Daten er in welchem Detailgrad betrachten möchte.

Im Rahmen dieser Arbeit soll das bestehende Monitoring- und Analyse-Framework

Kieker um ein Plugin zur Analyse mittels OLAP erweitert werden. Als Anwendungsbeispiel dient der JPetStore, ein Online-Tierhandel, der alle gängigen Funktionen eines Webshops bereitstellt.

Die Arbeit ist in 5 Kapitel gegliedert. Im folgenden zweiten Kapitel werden die Grundlagen zur multidimensionalen Performance-Analyse definiert und das Konzept des Online Analytical Processing erläutert. Im dritten Kapitel wird das der Analyse zugrunde liegende Datenmodell erklärt. Weiterführend wird die Transformation dieses Modells in relationale Strukturen beschrieben und mit der Aggregation ein für die Implementierung wichtiges Konzept erläutert. Im vierten Kapitel werden zunächst die Architektur von Kieker und die Einordnung des entwickelten Plugins zur Performance-Analyse beschrieben. Im Folgenden werden der freie OLAP-Server Mondrian und das Mondrian-Frontend Saiku als Bestandteile der Lösung vorgestellt. Das entwickelte Plugin erlaubt dem Anwender, die Performance eines Softwaresystems mittels multidimensionaler Sichten zu analysieren. Im letzten Kapitel werden schließlich die Ergebnisse der Arbeit zusammengefasst.

2 Grundlagen

In diesem Kapitel werden zuerst die Begriffe Software-Komponente, Software-Performance und Performance-Analyse erklärt. Anschließend wird der für das Verständnis der Arbeit wichtige Begriff des Online Analytical Processing (OLAP) beschrieben.

2.1 Software-Komponente

Die Idee, Software zum Großteil aus fertigen Bausteinen (Komponenten) zusammenzusetzen, wurde erstmals 1968 von M. D. McIlroy schriftlich festgehalten [13]. Ziele dieser Art der Softwareentwicklung sind vor allem Wiederverwendbarkeit, um die Wirtschaftlichkeit von Softwareproduktionen zu steigern, aber auch Qualitätsgewinn, Abstraktion und Erweiterbarkeit.

Eine viel zitierte Definition für den Begriff Software-Komponente wurde 1996 auf der European Conference on Object-Oriented Programming formuliert und beschreibt eine Software-Komponente als ein *Teil einer Komposition* mit *vertraglich spezifizierten Schnittstellen* und *expliziten Kontextabhängigkeiten*. Des Weiteren kann eine Software-Komponente *unabhängig* eingesetzt werden und unterliegt der *Zusammensetzung durch Dritte* [19].

Eine neuere Definition beschreibt eine Software-Komponente als eine architektonische Einheit, die eine Teilmenge der Funktionalität des Systems und/oder Daten kapselt, den Zugriff auf diese Teilmenge über eine explizit definierte Schnittstelle einschränkt und explizite Abhängigkeiten für ihren erforderlichen Ausführungskontext definiert [20].

Die Komplexität einer Komponente ist nicht eingeschränkt und kann somit stark variieren, besonders da Komponenten wieder aus anderen Komponenten zusammengesetzt werden können. Wichtig ist die Beschreibung einer Schnittstelle, über die die Dienste dieser Komponente genutzt werden können. Damit Komponenten

auch vermarktbar sind, und um nicht gewollten Änderungen oder Annahmen über das Verhalten einer Komponente vorzubeugen, soll die Implementierung verborgen bleiben. Eine Komponente ist also eine „Blackbox". Software-Komponenten verkörpern daher die Prinzipien des Software-Engineering Kapselung, Abstraktion und Modularität.

Der Ausführungskontext kann zum Beispiel andere benötigte Komponenten und Daten enthalten, genauso wie eine benötigte spezifische Hardware, ein Betriebssystem oder eine Laufzeitumgebung.

2.2 Software-Performance

Performance misst, wie effektiv ein Softwaresystem in Bezug auf Zeitbeschränkungen und Zuweisung von Ressourcen ist. Hier werden Größen wie Antwortzeit, Durchsatz und Auslastung betrachtet [6].

C.U. Smith definiert Performance als den Grad, zu welchem ein Softwaresystem oder eine Komponente seine Ziele in Bezug auf Zeitverhalten erfüllt [18]. Wobei damit primär das Antwortzeitverhalten gemeint ist. Die Antwortzeit ist die Zeit, die ein System braucht, um auf eine Eingabe zu reagieren. Einflussfaktoren sind unter anderem die Netzauslastung, der Durchsatz (Flaschenhälse), die Hardware (CPU, Speicher, Netzwerk-Bandbreite, Anzahl CPUs, etc.) und die Laufzeitumgebung. Die Performance wird also nicht nur von der Software selbst beeinflusst, sondern auch von allen darunterliegenden Ebenen wie dem Betriebssystem, der Middleware, der Hardware und der Kommunikationsnetzwerke.

Die Skalierbarkeit beschreibt das Verhalten eines Softwaresystems bezüglich Antwortzeit und Durchsatz bei steigender Last und ist somit ein wichtiges Maß der Performance. Da Performance die Bedienbarkeit eines Softwaresystems maßgeblich beeinflusst, zählt sie zu den wichtigsten nicht-funktionalen Anforderungen.

2.3 Performance-Analyse

Um die Performance eines Softwaresystems zu analysieren, existieren verschiedene Ansätze [17, 5]. Das Ziel besteht darin, Performance-Probleme zu erkennen und zu beheben. Je früher die Probleme erkannt werden, desto besser lassen sie sich

lösen, denn Änderungen an der Architektur der Software sind zu einem späten Entwicklungszeitpunkt sehr teuer.

Beim analytischen Ansatz wird, zum Beispiel auf Grundlage von UML-Diagrammen, ein Performance-Modell erstellt, das durch Annahmen über Funktionen, Ressourcen und Nutzerverhalten Prognosen bezüglich der Performance erstellt. Da es sich hierbei nur um ein theoretisches Modell handelt, können die tatsächlichen Werte von den prognostizierten abweichen, es eignet sich aber sehr gut für die Bestimmung bereitzustellender Ressourcen.

Simulation ist ein weiterer Ansatz zur Erkennung von Softwareproblemen. Dafür muss ein Prototyp der zu entwickelnden Software erstellt und unter realistischen Bedingungen getestet werden. Dieses Verfahren liefert bessere Prognosen, ist jedoch relativ teuer und birgt die Gefahr trotzdem ungenau zu sein, wenn zum Beispiel performancekritische Komponenten nicht Bestandteil des Prototyps waren.

Eine dritte Möglichkeit bilden sogenannte Betriebstests im Anschluss an die Implementierung. Dieses Verfahren liefert genaue Messdaten, allerdings sind zu diesem Zeitpunkt, wie eingangs beschrieben, Optimierungen schwerer vorzunehmen.

Das in dieser Arbeit behandelte Verfahren unterscheidet sich wesentlich von den obigen Ansätzen, da es sich beim Monitoring um die Beobachtung laufender Softwaresysteme handelt. Es geht dabei nicht um einen einmaligen Test, sondern um eine ständige Überwachung des Systems. So kann beispielsweise auf Änderungen der Anforderungen, des Nutzerverhaltens oder des Nutzeraufkommens gezielt reagiert werden. Des Weiteren bietet dieses Verfahren die Möglichkeit, auch bestehende Softwaresysteme zu analysieren.

2.4 OnLine Analytical Processing

Der Begriff Online Analytical Processing (OLAP) wurde 1993 in einem White Paper von E.F. Codd geprägt. In diesem Paper wurde der Begriff durch 12 Regeln definiert, die heute unter dem Begriff Codd'sche Regeln bekannt sind. Diese Regeln blieben allerdings nicht ohne Kritik, da sie nicht mathematisch, sondern eher vage formuliert wurden. Außerdem trennen die Regeln nicht strikt zwischen fachlich-konzeptionellen Anforderungen und technischen Realisierungsaspekten. 1995 fügte Codd 6 weitere Regeln hinzu und strukturierte die nun 18 Regeln, was aber nicht wesentlich zur Akzeptanz beitrug [12].

Im selben Jahr wurde der FASMI-Test veröffentlicht, der heute auch als Definition für OLAP-Systeme in Wissenschaft und Praxis verbreitet ist. FASMI steht für Fast-Analysis-of-Shared-Multidimensional-Information und somit für die 5 einprägsamen Anforderungen an OLAP-Systeme [15]:

Fast bedeutet, dass Resultate innerhalb von ca 5 Sekunden geliefert werden sollten, wobei einfache Abfragen höchstens eine Sekunde und nur sehr wenige Abfragen länger als 20 Sekunden dauern sollten, da Benutzer mit dem System sonst nicht effizient arbeiten können.

Analysis steht dafür, dass dem Endbenutzer domänenspezifische Analysefunktionalität zur Verfügung steht und die entsprechenden Abfragen intuitiv formuliert werden können, also insbesondere keine Programmierkenntnisse erfordern.

Shared steht für die Forderung nach Mehrbenutzerfähigkeit. Dies bedeutet insbesondere, dass entsprechende Schutzmechanismen für den Zugriff auf die Daten, wie auch Sperrverfahren bei konkurrierenden Schreibvorgängen existieren müssen.

Multidimensional ist die wichtigste Anforderung an ein OLAP-System. Es muss unabhängig von der zugrunde liegenden Datenhaltung eine multidimensionale Sicht auf die Daten bereitstellen und innerhalb von Dimensionen Hierarchien und multiple Hierarchien unterstützen, da dies die natürlichste Weise ist Daten zu analysieren.

Information steht für alle Daten und daraus abgeleitete Informationen, die das System bereitstellt. Hierbei ist die Datenmenge, die das System bei stabiler Antwortzeit analysieren kann, wichtig und nicht der Speicherplatzbedarf.

Das FASMI-Konzept stellt somit die Nutzeranforderungen in den Vordergrund und blendet die technischen Details zur Realisierung aus. Daher existieren unterschiedliche technische Realisierungen von OLAP-Systemen, die sich hauptsächlich in der Art der zugrunde liegenden Datenhaltung unterscheiden.

Werden die Daten in einem multidimensionalen Datenbankmanagementsystem gehalten, spricht man von *multidimensional OLAP* oder *MOLAP*. Hierbei werden sowohl die Detaildaten wie auch die Aggregationen in multidimensionalen Speicherstrukturen, meist in einem multidimensionalen Array, direkt auf dem OLAP-Server abgelegt. Dies sorgt für eine sehr gute Performance. Bei einem sehr hohen Datenaufkommen, oder einem Würfel mit vielen Dimensionen und damit vielen

Aggregationen, wird der OLAP-Würfel allerdings schnell zu groß für den Hauptspeicher des Servers, sodass die Antwortzeiten stark ansteigen [12]. Ein weiterer Nachteil ist, dass die Datenbestände nur zyklisch aktualisiert werden können. MOLAP ist also bei aktuellem technischen Stand eher für kleine Datenbestände bzw. wenige Dimensionen geeignet. Aufgrund technischer Entwicklungen wie der zunehmenden Verbreitung von 64-Bit-Rechner-Architekturen wird diese Technologie wieder interessanter.

In der Praxis basieren OLAP-Systeme häufig auf den bewährten relationalen DBMS. Diese Form von OLAP heißt *relational OLAP* oder kurz *ROLAP*. Es werden sowohl die Detaildaten als auch die Aggregationen in einer relationalen Datenbank gespeichert. Dabei müssen die Daten allerdings in einer Form vorliegen, die die multidimensionale Sichtweise unterstützt. Für diese Abbildung der multidimensionalen Sichtweise in relationale Tabellen haben sich das Star- und das Snowflake-Schema etabliert, die in Kapitel 3.3 näher erläutert werden. Der OLAP-Server greift nun direkt auf die relationale Datenbank zu, was natürlich im Vergleich zu MOLAP zu schlechteren Antwortzeiten führt. Dafür kann mit ROLAP eine wesentlich größere Datenmenge betrachtet werden. Außerdem unterstützt die robuste und bewährte relationale DB-Technologie die effiziente und sichere Verarbeitung dieser großen Datenmengen.

In Tabelle 2.4.1 sind die Vor- und Nachteile von ROLAP und MOLAP übersichtlich zusammengefasst.

Kriterium	ROLAP	MOLAP
Skalierbarkeit	+ Verwaltung großer Datenvolumen + flexibel in der Anzahl von Dimensionen	– Datenvolumen kann eingeschränkt sein – Anzahl von Dimension kann eingeschränkt sein
Antwortzeit	– vergleichsweise höhere Antwortzeiten	+ Performancevorteile bezüglich der Antwortzeit
Multi-dimensionalität	– durch relationale Speicherstruktur nicht explizit unterstützt	+ explizite Unterstützung durch Speicherstruktur
Reife/Stabilität	+ ausgereifte, robuste DB-Technologie + Umfang an Verwaltungsfunktionen (Sicherheit, Backup, Recovery, etc.)	– vergleichsweise neue Technologie
Standardisierung	+ hoher Grad an Standardisierung in Bezug auf Konzepte und Sprache (SQL)	– unzureichende Standardisierung/Offenheit, da meist proprietäre Technologie
Knowhow	+ bezüglich relationaler Datenbanken in den meisten Unternehmen verfügbar	– bezüglich multidimensionale Datenbanken ggf. nicht vorhanden

Abbildung 2.4.1: Vor- und Nachteile von ROLAP und MOLAP (Quelle: 7)

Hybrid OLAP (*HOLAP*) stellt eine Mischform dar, die versucht die positiven Eigenschaften beider Ansätze zu vereinen. Rechenintensive, hoch verdichtete Ag-

gregationen werden aufgrund der schnellen Antwortzeit in multidimensionalen, die große Menge an Detaildaten in relationalen Datenbanksystemen abgelegt.

Ein weiterer nicht so verbreiteter Architekturtyp ist das *Desktop OLAP* (*DOLAP*). Hierbei werden erst die Daten auf den lokalen Rechner importiert und dann lokal analysiert. Dies bietet den Vorteil, dass für die Analyse keine Netzwerkverbindung bestehen muss. Allerdings wird die Größe der zu analysierenden Datenmenge durch die dann zur Verfügung stehende Hardware begrenzt.

OLAP kann in nahezu allen Bereichen eingesetzt werden, bei weniger als 3 Dimensionen oder einer sehr kleinen Datenmenge erscheint dies jedoch nicht mehr sinnvoll.

3 Multidimensionales Datenmodell

Einer Datenanalyse mit OLAP liegt ein multidimensionales Datenmodell zugrunde, bei dem die unterschiedlichen Sichtweisen auf den Datenbestand im Mittelpunkt stehen. In diesem Kapitel werden die Elemente des multidimensionalen Datenmodells und die wichtigsten Operationen darauf anhand des Anwendungsbeispiels dieser Arbeit erläutert. Des Weiteren wird die Transformation des multidimensionalen Modells in ein logisches Modell beschrieben. Zum Abschluss des Kapitels werden das Erstellen von Aggregationen und ihr Nutzen vorgestellt.

3.1 Multidimensionaler Datenwürfel

Die Detaildaten und ihr semantischer Kontext machen das multidimensionale Datenmodell aus. Diese dem Modell zugrunde liegenden Daten werden Fakten genannt. Der semantische Kontext bildet die Struktur des Datenmodells, durch die der Nutzer eine auf seine Fragestellung angepasste Sicht auf die Daten erhält.

Diese mehrdimensionale Sicht auf die Daten wird durch die Metapher eines Würfels (engl. Cube) beschrieben (siehe Abbildung 3.1.1). Die Würfelkanten sind die Dimensionen des Modells, sie stellen den Analysekontext dar [8], wobei ein OLAP-Würfel nicht auf 3 Dimensionen beschränkt ist. Prinzipiell lassen sich beliebig viele Dimensionen abbilden. Die Anzahl der Dimensionen wird als Dimensionalität des Würfels bezeichnet. Bei mehr als 3 Dimensionen spricht man auch von einem Hypercube.

Dimensionen können hierarchisch in Dimensionsstufen (engl. Level) gegliedert werden [8], wie die Dimension Zeit mit ihren Dimensionsstufen Jahr, Quartal und Monat in Abbildung 3.1.2. Die Kantenlänge des Würfels wird durch die Anzahl der

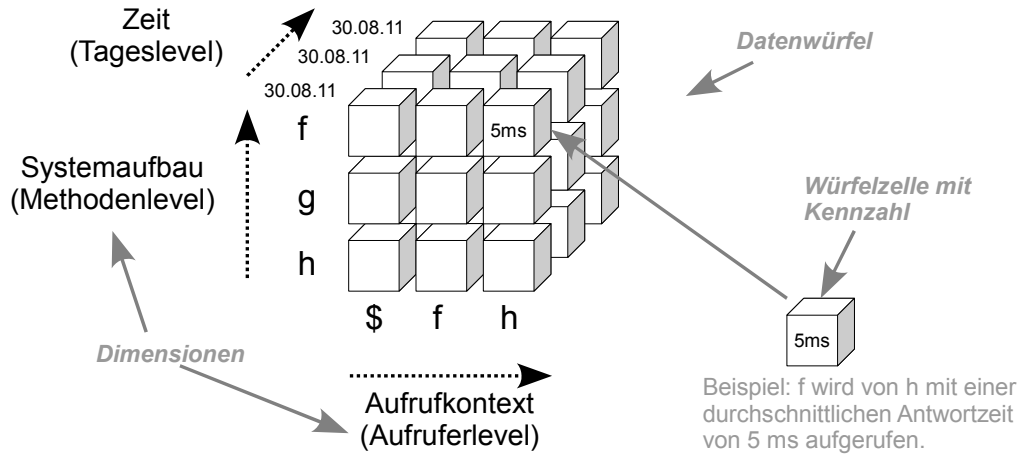

Abbildung 3.1.1: Würfel und Dimensionen

Dimensionselemente (engl. Member), also der Ausprägungen der jeweiligen Stufe bestimmt.

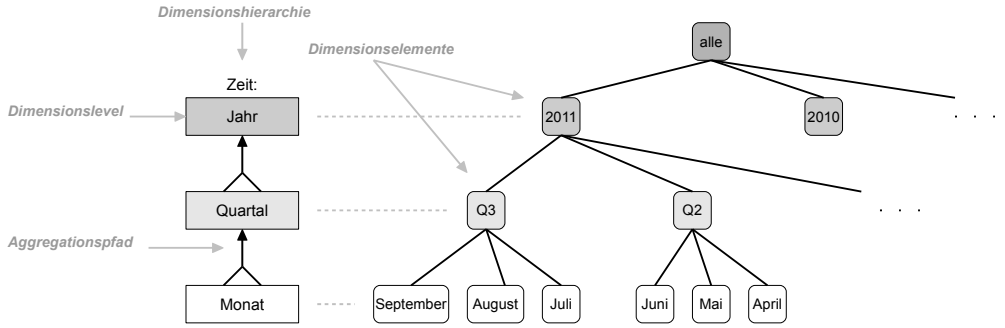

Abbildung 3.1.2: Dimensionshierarchie

Die Würfelzellen repräsentieren die Kennzahlen, wobei eine Kennzahl ein aus Fakten abgeleiteter Wert ist [8]. Entlang von Dimensionshierarchien können Kennzahlen über arithmetische Operatoren und Funktionen zu abgeleiteten Kennzahlen aggregiert werden, weshalb man auch von einem Aggregations- oder Konsolidierungspfad spricht (siehe Abbildung 3.1.2). Hier kommen je nach semantischem Kontext verschiedene Aggregationsmöglichkeiten wie Zählfunktionen, die Bildung von Summe, Produkt, Durchschnitt, Minimal- oder Maximalwert, etc. zum Einsatz. Die Granularität der Daten ist ein wichtiges Thema. Je detaillierter die Daten sind, desto feiner ist ihre Granularität. Einen Überblick über die Verdichtungsebenen der Dimension Zeit liefert Abbildung 3.1.3.

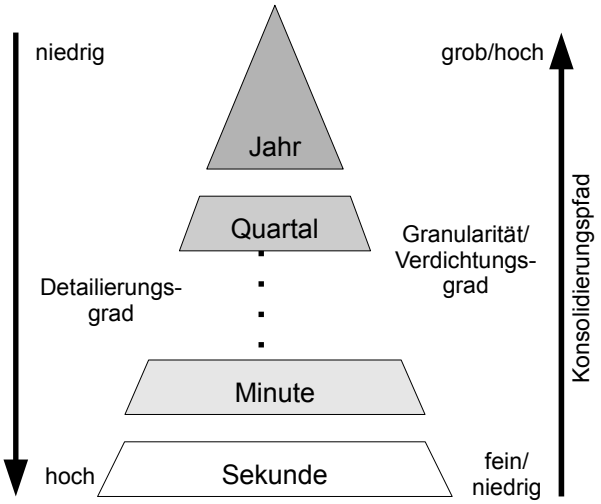

Abbildung 3.1.3: Verdichtungsebenen der Dimension Zeit

Eine hohe Verdichtungsebene bietet hier eine langfristige Sicht auf die Daten, was meist einer Durchschnittsbetrachtung entspricht. Für detailliertere Analysen greift man dann auf niedrigere Verdichtungsebenen zu. In welcher Granularität die Daten vorliegen, wirkt sich auf den Speicherplatzbedarf und die Verarbeitungsgeschwindigkeit aus, weshalb man bei hohem Datenaufkommen ältere Daten nur noch aggregiert vorhält [16]. Auf diese Thematik wird in Kapitel 3.4 genauer eingegangen.

Abbildung 3.1.4 zeigt die Hierarchien der Dimensionen und den Zusammenhang der Dimensionen und Fakten des Anwendungsbeispiels dieser Arbeit in einem multidimensionalen E/R-Modell. Am Beispiel der Zeitdimension wird deutlich, dass innerhalb einer Dimension parallele Hierarchien definiert werden können, um durch unterschiedliche Konsolidierungspfade eine flexiblere Analyse zu unterstützen.

3.2 Operationen zur Datenanalyse

Ein wesentliches Merkmal eines OLAP-Systems ist die einfache Navigierbarkeit durch die eben beschriebene multidimensionale Datenstruktur. Diese wird durch Operationen auf dem Datenwürfel erreicht, wobei laut Marquardt [12] Roll-up, Drill-down, Drill-across, Slicing und Dicing mindestens erwartet werden können. Im Folgenden werden diese Operationen und die ebenfalls gebräuchliche Pivotierung erklärt.

- Unter *Roll-up* versteht man das Wechseln von einer feingranularen zu einer

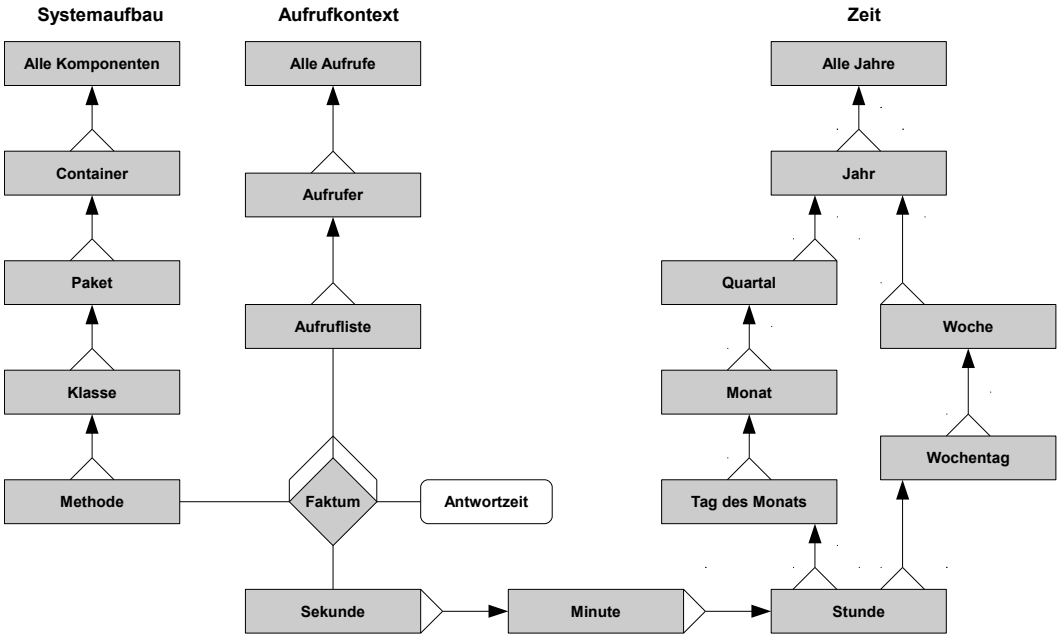

Abbildung 3.1.4: Dimensionshierarchien dargestellt in einem multidimensionalen E/R-Modell

aggregierten Ansicht. Es entspricht also einer Verdichtung der Daten entlang des Aggregationspfades. Zum Beispiel der Wechsel von Quartalen zu Jahren wie in Abbildung 3.2.1.

- *Drill-down* ist die zum Roll-up komplementäre Funktion, also der Wechsel von den aggregierten Daten zu den Detaildaten.

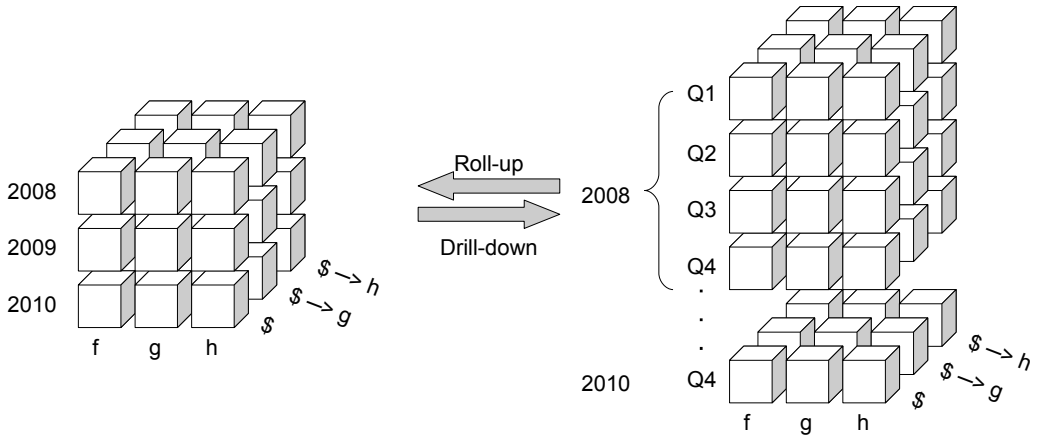

Abbildung 3.2.1: Roll-up und Drill-down

- Als *Pivotierung* wird das Vertauschen von Achsen bezeichnet. Es handelt

sich also um eine Drehung des Würfels, weswegen für diese Operation auch der Begriff *Rotation* gebräuchlich ist. Wie in Abbildung 3.2.2 zu sehen, wird dem Anwender dadurch eine andere Sicht auf die Daten gegeben, während der Informationsgehalt unverändert bleibt.

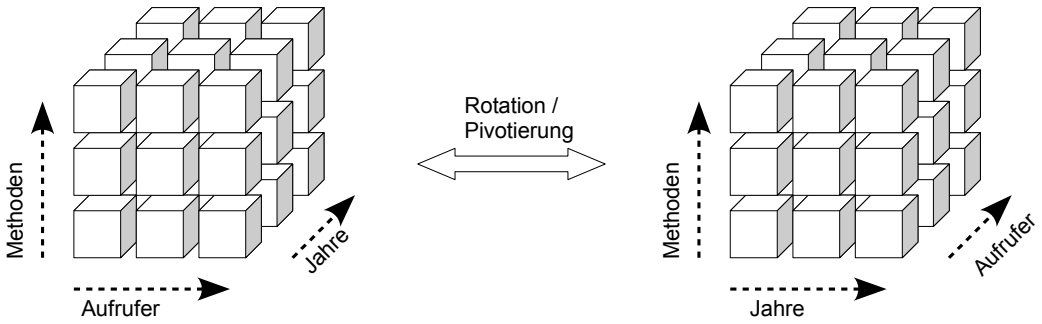

Abbildung 3.2.2: Rotation / Pivotierung

- Als *Drill-across* bezeichnet man das Austauschen einer Dimension gegen eine andere, wodurch man einen grundlegend veränderten Würfel erhält.
- Beim *Slicing* wird die Dimensionalität des Würfels um eins verringert, indem aus einer Dimension nur noch ein Element betrachtet wird, diese Dimension dann also wegfällt. Bildlich betrachtet entspricht dies dem Herausschneiden einer Scheibe aus dem Würfel wie in Abbildung 3.2.3, in der die Sicht auf eine bestimmte Klasse eingeschränkt wird.

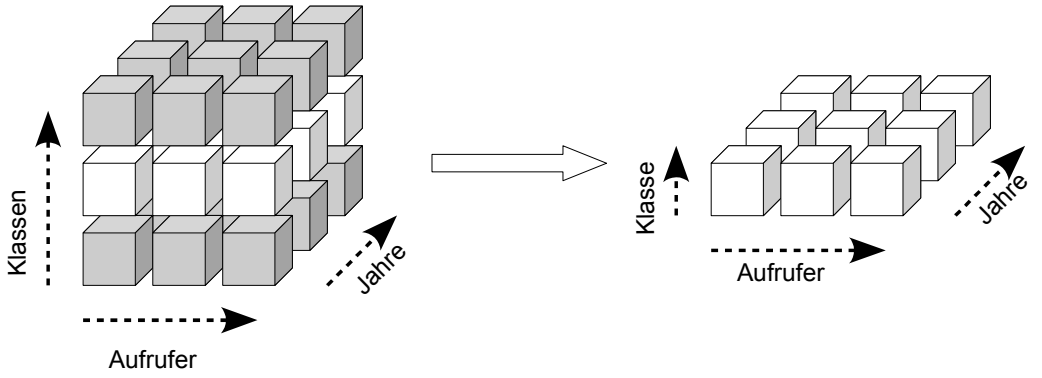

Abbildung 3.2.3: Slicing

- *Dicing* bietet die Möglichkeit einen Teilwürfel zu bilden, indem die Sicht auf beliebige Dimensionselemente und -Level eingeschränkt wird. Diese Opera-

tion ist also sehr allgemein und schließt so auch Slicing mit ein. Ein Beispiel ist in Abbildung 3.2.4 zu sehen.

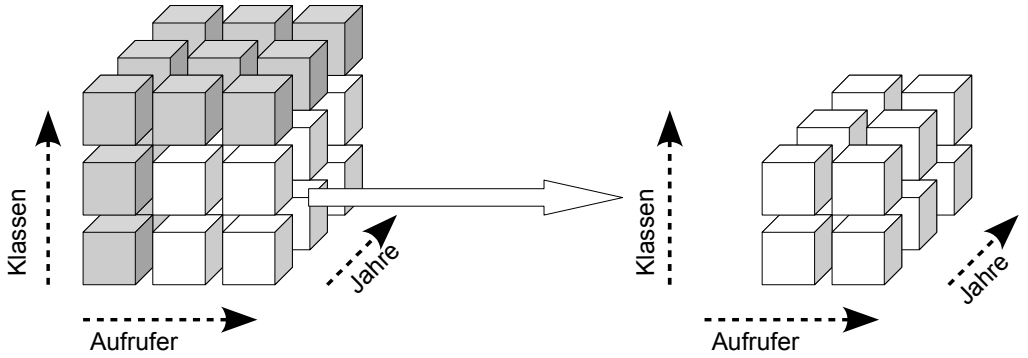

Abbildung 3.2.4: Dicing

Die letzten beiden Operationen bilden Teilmengen des Würfels, wodurch die Menge der dargestellten Daten übersichtlich gehalten werden kann und dadurch der Fokus der Analyse auf die für den Anwender interessanten Daten gelegt werden kann.

3.3 Logisches Datenmodell

Zur Abbildung eines mehrdimensionalen Modells auf relationale Datenstrukturen hat sich die Darstellung der Dimensionen und Fakten in einem Star-Schema als Standard etabliert. Für jede Dimension wird eine Dimensionstabelle angelegt, die einen (meist künstlichen) Primärschlüssel und als weitere Attribute die zugehörigen Dimensionslevel enthält. Den Kern des Modells bildet die Faktentabelle, die neben den Fakten die Fremdschlüssel zu den Dimensionstabellen enthält [9]. So steht jedes Faktum in Beziehung zu allen Dimensionen eines Würfels (siehe Abbildung 3.3.1). Der Begriff Star-Schema ist auf die sternförmige Anordnung der Dimensionstabellen um die Faktentabelle zurückzuführen.

Vom semantischen multidimensionalen E/R-Modell (3.1.4) zum logischen Modell gehen die Informationen über die hierarchische Anordnung der Dimensionen verloren, welche später an anderer Stelle wieder hinzugefügt werden müssen (siehe 4.2). Wie im Beispiel zu sehen, wird auf eine Normalisierung der Tabellen verzichtet, da die Dimensionen nicht als performancekritisch gelten [21].

Eine (Teil-)Normalisierung ist jedoch auch möglich. Hierbei entsteht ein sogenanntes Snowflake-Schema (siehe Abbildung 3.3.2). Dieses Modell ist wie in der Abbil-

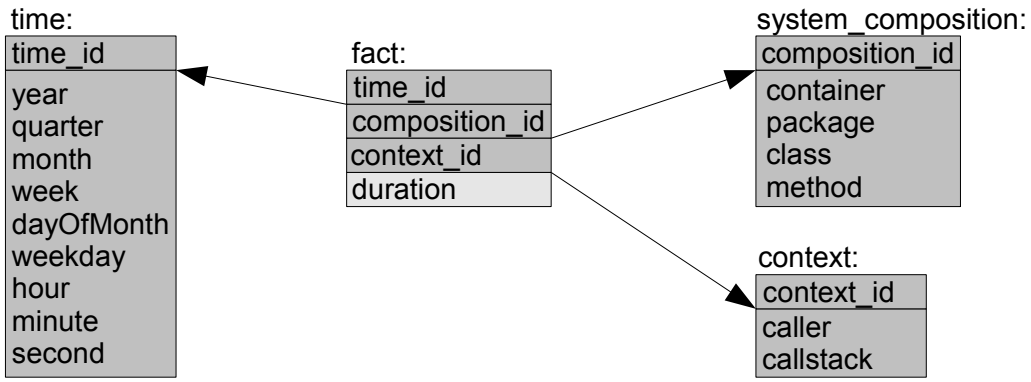

Abbildung 3.3.1: Star-Schema

dung zu sehen deutlich komplexer. In diesem Beispiel werden aus vier Tabellen im Star-Schema sechzehn Tabellen im Snowflake-Schema. Dadurch kann zwar das benötigte Speichervolumen durch weniger Redundanz verringert werden, jedoch fallen bei Abfragen auf hohem Aggregationsniveau viele Join-Operationen an, wodurch die Performance wieder verschlechtert wird.

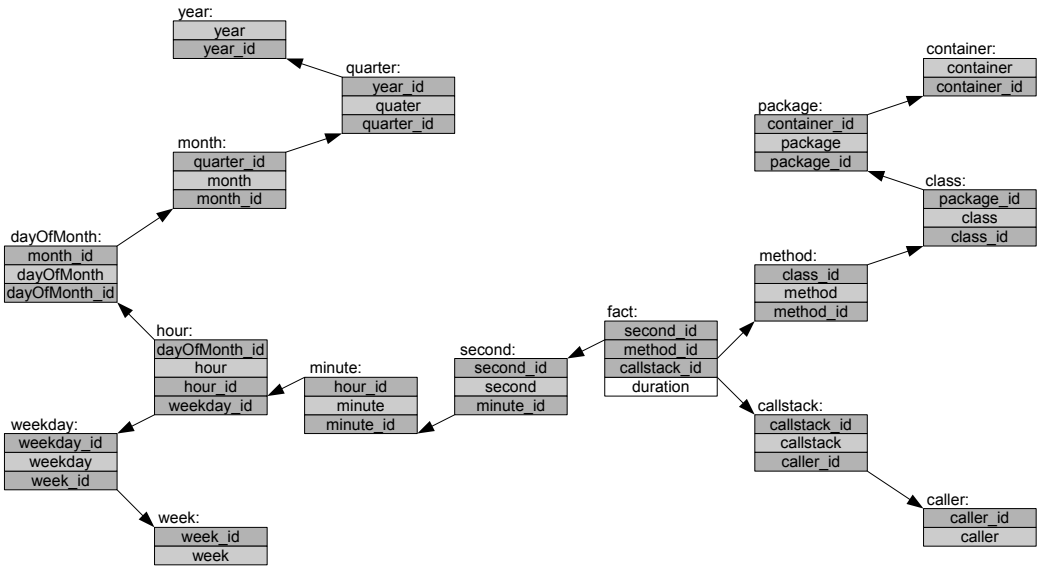

Abbildung 3.3.2: Snowflake-Schema

Möchte man verschiedene Würfel mit zum Teil gleichen Dimensionen abbilden, wird das Schema einfach um eine weitere Faktentabelle erweitert, die die gewünschten Dimensionstabellen ebenfalls nutzt. Das dabei entstehende Schema wird Constellation- oder Galaxy-Schema genannt [21].

3.4 Aggregation

Die Erstellung zusätzlicher Tabellen zur Speicherung aggregierter Daten hat verschiedene Gründe. Hauptsächlich sollen damit die an OLAP-Systeme geforderten kurzen Antwortzeiten erreicht werden, denn bei einer großen Menge an Detaildaten und einer Abfrage auf hohem Aggregationsniveau wäre das Resultat nicht innerhalb weniger Sekunden verfügbar.

Ein weiterer Grund ist die Reduzierung der Datenmenge, denn auch wenn Speicher in jedweder Form immer größer und günstiger wird, muss die Menge der zu verarbeitenden Daten begrenzt werden. Beim Monitoring von Softwareanwendungen, wie im Anwendungsbeispiel zu dieser Arbeit, entstehen in kurzer Zeit sehr viele Daten, sodass auch hier regelmäßig Detaildaten gelöscht werden müssen, um den Speicherbedarf zu begrenzen.

Eine Aggregationstabelle bezieht sich immer auf eine bestimmte Faktentabelle, wobei es verschiedene Möglichkeiten gibt, Aggregationstabellen zu bilden. Zum Beispiel können eine oder mehrere Dimensionen komplett weggelassen werden, sodass die Tabelle Kennzahlen enthält, die über die weggelassenen Dimensionen aggregiert sind. Diese Technik wird *lost dimension* genannt [10]. In der Beispielanwendung könnte die Dimension Zeit weggelassen werden, was zur Tabelle *agg_ld1_fact* in Abbildung 3.4.1 führt. Ein Eintrag aus dieser Tabelle gibt die durchschnittliche Antwortzeit (*duration_avg*) und die Anzahl der dafür aggregierten Einträge (*fact_count*) einer Methode in einem bestimmten Aufrufkontext wieder. Die Methode und die höheren Level des Systemaufbaus werden über *composition_id* und der Aufrufkontext über *context_id* bestimmt. Aggregiert werden also in diesem Fall alle Einträge aus der Faktentabelle mit gleicher *composition_id* und *context_id*. Da die Zeitdimensionstabelle sehr groß werden kann, würde diese Aggregationstabelle bei Abfragen über die gesamte Zeit eine enorme Performance-Verbesserung einbringen.

In Tabelle *agg_ld2_fact* wurde zusätzlich die Dimension Aufruferkontext weggelassen, wodurch Abfragen nach durchschnittlichen Antwortzeiten von Methoden ohne weitere Einschränkungen schnell beantwortet werden könnten.

Eine weitere Technik besteht darin, über einzelne Level von Dimensionen zu aggregieren. Diese Verkleinerung einer Dimension wird *collapsed dimension* genannt [10]. Tabelle *agg_c1_fact* in Abbildung 3.4.1 zeigt eine mögliche Aggregationstabelle, bei der das niedrigste Level der Dimension Zeit (second) weggelassen wurde. Ein

Eintrag aus dieser Tabelle entspricht also der Aggregation der Detaildaten über eine Minute. Tabelle *agg_c2_fact* lässt das nächste Zeitlevel weg, enthält also Aggregationen über eine Stunde.

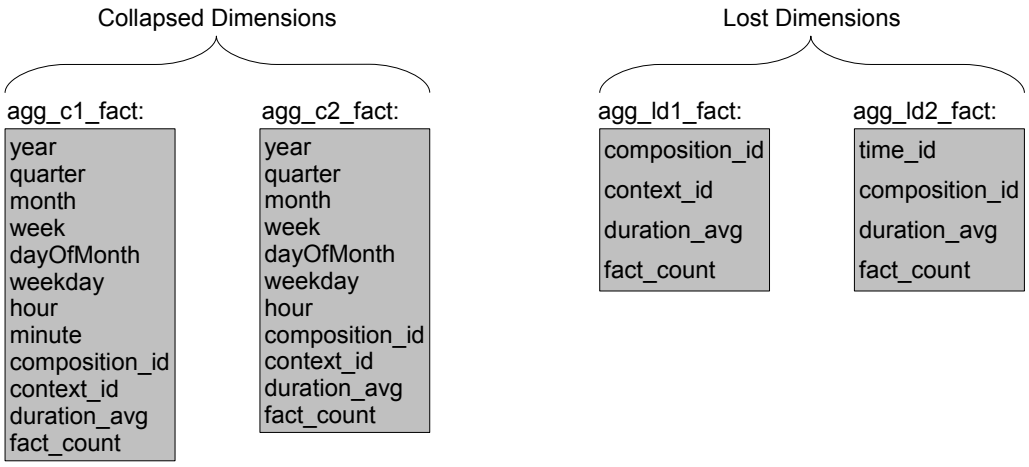

Abbildung 3.4.1: Aggregationen

Es gibt also sehr viele mögliche Aggregationstabellen, die bestimmte Abfragen beschleunigen, allerdings immer mit redundanter Datenhaltung verbunden sind. Im Allgemeinen können nicht alle möglichen Aggregationstabellen vorgehalten werden. Bei der Erstellung einer Datenbasis für OLAP-Systeme müssen also die Aggregationstabellen ausgewählt werden, die den größten Nutzen für das System bringen. Hierbei gilt es zu beachten welche Dimensionen besonders groß werden und welche Abfragen vermutlich häufig gestellt werden.

Für Performance-Analysen ist es ausreichend Detaildaten nur für kurze Zeit zu speichern, da bei Analysen auf lange Sicht Aggregate von größerem Interesse sind. In welcher Sekunde vor einem Monat eine Methode welche Antwortzeit hatte, ist selten von Bedeutung. Es genügt also, die Detaildaten für eine gewisse Zeitspanne vorzuhalten.

Die sekundengenauen Detaildaten müssen regelmäßig zu minutengenauen Daten aggregiert werden, die wiederum regelmäßig zu stundengenauen Daten aggregiert werden. Löscht man dann die Detaildaten, die älter als eine Stunde sind, und die aggregierten Daten, die älter als ein Tag sind, so können Abfragen für die letzte Stunde sekundengenau, für den letzten Tag minutengenau und für ältere Daten stundengenau bedient werden. Durch weitere Aggregationstabellen ließe sich die Datenmenge auf Kosten der Genauigkeit weiter einschränken.

Eine hohe Nutzung der Webseite vorausgesetzt, wird pro Sekunde ein Eintrag in der Zeittabelle erzeugt und in der Anzahl nach oben nicht begrenzte Einträge in der Faktentabelle. Die Dimensionen Aufrufkontext und Systemaufbau hängen hingegen von der Implementierung der Webseite ab und sind somit beide recht klein (beim JPetStore beide <30 Einträge). Gibt es *n* Einträge in der Dimension Aufrufkontext und *m* Einträge in der Dimension Systemaufbau, so ist die Anzahl der Einträge in der Aggregationstabelle *agg_c1_fact* damit auf *n*m* Einträge pro Minute und die Tabelle *agg_c2_fact* auf *n*m* Einträge pro Stunde begrenzt.

Mithilfe solcher Aggregationstabellen lässt sich die aufkommende Datenmenge also gut abschätzen.

3.5 Technische Standards

Mit dem Aufkommen des OLAP-Konzepts Mitte der 90er Jahre wurden auch Standardisierungsversuche zur Vereinheitlichung der Terminologie unternommen, da dies eine wesentliche Voraussetzung für eine schnelle Entwicklung und Verbreitung einer Technik ist. Prominentes Beispiel hierfür ist die Entwicklung der relationalen Datenbanken. Im Folgenden werden die Abfragesprache MDX (Multi-Dimensional eXpression) und die von Mondrian, den in dieser Arbeit verwendeten OLAP-Server, unterstützten Schnittstellen zur Datenübertragung kurz vorgestellt.

Multidimensionale Abfragesprache (MDX)

Die Abfragesprache für multidimensionale Datenräume MDX wurde Ende der 90er Jahre von Microsoft entwickelt und gilt heute als Standard. Die Syntax von MDX mit ihren *Select*-, *From*- und *Where*-Klauseln erinnert stark an die relationale Abfragesprache SQL. Allerdings können im Gegensatz zu SQL mehrdimensionale Würfel das Ergebnis einer Abfrage sein. Die weiteren Schlüsselwörter wie *Dimension*, *Measure*, *Member*, *Cell*, *Level* und *Hierarchie* verdeutlichen den Bezug zum in Kapitel 3.1 beschriebenen multidimensionalen Datenmodell.

Auf eine ausführlichere Beschreibung der Sprachelemente soll an dieser Stelle verzichtet werden, da das in dieser Arbeit verwendete OLAP-Frontend die Generierung der MDX-Abfragen übernimmt und somit eine detailliertere Beschreibung nicht weiter zum Verständnis der Arbeit beigetragen würde. Eine ausführliche Beschreibung der Sprache findet sich in [11].

XML for Analysis (XMLA)

XMLA ist der jüngste Versuch zur Schaffung einer standardisierten API im OLAP-Bereich. Im Gegensatz zu früheren Versuchen erhält XMLA eine breite Unterstützung von Firmen wie Hyperion (jetzt Oracle), Microsoft, SAP und SAS. XMLA ist ein auf SOAP, HTTP und XML basierendes Protokoll, mit dem MDX-Statements zwischen OLAP-Komponenten übers Internet ausgetauscht werden können [2].

Java API (OLAP4J)

OLAP4J ist eine Java-Schnittstelle für den Zugriff auf multidimensionale Daten, wie JDBC für den Zugriff auf relationale Daten.

4 Entwicklung eines OLAP-Plugin für Kieker

In diesem Kapitel werden die Bestandteile der in dieser Arbeit entstandenen Softwarelösung beschrieben. In Kapitel 4.1 wird das Framework, das durch diese Arbeit um eine Komponente erweitert wurde, vorgestellt. Im folgenden Kapitel wird der OLAP-Server Mondrian und seine Funktionsweise erläutert. Im dritten Unterkapitel wird auf das graphische Frontend eingegangen. Abschließend werden der Nutzen des Plugins und Möglichkeiten zur Erweiterung aufgezeigt.

Auf Grundlage des logischen Datenmodells wurde ein relationales Datenbankschema für eine MySQL-Datenbank erstellt (siehe Anhang 5). Diese Datenbank bildet die Basis für den in Kapitel 4.2 beschriebenen OLAP-Server.

4.1 Kieker

Kieker ist ein Monitoring- und Analyse-Framework für Java-Applikationen. Es zeichnet Daten über die Hardwareauslastung und den Kontrollfluss und die Antwortzeiten von Methoden auf und bietet Tools zum Lesen, Analysieren und Visualisieren der gemessenen Daten. Kieker ist so konzipiert, dass es kaum Overhead produziert, damit es für die ständige Kontrolle von Softwaresystemen eingesetzt werden kann und nicht nur zu Testzwecken.

Abbildung 4.1.1 gibt einen Überblick über das Framework bestehend aus den zwei Hauptkomponenten *Kieker.Monitoring* und *Kieker.Analysis*. Die Monitoring-Komponente sorgt für die Aufzeichnung der Daten. Die Analysis-Komponente liest, analysiert und visualisiert die aufgezeichneten Daten. *Kieker.TraceAnalysis* ermöglicht, als ein Plugin von Kieker.Analysis, die Rekonstruktion der Architektur und des Kontrollflusses des überwachten Systems. Weitere Plugins visualisieren die daraus

gewonnen Informationen, indem zum Beispiel Sequenz- oder Abhängigkeitsdiagramme erzeugt werden. Als weitere Visualisierung dieser Daten ist im Rahmen dieser Arbeit ein OLAP-Plugin entstanden, das Informationen über den Systemaufbau, den Kontrollfluss und das Antwortzeitverhalten in einer relationalen Datenbank speichert. Diese Daten werden über den in Kapitel 4.2 beschriebenen OLAP-Server Mondrian in einen multidimensionalen Kontext gesetzt und vom in Kapitel 4.3 beschriebenen Mondrian-Frontend Saiku in der Kieker-Analyseumgebung dargestellt.

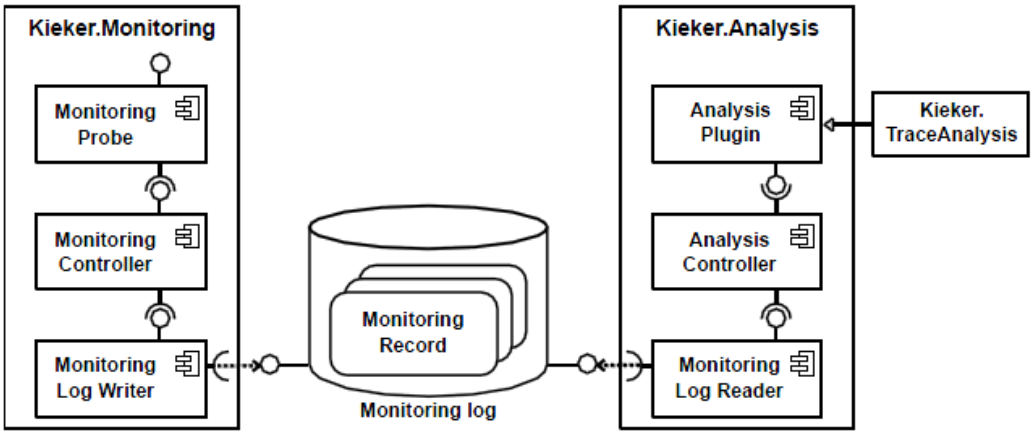

Abbildung 4.1.1: Überblick der Framework-Komponenten (Quelle: [14])

Die Architektur von Kieker und die bereits implementierten Analysemöglichkeiten werden in [3] und [14] detailliert beschrieben.

4.2 Der OLAP-Server Mondrian

Mondrian ist ein Java-basierter OLAP-Server, der auf einer relationalen Datenbank operiert. Mondrian wurde 2001 als Open-Source-Projekt veröffentlicht und ist seit 2006 Teil des Pentaho-Projekts. Die Tabellen der relationalen Datenbank müssen, wie in Kapitel 3.3 beschrieben, im Star- oder Snowflake-Schema angeordnet sein. Die Beziehungen zwischen den relationalen und den multidimensionalen Strukturen werden in einer XML-Konfigurationsdatei, dem sogenannten Mondrian-Schema, definiert. Die Tabellen und Spalten des relationalen Datenbankschemas werden damit auf die Dimensionen und Fakten des multidimensionalen Würfels abgebildet. Listing 4.1 und Listing 4.2 sind Ausschnitte des Mondrian-Schemas

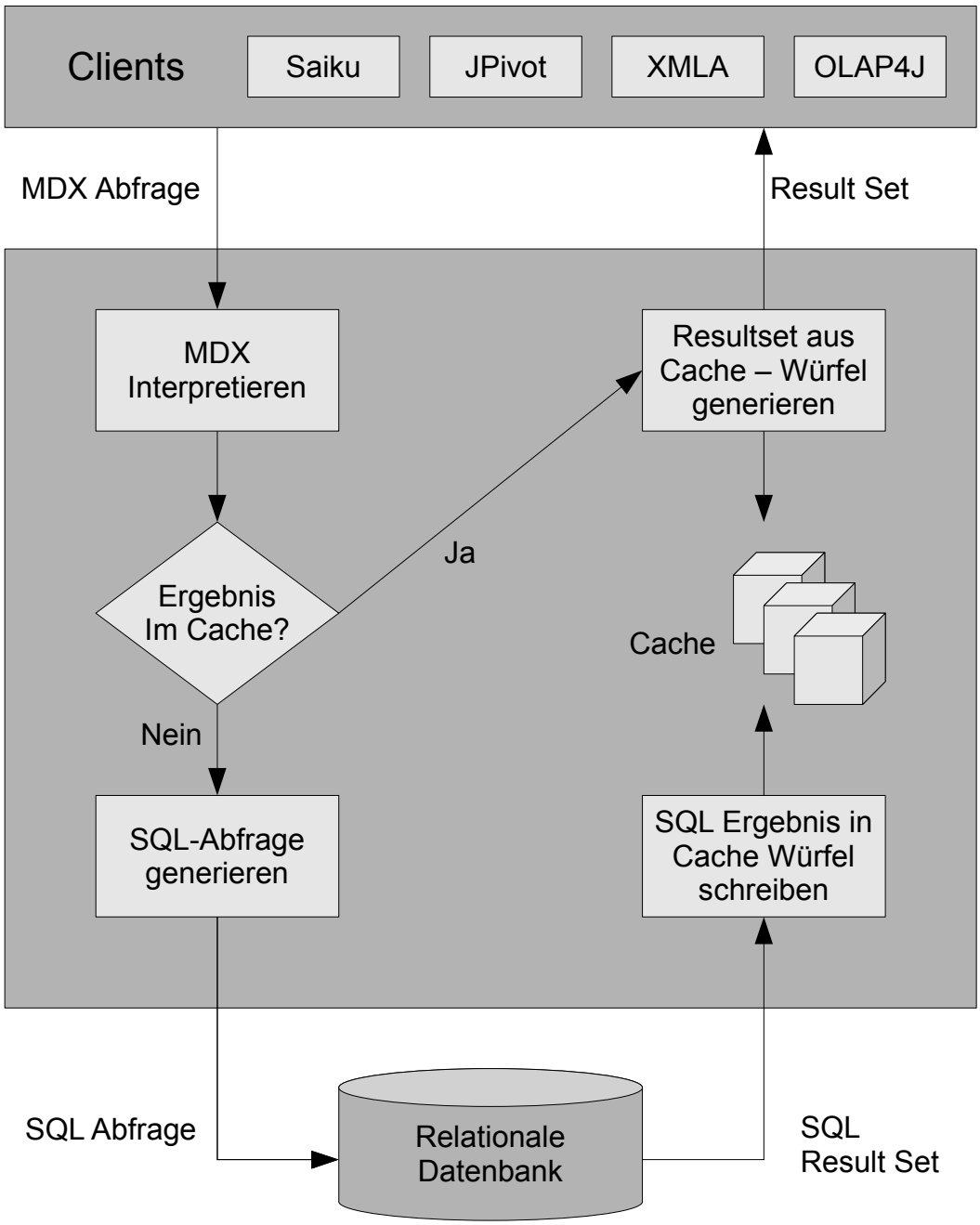

Abbildung 4.2.1: Überblick Mondrian OLAP-Server

der Beispielanwendung. Das gesamte Schema ist in Anhang 5 abgebildet. In Listing 4.1 ist beispielhaft gezeigt, wie die Spalten der Dimensionstabellen auf die Hierarchielevel der Dimensionen des Würfels abgebildet werden.

Liegen die Daten in einem Snowflake-Schema vor, wird die Schema-Definition we-

Listing 4.1: Ausschnitt Dimension

```
<Dimension name="System Composition">
  <Hierarchy hasAll="true" primaryKey="composition_id">
    <Table name="system_composition"/>
    <Level name="Container" column="container" type="String" uniqueMembers="
        true"/>
    <Level name="Package" column="package" type="String" uniqueMembers="true"/
        >
    <Level name="Class" column="class" type="String" uniqueMembers="false"/>
    <Level name="Method" column="method" type="String" uniqueMembers="false"/>
  </Hierarchy>
</Dimension>
```

sentlich komplexer, da die Dimensionen über Join-Konstrukte aus verschiedenen Tabellen erzeugt werden müssen.

Listing 4.2 zeigt eine Würfeldefinition über der Faktentabelle *fact*. Sie enthält neben den Kennzahlen des Würfels und den Dimensionen, von denen diese Kennzahlen abhängen, auch die Beschreibung einer Aggregationstabelle. Diese sorgt wie in Kapitel 3.4 beschrieben für eine Performancesteigerung. Es können auch Kennzahlen angegeben werden, die nicht aus der Faktentabelle stammen, sondern vom OLAP-Server berechnet werden, wie hier *average response time*.

Die Dimensionen können auch als Teil des Würfels definiert werden. Die separate Definition hat den Vorteil, dass die Dimensionen in anderen Würfeln desselben Schemas genutzt werden können. 10 enthält eine detaillierte Anleitung zur Erstellung eines Mondrian-Schemas.

Mondrian nutzt die in Kapitel 3.5 vorgestellte Abfragesprache *MDX* für Abfragen auf den multidimensionalen Datenstrukturen. Die Funktionsweise des Mondrian-OLAP-Servers ist in Abbildung 4.2.1 übersichtlich dargestellt. Beim Start des Servers wird das Mondrian-Schema geladen und validiert. Der Server nimmt MDX-Abfragen eines Clients entgegen, interpretiert diese anhand des erstellten Mondrian-Schemas und gibt das Ergebnis zurück, falls es bereits im Cache ist, oder anhand der Cache-Würfel generiert werden kann. Andernfalls wird mittels des Mondrian-Schemas die multidimensionale Abfrage in SQL-Abfragen übersetzt und auf der relationalen Datenbank ausgeführt. Das Ergebnis wird in multidimensionaler Form in den Cache geschrieben und an den Client gesendet. Eine detailliertere Darstellung und Beschreibung der Architektur des Mondrian-OLAP-Servers ist unter [10] zu finden.

Es existieren auch Open-Source-Projekte im Bereich des multidimensionalen OLAP wie Palo, allerdings wurde dies im Rahmen dieser Arbeit aufgrund der in Kapitel

Listing 4.2: Ausschnitt Würfel

```
<Cube name="Performance">
  <Table name="fact">
    <AggName name="agg_c1_fact">
      <AggFactCount column="fact_count"/>
      <AggForeignKey factColumn="composition_id" aggColumn="composition_id"/>
      <AggForeignKey factColumn="context_id" aggColumn="context_id"/>
      <AggMeasure name="[Measures].[avg]" column="duration_avg"/>
      <AggLevel name="[Time].[Year]" column="year"/>
      <AggLevel name="[Time].[Quarter]" column="quarter"/>
      <AggLevel name="[Time].[Month]" column="month"/>
      <AggLevel name="[Time].[DayOfMonth]" column="dayOfMonth"/>
      <AggLevel name="[Time].[Hour]" column="hour"/>
      <AggLevel name="[Time].[Minute]" column="minute"/>
    </AggName>
  </Table>

  <DimensionUsage name="Time" source="Time" foreignKey="time_id"/>
  <DimensionUsage name="System Composition" source="System Composition"
    foreignKey="composition_id"/>
  <DimensionUsage name="Calling Context" source="Calling Context"
    foreignKey="context_id"/>

  <Measure name="total viewings" column="composition_id" aggregator="count"
    formatString="Standard"/>

  <Measure name="avg" column="duration" aggregator="avg" visible="false"/>

  <!-- Umrechnung von Nano- in Millisekunden und Formatierung-->
  <CalculatedMember name="average response time" dimension="Measures"
    formula="[Measures].[avg]/1000000">
    <CalculatedMemberProperty name="FORMAT_STRING" value="0.00 ms"/>
  </CalculatedMember>
</Cube>
```

2.4 beschriebenen Beschränkung der zu verarbeitenden Datenmenge nicht als Alternative betrachtet.

4.3 Das OLAP-Frontend Saiku

Saiku ist ein leichtgewichtiges modulares Open-Source-OLAP-Frontend. Es ist einfach integrierbar, erweiterbar und konfigurierbar. Es nutzt OLAP4J als Schnittstelle zum Server und bietet nach außen eine REST-Schnittstelle, über die die Saiku-Webapp die Abfragen stellt [1]. Die Ergebnisse der Abfragen werden in Kreuztabellen dargestellt. Die intuitiv zu bedienende Oberfläche bietet die in Kapitel 3.2 beschriebenen Funktionen zur nutzerfreundlichen Formulierung von Abfragen. Weiter wird die Möglichkeit geboten Abfragen zu speichern und Ergebnisse zu exportieren.

Am linken Rand der Oberfläche befindet sich ein Drop-down-menü, aus dem man

einen der zur Verfügung stehenden Würfel wählen kann. Wurde ein Würfel gewählt, werden darunter in einem Baummenü dessen Dimensionen mit entsprechender Hierarchisierung angezeigt. Unter den Dimensionen befinden sich die zur Verfügung stehenden Kennzahlen. Per *Drag and Drop* können nun die zu betrachtenden Dimensionslevel und Kennzahlen auf die Zeilen oder Spalten gelegt werden. Durch einen einfachen Doppelklick auf eines der ausgewählten Dimensionslevel kann dieses auf beliebige Elemente eingeschränkt werden, was einer Slice- bzw. Dice-Operation entspricht.

Als Beispiel der durch das Plugin gewonnenen Funktionalität wurden die Monitoring-Daten des JPetStore wie beschrieben durch das Plugin verarbeitet und auf diesen Beispieldaten zwei Abfragen formuliert. In Abbildung 4.3.1 wurde die Abfrage *„Wie hoch sind die durchschnittlichen Antwortzeiten der beobachteten Methoden im Tagesverlauf?"* betrachtet. Dazu wurden die Kennzahl *average response time* und das Zeitlevel *hour* auf die Spalten und das Methodenlevel des Systemaufbaus auf die Zeilen gelegt.

In Abbildung 4.3.2 wurde die Frage *„Wie oft und mit welcher durchschnittlichen Antwortzeit wurden im September Methoden mit verschiedenem Aufrufkontext aufgerufen?"* formuliert. Dies soll unterstreichen, wie einfach auch komplexe Abfragen zu formulieren sind. Dazu wurden der Übersicht halber die Methoden und ihr zugehöriger *callstack* auf die Zeilen und die Kennzahlen *total viewings* und *average response time* auf die Spalten gelegt. Dann wurde die Tabelle auf die Methoden eingeschränkt, die über unterschiedliche *callstacks* aufgerufen wurden. Zuletzt wurde das Monatslevel als Filter genutzt und darin der Monat September gewählt. Zu sehen ist jeweils, wie viele Elemente des Methodenlevels und des Monatslevels ausgewählt wurden.

Die etablierte Alternative JPivot bietet zwar noch etwas mehr Funktionalität, jedoch gleicht Saiku dies durch eine gelungene Oberfläche mit intuitiver Menüführung wieder aus. Die erste Version von Saiku wurde 2010 veröffentlicht, die momentan aktuelle Version ist 2.1 [4]. Damit ist Saiku ein sehr junges Projekt, sodass weitere Funktionalität erwartet werden kann.

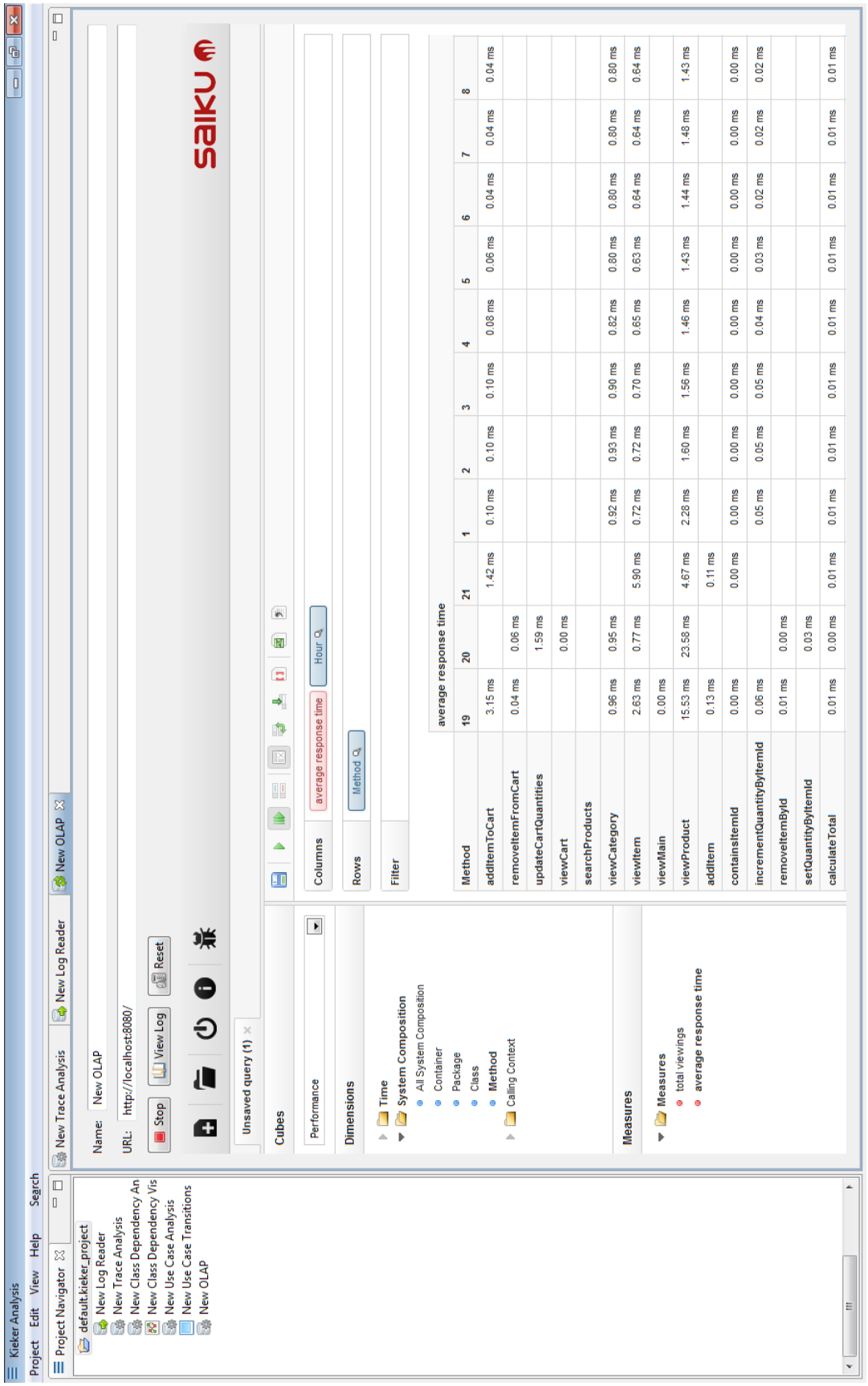

Abbildung 4.3.1: Beispielabfrage in entstandener Softwarelösung

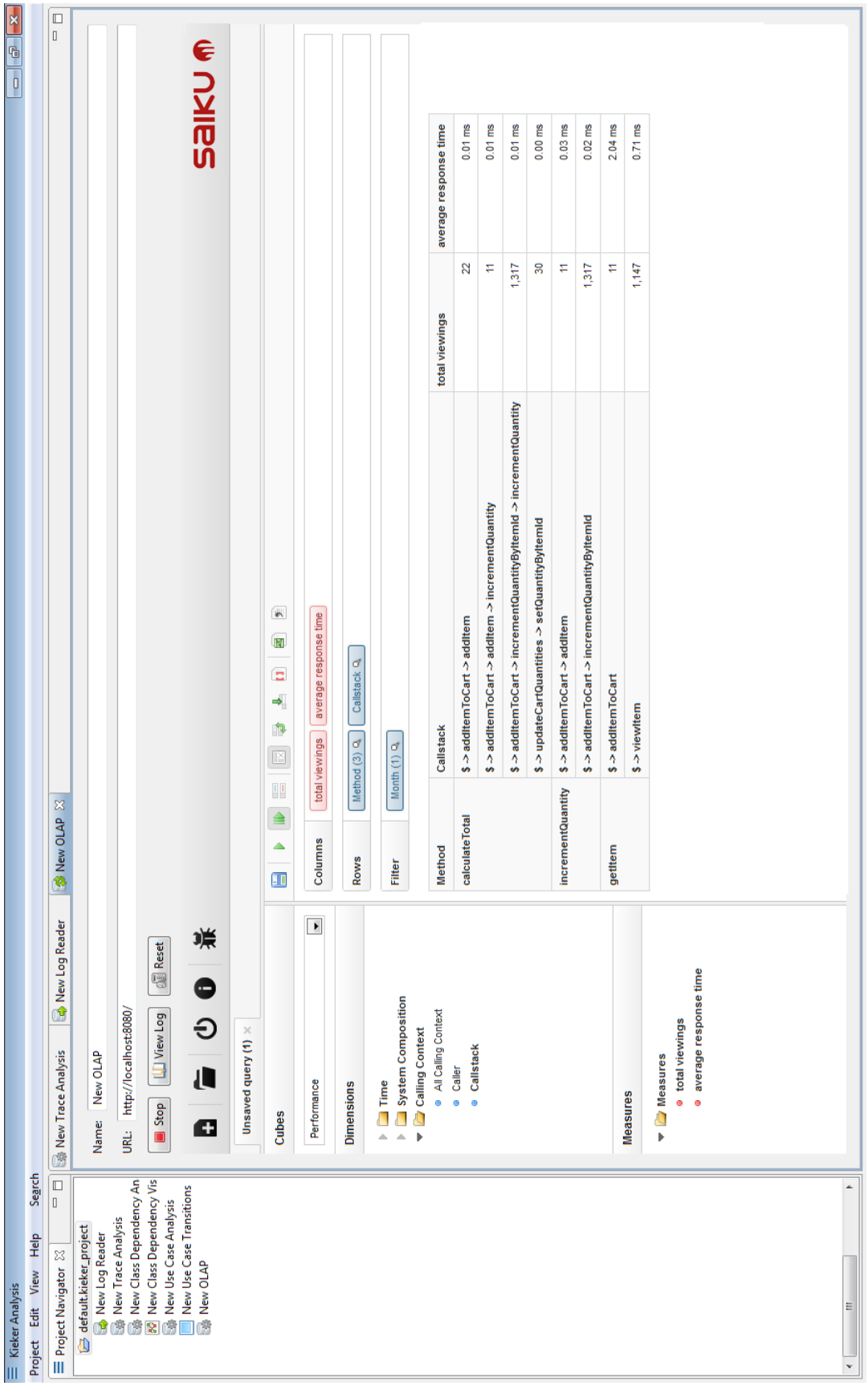

Abbildung 4.3.2: Komplexe Abfrage

4.4 Erweiterbarkeit der Anwendung

Durch das OLAP-Plugin wurde das Monitoring- und Analyse-Framework Kieker um eine weitere Möglichkeit zur Visualisierung der Performance-Informationen und Analyse dieser geschaffen. In dieser Arbeit wurde zur Analyse ein Würfel, mit drei Dimensionen und zwei Kennzahlen, erstellt. Für den praktischen Einsatz wären sicherlich weitere Größen interessant. Als weitere Kennzahl für diesen Würfel könnte der Durchsatz angegeben werden, aber auch ein weiterer Würfel, der die Systemauslastung der Hardwarekomponenten über die Zeit wiedergibt, und so einen Bezug zwischen Durchsatz, Antwortzeit und Systemauslastung zulässt, ist denkbar.

Im Bereich der Darstellung wären weitere graphische Aufbereitungen der Informationen als Alternativen zu den Kreuztabellen interessant, wie beispielsweise Säulen-, Balken- oder Kreisdiagramme zum Vergleich bestimmter Kennzahlen oder Kurven, um Entwicklungen und Tendenzen besser erkennen zu können. Letztere würden sich zum Beispiel hervorragend eignen, um das Antwortzeitverhalten der Methoden im Verlauf der Zeit aus Abbildung 4.3.1 darzustellen. Für OLAP-Anwendungen wird zwar explizit gefordert, Abfragen ohne jegliche Programmierkenntnisse formulieren zu können, jedoch könnte es für Analysten mit Kenntnissen in der Abfragesprache MDX interessant sein, die erzeugte MDX-Abfrage nicht nur betrachten, sondern auch verändern zu können.

5 Zusammenfassung und Fazit

Die vorliegende Arbeit hat die Modellierung eines multidimensionalen Datenmodells, wie es einem OLAP-System zugrunde liegt, und die nötigen Schritte zum operativen Einsatz aufgezeigt. Im Praxisteil wurde ein Modul zur multidimensionalen Performance-Analyse für das Monitoring- und Analyse-Framework Kieker erstellt. Dazu wurden die Open-Source-Komponenten Mondrian und Saiku in die Anwendung integriert.

Nach einer einleitenden Einführung in das Thema wurden im zweiten Kapitel die Grundlagen zum Verständnis der Arbeit gelegt und mit der Definition von Online Analytical Processing durch die FASMI-Regeln nicht nur die wichtigste Grundlage dieser Arbeit erklärt, sondern auch konkrete Anforderungen an die zu entwickelnde Software gestellt.

Das dritte Kapitel bildet mit der multidimensionalen Modellierung einen Schwerpunkt der Arbeit. Die Strukturen multidimensionaler Datenräume wurden anhand der für das Anwendungsbeispiel interessanten Größen zur Analyse der Performance beschrieben. Des Weiteren wurde die Navigation in diesen Datenräumen erklärt und die nötige Transformation des Modells in logische Strukturen beschrieben. Weiter wurde auf die Wichtigkeit von Aggregationen in der praktischen Anwendung eingegangen und die technischen Standards zur Verarbeitung mehrdimensionaler Ausdrücke vorgestellt.

Im letzten Kapitel wurde eine Übersicht über das zu erweiternde Framework Kieker gegeben. Des Weiteren wurden der OLAP-Server Mondrian und das OLAP-Frontend Saiku detailliert betrachtet, da sie Bestandteile der Softwarelösung dieser Arbeit sind. Abschließend wurden die Erweiterungsmöglichkeiten der entstandenen Anwendung aufgezeigt.

Die erstellte Software erlaubt dem Anwender eine Analyse der Performance eines Java-basierten Softwaresystems. Dabei wird die Systemarchitektur abgebildet und es können Antwortzeiten einzelner Methoden, oder aggregiert über Klassen oder

Pakete betrachtet werden. Des Weiteren kann der Kontrollfluss detailliert aufgezeigt werden. Die entsprechenden Abfragen können über die Oberfläche, wie in den Abbildungen 4.3.1 und 4.3.2 dargestellt, intuitiv formuliert werden. Die Lösung erfüllt somit die Anforderungen an OLAP-Systeme wie sie in Kapitel 2.4 formuliert wurden.

Literaturverzeichnis

[1] Saiku, 2011. URL http://www.analytical-labs.com/. zuletzt besucht 29. September 2011.

[2] What is xml for analysis (xmla)?, 2011. URL http://news.xmlforanalysis.com/what-is-xmla.html. zuletzt besucht 29. September 2011.

[3] Wilhelm Hasselbring JanWaller Jens Ehlers Sören Frey Dennis Kieselhorst André van Hoorn, Matthias Rohr. Continuous monitoring of software services: Design and application of the kieker framework, 2009. URL http://www.informatik.uni-kiel.de/uploads/tx_publication/vanhoorn_tr0921.pdf. zuletzt besucht 29. September 2011.

[4] Tom Barber. About analytical labs, 2011. URL http://projects.analytical-labs.com/knowledgebase/articles/8. zuletzt besucht 29. September 2011.

[5] Steffen Becker, Lars Grunske, Raffaela Mirandola, and Sven Overhage. Performance prediction of component-based systems. In Ralf Reussner, Judith Stafford, and Clemens Szyperski, editors, *Architecting Systems with Trustworthy Components*, volume 3938 of *Lecture Notes in Computer Science*, pages 169–192. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-35800-8.

[6] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. *Model-Based Software Performance Analysis*. Springer, 2011. ISBN 978-3-6421-3620-7.

[7] Jens Ehlers. Multidimensionale Analyse und Planung mit Open Source-Komponenten am Beispiel des Finanzdienstleistungsbereiches. Master's thesis, Uni Münster, 2008.

[8] Michael Hahne. Logische datenmodellierung für das data warehouse. In *Analytische Informationssysteme 3.Aufl.* Springer, 1998. ISBN 3-540-63364-2.

[9] Michael Hahne. Mehrdimensionale datenmodellierung für analyseorientierte informationssysteme. In *Analytische Informationssysteme 4.Aufl.* Springer, 2010. ISBN 978-3-642-04815-9.

[10] Julian Hyde. Mondrian documentation. URL `http://mondrian.pentaho.com/documentation/`. zuletzt besucht 29. September 2011.

[11] Mosha Pasumansky Mark Whitehorn, Robert Zare. *Fast Track to MDX*. Springer, 2002. ISBN 1-85233-681-1.

[12] Justus Marquardt. *Metadatendesign zur Integration von Online Analytical Processing in das Wissensmanagement*. Dr. Kovac, 2008. ISBN 978-3-8300-3598-5.

[13] M. D. McIlroy. Mass produced software components. In *Software Engineering: Report of a conference sponsored by the NATO Science Committee*, 1968.

[14] André van Hoorn Nils Ehmke and Reiner Jung. Kieker 1.2 user guide, 2010.

[15] Nigel Pendse. What is olap? an analysis of what the often misused olap term is supposed to mean, 2008. URL `http://www.olapreport.com/fasmi.htm`. zuletzt besucht 29. September 2011.

[16] Helge Petersohn. *Data mining : Verfahren, Prozesse, Anwendungsarchitektur*. Oldenbourg Verlag, 2005. ISBN 978-3-486-57715-0.

[17] Niklas Schlimm, Mirko Novakovic, Robert Spielmann, and Tobias Knierim. Performance-analyse und -optimierung in der softwareentwicklung. *Informatik Spektrum*, 30(4):251–258, 2007.

[18] Connie U. Smith and Lloyd G. Williams. New book - performance solutions: A practical guide to creating responsive, scalable software. In *Int. CMG Conference*, pages 355–358, 2001.

[19] Clemens A. Szyperski. *Component software - beyond object-oriented programming*. Addison-Wesley-Longman, 1998. ISBN 978-0-2011-7888-3.

[20] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009. ISBN 0470167742.

[21] Gottfried Vossen. *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, 5. Auflage*. Oldenbourg, 2008. ISBN 978-3-4862-7574-2.

Anhang

Relationales Schema

```
CREATE TABLE `time` (
  `time_id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `timestamp` TIMESTAMP NOT NULL,
  `year` SMALLINT NOT NULL,
  `quarter` VARCHAR( 2 ) NOT NULL,
  `month` VARCHAR( 10 ) NOT NULL,
  `week` TINYINT NOT NULL,
  `dayOfMonth` TINYINT NOT NULL,
  `weekday` VARCHAR( 10 ) NOT NULL,
  `hour` TINYINT NOT NULL,
  `minute` TINYINT NOT NULL,
  `second` TINYINT NOT NULL DEFAULT '0'
);

CREATE TABLE `system_composition` (
  `composition_id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `container` VARCHAR( 30 ) NOT NULL,
  `package` VARCHAR( 100 ) NOT NULL,
  `class` VARCHAR( 30 ) NOT NULL,
  `method` VARCHAR( 30 ) NOT NULL
);

CREATE TABLE `context` (
  `context_id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `caller` VARCHAR( 30 ) NOT NULL,
  `callstack` VARCHAR( 150 ) NOT NULL
);

CREATE TABLE `fact` (
  `time_id` INT NOT NULL,
  `composition_id` INT NOT NULL,
  `context_id` INT NOT NULL,
  `duration` INT NOT NULL,
  Foreign Key (`time_id`) references `time`(`time_id`) ON DELETE CASCADE,
  Foreign Key (`composition_id`) references `system_composition`(`
      composition_id`),
  Foreign Key (`context_id`) references `context`(`context_id`)
);
```

```
CREATE TABLE `agg_c1_fact` (
  `timestamp` TIMESTAMP NOT NULL,
  `year` SMALLINT NOT NULL,
  `quarter` VARCHAR( 2 ) NOT NULL,
  `month` VARCHAR( 10 ) NOT NULL,
  `week` TINYINT NOT NULL,
  `dayOfMonth` TINYINT NOT NULL,
  `weekday` VARCHAR( 10 ) NOT NULL,
  `hour` TINYINT NOT NULL,
  `minute` TINYINT NOT NULL,
  `composition_id` INT NOT NULL,
  `context_id` INT NOT NULL,
  `duration_avg` BIGINT NOT NULL,
  `fact_count` INT NOT NULL,
  Foreign Key (`composition_id`) references `system_composition`(`
      composition_id`),
  Foreign Key (`context_id`) references `context`(`context_id`)
);
```

Mondrian-Schema

```
1  <?xml version="1.0"?>
2  <Schema name="Pet Store">
3
4    <Dimension name="System Composition">
5      <Hierarchy hasAll="true" primaryKey="composition_id">
6        <Table name="system_composition"/>
7        <Level name="Container" column="container" type="String" uniqueMembers="
             true"/>
8        <Level name="Package" column="package" type="String" uniqueMembers="true
             "/>
9        <Level name="Class" column="class" type="String" uniqueMembers="false"/>
10       <Level name="Method" column="method" type="String" uniqueMembers="false"
             />
11     </Hierarchy>
12   </Dimension>
13
14   <Dimension name="Calling Context">
15     <Hierarchy hasAll="true" primaryKey="context_id">
16       <Table name="context"/>
17       <Level name="Caller" column="caller" type="String" uniqueMembers="false"
             />
18       <Level name="Callstack" column="callstack" type="String" uniqueMembers="
             false"/>
19     </Hierarchy>
20   </Dimension>
21
22   <Dimension name="Time" type="TimeDimension">
23     <Hierarchy hasAll="false" primaryKey="time_id">
24       <Table name="time"/>
25       <Level name="Year" column="year" type="Numeric" uniqueMembers="true"
26                levelType="TimeYears"/>
27       <Level name="Quarter" column="quarter" type="String" uniqueMembers="true
             "
28           levelType="TimeQuarters"/>
29       <Level name="Month" column="month" type="String" uniqueMembers="true"
30           levelType="TimeMonths"/>
31       <Level name="DayOfMonth" column="dayOfMonth" type="Numeric"
             uniqueMembers="true"
32           levelType="TimeDays"/>
33       <Level name="Hour" column="hour" type="Numeric" uniqueMembers="true"
34           levelType="TimeHours"/>
35       <Level name="Minute" column="minute" type="Numeric" uniqueMembers="true"
36           levelType="TimeMinutes"/>
37       <Level name="Second" column="second" type="Numeric" uniqueMembers="true"
38           levelType="TimeSeconds"/>
39     </Hierarchy>
40     <Hierarchy hasAll="true" name="Weekly" primaryKey="time_id">
41       <Table name="time"/>
42       <Level name="Year" column="year" type="Numeric" uniqueMembers="true"
43           levelType="TimeYears"/>
44       <Level name="Week" column="week" type="Numeric" uniqueMembers="true"
45           levelType="TimeWeeks"/>
```

```
46        <Level name="Weekday" column="weekday" type="String" uniqueMembers="true
              "
47            levelType="TimeDays"/>
48        <Level name="Hour" column="hour" type="Numeric" uniqueMembers="true"
49            levelType="TimeHours"/>
50        <Level name="Minute" column="minute" type="Numeric" uniqueMembers="true"
51            levelType="TimeMinutes"/>
52        <Level name="Second" column="second" type="Numeric" uniqueMembers="true"
53            levelType="TimeSeconds"/>
54      </Hierarchy>
55    </Dimension>
56
57  <Cube name="Performance">
58    <Table name="fact">
59      <AggName name="agg_c1_fact">
60        <AggFactCount column="fact_count"/>
61        <AggForeignKey factColumn="composition_id" aggColumn="composition_id"/>
62        <AggForeignKey factColumn="context_id" aggColumn="context_id"/>
63        <AggMeasure name="[Measures].[avg]" column="duration_avg"/>
64        <AggLevel name="[Time].[Year]" column="year"/>
65        <AggLevel name="[Time].[Quarter]" column="quarter"/>
66        <AggLevel name="[Time].[Month]" column="month"/>
67        <AggLevel name="[Time].[DayOfMonth]" column="dayOfMonth"/>
68        <AggLevel name="[Time].[Hour]" column="hour"/>
69        <AggLevel name="[Time].[Minute]" column="minute"/>
70      </AggName>
71    </Table>
72
73    <DimensionUsage name="Time" source="Time" foreignKey="time_id"/>
74    <DimensionUsage name="System Composition" source="System Composition"
75      foreignKey="composition_id"/>
76    <DimensionUsage name="Calling Context" source="Calling Context"
77      foreignKey="context_id"/>
78
79    <Measure name="total viewings" column="composition_id" aggregator="count"
80      formatString="Standard"/>
81
82    <Measure name="avg" column="duration" aggregator="avg" visible="false"/>
83
84    <!-- Umrechnung von Nano- in Millisekunden und Formatierung-->
85    <CalculatedMember name="average response time" dimension="Measures"
86      formula="[Measures].[avg]/1000000">
87      <CalculatedMemberProperty name="FORMAT_STRING" value="0.00 ms"/>
88    </CalculatedMember>
89
90  </Cube>
91
92  </Schema>
```

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Kiel, 30. September 2011

Unterschrift