Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3270

Agiles Data Warehousing am Beispiel von Prozessmonitoring

Agile Data Warehousing to support Process Monitoring

Philipp Sigloch

Studiengang:	Softwaretechnik
Prüfer:	*Dr. Holger Schwarz*
Betreuer:	*Dipl.Inf. Falko Kötter*
begonnen am:	15. November 2011
beendet am:	15. Mai 2012
CR-Klassifikation:	H.2, H.3

Abstract

Heutige Geschäftsprozesse nehmen ständig an Komplexität zu. Hiermit einhergehend werden die Anpassung und Optimierung einzelner Teilprozesse oder des Gesamtprozesses nötig. Gleichzeitig können über alle Bereiche eines Prozesses hinweg Daten gesammelt werden, welche durch Analyse diese Anpassung und Optimierung ermöglichen. Hierfür müssen die anfallenden Daten dauerhaft gespeichert und bereitgestellt werden. Dies geschieht in Form von Data Warehouses. Da die Erstellung und Wartung dieser Data Warehouse Strukturen teuer und zeitaufwändig ist, werden Versuche unternommen, die hierfür nötigen Arbeiten zu minimieren bzw. zu automatisieren.

In dieser Arbeit wird, aufbauend auf Praxisanforderungen und bisherigen Ansätzen, zur automatisierten Prozessüberwachung ein Konzept für die Erzeugung eines agilen Data Warehouses entwickelt. Dies erfordert die automatisierte Erstellung und Einrichtung der für ein Data Warehouse nötigen Strukturen sowie deren automatische Anpassung bei Veränderung der Umgebungsbedingungen. Zusätzlich müssen Möglichkeiten für das Befüllen dieser Strukturen mit Daten geschaffen werden. Hierfür werden zwei Komponenten entwickelt, welche alle benötigten Data Warehouse Strukturen erzeugen und diese mit Hilfe eines BI-Servers bereitstellen. Hierbei werden Datenbankschemata, OLAP-Schemata sowie ETL-Prozesse automatisiert erzeugt.

Das entwickelte Konzept wird in einen Gesamtprozess integriert, welcher eine Business Intelligence Lösung als Software-As-A-Service-Lösung ermöglicht.

Abstract

Today's business processes are constantly increasing in complexity. This necessitates the adaptation and optimization of individual processes or parts of the overall process. Data is collected during the whole process, allowing analysis for following adaption and optimization of the process. To make this possible, the collected data has to be permanently stored and made available by a data warehouse. Since the creation and maintenance of such a data warehouse is expensive and time-consuming the goal is to minimize or automate this development process.

This work presents a concept for the creation of an agile data warehouse, based on practical requirements and existing approaches for automated process monitoring. This requires the automatic creation and configuration of the data neccessary for a data warehouse, as well as their automatic adaption to changes. In addition a possibility to fill these structures with the collected data is created. Therefore, two components will be implemented, which generate all the required configurations and deploys those information to a BI-Server. For this purpose we create database schemas, OLAP schemas and ETL processes automaticly.

The developed concept is integrated into an overall process, which allows a business intelligence solution to be run as Software-As-A-Service.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Heutige Geschäftsprozesse nehmen ständig an Komplexität zu. Hiermit einhergehend werden auch die Anpassung und Optimierung einzelner Teilprozesse oder des Gesamtprozesses nötig. Damit dies möglich wird, werden an vielen Stellen eines solchen Prozesses Informationen, Kennzahlen und weitere Kerndaten erfasst und gespeichert. Diese Daten sind nötig für die Anpassung und Erweiterung des Prozessablaufs und müssen daher überwacht und ausgewertet werden. Sie können jedoch auch wichtige Informationen in Bezug auf Kundenverhalten, zukünftige Produkte und Absatzwege enthalten.

Diese Daten auszuwerten und aufzuarbeiten ist eine schwierige Aufgabe. Erschwert wird dies unter anderem dadurch, dass die erfassten Informationen dezentral und auf unterschiedliche Weise erfasst und gespeichert werden. Hinzu kommt, dass der Bezug zwischen Daten von unterschiedlichen Speicherorten meist nicht vollständig vorliegt oder aufgrund des Speicherformates nicht eindeutig herzustellen ist. Entsprechend müssen die gesammelten Informationen aus ihren einzelnen, lokal isolierten Speicherorten in eine zentrale Struktur überführt werden. Während dieses Prozesses müssen sie aufgearbeitet und in Abhängigkeit zueinander gesetzt werden. Dies ermöglicht es im Anschluss die erfassten Daten auszuwerten und auf ihrer Basis eine Entscheidung für die Optimierung oder Veränderung des zugrundeliegenden Prozesses zu treffen.

Die hierfür nötige zentrale Struktur wird in vielen Fällen mit Hilfe eines Data Warehouses realisiert [BG04]. Gefüllt mit den zuvor strukturierten und zueinander in Bezug gesetzten Daten bietet es eine einfache, aber mächtige Möglichkeit für deren Auswertung. Jedoch ist die Erzeugung eines Data Warehouses ein aufwändiger und zeitintensiver Vorgang [Lio6]. Neben der Erzeugung einer passenden Datenstruktur müssen in vorgelagerten Prozessen Daten aufgearbeitet werden. Hinzu kommt, dass bei Änderungen des Geschäftsprozesses, der erfassten Daten oder der Speicherart diese Vorgänge ebenfalls angepasst werden müssen.

Aufgrund der großen Anzahl der aufeinanderfolgenden und ineinander greifenden Einzel- und Prozessschritte, welche zum Betrieb eines Data Warehouse nötig sind, wird die Erstellung, Pflege und Weiterentwicklung von entsprechend ausgebildeten IT-Fachkräften durchgeführt. Bei einem neuen Projekt kann die Erzeugung der nötigen Strukturen simultan zur Projektplanung und -durchführung stattfinden. Somit stehen die benötigten Strukturen und Funktionen bereits zeitnah zu Beginn der Produktiveinführung zur Verfügung. Bei bereits existierenden Prozessen ist das nachträgliche Anbinden eines Data Warehouses dagegen ein aufwändiger Prozess, da bestehende Daten sowie Strukturen berücksichtigt und gegebenenfalls migriert oder transformiert werden müssen. Dies führt dazu, dass möglicherweise wichtige Informationen erst zeitverzögert in Entscheidungsprozesse integriert werden können [Lio6]. Des Weiteren stellt sich jedoch, wie bei allen Geschäftsentscheidungen, die

Kostenfrage, welche gerade für mittelständische Unternehmen von zentraler Bedeutung ist.

Ein Ziel bei der Forschung an Data Warehouses ist es daher, den Aufwand zur Erstellung und Einrichtung zu minimieren und diese agil zu gestalten. Dies hat den Vorteil, dass auch auf nachträgliche Änderungen an den Anforderungen oder Ausgangsdaten schnell eingegangen werden kann. Voraussetzung hierfür ist eine schnelle und fehlerfreie Anpassung der vorhandenen Umgebung, ohne Verlust der Aussagekraft und Qualität der aufbereiteten Daten. Dies würde es ermöglichen ein Data Warehouse als Teil einer Software-As-A-Service Lösung anzubieten, ohne hohe Kosten zu erzeugen oder Personal zu binden.

1.1. Aufgabenstellung

Ziel dieser Arbeit ist es, ein Werkzeug für die agile Erstellung eines Data Warehouses zu entwickeln und zu implementieren. Agil ist in diesem Zusammenhang wie folgt definiert: automatisierte Erstellung und Einrichtung der für ein Data Warehouse nötigen Strukturen sowie deren automatische Anpassung bei Veränderung der Umgebungsbedingungen. Zusätzlich müssen Möglichkeiten für das Befüllen dieser Strukturen mit Daten geschaffen werden. Als Grundlage hierfür dient eine Definition der zu messenden Ziele und Key Performance Indikatoren (KPIs).

Um dies umsetzen zu können, sollen zunächst bereits vorhandene Ansätze für die Erzeugung eines agilen Data Warehouses untersucht und bewertet werden. Auch der aktuelle Stand der Forschung bei der Erzeugung von Data Warehouse-Strukturen soll hierbei untersucht werden.

Mit Hilfe der hieraus gewonnenen Erkenntnisse soll eine Möglichkeit erarbeitet werden, welche es erlaubt aus definierten Zielen und KPIs Kerndokumente zu erstellen, die für die Erzeugung eines Data Warehouses benötigt werden. Als Kerndokumente gelten Datenbankschemata, OLAP-Schemata sowie zugehörige Extract-Transfer-Load (ETL) Anweisungen. Diese können im Anschluss genutzt werden, um eine automatische Einrichtung eines Data Warehouses zu realisieren.

Die entstandenen Strukturen sollen im Anschluss so erweitert werden, dass sich das Warehouse automatisch auf Veränderungen in den Zielen und KPIs einstellt. Hierfür muss das eingesetzte Data Warehouse entsprechend erweitert werden und eine Schnittstelle zur Bekanntgabe der Änderungen im Geschäftsprozess bereitgestellt werden. Ziele und KPIs werden hierbei in Form einer XML-Datei, der sogenannten ProGoalML, kommuniziert. Am Ende sollte es möglich sein Monitoring- und Analysewerkzeuge auf Basis des Data Warehouse zu betreiben.

Die implementierte prototypische Software sowie deren Schnittstellen und Architektur müssen vollständig dokumentiert werden.

1.2. Aufbau der Arbeit

Kapitel 1 bietet eine kurze Einführung in das Thema dieser Arbeit. Des Weiteren findet sich hier eine Übersicht über die Aufgabenstellung und die daraus resultierenden Ziele.

In Kapitel 2 finden sich Grundlagen, die für das Verständnis der Entscheidungsprozesse und Entwicklungsschritte nötig sind. Hierzu zählen ETL-Prozesse, Datenbankschemata und OLAP-Schemadefinitionen.

Nach Schaffung des nötigen Grundlagenwissens soll im folgenden Kapitel 3 ein grober Überblick über den Kontext dieser Arbeit gegeben werden. So wird eine Übersicht über vor- und nachgelagerte Prozesse vermittelt. Es wird insbesondere auf die relevanten Dokumente eingegangen, auf welche im weiteren Verlauf der Durchführung zurückgegriffen wird und die eingesetzte Data Warehouse Lösung Pentaho eingeführt.

Kapitel 4 bietet einen Überblick über bereits existierende Ansätze zum Thema agiles Data Warehouse.

Das Entwicklungskonzept und zugehörige Entscheidungsprozesse werden in Kapitel 5 behandelt.

Kapitel 6 beschäftigt sich mit der Implementierung eines Prototyps, der die zuvor erarbeiteten Konzepte umsetzt.

Im letzten Kapitel, Kapitel 7, finden sich eine Zusammenfassung der Ergebnisse sowie ein Ausblick auf mögliche Verbesserungen und Anwendungsbereiche bzw. Erweiterungen.

2. Grundlagen und Definitionen

> Ein Data Warehouse ist eine physische Datenbank, die eine integrierte Sicht auf beliebige Daten zu Analysezwecken ermöglicht [BG04].

Nach dieser Definition handelt es sich bei einem Data Warehouse zunächst um eine herkömmliche, physische Datenbank. Durch das Vorhalten von Daten aus vielen unterschiedlichen Quellen über einen längeren Zeitraum ermöglicht ein Data Warehouse eine globale Sicht auf diesen Datenbestand. Dies ermöglicht es die gespeicherten Daten zu analysieren. Um dieser Anforderung gerecht zu werden, müssen diverse Vorraussetzungen erfüllt werden. Eine zentrale Rolle spielt hierbei die Entwicklung von Schemata zum Speichern und Verarbeiten von Daten sowie deren Integration aus unterschiedlichen Datenquellen.

Diese Anforderungen müssen sich auch in der physischen Datenhaltung wiederspiegeln. Die zentrale Komponente eines Data Warehouse ist deshalb das sogenannte multidimensionale Datenmodell. Es ermöglicht die Speicherung der Daten in Bezug auf Dimensionen und Klassifikationshierarchien und bietet eine globale Sicht auf Daten in konsistenter Form. Mit Hilfe dieser können weitere Anwendungen, wie das im Kapitel 2.2 besprochene On Line Analystical Processing (OLAP), die gespeicherten Daten auswerten.

Das bereitgestellte Datenmodell wird mittels eines Extract-Transform-Load-Prozesses (ETL) mit Daten aus unterschiedlichen Quellen gefüllt. Dieser garantiert, dass Daten aus unterschiedlichen Systemen in aufgearbeiteter Form und in einem einheitlichen Format im Data Warehouse zur Verfügung gestellt werden können.

Ein weiterer Unterschied eines Data Warehouse gegenüber herkömmlicher Datenhaltung besteht darin, dass gespeicherte Daten nicht mehr verändert werden. Da die gesammelten Daten für Analysezwecke gespeichert werden, ist dies Bedingung für den Erhalt ihrer Aussagekraft. Das Erweitern der Datenbasis ist dagegen jederzeit möglich und erwünscht [BG04].

Der Gesamtprozess, auch Data Warehousing genannt, ist die Durchführung jedes einzelnen der bereits genannten Schritte. Angefangen bei der Datenbeschaffung mit anschließender Umwandlung durch den ETL-Prozess, Speicherung der Ergebnisse dieses Prozesses mittels eines multidimensionalen Datenmodells, bis zum mittels OLAP erzeugten Analyseergebnis am Ende der Prozesskette [Leh03].

Dieser Prozess kann mit Hilfe unterschiedlicher Software abgebildet werden. Neben diversen kostenpflichtigen Lösungen von Anbietern wie Oracle oder SAP existieren auch freie Implementierungen, wie die für diese Arbeit genutzte Community Edition der Penthao BI-Suite.

Die Grundlagen der angesprochenen Techniken sollen innerhalb dieses Kapitels vermittelt werden, sodass später getroffene Entscheidungen nachvollzogen werden können.

2.1. Extract-Transform-Load

Um ein Data Warehouse zu betreiben, wird eine große Anzahl an Daten benötigt. Diese Daten stammen, nach der Definition aus Kapitel 2, aus vielen, heterogenen Quellen und weisen unterscheidliche Formate auf. Um diese Daten nun als homogene Datenmenge bereitstellen zu können, wird ein ETL-Prozess benötigt.

Unter ETL versteht man einen dreistufigen Prozess, bestehend aus den Phasen Extract, Transform und Load. Diese beschreiben den Vorgang der regelmäßigen Aktualisierung der Daten eines Data Warehouses bzw. einer OLAP-Applikation. Für die Umsetzung müssen die Daten zunächst aus einem oder mehreren Quellsystemen extrahiert werden. Darauffolgend werden sie auf Konsistenz geprüft und gegebenenfalls in die Struktur des Zielsystems transformiert. Im Anschluss können sie in das eigentliche Data Warehouse geladen werden.

Die Erstellung eines ETL-Prozesses ist oft einer der aufwändigsten Schritte bei der Erzeugung eines Data Warehouses [Lio6], da ohne den Aufbau und die Pflege einer gut strukturierten Datenbasis die Funktionen eines Data Warehouses stark eingeschränkt sind. Der ETL-Prozess dient zur Schaffung und Aktualisierung eben dieses Datenbestandes.

Die nachfolgende Abbildung 2.1 verdeutlicht die Zusammenarbeit der drei Teilschritte eines ETL-Prozesses.

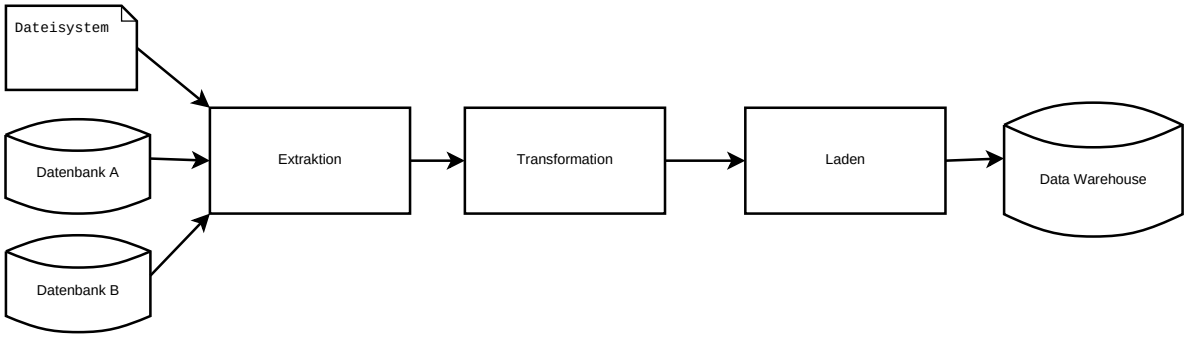

Abbildung 2.1.: Übersicht über einen ETL-Prozess (eigene Darstellung)

Extraktion Ziel dieser ersten Phase ist es, einen Teil der Daten aus einem oder mehreren Quellsystemen zu extrahieren. Hierbei kann die Art der Quelle sowie ihr Aufbau und die Art der enthaltenen Daten stark variieren. Aufgrund möglicher unterschiedlicher Datenstrukturen erfolgt in diesem Schritt auch eine Transformation in ein gemeinsames Datenschema. Dies gewährleistet, dass Daten aus unterschiedlichen Quellen in der Transformationsphase verarbeitet werden können [Leho3].

Neben der eigentlichen Datenextraktion werden in dieser Phase noch zwei weitere Eigenschaften des ETL-Prozesses festgelegt: die Art der extrahierten Daten sowie die Häufigkeit der Extraktion. Je nach Art der Quelle können unterschiedliche Arten von Daten extrahiert werden, beispielsweise Snapshots oder Logs.

Während ein Snapshot stets die volle Datenmenge liefert, erhält man durch Abfragen eines Logs lediglich eine inkrementelle Menge von Änderungen seit der letzten Abfrage. Mischformen oder weiter eingeschränkte Varianten sind hierbei denkbar. Der Zeitpunkt und damit die Häufigkeit der Extraktion können sich ebenfalls stark unterscheiden. So ist zum Einen eine synchrone Benachrichtigung möglich. Die Quelle würde hierbei jede einzelne Änderung propagieren. Häufiger werden jedoch sogenannte asynchrone Benachrichtigungen genutzt. Diese können periodisch, anfragegesteuert oder ereignisgesteuert erfolgen [BG04].

Transformation Die Transformation stellt die zentrale Aufgabe des ETL-Prozesses dar. Hierbei werden die durch die Extraktion gelieferten Ausgangsdaten an das gewünschte Zielschema angepasst. Dieser Vorgang wird auch Schema-Mapping genannt. Um dies zu bewerkstelligen, wird zwischen zwei Transformationsarten unterschieden: syntaktische und semantische Transformation. Bei der syntaktischen Transformation werden die erhaltenen Daten auf formaler Ebene aufbereitet. So werden Daten aus unterschiedlichen Quellen auf eine einheitliche Syntax transformiert. Als Beispiel dient hier die Transformation unterschiedlicher Datumsformate in eine einheitliche Variante.

Mit der semantischen Transformation werden die Daten auf inhaltliche Aspekte überprüft und gegebenenfalls versucht fehlende Daten zu ergänzen. Da in diesem Schritt Daten aus unterschiedlichen Quellen verbunden werden sollen, müssen unterschiedliche Strukturen, welche jedoch gleiche Daten enthalten, auf eine gemeinsame Basis transformiert werden. So müssen zum Beispiel Maßeinheiten umgerechnet, Datenwerte angepasst (Beispielsweise '1' oder 'true' auf 't', '0' oder 'false' auf 'f'), Aggregationen oder ähnliche Operationen auf die einzelnen Datensätzen angewendet werden [Leh03].

Der Zeitaufwand, der mit diesem Vorgang verbunden ist, steigt mit der Datenmenge, da jeder Datensatz einzeln transformiert werden muss. Daher ist es vorteilhaft, wenn die Unterschiede zwischen den einzelnen Quellsystemen nicht zu groß sind. Des Weiteren sollten die zur Transformation genutzten Algorithmen möglichst effizient gewählt werden [BG04].

Laden Die durch die Transformation erzeugten Daten müssen nach der Transformation möglichst effizient in das Zielsystem eingebracht werden. Da es sich hierbei meist um eine Datenbank handelt, ist darauf zu achten, dass die Dauer dieses Vorgangs möglichst kurz gehalten wird. Durch lange Schreib- und Updateprozesse wird ein anderweitiger Zugriff auf die Datenbank blockiert, wodurch beispielsweise angeschlossene Monitoring-Tools keine Daten erhalten könnten. Dies führt dazu, dass in diesem Schritt darauf geachtet werden muss, ob, wie und welche Daten hinzugefügt oder bearbeitet werden müssen. Vor allem bei zeitkritischen Systemen müssen bestimmte Zeitfenster zur Aktualisierung des Datenbestandes eingehalten werden. Daher wird

das synchrone Einbringen von Änderungen aus den Quellsystemen nur in wenigen Fällen angewendet.

2.2. On Line Analytical Processing

OLAP ist ein Akronym für On Line Analytical Processing und beschreibt einen Ansatz der Datenanalyse. Durch den Einsatz von OLAP ist es möglich multidimensionale Analysen auf Geschäftsdaten auszuführen. Es bietet Möglichkeiten für komplexe Berechnungen sowie Trendanalysen, welche als Hilfsmittel dienen können, um Innovationen und neue Geschäftsideen zu entwickeln oder vorhandene Geschäftsmodelle zu optimieren. Ein Einsatzgebiet findet sich an jenen Stellen, an welchen einfache Abfrage-Werkzeuge oder -Sprachen aufgrund der Anfragekomplexität an die Grenzen ihrer Leistungsfähigkeit stoßen. Dies gilt insbesondere für die Performance solcher Abfragen und dem eingeschränkten Funktionsumfang von anderen Anfragesprachen, wie SQL. So sind komplexe analytische Abfragen, wie „Zeige alle Verkäufe des Buches XYZ in Stuttgart von Juni bis Dezember 2011, im Vergleich zu denselben Daten im Vorjahreszeitraum“, mit herkömmlichem SQL schwer umzusetzen. Steigt die Komplexität einer Anfrage noch, so kann die Abfrage mittels reinem SQL sogar unmöglich werden.

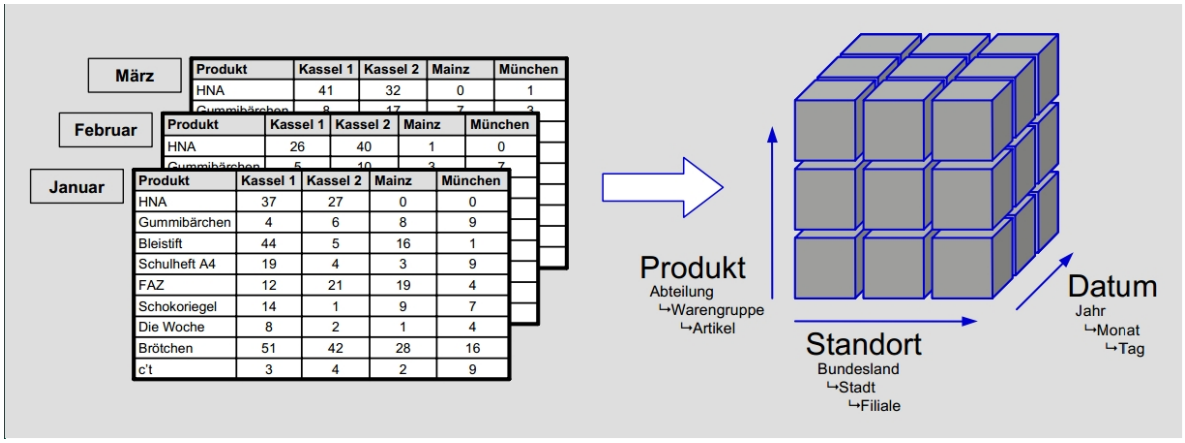

Abbildung 2.2.: Darstellung eines OLAP-Würfels mit zugehörigen Werten (Quelle: [SM11])

Um solch komplexe Anfragen zu ermöglichen, werden Daten in sogenannten OLAP-Würfeln (engl. Cubes) gehalten. Ein Würfel ist ein logisches Konstrukt zur Darstellung von mehrdimensionalen Daten. Hierfür werden die Informationen als Elemente eines mehrdimensionalen Würfels angelegt. Abbildung 2.2 zeigt beispielhaft einen Würfel mit drei Dimensionen: Produkt, Standort sowie der zugehörige Zeitraum. Jede einzelne dieser Dimensionen kann unterschiedliche Detailgrade enthalten. So kann das Datum wiederum durch Informationen in Jahre, Monate und Tage unterteilt werden. Als Ergebnis erhält man einen Datenwürfel, der je nach Zustand unterschiedliche Daten anzeigen bzw. zurückgeben kann [Tho02].

Auf diesen Würfeln können unterschiedliche Operationen ausgeführt werden. Drei Beispiele hierfür werden in Abbildung 2.3 dargestellt. Mittels Slice können Scheiben aus einem Würfel „herausgeschnitten“ werden. Dies ermöglicht die gezielte Untersuchung einzelner Sektionen des Datensatzes. Dicing ist eine Erweiterung der Slice-Operation und ermöglicht das gleichzeitige Slicing in unterschiedlichen Dimensionen. Als Ergebnis erhält man einen Teilbereich des Ursprungswürfels. Hereinzoomen oder Drilldown wird jener Vorgang genannt, welcher es ermöglicht auf detailliertere Werte einer Aggregation zuzugreifen. Zusätzlich zu den hier genannten Operationen existieren noch diverse weitere Operationen, mit deren Hilfe ein einzelner Würfel manipuliert werden kann. Ein OLAP-Würfel ist jedoch nicht, wie in diesem Beispiel, auf drei Dimensionen beschränkt, sondern kann mit beliebig vielen Dimensionen versehen werden.

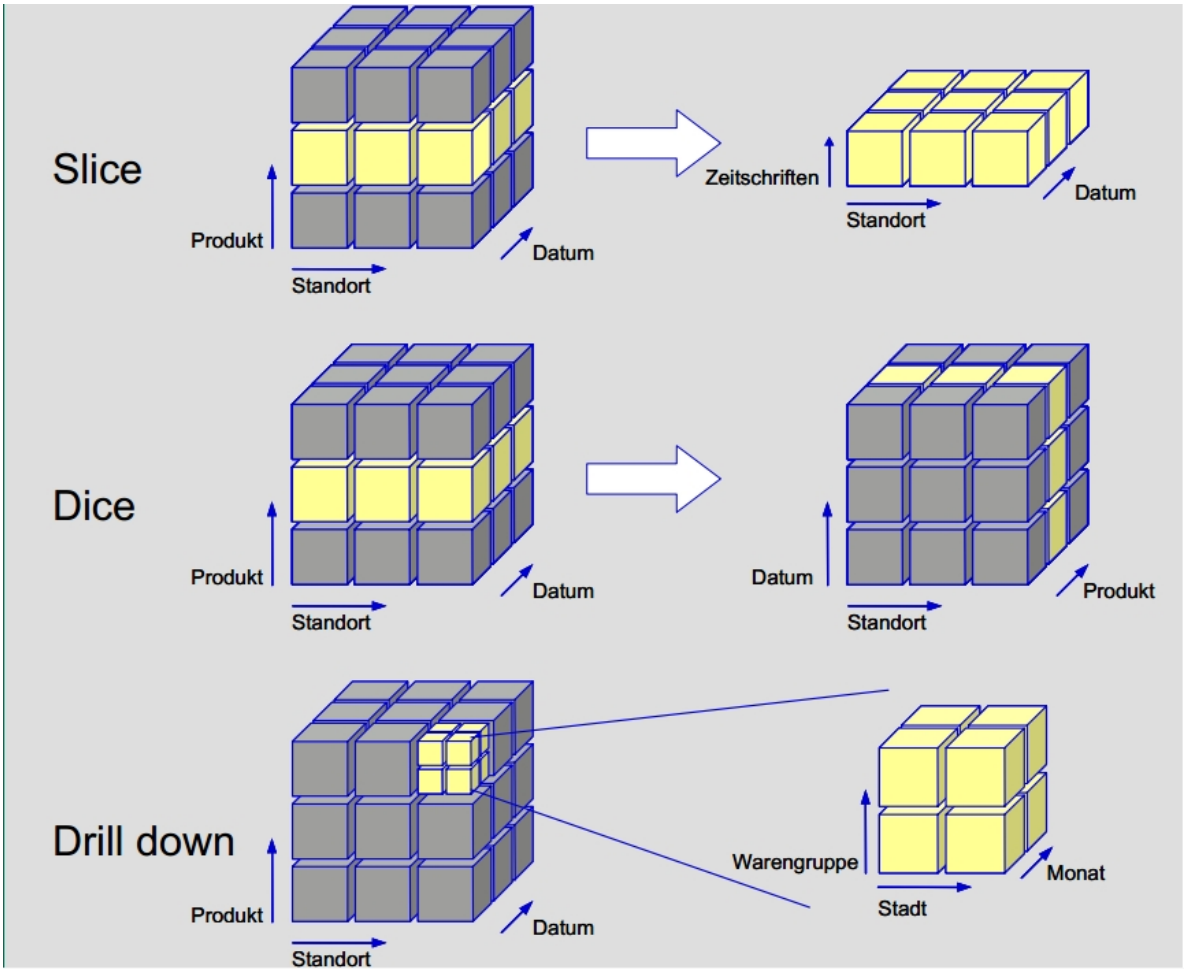

Abbildung 2.3.: Beispieloperationen auf einem OLAP-Würfel (Quelle: [SM11])

Es wird zwischen zwei unterschiedlichen OLAP-Varianten unterschieden. Das sogenannte MOLAP (multidimensionales OLAP), welches auf eine multidimensionale Datenbank zugreift sowie ROLAP (relationales OLAP) mit Zugriff auf eine relationalen Datenbank. Bei letzterer Variante wird in einer relationale Datenbank ein Sternschema angelegt. Dies

ermöglicht die Speicherung eines Würfels innerhalb einer relationalen Datenbank. OLAP Funktionen wie Slice und Dice können dann mittels klassischer Datenbankanfragen realisiert werden. Ein großer Nachteil des ROLAP liegt jedoch darin, dass Standard-SQL für multidimensionale Analysen nur bedingt ausreicht, hierdurch müssen Teile der Abfragelogik im OLAP-Server integriert werden.

Um Anfragen an auf ROLAP basierende Systeme zu vereinfachen, wurden unterschiedliche Anfragesprachen für OLAP-Würfel entwickelt. Diese Anfragen sind prinzipiell auch mittels SQL denkbar, jedoch wurde SQL für Datenbanken im operationalen Betrieb entworfen. Hierdurch ist SQL nur begrenzt für die Formulierung von multidimensionalen Anfragen geeignet [Leh03]. Von diesen neuen Sprachen hat sich vor allem die Multidimensional Expression, kurz MDX, etabliert [Nol99].

Gemäß seines Namens handelt es sich bei Multidimensional Expression um eine Sprache zur Formulierung von multidimensionalen Ausdrücken. Ursprünglich von der Firma Microsoft entwickelt, stellt MDX den Industriestandard im Bereich OLAP-Anfragen dar. Die Syntax ist dabei lose an die in SQL-Anfragen genutzte Syntax angelehnt und nutzt ähnliche Schlüsselworte. Mit Hilfe dieser Syntax ist es möglich, OLAP-Würfel zu manipulieren und Anfragen zu stellen [Nol99].

2.3. Sternschema

Beim Sternschema handelt es sich um ein Datenbankschema welches genutzt wird, um multidimensionale Daten mit Hilfe einer relationalen Datenbank abzubilden. Ein Beispiel stellt die datenbankinterene Repräsentation der in Kapitel 2.2 beschriebenen OLAP-Würfel dar. Zentrales Element eines Sternschemas ist eine sogenannte Faktentabelle. Diese wird um eine beliebige Anzahl von Dimensionstabellen ergänzt. Diese Dimensionstabellen sind einzeln mit der Faktentabelle verbunden und besitzen keine Abhängigkeiten zueinander. Die graphische Darstellung entspricht hierbei einem Stern, woraus sich der Name dieses Schemas ableitet [Leh03]. Dies wird von Abbildung 2.4 dargestellt.

Die Faktentabelle enthält sämtliche zu verwaltenden Daten (Fakten) bzw. Kennzahlen. Dies führt dazu, dass Faktentabellen eine große Anzahl von Daten halten müssen und deshalb zur Abfrageoptimierung um Indizes erweitert werden sollten. Als Fakt könnte beispielsweise die Anzahl der Verkäufe eines einzelnen Produkts innerhalb eines Monats gespeichert werden.

Im Gegensatz hierzu werden in einer Dimensionstabelle die beschriebenen Daten gespeichert. Ein Beispiel hierfür wäre die Darstellung einer Dimension Standort, welche den Namen einer Stadt, die zugehörige Postleitzahl, das Bundesland sowie den Namen der Filiale enthält. Die einzelnen Dimensionstabellen liegen in denormalisierter Form vor. Das heißt, dass zwischen Nicht-Schlüsselattributen funktionale Abhängigkeiten existieren, wodurch die 3. Normalform (3NF) verletzt wird. Hierdurch erhält man eine leicht erhöhte Abfragegeschwindigkeit zu Lasten des Speicherplatzes sowie der Datenintegrität [BG04].

Die Faktentabelle enthält Fremdschlüssel auf die Dimensionseinträge. Ein Fremdschlüssel ist die eindeutige Identifizierung eines Wertes in seiner Tabelle und wird in vielen Fällen

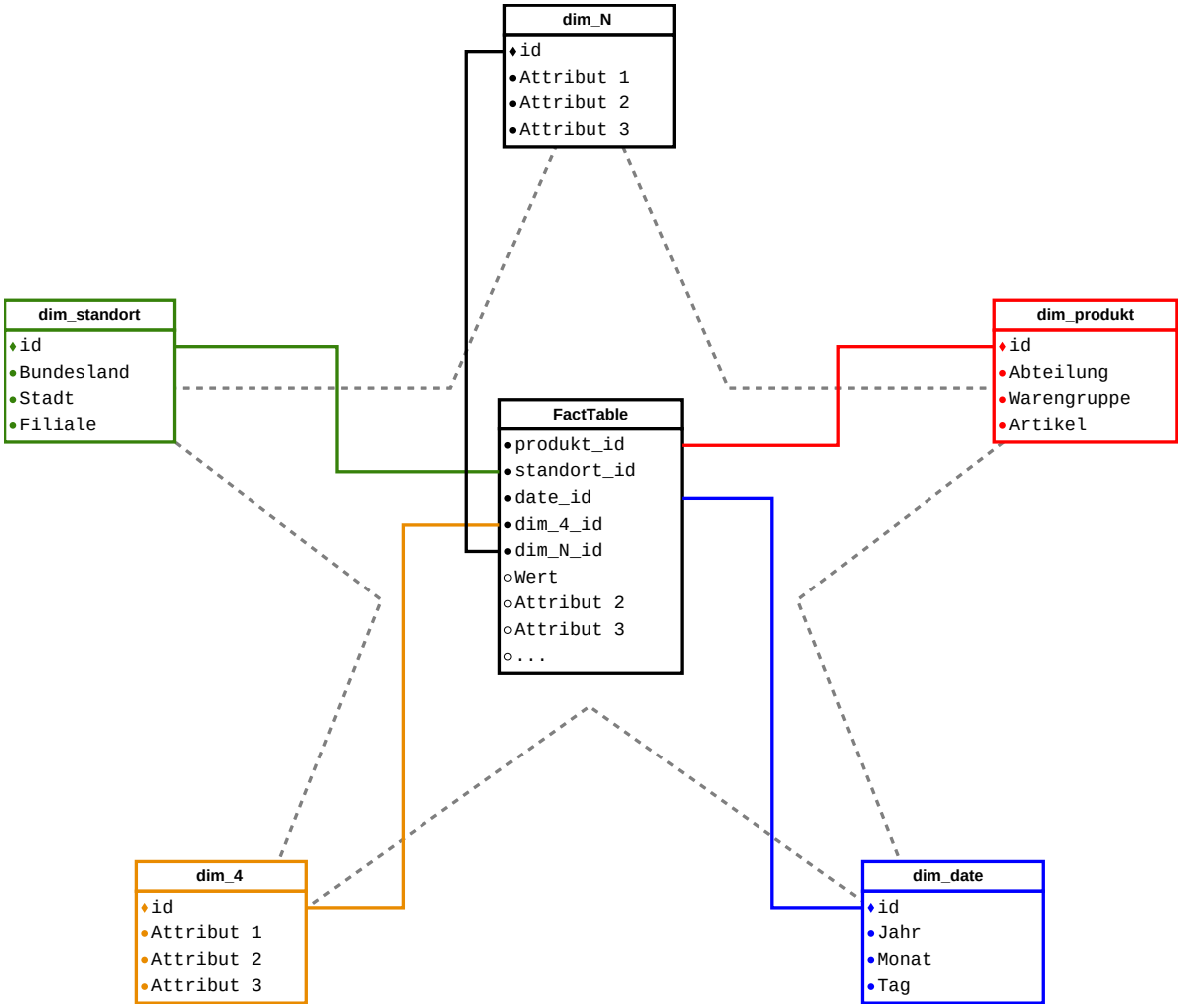

Abbildung 2.4.: Übersicht über eine Faktentabelle und ihre Dimensionstabellen eines Sternschemas (eigene Darstellung)

von einem numerischen Wert repräsentiert. In vielen Implementierungen wird der Primärschlüssel der Faktentabelle aus der Gesamtmenge der Fremdschlüssel zusammengesetzt. Dies impliziert, dass jeder Eintrag in der Faktentabelle in Bezug auf seine Dimensionen einmalig ist [RC02].

Anhand des in Abbildung 2.2 genutzen Beispiels würde das Erzeugen der Tabelle *Januar* Werte aus den Dimensionen Produkt, Standort, Datum sowie der Faktentabelle benötigen. Dies führt zu folgendem SQL-Statement:

```
1  SELECT produkt.name, standort.stadt, ft.wert
2  FROM facttable AS ft, dim_standort AS standort, dim_produkt AS produkt
3  WHERE
4          ft.datum_fk IN
5                  (SELECT id FROM dim_date WHERE Monat = "Januar")
```

```
6         AND ft.standort_id = standort.id
7         AND ft.produkt_id = produkt.id;
```

Listing 2.1: Eine Beispielabfrage von Werten in einem Sternschema (eigene Darstellung)

Mithilfe von Joins können so Informationen aus den Dimensionstabellen und der Faktentabelle zusammengeführt werden.

3. Kontext der Arbeit

In diesem Kapitel wird ein Überblick über den Kontext gegeben, in welchem das agile Data Warehouse zum Einsatz kommt. Hierfür wird zunächst der Gesamtprozess dargestellt. Aufbauend hierauf wird detailliert auf das in diesem Prozess erzeugte ProGoalML sowie darauf aufbauende Komponenten eingegangen. Abschließend wird das agile Data Warehouse sowie der dafür nötige Pentaho BI-Server in dieses Konzept eingeführt.

3.1. Einordnung in den Gesamtprozess

Das in dieser Arbeit zu erzeugende agile Data Warehouse soll als Teilkomponente einer komplexen BI-Lösung für Geschäftsprozesse eingesetzt werden. Sie dient hierbei als Quelle für Optimierungs- und Analyse-Werkzeuge. Diese BI-Lösung soll auf lange Sicht als Software-As-A-Service (SAAS) angeboten werden. Um innerhalb einer SAAS Lösung eingesetzt werden zu können, ist es nötig, dass alle Prozessschritte zur Einrichtung der BI-Lösung automatisiert erfolgen. Dies hat zur Folge, dass ein unter diesen Vorraussetzungen nutzbares Data Warehouse automatisiert eingerichtet und mit Daten befüllt werden muss. Dies entspricht der in Kapitel 1.1 getroffenen Definition von agil: eine automatisierte Erstellung und Einrichtung der für ein Data Warehouse nötigen Strukturen sowie deren automatische Anpassung bei Veränderung der Umgebungsbedingungen. Zusätzlich müssen Möglichkeiten für das Befüllen dieser Strukturen mit Daten geschaffen werden.

Entsprechend dieser Definition müssen Kerndokumente (das Datenbankschema, das OLAP-Schema sowie eine ETL-Beschreibung) erzeugt und mit ihrer Hilfe ein Data Warehouse konfiguriert werden. Um diese Kerndokumente erzeugen zu können, werden Informationen aus anderen Bereichen des Gesamtprozesses benötigt. Dieses Kapitel soll einen Überblick über diese vorgelagerten Prozesse bieten sowie auf ihre Eigenschaften und Ergebnisse eingehen. Der Gesamtprozess (APro) wird in Abbildung 3.1 dargestellt.

Ausgangspunkt für den Gesamtprozess ist ein BPMN 2.0 Model, welches den Ablauf eines Geschäftsprozesses modelliert. Business Process Model and Notation (BPMN) ist eine grafische Spezifikation zur Beschreibung von Geschäftsprozessen. Hierfür werden diverse Symbole bereitgestellt, welche die grafische Darstellung ermöglichen. Mit Hilfe von BPMN ist es möglich Prozesse festzuhalten und zu optimieren [All09].

Die mit APro erzeugten BPMN Modelle basieren auf der BPMN Spezifikation 2.0 und werden mit der webbasierten Software Oryx [ory11] erzeugt. Die erzeugten Modelle bilden Prozesse ab, welche optimiert und analysiert werden sollen. Hierfür müssen jedoch zunächst Kennwerte, sog. Key-Performance-Indikatoren (KPI) und Ziele definiert und erfasst werden,

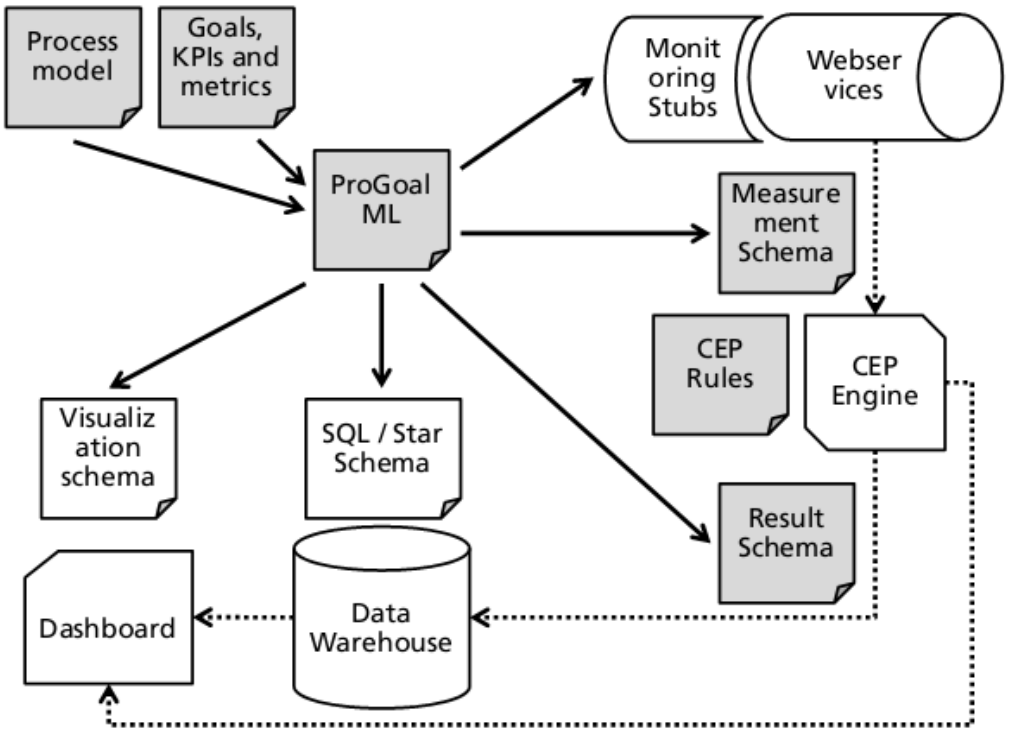

Abbildung 3.1.: Übersicht über den Gesamtprozess APro (Quelle: [KW12])

auf deren Basis eine Analyse des Prozesses möglich ist. BPMN bietet hierfür keine Möglichkeit, weshalb Oryx um ein spezielles Plugin erweitert wurde, mit welchem sich diese Elemente grafisch modellieren lassen. Abbildung 3.2 zeigt einen mit Hilfe dieser Erweiterung erzeugten Prozess. Die Erweiterung von BPMN um diese zusätzlichen Definitionen wird in einem neuen Dateiformat gespeichert: der sog. ProGoalML.

Dieses Dokument kann nun, wie in Abbildung 3.1 dargestellt, als Ausgangspunkt für nachfolgende Prozesse genutzt werden.

3.2. ProGoalML

Beim Datenformat *ProGoalML* handelt es sich um eine XML-Schemadefinition (XSD), die eine Beschreibung zur Modellierung von Zielen, Messpunkten, Kennwerten und KPIs in Bezug auf ein BPMN-Prozessmodell bietet. Mit Hilfe einer zusätzlichen Notation innerhalb eines BPMN-Modells ist es möglich, Kennwerte an bestimmten Stellen eines BPMN-Prozesses zu erfassen, diese zu aggregieren und in Beziehung zueinander zu stellen.

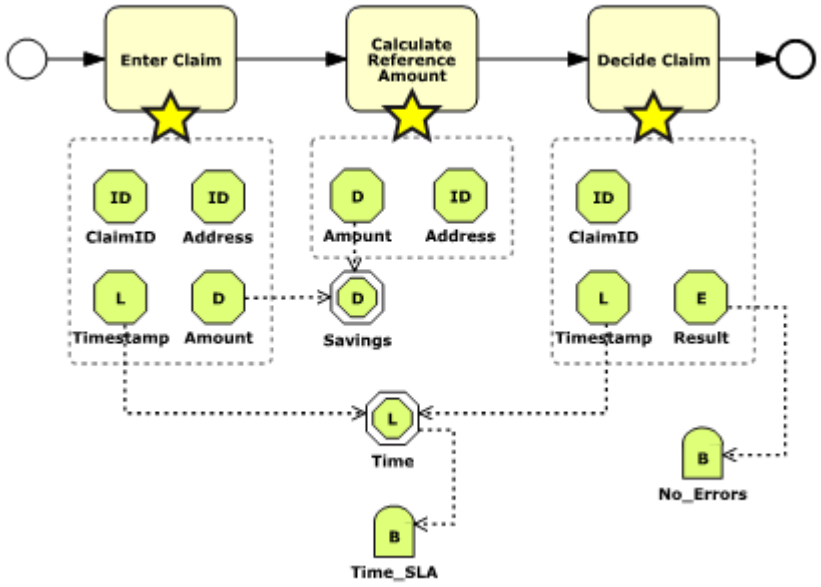

Abbildung 3.2.: Beispielprozess mit MeasuringPoints, KPIs und Goal (Quelle: [KW12])

Ein ProGoalML-Dokument besteht aus Metadaten, einem GoalModel sowie dem dazugehörigen Prozessmodel.

Meta: Hierin finden sich allgemeine Informationen wie das Erstellungsdatum, eine Beschreibung oder der Titel.

ProcessModel: Hierbei handelt es ich um ein standardisiertes BPMN 2.0 Model. Es enthält alle Elemente des Prozesses, welche im GoalModel mittels des *RefBpmn*-Tags referenziert werden können (siehe Listing 3.1 Zeile 5).

GoalModel: Hierin finden sich alle Informationen, welche sich auf Ziele, Messpunkte oder KPIs beziehen.

MeasuringPoint: Ein MeasuringPoint besteht aus einer beliebigen Anzahl von Parametern sowie einer Referenz auf das zugehörige BPMN-Element. Jeder Parameter enthält neben seinem Namen noch den zu erwartenden Datentyp. Ein Parameter stellt einen zu erfassenden Wert dar. So definiert der in Listing 3.1 Zeile 6-8 dargestellte Parameter eine Postleitzahl. Der definierte Kennwert hat den Namen „Address“ und ist vom Datentyp *Double*.

KeyPerformanceIndicator: Eine KPI enthält neben dem eigenen Namen ebenfalls den zugehörigen Datentyp. Zusätzlich wird ein Formel-Element für Berechnungen bereitgestellt. Hierin könnte beispielsweise die Differenz zweier Paramter berechnet werden. Damit die Parameter einer Formel mit einem Messpunkt assoziert werden können, müssen diese innerhalb von *RefParamter*-Tags referenziert werden. Der in

Listing 3.1 Zeile 13-23 spezifizierte KeyPerformanceIndicator soll beispielsweise die gesparten Kosten des Prozesses erfassen. Hierfür werden von den tatsächlichen Kosten die Referenzkosten abgezogen. Die tatsächlichen Kosten werden innerhalb des MeasuringPoints *Enter-Claim* als Parameter *Amount* erfasst. Die Referenzkosten im MeasuringPoint *Calculate_Reference_Amount* werden ebenfalls als Parameter *Amount* bereitgestellt. Hierdurch wird deutlich, warum die zusätzliche Referenzierung des MeasuringPoints benötigt wird. Das Ergebnis der Berechnung soll vom Typ *Double* sein.

Goal: Wie die Definition eines KPI enthält ein Goal seinen Namen sowie Referenzinformationen zu den genutzten Variablen. Der Datentyp eines Goals muss vom Typ *Boolean* sein und dient somit zur Überprüfung von Werten und Prozessabschnitten. So wird im vorliegenden Beispiel das Ziel immer dann erfüllt, wenn der in MeasuringPoint *Decide_Claim* erfasste Parameter *Result* nicht dem Wert „Error“ entspricht. Dies ist immer dann erfüllt, wenn der Prozess korrekt durchlaufen wurde.

```
1  <Progoalml version="1.0">
2         <Meta>...</Meta>
3         <GoalModel>
4                 <MeasuringPoint name="Calculate_Reference_Amount">
5                         <RefBpmn>Calculate_Reference_Amount</RefBpmn>
6                         <Parameter name="Address">
7                                 <DataType>double</DataType>
8                         </Parameter>
9                         <Parameter name="Amount">...</Parameter>
10                </MeasuringPoint>
11                <MeasuringPoint name="Decide_Claim">...</MeasuringPoint>
12                <MeasuringPoint name="Enter_Claim">...</MeasuringPoint>
13                <KeyPerformanceIndicator name="Savings">
14                        <Formula>
15                                Enter_Claim.Amount - Calculate_Reference_Amount.Amount
16                        </Formula>
17                        <RefParameter>
18                                <ParameterName>Amount</ParameterName>
19                                <MeasuringPointName>Enter_Claim</MeasuringPointName>
20                        </RefParameter>
21                        <RefParameter>...</RefParameter>
22                        <DataType>double</DataType>
23                </KeyPerformanceIndicator>
24                <Goal name="No_errors">
25                        <Formula>Result != "ERROR"</Formula>
26                        <RefParameter>
27                                <ParameterName>Result</ParameterName>
28                                <MeasuringPointName>Decide_Claim</MeasuringPointName>
29                        </RefParameter>
30                        <DataType>boolean</DataType>
31                </Goal>
32        </GoalModel>
33        <ProcessModel>...</ProcessModel>
34 </Progoalml>
```

Listing 3.1: Eine Beispiel ProGoalML (Quelle: [KW12])

Alle für die Generierung des Data Warehouses nötigen Informationen können aus diesen Komponenten gewonnen werden. Hilfreich hierbei ist die sogenannte APro-Bibliothek, welche diverse Funktionen zur Verarbeitung der ProGoalML zur Verfügung stellt.

Neben den definierten MeasuringPoints, KeyPerformanceIndicatoren und Goals können aus der ProGoalML noch weitere Informationen gewonnen werden: Jedes Element kann einer Überdeckungsklasse zugeordnet werden. Eine Überdeckungsklasse definiert einen Datensatz, welcher einen kontextuellen Zusammenhang seiner Elemente beschreibt. Eine Beziehung zwischen zwei Messpunkten ist beispielsweise dann gegeben, wenn in beiden Messpunkten die gleiche ID erfasst wird. Können Elemente nicht in Beziehung zueinander gesetzt werden, so erhält man eine zweite Überdeckungsklasse.

Mit Hilfe dieser Überdeckungsklassen ist die gewünschte Auswertung und Analyse der gesammelten Informationen möglich, denn erst durch die klare Abhängigkeit der Daten untereinander können diese in einen Kontext gebracht werden und somit Aussagekraft erhalten. Eine Überdeckungsklasse kann somit als Grundlage für einen zu erzeugenden OLAP-Würfel gesehen werden.

Überdeckungsklassen lassen sich mit Hilfe der APro-Bibliothek aus einer ProGoalML generieren. Listing 3.2 zeigt ein solches Ergebnis-Schema. Wie man an diesem Beispiel sieht, handelt es sich hierbei um eine XML-Schema-Definition. Das hierdurch definierte Format dient als Datenquelle des ETL-Prozesses für das agile Data Warehouse.

```
1  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
2  <xs:element name="Handle_Claim" type="Handle_Claim"/>
3          <xs:complexType name="Handle_Claim">
4                  <xs:sequence>
5                          <xs:element name="Case_ID" type="xs:id"/>
6                          <xs:element name="Address" type="xs:id"/>
7                          <xs:element name="Enter_Claim.Timestamp" type="xs:long"/>
8                          <xs:element name="Savings" type="xs:double"/>
9                          <xs:element name="No_Errors" type="xs:boolean"/>
10                 </xs:sequence>
11         </xs:complexType>
12 </xs:schema>
```

Listing 3.2: Ein Beispielschema einer Ergebnisdatei (Quelle: [KW12])

Aus einem Ergebnis-Schema lassen sich neben dem Namen des Schemas auch die einzelnen zu erwartenden Parameter sowie ihre Datentypen auslesen. Diese Parameter entsprechen den in der ProGoalML definierten KPIs, Goals und Parametern der MeasuringPoints, sofern diese zur entsprechenden Überdeckungsklasse gehören.

Die eigentlichen Ergebnisdateien, welche in einem späteren Teilprozess generiert werden, enthalten die zu speichernden Daten. Listing 3.3 zeigt ein Beispiel für eine mögliche Ergebnisdatei basierend auf dem Ergebnis-Schema aus Listing 3.2.

```
1  <Handle_Claim>
2          <Case_ID>1</Case_ID>
3          <Address>70191</Address>
4          <Enter_Claim.Timestamp>1325419200</Enter_Claim.Timestamp>
```

```
5         <Savings>100.00</Savings>
6         <No_Errors>true</No_Errors>
7  </Handle_Claim>
```

Listing 3.3: Eine mögliche Ergebnisdatei für das Ergebnisschema in Listing 3.2

3.3. Bisherige Arbeiten mit ProGoalML

Neben dem bereits erwähnten ProGoalML Generator, welcher mittels eines Oryx-Plugins realisiert wurde, existieren bereits weitere Arbeiten, welche auf die ProGoalML zurückgreifen. So existiert ein Projekt, welches es ermöglicht Web-Services aus einer ProGoalML zu generieren. Diese Web-Services sind in der Lage Messungen einzelner Messpunkte entgegen zu nehmen. Messpunkte können dabei Teil eines ausführbaren Prozessmodells sein, welches mittels einer Process Engine ausgeführt wird. Alternativ können sie Teil eines Anwendungssystems sein, welches durch aus der ProGoalML generierte Codestubs realisiert wird. Mit diesem Vorgehen lassen sich unterschiedliche Systeme für Messungen instrumentieren.

Diese Web-Services dienen zur Entgegennahme von Werten der in der ProGoalML spezifizierten Messpunkte, KPIs und Ziele.

Hierzu wird die Ausführung des BPMN Models mittels einer Process Engine simuliert.

Des Weiteren wurde eine Complex Event Processing (CEP) Engine integriert. "*Eine CEP-Engine ermöglicht das Erkennen und Reagieren auf komplexe Ereignisse, die sich durch die Verknüpfung vieler atomarer Ereignisse zu Einheiten einer semantisch höheren Ordnung zusammensetzen*" [GSS10].

Im vorliegenden Fall bedeutet dies, dass die eingesetzte CEP-Engine die von den Web-Services gesammelten Daten entgegennimmt. Mit Hilfe der aus der ProGoalML erzeugten Ergebnis-Schemata sowie der für KPIs und Goals spezifizierten Formeln werden aus diesen Daten die bereits beschriebenen Ergebnisdateien erzeugt. Jede Ergebnisdatei enthält die gesammelten und aggregierten Werte eines Datensatzes einer Überdeckungsklasse (siehe Listing 3.3).

3.4. Pentaho BI-Suite

BI ist eine Abkürzung für den Begriff Business Intelligence. Hierbei handelt es sich um Verfahren und Prozesse zur Analyse von zuvor erfassten Daten. Diese Analyse erfolgt durch Softwarelösungen, welche entsprechend den angegebenen Vorgaben Berichte erzeugen und zur Verfügung stellen. Die hierfür benötigten Rohdaten entstammen einer Datenbank oder einem Data Warehouse und werden mit unterschiedlichen Analysemethoden ausgewertet. Hierzu zählen neben Data Mining und Text Mining auch das in Kapitel 2.2 näher beschriebene OLAP.

Neben der unter GPL veröffentlichten Community Edition werden auch drei kommerzielle Varianten vertrieben. Pentaho ist vollständig in Java implementiert [pen12]. Neben verbesserten Werkzeugen enthält die kommerzielle Variante einen professionellen Support.

Die Pentaho BI-Suite (im Folgenden nur Pentaho genannt) bietet in diesem Zusammenhang eine Sammlung von Werkzeugen, mit deren Hilfe sowohl die Datensammlung (ETL-Prozess, siehe Kapitel 2.1) als auch die Auswertung mittels OLAP und anschließende Reportgenerierung realisiert werden kann. Je nach Anforderung ist es möglich der Suite weitere Komponenten hinzuzufügen, welche beispielsweise die Reporting-Eigenschaften erweitern oder neue Funktionen wie Data Mining hinzufügen.

Um dies zu ermöglichen besteht Pentaho aus unterschiedlichen Modulen, deren Kern der sogenannte BI-Server darstellt. Hierbei handelt es sich um eine J2EE-kompatible Webanwendung, die auf gängigen Java-Applikationsservern wie Tomcat, JBoss AS, IBM WebSphere, BEA WebLogic und Oracle Application Server läuft. Als Bedienoberfläche wird ein Webfrontend zur Verfügung gestellt (Abbildung 3.3 *Presentation Layer*). Alternativ kann die Anwendung auch als Webservice über Java- oder AJAX-Schnittstellen angesprochen werden.

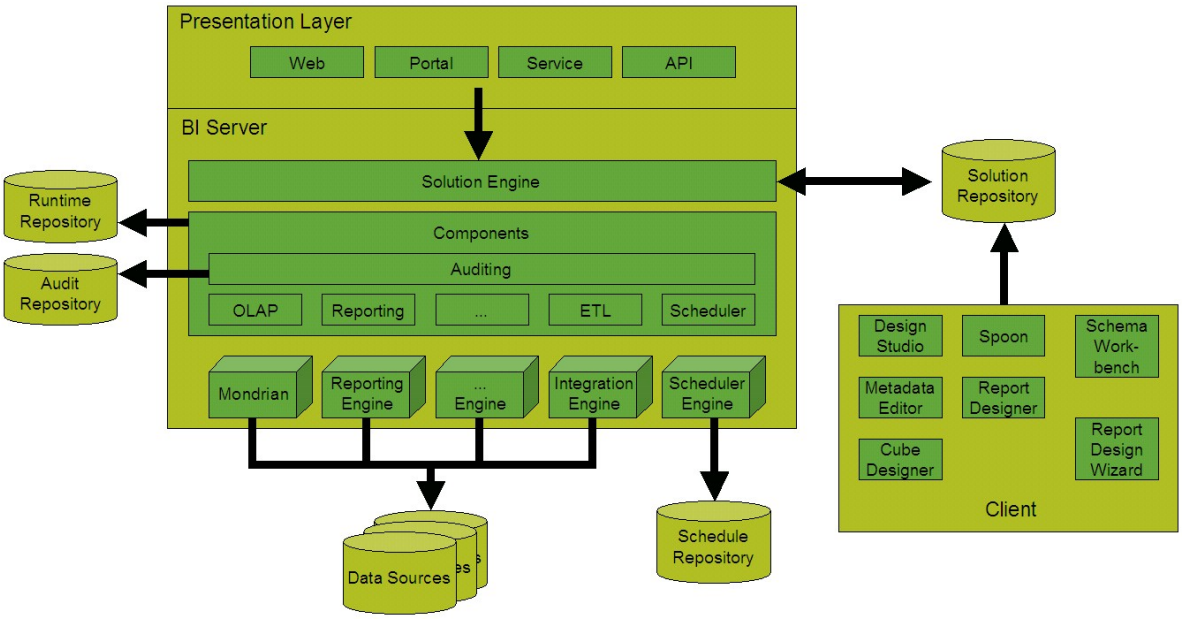

Abbildung 3.3.: Der Pentaho BI-Server (Quelle: [HK07])

Das Kernelement des Servers ist die Solution Engine. Sie verwaltet den Zugriff auf das Solution Repository, in welchem die zur Verfügung stehenden OLAP-Cubes sowie ETL-Prozesse abgelegt werden. Zusätzlich übernimmt sie die Steuerung und das Scheduling von sog. Action Sequences. Bei der Ausführung von Action Sequences werden die Server-Engines der einzelnen Module angesprochen. Der Pentaho BI-Server enthält bereits alle benötigten Java-Libraries für die Server-Engines der einzelnen mitgelieferten Module sowie einige zusätzliche Engines, wie BIRT oder JasperReporting.

Als mögliche Datenquellen (*Data Sources*) stehen ebenfalls unterschiedliche Technologien zur Verfügung. In der Standardkonfiguration des BI-Servers ist dieser mit einer HSQL-Datenbank ausgestattet. Hierbei handelt es sich um eine in Java implementierte relationale SQL-Datenbank, welche ohne zusätzliche Installation durch den Anwender betrieben werden kann. Alternativ ist es jedoch auch möglich weitere Datenquellen, wie PostgreSQL, MySQL oder einfache Textdateien, zu integrieren.

Zusätzlich zu den serverseitigen Modulen existiert eine Sammlung von Tools, welche die Erstellung und Bearbeitung der unterschiedlichen Solutions ermöglichen (vgl. Abbildung 3.3 *Client*). Mit diesen können OLAP-Cubes erzeugt, Reports gestaltet oder ETL-Prozesse erzeugt werden. Diese grafischen Editoren sind für den Betrieb des Servers jedoch nicht erforderlich.

Im Folgenden werden die beiden für diese Arbeit wichtigen Komponenten Mondrian, der eingesetzte OLAP-Server und Kettle, das ETL-Modul, genauer untersucht und ihre Einsatzgebiete beschrieben.

3.4.1. Pentaho Data Integration – Kettle

Die Pentaho Data Integration (PDI) dient zur Datenintegration und als ETL-Prozess-Engine. Diese ist aus dem ehemaligen Open-Source-Projekt Kettle hervorgegangen. Mit Hilfe dieser Erweiterung ist es möglich die in Kapitel 2.1 beschriebenen ETL-Prozesse zu modellieren und auszuführen.

Der ETL-Prozess wird durch die Aneinanderreihung diverser Einzelbausteine beschrieben. Hierbei bildet jeder einzelne Baustein einen Teilprozess ab. Die Abbildung 3.4 stellt einen einfachen, aber typischen ETL-Prozess nach:

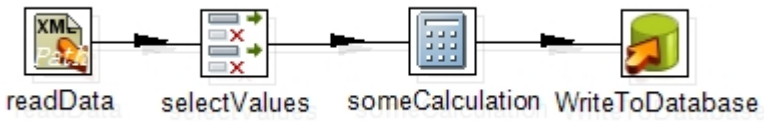

Abbildung 3.4.: Beispiel für einen ETL-Prozess in Kettle (eigene Darstellung)

- Im ersten Schritt (readData) werden Daten aus einer Datenquelle eingelesen.
- Im zweiten Schritt (selectValues) wird eine Selektion der Daten durchgeführt.
- Im dritten Schritt (someCalculation) werden diverse Berechnungen auf den eingelesenen Daten durchgeführt.
- Im letzten Schritt (WriteToDatabase) werden die so erzeugten Daten in eine bereitgestellte Datenbank transferiert.

Kettle bietet hierbei eine große Auswahl an Einzelbausteinen, sodass durch Kombination nahezu jedes ETL-Problem gelöst werden kann. Jedoch können bei komplexen Abfolgen die entstehenden Sequenzen sehr groß und unübersichtlich werden. Gleichzeitig steigt mit wachsender Anzahl von Einzelbausteinen auch die Ausführungsdauer der Gesamtsequenz.

Zur Modellierung und Optimierung dieser Sequenzen wird daher das Tool "Spoon" zur Verfügung gestellt. Hierbei handelt es sich um einen grafischen Editor, mit dessen Hilfe die Einzelbausteine zusammengestellt und konfiguriert werden können. Des Weiteren werden Optionen zum Testen und Optimieren der erzeugten ETL-Prozesse bereitgestellt.

Die Sequenz wird als Dokument mit XML-Syntax gespeichert und in dieser Form im BI-Servers bereitgestellt. Zusätzlich zu dieser Definition wird für die Ausführung auf dem BI-Server noch eine *XAction* benötigt. Hierbei handelt es sich um ein speziell strukturiertes Dokument, welches den Zugriff auf die Kettle-Sequenz bereitstellt. Erst wenn beide Dokumente zur Verfügung stehen, kann die Transformation vom BI-Server bereitgestellt werden und mittels des Solution-Repositories aufgerufen werden.

Im Folgenden wird das Modul "Pentaho Data Integration" stets "Kettle" genannt.

3.4.2. Pentaho Analysis Services Community Edition – Mondrian

Bei Mondrian handelt es sich um die in Pentaho integrierte OLAP-Komponente. Sie ist in der Lage OLAP-Schemata, welche als XML-Dokument im Solution Repository bereitgestellt werden, zu interpretieren. Außerdem ermöglicht Mondrian die Analyse der hierdurch bereitgestellten OLAP-Würfel durch Anfragen in Form von multidimensionalen Ausdrücken (bspw. MDX).

Zur Erstellung einer von Mondrian lesbaren Schema-Definition wird der sogenannte "Cube-Designer" zur Verfügung gestellt. Mit Hilfe dieses Tools ist es möglich eine bereitgestellte Datenquelle für die Analyse in das benötigte XML-Schema zu überführen. Alternativ kann dieses auch manuell erstellt werden. Ein Beispiel findet sich im Kapitel 5.1.4.

3.5. Anbinden eines agilen Data Warehouse

Dank der in Kapitel 3.3 beschriebenen, bereits existierenden Projekte ist eine für die Anbindung eines Data Warehouses nötige Umgebung geschaffen. Die in der ProGoalML spezifizierten Werte werden durch die erzeugten Web-Services erfasst und im Anschluss durch die CEP-Engine weiter aufgearbeitet. Die hierdurch erzeugten Ergebnisdateien dienen dem Data Warehouse als Datenquelle, welche mittels eines zu erzeugenden ETL-Prozesses ausgelesen werden müssen.

Als direkte Folge aus dieser Anbindung ergibt sich eine entscheidende Vereinfachung des nötigen ETL-Prozesses. Da alle zu speichernden Daten bereits die CEP-Engine durchlaufen haben, besitzen sie ein einheitliches, bereits bekanntes Datenformat. Des Weiteren liegen

diese Daten bereits in ihren Überdeckungsklassen, wodurch ein Großteil der nötigen Transformationsarbeit entfällt.

Berücksichtigt man nun noch die zur Verfügung gestellte ProGoalML, so werden alle für die Erstellung und den Betrieb eines Data Warehouse nötigen Informationen durch den Gesamtprozess bereitgestellt. Aus diesen Informationen müssen ein Datenbankschema, ein OLAP-Schema sowie der ETL-Prozess erzeugt werden. Mit Hilfe dieser Kerndokumente kann dann die Datenbank sowie der Pentaho BI-Server konfiguriert werden. Ist dies erfolgt, kann mit Hilfe des erzeugten ETL-Prozesses das Data Warehouse mit den einkommenden Daten befüllt werden.

Sollte sich die zugrundeliegende ProGoalML verändern, so müssen sich auch alle nachgelagerten Prozesse anpassen. Dies gilt auch für die dem Data Warehouse zugrundeliegenden Strukturen. Neben der Neugenerierung von OLAP-Schemata und ETL-Prozessen muss hierfür auch die Datenbank verändert werden. Hierbei dürfen jedoch keine bereits erfassten Daten verfälscht werden oder verloren gehen.

Sind diese Bedingungen erfüllt, so können unterschiedliche, nachgelagerte Funktionen an das Data Warehouse angebunden werden. So ist beispielsweise ein Dashboard zur Echtzeitanalyse der erfassten Daten integrierbar. Aber auch Bewertungs- und Analysewerkzeuge sowie Monitoring-Tools können mit Daten aus dem Data Warehouse interagieren.

4. Verwandte Arbeiten

Das Vermindern des Arbeitsaufwands und gleichzeitige Beschleunigen des Entwicklungsprozesses ist in jeder Technologieentwicklung im Bereich der Informationstechnologie ein Thema. Entsprechend finden sich auch im Bereich der Data Warehouse Erstellung unterschiedliche Ansätze diesen Vorgang zu automatisieren oder zumindest zu beschleunigen. Diese Ansätze lassen sich in zwei Hauptkategorien unterteilen. (Semi-)automatische Erzeugung von Teildokumenten, welche für das Einrichten eines Data Warehouses benötigt werden sowie die Veränderung des Entwicklungsprozesses.

Bei der automatischen Erzeugung wird auf unterschiedliche vorgelagerte Informationen, Definitionen oder Strukturen zurückgegriffen, um mit diesen zusätzlichen Informationen benötigte Dokumente für ETL-Prozesse oder OLAP-Schemata zu erzeugen. Bei der Veränderung des Entwicklungsprozesses wird hingegen das eingentliche Vorgehen der Entwicklung verändert. Ziele hierbei sind vor allem die schnellere Erzeugung von Teilergebnissen sowie eine agilere Gestaltung des Entwicklungsprozesses. Dies soll es ermöglichen möglichst schnell auf Veränderungen in den Anforderungen oder in den vorgelagerten Datenstrukturen zu reagieren.

4.1. Veränderung des Entwicklungsprozesses

Bei der Erstellung von Data Warehouses wird meist auf klassische Entwicklungsansätze aus der Softwareentwicklung zurückgegriffen. Ein Beispiel hierfür ist das Wasserfallmodell. Diese Modelle folgen klar strukturierten Abläufen. Aufgrund dieser starren Entwicklungsstruktur ist es mit ihnen jedoch nur schwer möglich, auf kurzfristige Änderungen zu reagieren. So muss bei umfassenderen Änderungen der Anforderung oder der Ausgangslage ein großer Teil der bereits durchlaufenen Planungs- und Entwicklungsschritte erneut durchgeführt werden. Es fehlt diesen Modellen in ihrer Reinform an Agilität in Bezug auf ihren Ablauf. Dies führt bei nicht vollständig vorhersehbaren Projekten zu erhöhter Entwicklungsdauer und entsprechend zu höheren Kosten [Hugo8].

In der Softwareentwicklung werden diesen „alten“ Modellen neue Vorgehensweisen mit agilen und entsprechend flexibleren Entwicklungsprozessen gegenübergestellt. Prominenteste Beispiele hierfür sind das sogenannte Prototyping sowie der Scrum-Ansatz. Der Hauptunterschied besteht darin, dass die neuen Prozesse bereits mit Teilinformationen den Entwicklungsprozess anstoßen können. Dies führt zu Teilprodukten (Prototypen), welche schrittweise weiterentwickelt und auf vollständigen Funktionsumfang ausgebaut werden [PRo6]. Hierdurch ist es möglich, dass bereits zu einem frühen Zeitpunkt in der Entwicklung erste

Ergebnisse bewertet und Probleme und Fehlplanungen erkannt werden können. Gleichzeitig kann in regelmäßigen Abschnitten die Richtung der Entwicklung korrigiert oder neu ausgerichtet werden.

Dieser Ansatz wird seit wenigen Jahren auch auf die Entwicklung von Data Warehouse Strukturen übertragen und insbesondere von Herrn Ralph Huges [Hug08] propagiert. Die Ziele hierin entsprechen jenen, die auch in der reinen Softwareentwicklung verfolgt werden: Verringerung der benötigten Zeit, Minimierung der Kosten sowie Verkleinerung des benötigten Personalpools. Gleichzeitig kann bereits während der Entwicklung auf Teilergebnisse zugegriffen werden. Diese können frühzeitig getestet und mit (Kunden-)Feedback weiter verbessert und verfeinert werden. Vorher unklare oder nicht eindeutige Anforderungen werden so mit Fortschreiten der Entwicklung vervollständigt.

4.2. (Semi-)Automatisierung

Ein weiterer Ansatz, die Erstellung und Verwaltung von Data Warehouses agiler zu gestalten, beschäftigt sich mit der (semi-)automatisierten Generierung von Informationen, welche zur Initialisierung und Wartung eines Data Warehouse benötigt werden [HSB00]. So ist die Erstellung von ETL-Prozessen, OLAP-Schemata, Stern-Schemata, SQL-Scripten und ähnlichen Dokumenten denkbar. Voraussetzung für die Erstellung dieser Kerndokumente ist jedoch, dass unterschiedlichste Informationen und Metadaten zur Verfügung stehen. Innerhalb des in Kapitel 3 vorgestellten APro-Prozesses können diese Informationen größtenteils aus den vorgelagerten Prozessschritten ausgelesen werden (ProGoalML). Alternativ könnten die fehlenden Informationen aus zusätzlichen Quellen bereitgestellt werden.

Das automatisierte Erzeugen dieser Kerndokumente führt zur Verringerung des für die Erstellung und Anpassung eines Data Warehouse nötigen Aufwands. Dies ermöglicht schnelle Änderungen und Erweiterungen der entsprechenden Data Warehouse Strukturen mit minimalen Kosten [MnMT09]. Ziel dieser Ansätze wird es sein, ein voll automatisiertes Data Warehouse zu erzeugen, welches ohne weitere Eingriffe durch eine Person erzeugt und verändert werden kann.

Im Folgenden werden mehrere Arbeiten untersucht, welche sich mit dieser Automatisierung beschäftigen.

4.2.1. OLAP-Schemata mittels Conceptual Graphical Models

Im Zuge der Erstellung eines Data Warehouse werden oft sogenannte *Computer Aided Warehouse Engineering* (CAWE) Tools eingesetzt [HSB00]. Mit diesem Hilfsmittel ist es einem Entwickler möglich, ein Data Warehouse mit Hilfe von graphischen Notationen zu beschreiben. Dies ist vergleichbar mit der graphischen Beschreibung von Geschäftsprozessen durch BPMN-Modelle. Diese Notation ist vom Zielsystem unabhängig und kann somit für unterschiedliche BI-Lösungen genutzt werden.

Ausgehend von solchen graphischen Notationen werden in [HSB00] des FORWISS München OLAP-Schemata erzeugt. Hierzu werden mit dem CAWE-Tool BabelFish multidimensionale Schemata generiert, welche als Grundlage zur Generierung von OLAP-Schemata genutzt werden. Diese können grafisch als ME/R (Multidimensional Entity-Relationship Model) dargestellt werden. Bei ME/R handelt es sich um eine Erweiterung des klassischen Entity-Relationship Models (ER) speziell für multidimensionale Modellierung, wie sie bei OLAP-Würfeln benötigt wird.

Die Transformation dieser Rohdaten wird durch den Einsatz einer Grammatik ermöglicht. Die von BabelFish generierten Schemata können durch einen typisierten Graphen dargestellt werden. Dieser wird mit Hilfe der in [SBH00] näher definierten Grammatik geparst und im Anschluss an das Datenmodell des Zielsystems angepasst. Hierdurch kann sich der Entwickler auf die Erzeugung der nötigen Data Warehouse Strukturen konzentrieren, ohne zunächst die Zielplattform berücksichtigen zu müssen.

4.2.2. ETL Prozesse auf Basis von Conceptual Models

Die Generierung und Wartung von ETL-Prozessen ist in Data Warehouse Projekten ein entscheidender Faktor [Solo5]. Entsprechend wurden diverse Versuche unternommen, den Erstellungsprozess zu vereinfachen. Eine der Schwierigkeiten hierbei liegt darin, dass die unterschiedlichen ETL-Frameworks eigene Notationen besitzen und keinen gemeinsamen Standard verwenden. Um diese Problematik zu umgehen wird ein ETL-Prozess nach [Simo5] zunächst als Conceptual Model bereitgestellt. Hierbei handelt es sich um eine plattformunabhängige Beschreibung von ETL-Prozessen. Das Ziel von Conceptual Models ist die abstrakte Darstellung von Anforderungen, welche von Menschen verstanden werden kann und gleichzeitig formal und vollständig ist. Hierdurch können sie eindeutig in das nächste logische Schema transformiert werden [TBC99].

Das Conceptual Model wird anschließend unter Einsatz von sogenannten Mappings in ein logisches Modell transformiert. Mappings beschreiben, wie die einzelnen Elemente des Conceptual Models in entsprechende Elemente im logischen Modell transformiert werden können. Dieses Vorgehen ist vergleichbar mit der in Kapitel 4.2.1 verwendeten Grammatik zur Übersetzung von Graphen in OLAP-Schemata. Im vorliegenden Vorschlag geschieht dies mittels eines semiautomatischen Ansatzes.

Einen ähnlichen Ansatz verfolgt das Department of Software and Computing Systems der Universität von Alicante. Im Gegensatz zum Ansatz der Forscher aus München ist ihr Ziel jedoch die automatische Erstellung von ETL-Prozessen.

Grundidee dieses Ansatzes ist es, sogenante Platform Independent Models (PIM) zu erstellen, welche das Conceptual Model darstellen. Ein solches Modell ist, wie der Name schon andeutet, eine plattformunabhängige Beschreibung eines ETL-Prozesses. Diese kann mit Hilfe der deklarierenden Sprache QVT (Query/ View/ Transformation) in Plattform Specific Models (PSM) transformiert werden. Dieser Vorgang wird in Abbildung 4.1 dargestellt. Das plattformunabhängige PIM wird mittels einer Transformation in ein an eine Plattform

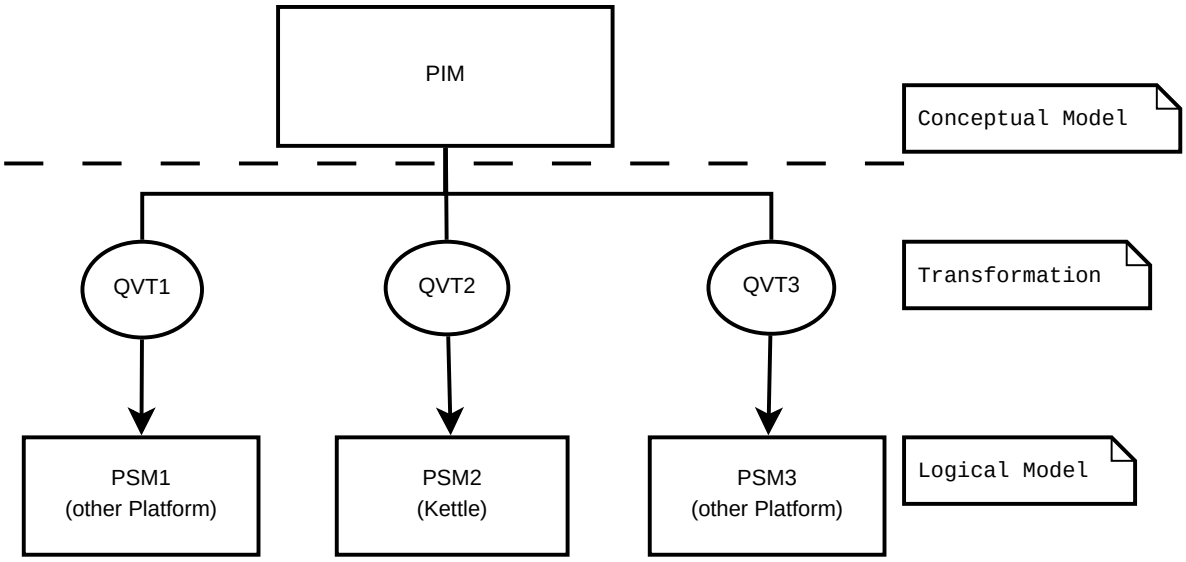

Abbildung 4.1.: Transformation eines Conceptual Models in ein logisches Modell am Beispiel von PIM, QVT und PSMs (eigene Darstellung nach [MnMT09])

gebundenes logisches Model (PSM) transformiert. In [MnMT09] wird als Grundlage zur Modellierung von PIMs eine primitive UML Notation vorgeschlagen.

Hierdurch gelingt es, den Erzeugungsprozess von ETL-Prozessen zu generalisieren und zu modularisieren, da einmal erzeugte PIMs für jede Plattform genutzt werden können. Hierfür muss lediglich eine entsprechende QVT-Transformation entwickelt werden.

4.2.3. Automatisierung von Schema Design

Neben der (semi-)automatischen Erstellung von OLAP-Schemata und ETL-Prozessen existieren auch Ansätze, mit denen die Erzeugung von Datenbankschemata erfolgen kann. Dieser Ansatz wird in [PD02] beschrieben.

Wie bereits bei der Generierung von ETL-Prozessen und OLAP-Schemata dient erneut ein Conceptual Model als Ausgangspunkt für die Schema-Generierung. Als Conceptual Model dient, wie in dem in Kapitel 4.2.1 vorgestellten Ansatz zur Generierung von OLAP-Schemata, ein ME/R Model. Als Ziel der Transformation in ein logisches Schema wird das in Kapitel 2.3 beschriebene Sternschema verwendet.

Die Schema-Generierung beruht auf zwei Voraussetzungen:

1. Numerische Felder im ME/R Model repräsentieren Measures. Je mehr dieser Felder eine bestimmte Entität besitzen, umso wahrscheinlicher ist, dass es sich bei dieser Entität um einen Fakt im Sinne der Faktentabelle des Sternschemas handelt.
2. Die Kardinalität einer Verbindung bestimmt, wie nützlich eine zugehörige Entität in Bezug auf die Schema-Generierung ist.

Unter Berücksichtigung dieser beiden Punkte wird aus einem gegebenen ME/R Model in mehreren Schritten ein Sternschema erzeugt:

1. Für Entitäten mit numerischen Feldern werden Fakt-Knoten erzeugt.
2. Basierend auf diesen Feldern werden numerische Attribute im erzeugten Knoten erstellt.
3. Erzeugen von Datums- und Zeit-Dimensionen für entsprechende Felder einer Entität.
4. Erzeugen von Dimensionen für die verbleibenden Attribute.
5. Rekursive Abarbeitung der Beziehung der einzelnen Entitäten, um zusätzliche Dimensionen zu erzeugen.

Mit diesem Ansatz gelingt es, aus einem gegebenen ME/R Model ein Sternschema zu erzeugen. Werte mit einem numerischen Datentyp werden als Fakten in die Faktentabelle aufgenommen. Nicht numerische Werte werden als Dimensionen angelegt. Das hierdurch erzeugte Sternschema kann nach Umwandlung in die Zielplattform als Datenbankschema für ein Data Warehouse genutzt werden.

4.3. Entwurfsentscheidungen

Wie bereits in Kapitel 3 beschrieben, soll das durch diese Arbeit vorgestellte agile Data Warehouse in einen komplexen Gesamtprozess integriert werden. Dieser soll nach Möglichkeit als Software-As-A-Service angeboten werden können. Hierdurch werden für jeden einzelnen Kunden eigene individuelle Warehouse Strukturen benötigt. Diese mittels eines Teams von Experten zu erzeugen, anzupassen und auf aktuellem Stand zu halten ist möglich, aus wirtschaftlicher Sicht jedoch nicht vorteilhaft. Ziel sollte es folglich sein, eine möglichst eigenständige, automatisierte Lösung zu erarbeiten, welche wenige bis keine manuell auszuführenden Schritte beinhaltet.

Entsprechend ist die Anwendung eines Scrum-Prozesses auf die Erstellung eines Data Warehouses im vorliegenden Fall nicht zielführend. Zwar würden die Kosten sowie Zeit- und Personalbedarf gegenüber einem „klassischen“ Vorgehen geringer ausfallen, wären aber dennoch wirtschaftlich unrentabel. Außerdem entspricht dieser Ansatz nicht der in Kapitel 1.1 gegebenen Definition von „agil“. Dennoch ist der Einsatz von Fachkräften in Hinsicht auf die Optimierung der Datenstrukturen gegenüber der automatisierten Variante im Vorteil. Durch jede Automatisierung besteht die Gefahr, dass ein gewisser Grad an Optimierungspotential aufgrund des generischen Ansatzes verloren geht.

Die (semi-)automatische Generierung von Kerndokumenten verspricht unter den gegebenen Randbedingungen das gewünschte Ziel zu erreichen. Dank der in Vorprozessen bereitgestellten Dokumente in Form der ProGoalML sowie generierten Informationen wie die Ergebnisdateien, ist bereits ein großer Teil der nötigen Informationen zur OLAP-Schema

Generierung vorhanden. Hinzu kommen durch die CEP-Engine aufgearbeitete und aggregierte Werte. Diese verringern den Aufwand des ETL-Prozesses und vereinfachen somit die Erstellung der hierfür nötigen Informationen.

Dank des Solution-Repositories des Pentaho BI-Servers bietet sich die Möglichkeit für die Erstellung eines vollautomatisierten Data Warehouses, welches sich unter Berücksichtigung zuvor generierter Kerndokumente selbstständig einrichten kann. Auch auf nachträgliche Veränderungen im Prozessmodell oder der gesammelten Werte, kann durch die Neugenerierung der einzelnen Kerndokumente reagiert werden. Im Anschluss müssen diese veränderten Kerndokumente in das bestehende System eingebracht und angewendet werden. Dies ermöglicht eine Einbindung in eine SAAS-Umgebung bei gleichzeitig klein gehaltenen Betriebs- und Erstellungskosten.

Im Kapitel 4.2 wurden bereits drei Ansätze vorgestellt, die sich mit der automatischen Erzeugung von Teilaspekten eines Data Warehouse beschäftigen. Alle drei Ansätze bauen auf eine vom Zielsystem unabhängige Definition auf, um im Folgenden mit Hilfe von Transformationsschritten einzelne Teildokumente, welche für die Erzeugung des Data Warehouses nötig sind, zu erstellen. Die einzelnen vorgestellten Lösungen bieten jedoch jeweils nur einen Lösungsansatz für die Generierung eines einzelnen Kerndokuments. Hinzu kommt, dass die Generierung von ETL-Prozessen in Kapitel 4.2.2 nicht mit Hilfe von ME/R Modellen erfolgt. Des Weiteren steht im vorliegenden Gesamtprozess, beschrieben in Kapitel 3, kein ME/R Modell oder PIM zur Verfügung. Eine einfache Kombination dieser drei vorhandenen Ansätze sowie die Integration in den vorliegenden Gesamtprozess ist folglich nicht möglich.

Dennoch steht in Form der ProGoalML ein Dokument zur Verfügung, welches als Conceptual Model genutzt werden kann. Hierdurch kann als Ausgangspunkt ein plattformunabhängiges Modell mit anschließenden Transformationsschritten für die Umsetzung eines agilen Data Warehouse genutzt werden. In den folgenden Kapiteln werden Möglichkeiten beschrieben, wie mit Hilfe dieses Ansatzes aus den in der ProGoalML bereitgestellten Informationen die drei Kerndokumente erzeugt werden können. Hierbei werden auch die in Kapitel 4.2 vorgestellten Ansätze und bereits beschriebene Vorgehen berücksichtigt. So dient beispielsweise der Datentyp eines Wertes zur Einordnung in Dimensionen und Fakten (vgl. Kapitel 5.1.3). Auch die Umsetzung selbst wird dem in Abbildung 4.1 dargestellten Vorgehen folgen: Ausgehend von der ProGoalML werden mittels Transformationsschritten die gewünschten Kerndokumente erzeugt.

Unter diesen Voraussetzungen bietet der (semi-)automatische Ansatz folglich klare Vorteile in vielen der genannten Bereiche. Außerdem deckt er sich mit der in Kapitel 1.1 getroffenen Definition von „agil“. Daher wird im Folgenden dieser Ansatz weiter verfolgt und es werden mit seiner Hilfe alle benötigten Schritte durchgeführt.

5. Konzept

Unter Berücksichtigung der im vorangegangenen Kapitel getroffenen Entwurfsentscheidung zu Gunsten der (semi-)automatischen Generierung von Kerndokumenten soll in diesem Kapitel ein Konzept für deren Umsetzung dargestellt werden. Das Konzept soll hierbei in das im Kapitel 3 dargestellte Gesamtkonzept integriert werden. Hierzu wird auf die im Kapitel 2 beschriebenen Grundlagen zurückgegriffen.

Zur Einrichtung eines Data Warehouses mit Hilfe des Pentaho BI-Servers müssen drei Kerndokumente erzeugt werden (siehe Kapitel 3). Diese Kerndokumente müssen aus Informationen erstellt werden, welche in der ProGoalML enthalten sind. Da das Data Warehouse nicht nur initialisiert werden muss, sondern auch an Veränderungen des zugrundeliegenden Prozesses angepasst werden muss, kann prinzipiell zwischen zwei Situationen unterschieden werden:

Situation A: Das Data Warehouse wurde noch nicht initialisiert. In diesem Fall muss keine Rücksicht auf bereits vorhandene Daten oder alte Datenstrukturen genommen werden. Diese Situation wird im Kapitel 5.1 näher erläutert.

Situation B: Es existiert bereits eine ältere Instanz des Data Warehouse, welche bereits gesammelte Informationen enthält und produktiv eingesetzt wird. Diese soll nun modifiziert werden. Um konsistente Daten und Analysen zu gewährleisten, muss hierfür auf bereits vorhandene Strukturen und Datensätze Rücksicht genommen werden. Dieser Fall wird in Kapitel 5.2 untersucht.

5.1. Erstellung von Kerndokumenten

Für die erstmalige Einrichtung eines Data Warehouses werden drei Kerndokumente benötigt: ein Datenbankschema, eine Definition der zugehörigen OLAP-Würfel sowie die benötigten ETL-Prozesse. Als zentrale Informationsquelle soll eine bereitgestellte ProGoalML genutzt werden. Aus dieser lassen sich, wie in Kapitel 3.2 erwähnt, alle für die Generierung der Kerndokumente nötigen Informationen auslesen. Im Folgenden werden die für diese Transformation nötigen Schritte sowie hierin auftretende Probleme beschrieben.

5.1.1. Multiple OLAP-Würfel

Eine einzelne ProGoalML kann mehrere Überdeckungsklassen enthalten. Jede Überdeckungsklasse stellt einen eigenen Kontext dar. Um dies mittels eines Data Warehouses umzusetzen, wird für jede Überdeckungsklasse ein eigener OLAP-Würfel benötigt. Dies ist nötig, da Werte aus zwei unterschiedlichen Überdeckungsklassen keinen kontextuellen Zusammenhang besitzen. Dies hat zur Folge, dass für jede dieser Klassen auch ein eigenes Sternschema sowie zugehörige ETL-Prozesse erzeugt werden müssen.

Durch Umwandeln der ProGoalML in ihre einzelnen Ergebnis-Schemata erhält man die in der ProGoalML spezifizierten Überdeckungsklassen. Diese können als Grundlage für die Generierung der Kerndokumente genutzt werden. Informationen, welche in einem Ergebnis-Schema nicht vorhanden sind, wie zum Beispiel die Berechnungsformel einer KPI, müssen weiterhin aus der ProGoalML ausgelesen werden.

5.1.2. Abhängigkeit der Kerndokumente

Als Ausgangsdokument für die Generierung von Kerndokumenten dient die ProGoalML. Für die Transformation der ProGoalML in die einzelnen Kerndokumente sind jedoch weitere Informationen nötig. So müssen Ergebnis-Schemata generiert werden, um Zugriff auf die zu erzeugenden Überdeckungsklassen zu erhalten. Diese führen zu multiplen OLAP-Würfeln, Sternschemata sowie dazugehörigen ETL-Prozessen. Darüber hinaus können ETL-Prozesse und OLAP-Schemata nicht ohne Kenntnis der Datenbankstruktur erzeugt werden, da sie direkt mit der Datenbank interagieren müssen.

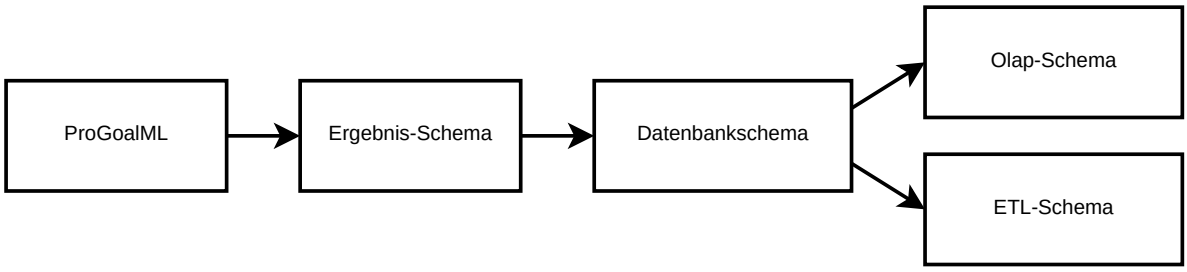

Abbildung 5.1.: Darstellung eines Generierungsprozesses mit Datenbankschema als zentrales Dokument (eigene Darstellung)

Diese Bedingungen führen zu dem in Abbildung 5.1 dargestellten Abhängigkeitsgraphen. Als Ausgangspunkt dient die ProGoalML, aus welcher im folgenden Schritt die Ergebnis-Schemata erzeugt werden. Unter Berücksichtigung dieser beiden Dokumententypen kann ein Datenbankschema erzeugt und bereitgestellt werden. Steht auch dieses zur Verfügung, können OLAP-Schema und ETL-Prozess erzeugt werden.

Als positiver Nebeneffekt dieser Abhängigkeit ergibt sich eine Vereinfachung der Umsetzung des Change-Managements. In die ProGoalML eingebrachte Veränderungen müssen lediglich bei der Erzeugung des Datenbankschemas berücksichtigt werden. Da nachfolgend erzeugte

Dokumente auf dieser basieren, sind auch Anpassungen an die neuen Bedingungen bereits berücksichtigt.

5.1.3. Generierung des Datenbankschemas

Die Erstellung eines geeigneten Datenbankschemas ist zentrales Thema bei der Erzeugung jedes Data Warehouse. Fehler hierin können zu fehlerhaften Aussagen und Performanceeinbrüchen bei Abfrage- oder Schreiboperationen führen [Leh03]. Zur Nutzung innerhalb von Pentaho benötigt man eine Datenbank auf Basis des Sternschemas (siehe Kapitel 2.3). Dies hat zur Folge, dass man eingehende Daten in zwei Gruppen unterteilen muss: *Fakten* und *Dimensionen*.

Kennt man den Kontext, in welchem die eingehenden Daten erfasst werden, ist diese Unterscheidung einfach. Bei einem (semi-)automatischen Ansatz, wie er in dieser Diplomarbeit verfolgt wird, ist diese Unterscheidung jedoch oft sehr schwierig. Um dennoch eine praktikable Einteilung zu erreichen, können unterschiedliche Lösungsansätze verfolgt werden.

Ein Fallbeispiel: Es soll ein Schema auf Basis folgender Informationen erzeugt werden:

ID: INTEGER; NAME: STRING; WERT: LONG; STARTZEIT: INTEGER; PLZ: INTEGER; TYP: ENUM

Statische Einordnung nach Datentyp: Zum Zeitpunkt der Erzeugung der Kerndokumente sind neben den Namen der zu erfassenden Werte auch deren Datentypen bekannt. Anhand dieses Datentyps kann eine Unterscheidung in Fakt oder Dimension erfolgen. Dies birgt jedoch diverse Einschränkungen.

Numerische Werte können in vielen Fällen als Fakten angesehen werden, wohingegen String-, Enum- und Boolsche-Werte als Dimensionen angesehen werden können [Leh03]. Leider handelt es sich hierbei nur um einen groben Richtwert, dessen Grenzen fließend sind. Je nach Kontext der erfassten Werte sollten numerische Werte als Dimensionen behandelt werden. Ein Beispiel hierfür wäre die Erfassung von Zeitpunkten. Diese sollten als Dimension betrachtet werden, um eine bessere Analyse zu gewährleisten (vgl. Kapitel 4.2.3).

Am Fallbeispiel: Nach dem beschriebenen Muster würde sich folgende Einteilung ergeben:

Fakt: ID, Wert, Startzeit, PLZ

Dimension: Name, Typ

Jedoch ist aus dem Kontext heraus ersichtlich, dass es sich bei der Startzeit um einen Zeitpunkt handelt. Dieser sollte als Dimension dargestellt werden. Ähnlich verhält es sich beim Feld PLZ. Dieses möchte man innerhalb einer Analyse meist zur örtlichen Gruppierung von Daten verwenden und nicht aggregieren.

Die Einordnung nach Datentyp bietet zwar die Möglichkeit zur vollautomatisierten Erzeugung von Sternschemata, führt jedoch in den meisten Fällen zu Ergebnissen welche die Auswertung erschwerend gestalten.

Einordnung nach Datentyp und Kontextanalyse: Eine Möglichkeit die zuvor beschriebene Kontextproblematik abzuschwächen bietet eine Kontextanalyse. Zunächst wird eine statische Einordung nach Datentyp durchgeführt. Im Anschluss werden die einzelnen Felder mit Hilfe von Kontextinformationen erneut überprüft und gegebenenfalls neu eingeordnet.

Voraussetzung für dieses Vorgehen ist die Möglichkeit zur Kontextanalyse. Steht kein Kontext zur Verfügung oder ist dieser nicht auswertbar, so führt dies zur selben Unterteilung wie sie durch die statische Einordnung erzeugt würde. Zusätzlich ist eine Kontextanalyse meist mit aufwändigen Algorithmen und einer hohen Laufzeit verbunden.

Betrachtet man das Fallbeispiel, so bestände die einzige Möglichkeit einer Kontextanalyse in der Auswertung der Feldnamen, da keine weiteren Informationen angeboten werden.

Die im Kontext dieser Diplomarbeit als Ausgangspunkt dienende ProGoalML bietet hingegen eine größere Menge von Informationen, die mit Hilfe einer Kontextanalyse ausgewertet werden könnten. So ist es beispielsweise möglich, auf das BPMN-Modell zurückzugreifen und dieses als Grundlage einer Kontextanalyse zu verwenden.

Gelingt es einen solchen Algorithmus zu finden, welcher eine erfolgreiche Kontextanalyse gewährleistet, so ist auch mit diesem Ansatz eine vollautomatisierte Erzeugung des Sternschemas möglich. Im Gegensatz zur statischen Einordnung sind die hierdurch erzeugten Schemata detaillierter und besser für die Nutzung mittels OLAP geeignet.

Einordnung mit externen Informationen: Als dritter Lösungsansatz ist es möglich externe Zusatzdefinitionen zu nutzen. Grundlage bildet zunächst wieder die statische Einordnung nach Datentypen. Im Gegensatz zur Kontextanalyse wird jedoch auf eine externe Informationsquelle zurückgegriffen, welche zusätzliche Informationen und Anweisungen enthält.

Am Beispiel würde dies bedeuten, dass unsere externe Zusatzdefinition Argumente enthält, die besagen, dass Integer-Werte mit dem Namen PLZ oder Startzeit als Dimension zu behandeln sind. Dies würde zu folgender Zuordnung führen:

Fakt: ID, Wert

Dimension: Name, Typ, Startzeit, PLZ

Dieser Ansatz bringt jedoch seine eigenen Einschränkungen mit sich. Zunächst gewinnt man Flexibilität in Bezug auf Dimensionen und Fakten sowie die Möglichkeit diese Einstellungen situationsbedingt zu verändern. Andererseits führt dies dazu, dass nur noch eine semiautomatische Erzeugung des Sternschemas möglich ist. Gleichzeitig hängt die Qualität des Sternschemas vom Detailgrad und der Vollständigkeit der externen Definition ab.

5.1.4. Generierung des OLAP-Schemas

Die Erstellung des zweiten Kerndokumentes, dem OLAP-Schema, ist im Vergleich zu den vorangegangenen Ausführungen einfach. Ein OLAP-Schema, wie es für Pentaho benötigt wird, ist eine XML-Datei, die nach dem Mondrian-Schema aufgebaut werden muss. Abbildung 5.1 zeigt ein Minimalbeispiel auf Basis des bereits bekannten Fallbeispiels.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <Schema name="BeispielSchema">
3      <Cube name="BeispielWuerfel" cache="true" enabled="true">
4          <Table name="beispielTabelle"/>
5          <Dimension name="dim_Name" foreignKey="name_id">
6              <Hierarchy primaryKey="name_id">
7                  <Table name="dim_name"/>
8                  <Level name="Name" column="Name" type="String" />
9              </Hierarchy>
10         </Dimension>
11                 <Dimension name="dim_Startzeit" foreignKey="startzeit_id">
12             <Hierarchy primaryKey="startzeit_id">
13                 <Table name="dim_startzeit"/>
14                 <Level name="Startzeit" column="Startzeit" type="integer" />
15             </Hierarchy>
16         </Dimension>
17         <Measure name="Wert" column="Wert" datatype="Numeric" aggregator="sum"/>
18         <Measure name="Wert_AVG" column="Wert" datatype="Numeric" aggregator="avg"/>
19                 <Measure name="PLZ" column="PLZ" datatype="Numeric" aggregator="sum"/>
20     </Cube>
21 </Schema>
```

Listing 5.1: Ein Beispiel OLAP-Schema (eigene Darstellung)

Ein Schema besteht aus beliebig vielen Cube-Blöcken, welche jeweils für einen einzelnen OLAP-Würfel stehen. Hierin werden Dimensionen und Measures angelegt. Eine Dimension enthält beliebig viele Hierarchien (im Beispiel: „Hierarchy“) und Level. Verzweigungen innerhalb einer Hierarchie werden als seperate Hierarchie modelliert. Measures entsprechen Fakten mit beliebiger Aggregation. Die möglichen Aggregationen reichen hierbei von einer einfachen Berechnung der Summe, über Min- und Max-Werte bis hin zur Durchschnittsberechnung.

Das Erzeugen dieser Strukturen ist eine leicht zu lösende Aufgabe, sofern man bereits auf eine fertige SQL-Definition zurückgreifen kann. Hierin definierte Dimensionstabellen werden als Dimensionen in den Würfel übernommen. Fakten werden als Measures zur Verfügung gestellt. Steht kein entsprechendes Stern-Schema zur Verfügung, so muss dieses erstellt oder ausgelesen werden, bevor das OLAP-Schema erzeugt werden kann.

Allerdings birgt auch diese Transformation ihre Schwierigkeiten. Das Ziel auf Ebene der Measures ist es, möglichst sinnvolle Aggregationsmöglichkeiten auf den einzelnen Werten zu erzeugen. Diese sind jedoch vom Kontext des Wertes abhängig. So ist eine Summe über einer Menge von Postleitzahlen, wie in Abbildung 5.1 gezeigt, unbrauchbar bei der

Auswertung. Zusätzlich können auch beim Erzeugen von Dimensionen Fragen auftreten. So ist die Reihenfolge der Level einer Hierarchie vom Kontext der Dimension abhängig. Es muss klar definiert werden, in welcher Reihenfolge die einzelnen Level einer Datumsdimension bereitgestellt werden sollen. Ansonsten ist nicht ersichtlich, ob bspw. der gespeicherte Wert 3 für einen Tag oder den Monat März steht. Hinzu kommt, dass das Anlegen von zusätzlichen Hierarchien ohne zusätzliche Informationen zu deren Struktur nicht möglich ist.

Ähnlich den Lösungsansätzen bei der Generierung des Datenbankschemas führt ein Ansatz zu einer vollautomatisierten, ein anderer mit Zusatzinformationen zu einer semiautomatischen Lösung.

Lösung mittels festen Werten: Bei diesem Lösungsansatz wird auf Basis des Datentyps eine festgelegte Anzahl von Aggregationen pro Fakt der Faktentabelle angelegt. Ob diese in Bezug auf den Kontext des Wertes sinnvoll genutzt werden können, wird hierbei nicht berücksichtigt. Ein Problem ergibt sich bei der Erzeugung von Hierarchien und Level, da lediglich die Dimensionstabellen und ihre Attribute bekannt sind. Hieraus lassen sich keine sinnvollen Hierarchiestufen ableiten. Die einzige Möglichkeit ist die Erstellung von Hierarchien auf Basis eines zuvor festgeschriebenen Schemas, in Abhängigkeit von Datentyp und Schlagwörten.

Dies kann je nach Umfang der festgelegten Werte zu zwei unterschiedlichen Extremen führen. Das Erste enthält viele und größtenteils unnötige Measures, Level und Hierarchien, um den Analyse-Werkzeugen eine möglichst große Auswahl an Optionen zu bieten. Das Zweite bietet nur eine sehr eingeschränkte Auswahl, welche aber im Allgemeinen sinnvoll genutzt werden kann.

Lösung mittels externer Informationen: Erlaubt es das Projekt auf einen semiautomatisierten Prozess zurückzugreifen, so besteht die Möglichkeit externe Zusatzinformationen in die Lösung mit einzubeziehen. Den Kern bildet weiterhin eine minimale Anzahl von festgelegten, allgemeingültigen Definitionen zur Erzeugung eines minimalen Würfel-Schemas. Dieses wird im Folgenden durch Informationen und Anweisungen aus einer externen Quelle verfeinert.

Im Gegensatz zum zuvor beschriebenen Lösungsansatz ermöglicht dieses Vorgehen speziell an das Projekt angepasste OLAP-Würfel. Jedoch gilt auch hier: Die Qualität des OLAP-Schemas hängt vom Detailgrad und der Vollständigkeit der externen Informationen ab.

5.1.5. Bucket-Dimensionen

In vielen Situationen können erfasste Werte durch Transformation zusätzliche Informationen bereitstellen. Eines der gängigsten Beispiele stellt hierbei der Timestamp dar. So steht der Integer Wert *1325419200* beispielsweise für den *1.1.2012, 12:00Uhr*. Diese Informationen können durch eine Analyse einfacher verarbeitet werden als der ursprüngliche Integer Wert.

Ein Bucket ist eine vorberechnete Wertemenge. Am obigen Beispiel könnte diese beispielsweise folgende Informationen enthalten: *Tag:1; Monat:1; Jahr:2012; Uhrzeit:"12:00"*. Ziel ist es nun, diese Werte innerhalb des Data Warehouses bereitzustellen. Dies hat den Vorteil, dass bei Anfragen direkt auf die berechneten Werte zugegriffen werden kann. Sofern im angefragten Bucket große Datenmengen verarbeitet wurden, sind dadurch erhebliche Laufzeitvorteile möglich.

Im Kontext eines Data Warehouses werden die gewonnenen Buckets zusätzlich als Dimension genutzt. Dies führt zu einfacheren Anfragen und zu leichter zu verarbeitenden OLAP-Würfeln. Mit oben gegebenen Timestamp, ist es z.B. möglich eine Zeitdimension zu erzeugen.

Für die Umsetzung müssen zusätzliche Informationen bereitgestellt werden. Der zu verarbeitende Wert sowie die Eigenheiten der einzelnen Buckets müssen definiert werden. Am Beispiel des bereits bekannten Fallbeispiels könnte dies wie folgt erfolgen:

Bedingung: *Feldname == "Startzeit"*

Buckets: *Tag; Monat; Jahr; Uhrzeit*

Da es sich hierbei meist um kontextabhängige Entscheidungen handelt, sollten diese Informationen aus einer externen Datenquelle bereitgestellt werden. Eine statische Lösung ist lediglich auf Datentyp-Ebene praktikabel und führt in den meisten Fällen zu Problemen in der Transformation.

5.1.6. Generierung des ETL-Prozess

Der Umfang und der Aufwand eines ETL-Prozesses sind abhängig von den zu verarbeitenden Daten. Dank der vorgelagerten CEP-Engine (siehe Kapitel 3.1) ist ein Großteil der nötigen ETL-Arbeit bereits erfolgt. Die zu verarbeitenden Daten werden von einer einzelnen Quelle, der CEP-Engine, in einem einheitlichen, bekannten Datenformat zur Verfügung gestellt. Diese Datenformate entsprechen den aus der ProGoalML erzeugbaren Ergebnis-Schemata.

Zusätzlich wird das zu verwendende Datenbankschema benötigt. Ohne dieses ist eine Interaktion mit der Datenbank nicht möglich, wodurch der ETL-Prozess überflüssig wäre.

Ein ETL-Script unter diesen Voraussetzungen muss folgende Schritte umsetzen:

1. Einlesen von Werten aus einer Ergebnisdatei (siehe 3.4).
2. Falls nötig: Werte für Bucket-Dimensionen transformieren und erzeugen.
3. Aktualisieren der Dimensionen sowie Auslesen der nötigen Fremdschlüssel aus den Dimensionstabellen.
4. Schreiben der Fakten und Fremdschlüssel in die Faktentabelle.

Sofern keine Bucket-Dimensionen erlaubt werden bzw. verfügbar sind, kann dieses Kerndokument vollständig automatisiert erzeugt werden. Sind Bucket-Dimensionen erlaubt, so wird auch für dieses Kerndokument eine externe Informationsquelle benötigt. Diese muss die für die Bucket-Generierung nötigen Zusatzinformationen bereitstellen.

5.2. Change-Management

Im Gegensatz zu der erstmaligen Erzeugung eines Data Warehouses, wie im Kapitel 5.1 beschrieben, müssen bei Änderungen an einer bestehenden Struktur unterschiedliche Situationen betrachtet und entsprechend verarbeitet werden können.

Da bei einer Änderung in den vorgelagerten Systemen alle Teile eines Data Warehouses betroffen sind, kann es zu umfangreichen Umbauten innerhalb der einzelnen Teildokumente kommen. Am Kritischsten hierbei sind Veränderungen innerhalb der Datenbank, da hier bereits gesammelte Daten verloren gehen können. So gehen beispielsweise durch Löschen eines KPI innerhalb der ProGoalML alle bisher gesammelten Kennwerte für diesen Berechnungspunkt verloren. Aber auch Änderungen an OLAP-Würfeln oder innerhalb der Transformationsprozesse können zu Problemen und inkonsistenten Daten führen. Daher ist es sinnvoll vor der Durchführung von Änderungen Sicherheitsmaßnamen zu ergreifen, welche Datenverlust und Konsistenzprobleme verhindern oder korrigieren können.

Im Zuge dieser Arbeit wurde dieser Sicherheitsmechanismus mittels eines Snapshot-Verfahrens realisiert. Hierbei wird vor Einbringen der Änderungen eine Sicherheitskopie des aktuellen Systems erzeugt, welche bei Fehlern oder Problemen erneut eingespielt werden kann. Hierzu zählen das SQL-Script zur Erzeugung der Datenbank, das OLAP-Schema sowie das zugehörige Kettle-Script. Zusätzlich wird noch ein Backup der im Data Warehouse enthaltenen Daten erstellt, so dass ein vollständiger Datenverlust ausgeschlossen werden kann.

Bei einem bestehendem System, das Veränderung unterworfen wird, können die entstehenden Situationen in drei Kategorien unterteilt werden: Löschen von vorhanden Informationen, hinzufügen neuer Dimensionen oder Fakten oder verändern existierender Fakten. Tabelle 5.1 zeigt eine kurze Übersicht über alle Fälle und Lösungsansätze.

5.2.1. Hinzufügen von neuen Informationen

Oftmals ist es aufgrund von Veränderungen nötig, dass zusätzliche Informationen in das Data Warehouse aufgenommen werden müssen. Dies ist der Fall, wenn in der ProGoalML ein neuer KPI, ein neues Ziel oder ein neuer Messpunkt hinzugefügt wird, welcher bisher noch nicht erfasst wurde.

Lösungsansatz:

Das Hinzufügen von weiteren Informationen stellt einen einfachen Problemfall dar. Hierfür müssen lediglich alle vorhandenen Kerndokumente um die hinzugekommenen

Informationen erweitert werden. Dies bedeutet eine Erweiterung der Datenbank sowie die Ergänzung der Transformation und des OLAP-Schemas. Dies kann analog zu dem zur Erzeugung des Data Warehouse genutzten Vorgehen umgesetzt werden und erfordert keine weiteren Anpassungen.

Probleme und Hindernisse:

Im Gegensatz zu anderen Veränderungen muss beim reinen Hinzufügen von Informationen kein Backup der Datenbank erfolgen, da hierbei keine Informationen verändert oder entfernt werden. Wird eine Dimension oder ein Fakt hinzugefügt müssen jedoch sämtliche bereits vorhandenen Daten um die fehlenden Werte ergänzt werden.

Da dies, solange es sich nicht um einen berechneten Kennwert aus einer anderen Spalte handelt, nicht möglich ist, muss hierfür eine Regelung gefunden werden. Die einfachste Möglichkeit ist das Erzeugen von Nullwerten für diese Reihen. Alternativ kann ein Default-Wert eingetragen werden. Handelt es sich um eine Berechnung aus bereits vorgehaltenen Informationen, so kann diese auch für ältere Kennwerte durchgeführt werden.

Beim Hinzufügen von Dimensionen kann auf die gleiche Weise verfahren werden.

5.2.2. Löschen von vorhandenen Strukturen

Durch Veränderungen im BPMN-Modell oder im Bereich der erfassten Messpunkte und KPIs kann es passieren, dass bisher erfasste Werte nicht mehr zur Verfügung gestellt werden. Dies kann auf unterschiedliche Art und Weise behandelt werden.

Probleme und Hindernisse:

Das Löschen von vorhandenen Strukturen stellt den einfachsten Problemfall dar. Neben einzelnen Fakten können ganze Dimensionen oder lediglich einzelne Spalten gelöscht werden. Dabei ist jedoch folgende Überlegung zu tätigen: Wird die entsprechende Information innerhalb der Datenbank gelöscht, so ist diese Löschung permanent. Ein Zurückholen der gelöschten Informationen ist innerhalb des in Kapitel 3 beschriebenen Systems nicht möglich, da das Data Warehouse den einzigen dauerhaften Speicher, neben den erzeugten Backups, darstellt.

Lösungsansätze:

Aufgrund dieser Schwierigkeiten sollten Alternativen zum dauerhaften Löschen von Informationen betrachtet werden.

Lösungsansatz A: Es besteht die Möglichkeit, die bereits gesammelten Daten im System weiterhin vorzuhalten und lediglich innerhalb des OLAP-Schemas nicht mehr zu referenzieren. Hierfür müssten in zukünftigen Datensätzen diese Informationen stets mit Nullwerten aufgefüllt werden. Während hierdurch die Problematik eines Datenverlustes bekämpft werden kann, führt dieses Vorgehen bei vielen Änderungen auf die Dauer zu einer großen Anzahl von nicht mehr gepflegten Kennwerten. Dies kann negative Auswirkungen auf Geschwindigkeit und Wartbarkeit der eingesetzten Datenbank mit sich bringen.

Lösungsansatz B: Durch die Kombination mit der Nutzung eines Snapshot-Verfahrens bei einer Veränderung, kann die Löschung dieser Informationen jedoch ohne direktes Verlustrisiko durchgeführt werden. Die gelöschten Informationen können jederzeit aus dem erzeugten Backup gewonnen werden. Diese Kennwerte sind dann auch noch nicht mit überflüssigen Nullwerten verunreinigt, was eine Auswertung vereinfacht. Hierbei ist jedoch zu beachten, dass diese Informationen bei der Löschung des entsprechenden Backups ebenfalls unwiderruflich verloren gehen.

5.2.3. Verändern von existierenden Informationen

Bei einer Veränderung von existierenden Informationen handelt es sich um den komplexesten Fall einer Aktualisierung, weshalb im Folgenden dieser Fall differenziert und detailliert beschrieben und mögliche Lösungsansätze diskutiert werden.

Verändern eines Datentyps

Ein in der ProGoalML erfasster Messwert, ein KPI oder ein Ziel legt den Datentyp des zugehörigen Kennwertes fest. Dieser Datentyp wird verändert, um einer Änderung im Erfassungsprozess gerecht zu werden.

Probleme und Hindernisse:

Bei der Veränderung eines Datentyps einer Spalte können zwei unterschiedliche Fälle auftreten.

Fall A: Im ersten Fall kann der vorhandene Datentyp in den neuen umgewandelt werden. Dies wäre beispielsweise bei einer Konvertierung von Integer-Werten in Long-Werte der Fall. Ist dies möglich, so können bereits gesammelte Kennwerte in das neue Format konvertiert werden. Jedoch kann auch die Konvertierung von Informationen zu Datenverlust führen. So ist zwar eine Transformation von Float-Werten in Integer-Werte möglich, jedoch gehen viele Informationen dabei verloren.

Fall B: Der zweite Fall tritt ein, wenn eine Konvertierung zwischen dem Ursprungstyp und dem Zieltyp nicht möglich ist. In diesem Fall müssten vorhandene Kennwerte gelöscht werden und durch Default- oder Nullwerte ersetzt werden. Dies würde zu einem vollständigen Datenverlust der entsprechenden Spalte führen.

Lösungsansätze:

Lösungsansatz A: Es besteht die Möglichkeit bereits vorhandene Kennwerte in ihrem eigenen Datenformat vorzuhalten und für neue Datensätze eine zusätzliche Spalte im neuen Datenformat zu erzeugen. Dieses Vorgehen führt jedoch zu Problemen bei der Auswertung der gesammelten Informationen, da die auswertende Instanz Kenntnis über beide Spalten und Gründe für Nullwerte in den entsprechenden Informationen haben muss.

Lösungsansatz B: Alternativ zum Lösungsansatz A könnten Kennwerte grundsätzlich zu jenem Typ konvertiert werden, welcher die höhere Genauigkeit besitzt. So könnten neue Integer-Werte ohne Informationsverlust in das bereits vorhandene Float-Format konvertiert werden. Jedoch verändert sich bei diesem Vorgehen ohne ersichtlichen Übergang der Detailgrad der erfassten Werte. Dies kann beim Verarbeiten und Auswerten zu unerwarteten und ungewünschten Effekten, beispielsweise bei der Generierung von Schaubildern, führen. Deshalb sind auch bei diesem Vorgehen zusätzliche Informationen über Eigenheiten der erfassten Kennwerte in den Folgeschritten nötig.

Namensänderung eines Fakts oder einer Dimensionsspalte

Ein in der ProGoalML erfasster Messwert, ein KPI oder ein Ziel besitzt einen eindeutigen Namen. Dieser wird beispielsweise aufgrund eines Schreibfehlers verändert.

Probleme und Hindernisse:

Die Änderung eines Bezeichners innerhalb des Systems ist einer der schwierigsten Fälle für die in dieser Arbeit beschriebene generische Lösung. Ein Grund hierfür ist die Generierung der einzelnen Informationen aus der bereits beschriebenen ProGoalML. Verändert sich hierin beispielsweise der Bezeichner eines Messpunktes, so ist nicht ersichtlich, ob dieser Vorgang tatsächlich nur den bezeichnenden Namen verändert oder ob ein alter Kennwert gelöscht und ein neuer hinzugefügt wurde. Sollte es sich lediglich um eine Umbenennung handeln, so könnte innerhalb der erzeugten Kerndokumente ebenfalls lediglich der Bezeichner umbenannt werden. Wurde jedoch ein vorhandener Messpunkt gelöscht und ein neuer hinzugefügt, so dürfen vorhandene Kennwerte nicht mit neuen Kennwerten in der gleichen Spalte kombiniert werden, da sie unterschiedliche Aussagen enthalten.

Lösungsansätze:

Lösungsansatz A: Da nicht eindeutig entscheidbar ist, ob eine Veränderung oder ein Lösch- und Hinzufügungsvorgang vorliegt, wird die Änderung eines Bezeichners grundsätzlich als eine Kombination aus Löschen und Hinzufügen betrachtet. Diese beiden Vorgänge wurden im Vorfeld bereits diskutiert und Lösungen hierfür erarbeitet.

Lösungsansatz B: Die Entscheidung wird durch Konfigurationsmöglichkeiten oder anderweitige Informationen gestützt, mit deren Hilfe eine eindeutige Entscheidung möglich ist. Besteht eine solche Möglichkeit, so kann ein entsprechendes Update-Script für das Data Warehouse erzeugt werden. In OLAP-Schema und Kettle-Script muss lediglich der veränderte Bezeichner angepasst werden.

Verändern einer Berechnungsgrundlage

Ein in der ProGoalML erfasster KPI oder ein Ziel besitzen eine Berechnungsgrundlage beschrieben durch eine Formel. Diese Formel wird nun verändert. Dies könnte beispielsweise bei einer Mehrwertsteuererhöhung der Fall sein.

Probleme und Hindernisse:

Eine Veränderung der Berechnungsgrundlage eines KPI oder Ziels führt oft zu einer veränderten Aussage der hierin erfassten Kennwerte. Da bereits vorhandene Informationen jedoch auf einer älteren Berechnungsgrundlage erstellt und gespeichert werden, kann dies zu Konsistenzproblemen führen.

Ein weiteres Problem stellt der mögliche Verlust von bereits berechneten und erfassten Kennwerten dar. Diese könnte am Beispiel der Mehrwertsteuer zu einer fehlerhaften Bilanzierung führen. Ein Datenverlust kann dann auftreten, wenn vorhandene Kennwerte aufgrund des gewählten Lösungsansatzes überschrieben oder gelöscht werden.

Lösungsansätze:

Lösungsansatz A: Ein erster Ansatz wäre die Erzeugung einer zusätzlichen Spalte, in welcher die neu erfassten Werte gespeichert werden. Die bereits existierende Spalte würde für zukünftige Datensätze mit Nullwerten aufgefüllt. Gleichzeitig können in der neu generierten Spalte auch alte Kennwerte auf Basis der veränderten Berechnungsgrundlage erzeugt werden. Hierdurch stehen für bereits erfasste Kennzahlen Ergebnisse auf Basis der alten und der neuen Formeln zur Verfügung. Diese Lösung entspricht jener, welche bereits beim Verändern eines Datentyps zum Einsatz kommt. Leider ergeben sich hierbei auch dieselben Probleme, wie die zusätzlich benötigten Informationen in der nachgelagerten Informationsverarbeitung.

Lösungsansatz B: Eine weitere Möglichkeit wäre das Nachberechnen bereits existierender Daten. Hierbei würden die bereits gesammelten Informationen nachträglich mittels der neuen Berechnungsgrundlage neu erzeugt und vorhandene Kennwerte überschrieben werden. Dies ist jedoch nur dann möglich, wenn die benötigten Informationen auch innerhalb der Datenbank zur Verfügung stehen. Außerdem stellt sich die Problematik, dass in diesem Fall die alte, bereits gesammelte Information und Aussage verloren geht.

Lösungsansatz C: Sollte das Nachberechnen von Kennwerten nicht möglich sein, so besteht die Möglichkeit einer Mischspalte. In einer solchen wird jener Kennwert gespeichert, welcher auf der Berechnung des Erfassungszeitpunktes basiert. Eine anschließende Bewertung dieser Daten ist jedoch sehr schwierig und nur mit entsprechenden Informationen möglich, da nicht eindeutig ist, mit welcher Formel einzelne Werte berechnet wurden.

Lösungsansatz D: Eine weitere Lösung stellt die Kombination der Lösungsansätze A und B dar. Hierfür werden die bereits erfassten Kennwerte vorgehalten sowie bereits gesammelte Informationen in einer zusätzlichen Spalte neu berechnet. Hierdurch wäre es beispielsweise auch möglich, auf Änderungen bei einer Mehrwertsteuerberechnung zu reagieren, ohne die bisherigen Informationen für die Bilanzierung zu verlieren. So stehen sowohl die alten Mehrwertsteuersätze zur Nutzung bereit, als auch alle neu berechneten Sätze. Die Berechnung der entsprechenden Kennwerte für die neue Spalte setzt jedoch voraus, dass alle zur Berechnung nötigen Daten im System abrufbar sind.

5.2.4. Kombination aus allen drei Kategorien

Während die einzelnen Kategorien für sich selbst gesehen lösbare Problemstellungen beinhalten, wird eine Kombination aller drei Kategorien schnell umfangreich und erfordert ein strukturiertes Abarbeiten der einzelnen Fälle. Um dies zu verdeutlichen ein kleines Beispiel: Wir betrachten eine Faktentabelle mit den drei Spalten A, B und C. Spalte A und B enthalten Kennwerte die mittels eines Messpunktes erfasst wurden. Spalte C stellt einen KPI dar, welcher auf einer einfachen Addition der Spalte A und B basiert.

Nun wird das System wie folgt verändert:

- Spalte A soll gelöscht werden.
- Eine neue Spalte D soll als weiterer Fakt hinzugefügt werden.
- Die Berechnungsgrundlage der Spalte C wird auf die Differenz der Spalten B und D abgeändert.

Anhand dieses Beispiels kann nun eine allgemein gültige Abarbeitungsreihenfolge der einzelnen Kategorien hergeleitet werden:

- Um die bereits besprochenen Veränderungsmöglichkeiten anwenden zu können, müssen zuvor alle hinzugefügten Spalten bekannt sein. Daher muss als erster Schritt Spalte D der Datenbank hinzugefügt werden. Hierbei muss entschieden werden, was für bereits vorhandene Reihen in diese Spalte geschrieben werden sollen.
- Im zweiten Schritt kann die Berechnungsgrundlage verändert werden, wobei eines der bereits beschriebenen Vorgehen angewendet wird.
- Da gelöschte Spalten in einer Berechnung nach Möglichkeit nicht mehr genutzt und referenziert werden sollten, muss der Löschvorgang noch vor dem Hinzufügen von neuen Spalten durchgeführt werden. Ansonsten könnte bei einer (Neu-)Berechnung eines KPI oder Ziels auf einen zu löschenden Kennwert zurückgegriffen werden. Dies führt zu falschen Ergebnissen.

Ein weiteres Problem stellen Berechnungen dar, welche neu hinzugefügt werden. Diese können sich auf ein bereits vorhandenes oder neu hinzugekommenes Ziel oder KPI beziehen. Um diese dennoch auch auf vorhandene Reihen der Datenbank anwenden zu können, muss gewährleistet werden, dass vor der Berechnung dieser Kennwerte alle Spalten hinzugefügt wurden. Daher wird die Nachberechnung dieser erst in der Veränderungsphase durchgeführt.

Da Berechnungen nicht ausschließlich auf Kennwerten aus Messpunkten beruhen müssen, sondern auch Ergebnisse anderer Berechnungen mit einbeziehen können, muss eine eindeutige Reihenfolge zur Abarbeitung festgelegt werden. Da alle referenzierten Felder zu diesem Zeitpunkt bekannt sind, ist dieses Problem mit einer topologischen Sortierung lösbar. Eine topologische Sortierung bezeichnet eine Reihenfolge von Sachverhalten, bei der vorgegebene Abhängigkeiten erfüllt sind. Im hier besprochenen Fall ist dies die Abhängigkeit

der einzelnen KPI und Ziele voneinander und die hierauf basierenden Berechnungen. Die eingesetzte ProGoalML Bibliothek bietet hierfür bereits entsprechende Methoden.

Unter diesen Annahmen und Feststellungen ergibt sich folgendes Vorgehen:

1. Durchführen aller Löschvorgänge.
2. Hinzufügen neuer Fakten und Dimensionen.
3. Durchführen aller weiteren Änderungen, ohne Neuberechnung von Kennwerten.
4. Sortieren der nötigen Neuberechnungen.
5. Durchführen der nun sortierten Berechnungen.
6. Erzeugen von ETL-Prozessen und OLAP-Schemata.

Mit steigender Anzahl von Änderungen sowie steigender Anzahl von Änderungen der dritten Kategorie nehmen auch die Komplexität und der Aufwand zur automatischen Anpassung des Data Warehouses zu. Dies führt zu komplexeren Statements für die Datenbank. OLAP-Schema und ETL- Prozess können dagegen wie bisher erstellt werden.

5.2.5. Überblick

Die nachfolgende Tabelle dient als kurze Übersicht zu den bereits besprochenen Problemfällen und ihrer Lösungsansätze:

Beschreibung	Probleme	Lösung
Hinzufügen von neuen Informationen		
Hinzufügen eines neuen Wertes	Nachberechnen von Werten	Hinzufügen in den Kerndokumenten
Löschen von vorhandenen Strukturen		
Bisher erfasster Wert wird gelöscht	Permanentes Löschen eines KPI, eines Ziels oder eines Messwertes	A: Vorhalten im System, nicht referenzieren im Olap-Würfel B: Erzeugen eines Backup, anschließendes Löschen
Verändern von existierenden Informationen		
Verändern eines Datentyps	A: Umwandlung ist möglich, jedoch mit Verlust der Genauigkeit B: Umwandlung in den neuen Datentyp ist nicht möglich	A: Vorhalten des alten Datensatzes, zusätzliches Erzeugen des neuen Datensatzes B: Konvertieren in Datentyp mit höherer Genauigkeit
Namensänderung eines Fakts oder einer Dimensionsspalte	Es ist unklar ob eine Umbenennung vorliegt oder die Kombination aus löschen und hinzufügen.	A: Grundsätzliche Behandlung als getrenntes Löschen und Hinzufügen B: Entscheidung gestützt durch zusätzliche Informationen
Verändern einer Berechnungsgrundlage	Inkonsistenz in den Berechnungen. Verlust bereits erfasster Werte	A: Vorhalten alter Werte innerhalb einer eigenen Spalte B: Vollständige Neuberechnung C: Erzeugung einer Mischspalte mit alten und neuen Werten D: Vorhalten alter Werte in eigener Spalte, vollständige Neuberechnung in neuer Spalte

Tabelle 5.1.: Übersicht über Problemfälle im Change-Management

6. Implementierung

In diesem Kapitel wird beschrieben, wie die in Kapitel 5 erarbeiteten Konzepte umgesetzt wurden. Als Programmiersprache wird JAVA eingesetzt. Zur Verarbeitung der ProGoalML kann auf die APro-Bibliothek zurückgegriffen werden. Diese stellt diverse Funktionen aus dem in Kapitel 3 beschriebenen Gesamtprozess bereit.

Analog zur Beschreibung des Konzepts wird die Implementierung in die Erstellung der Kerndokumente sowie das Change-Management unterteilt. Zusätzlich wird ein Überblick über die Konfigurationsmöglichkeiten gegeben, welche externe Informationen für die einzelnen Kerndokumente bereitstellt. Als letztes wird das Bereitstellen (Deployment) der Kerndokumente für die Nutzung durch den Pentaho BI-Server behandelt.

Eine Benutzungsanleitung befindet sich in Anhang A.1.

6.1. Überblick

Durch die geplante Trennung von Deployment und der Erstellung der Kerndokumente wurden zwei voneinander getrennte Komponenten erzeugt. Während das Projekt zur Erzeugung von Kerndokumenten, im Folgenden ADWCore genannt, als Abhängigkeit lediglich auf die APro-Bibliothek angewiesen ist, müssen für das Deployment zusätzliche externe Bibliotheken bemüht werden. Hierunter fallen neben dem ADWCore als zentrale Komponente Bibliotheken zur Kommunikation mit den eingesetzten Datenbanken MySQL [Ora12] sowie HSQLDB [The12]. Abbildung 6.1 zeigt einen groben Überblick über die Komponenten ADWCore und ADWDeployment.

ADWCore

Der ADWCore folgt dem Model-View-Controller (MVC) [GHJJ96] Prinzip. Jedoch wird bei der Implementierung auf View-Komponenten verzichtet, da ADWCore als Bibliothek innerhalb des Backends genutzt werden soll. Eine Oberfläche ist deshalb nicht nötig.

Als zentrale Komponente dient die sogenannte ADWFactory. Diese als Singleton implementierte Klasse bietet alle nötigen Funktionen, um aus einer ProGoalML die drei Kerndokumente zu erzeugen. Zusätzlich ist über sie auch der Zugriff und die Steuerung der aktuellen Konfiguration möglich.

Die Erstellung selbst wird in drei Stufen unterteilt:

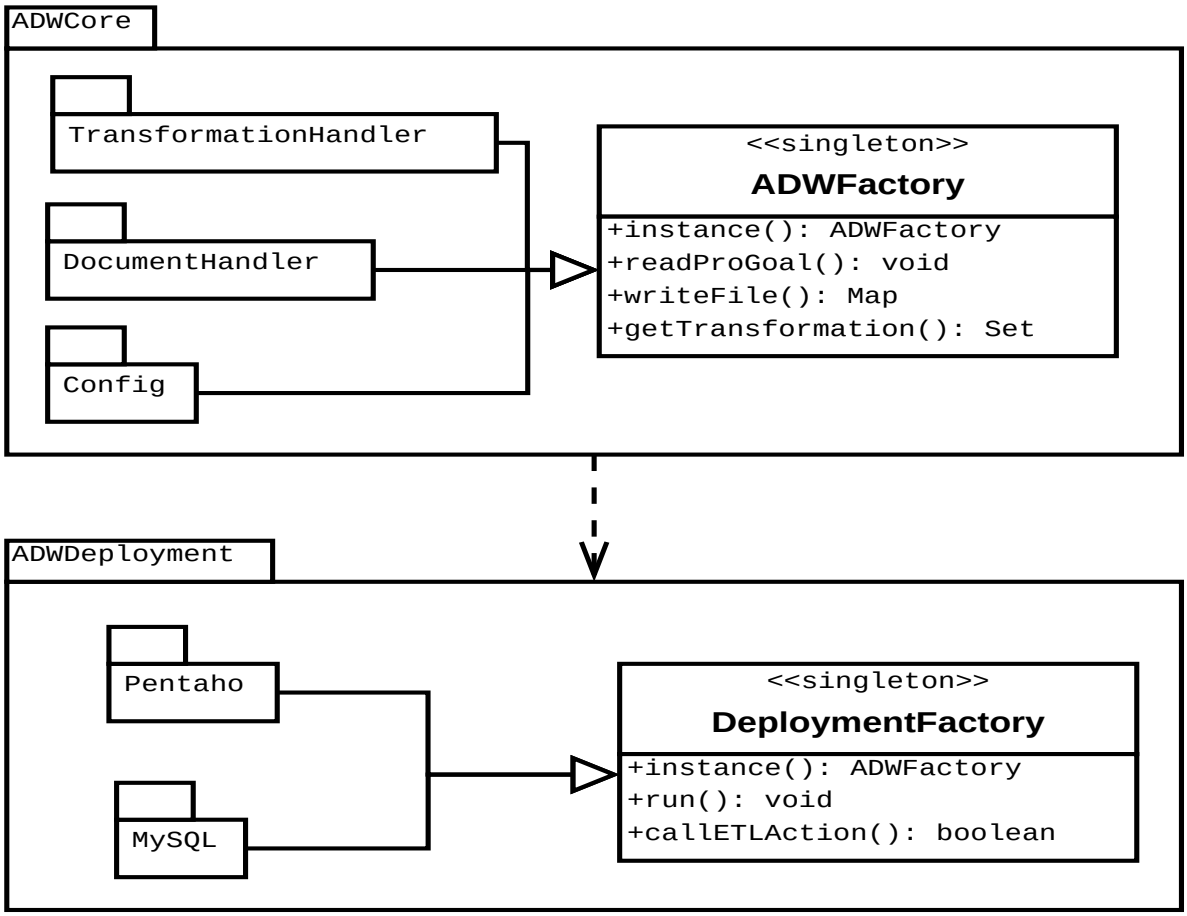

Abbildung 6.1.: Übersicht über die Komponenten ADWCore und ADWDeployment (eigene Darstellung).

1. Lesevorgänge: Einlesen der ProGoalML sowie Erzeugung der zugehörigen Ergebnis-Schemata.
2. Transformation dieser Daten in die für die Kerndokumente nötigen Formate.
3. Schreiben der Kerndokumente.

Um dies zu erreichen ist das ADWCore Projekt in zwei Pakete unterteilt: Transformation-Handler und den sogenannten Document-Handler. Im ersten Paket werden die nötigen Transformationsschritte implementiert, im zweiten Lese- und Schreiboperationen.

Das dritte große Paket innerhalb von ADWCore bilden die Konfigurationsmechanismen, welche sich innerhalb des Config-Pakets wiederfinden.

ADW-Deployment

Bei diesem Projekt handelt es sich um eine Sammlung von Klassen und Methoden, welche speziell auf das Ausliefern der erzeugten Kerndokumente im Kontext des Pentaho

BI-Servers angepasst wurden. Da die eingesetzten Kerndokumente für eine Vielzahl von BI-Lösungen sich ähneln, bzw. meist sogar gleichen, bietet die Entkopplung des eigentlichen Deployment-Prozesses einen Vorteil bei der Wiederverwertung und Pflege des ADWCore Paktes (Modularisierung).

Auch das ADW-Deployment bietet mit der Klasse DeploymentFactory eine zentrale Funktionssammlung, welche den Einsatz des gesamten Projektes ermöglicht.

6.2. Konfiguration

Bei der Konfiguration handelt es sich um ein XML-Dokument, welches für die Generierung der Transformation wichtige zusätzliche Informationen enthält. Diese Informationen sind nicht in der ProGoalML enthalten, für die Transformation und das Deployment jedoch nötig.

Eine Konfigurationsdatei besteht aus drei Teilen: Globalen Informationen im Abschnitt *Global*, Informationen bezogen auf die Kerndokumente im Bereich *Output* sowie *Options*-Elementen, welche bei der Transformation der Daten aus der ProGoalML berücksichtigt werden.

Der Bereich *Global* enthält Informationen zur Datenbankanbindung, dem Pentaho BI-Server sowie Einstellungen in Bezug auf das integrierte Logging.

Im Abschnitt *Output* werden Informationen für die Erzeugung der einzelnen Kerndokumente hinterlegt. Hierzu zählen Zielpfade, Dateinamen, Dateiendungen sowie Kerndokument bezogene weitere Einstellungen.

Der dritte und letzte Bereich *Options* bietet zusätzliche Informationen, welche innerhalb der Transformation berücksichtigt werden. Hierzu zählen beispielsweise Informationen, die wie in Kapitel 5.1.5 beschrieben, Buckets oder erweiterte Dimensionen ermöglichen. So können auf Basis von Datentypen neue Dimensionen erzeugt werden. Die in Frage kommenden Kennwerte können zusätzlich mittels eines *identifiers* eingeschränkt werden. Die hierdurch gewonnene Dimension kann mit Hilfe des *column*-Tag um Level erweitert werden. Werte sowie der Datentyp eines Levels werden mittels eines *transform*-Objekts erzeugt. Hierfür stehen diverse Transformations-Typen zur Verfügung, mit deren Hilfe Zeitdimensionen und Bucket-Dimensionen erzeugt werden können.

Zusätzlich zu Dimensionen lassen sich auf ähnliche Art und Weise Measures für Fakten definieren. Neben allgemein gültigen Measures, welche für jeden Fakt angelegt werden, besteht auch die Möglichkeit Measures für bestimmte Datentypen anzulegen.

Im Anhang A.2 befindet sich ein Beispiel für eine vollständige Konfigurationsdatei. Eine detaillierte Aufschlüsselung der einzelnen Elemente einer Konfigurationsdatei befindet sich in der Anleitung (Anhang A.1).

Das ADWCore Projekt besitzt eine vordefinierte Konfigurationsdatei, welche Standardinformationen enthält. Diese kann in den meisten Fällen ohne Veränderung genutzt werden.

Sollte das Einsatzgebiet eine Änderung dieser Standardwerte erfordern, so kann dies jederzeit durch eine externe Konfigurationsdatei erfolgen. Diese muss jedoch Schema-Konform erstellt werden. Alternativ ist die programmatische Generierung bzw. Modifizierung der Konfigurationsinformationen möglich.

6.3. Erstellen der Kerndokumente – ADWCore

Der ADWCore bildet das zentrale Projekt zur Erzeugung von Kerndokumenten. Durch Eingabe einer ProGoalML und unter Berücksichtigung der zusätzlichen Informationen aus der bereitgestellten Konfigurationsdatei werden für sämtliche Überdeckungsklassen die zugehörigen Kerndokumente erzeugt.

Hierzu wird die ProGoalML eingelesen und direkt in ihre einzelnen Überdeckungsklassen zerlegt. Das Ergebnis ist mindestens ein Ergebnis-Schema. Mithilfe dieser Ergebnis-Schemata wird der Transformationsprozess für das Datenbankschema angestoßen. Eine eingelesene ProGoalML sowie die daraus erzeugten Ergebnis- und Datenbankschemata werden innerhalb der ADWFactory für weitere Verwendung vorgehalten. Hierdurch wird vermieden, dass einzelne Dokumente mehrfach verarbeitet werden müssen.

Um zukünftige Erweiterungen des ADWCores zu ermöglichen werden Lese-, Schreib- und Transformationsschritte jeweils mittels einer eigenen Schnittstelle bereitgestellt. Dies gewährleistet eine einfache Erweiterung, z.B. um weitere Kerndokumenttypen.

6.3.1. Lese- und Schreibprozesse

Im Transformation-Handler-Paket befinden sich sämtliche Funktionen, welche zur Generierung oder Verarbeitung von externen Dokumenten benötigt werden. Dieser ist wiederum in zwei Teilbereiche unterteilt:

Mithilfe der sich im Reader-Paket befindlichen Klassen und Methoden können externe Dokumente gelesen und somit für Transformationen zur Verfügung gestellt werden. Jede Klasse, die das Einlesen einer Datei ermöglicht, muss hierfür das Interface *IDocumentReader* implementieren. Die im Zuge dieser Arbeit erstellten Reader ermöglichen das Einlesen der ProGoalML sowie das Einlesen der Ergebnis-Schemata. Hierfür werden Funktionen aus der APro-Bibliothek genutzt.

Das Erzeugen von externen Dokumenten ist hingegen mit den Klassen und Methoden im Generator-Paket möglich. Die hierin implementierten Klassen stellen das *IDocumentGenerator* Interface bereit. Dieses akzeptiert als Eingabe das Ergebnis einer Transformation und erzeugt hieraus das entsprechende Kerndokument. Hierfür werden auf die in der Konfigurationsdatei im Abschnitt Output definierte Informationen zurückgegriffen.

Es existieren Generatoren für alle drei benötigten Kerndokumentarten.

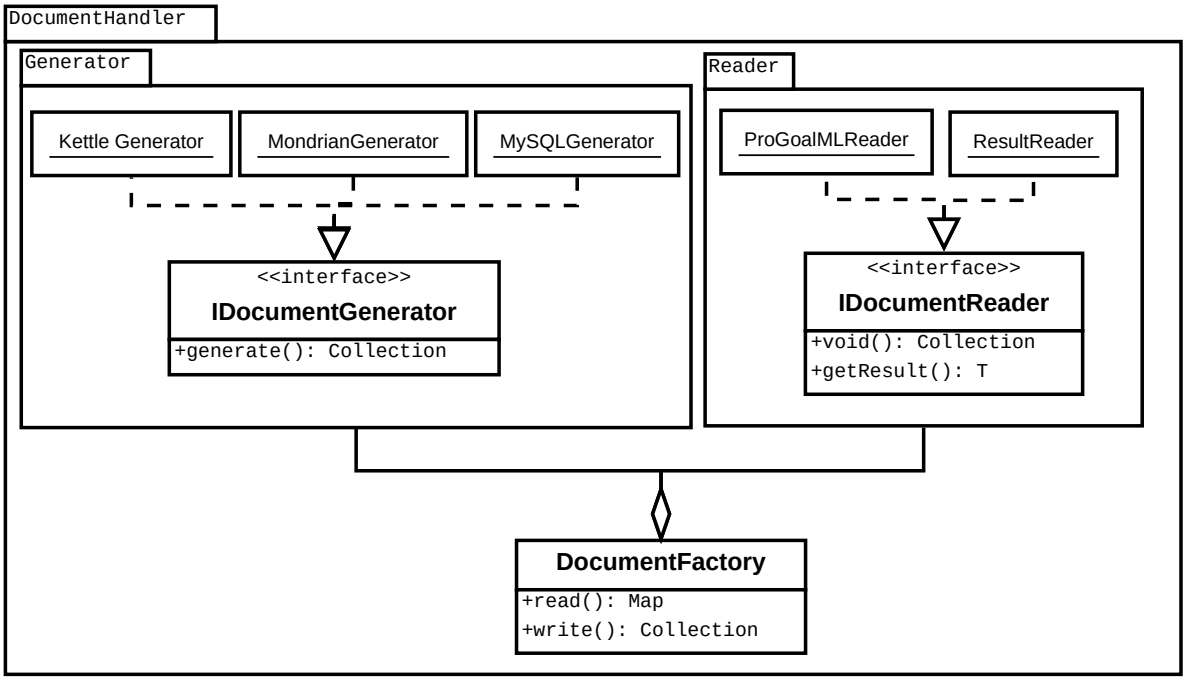

Abbildung 6.2.: Übersicht über den DocumentHandler (eigene Darstellung).

Sowohl Lese- als auch Schreiboperationen werden über die Klasse DocumentFactory bereitgestellt und aufgerufen.

6.3.2. Transformation

Die Transformation der in der ProGoalML vorhandenen Informationen in die einzelnen Kerndokumente ist die zentrale Aufgabe der ADWCore Komponente. Dies ist mit Hilfe der Klassen und Methoden im Paket TransforamtionHandler möglich.

Transformationen werden einzeln ausgeführt und müssen das ITransformation Interface implementieren. Analog zum Vorgehen innerhalb des DocumentHandlers, wird für jedes der benötigten Kerndokumente eine eigene Transformation erzeugt. Die für das Datenbankschema nötigen Transformationen werden auf Basis der übergebenen ProGoalML erstellt. Alle weiteren Transformatoren greifen auf die hierbei erzeugten Informationen als Quelle zurück.

Auf die zur Verfügung stehenden Transformationen kann mittels der TransformationFactory zugegriffen werden.

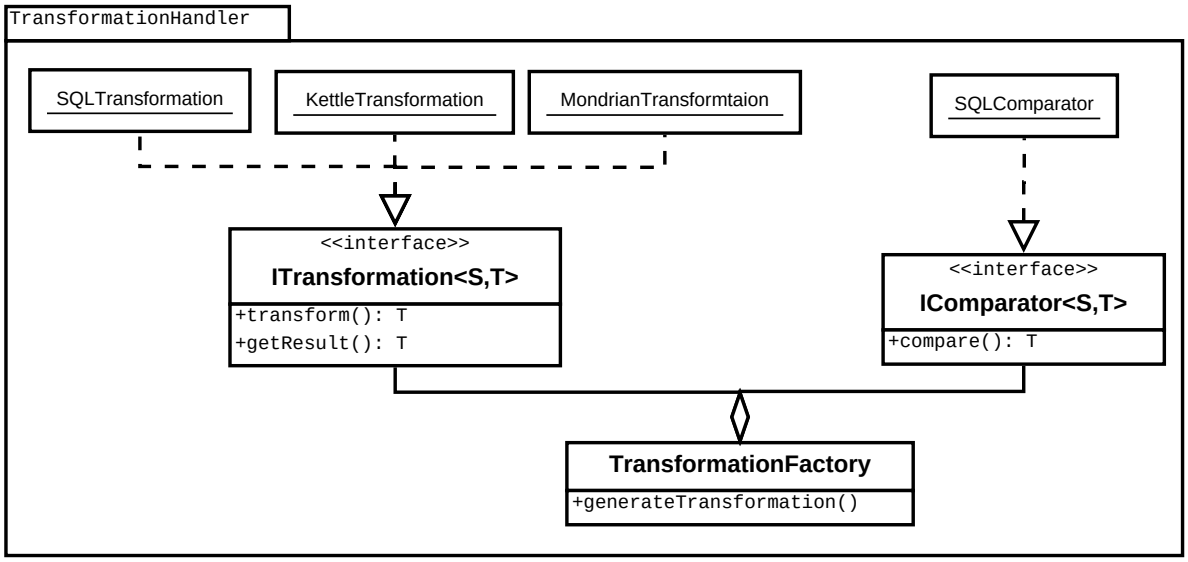

Abbildung 6.3.: Übersicht über den TransformationHandler (eigene Darstellung).

SQLTransformator

Mit Hilfe des SQLTransformators lässt sich aus einem Ergebnis-Schema ein Datenbankschema erzeugen. Hierfür werden, analog zum in Kapitel 5.1.3 beschriebenen Vorgehen, die einzelnen Datenwerte anhand ihres Datentyps in die Gruppen Fakt oder Dimension unterteilt. Sämtliche numerischen Datentypen werden hierbei als Fakten behandelt, Boolsche Werte und auf Buchstaben basierende Werte als Dimension.

Das bedeutet, dass in der ProGoalML erfasste Parameter und KPIs je nach angegebenem Datentyp entsprechend dieser Gruppierung eingeteilt werden. Ziele, welche nur Boolsche Werte annehmen können, werden stets als Dimension angelegt.

Da diese statische Einteilung nicht ausreicht um alle gewünschten Funktionalitäten zu modellieren, können in der Konfigurationsdatei zusätzliche Informationen für diesen Schritt hinterlegt werden. So ist es beispielsweise möglich, auf dem Datentyp Long einen Timestamp zu definieren (siehe Anhang A.2). Hiermit assoziierte Daten werden als Dimension behandelt und die in der Konfiguration definierten zusätzlichen Dimensionsspalten werden erzeugt. Alternativ können Elemente aus der ProGoalML, beispielsweise eine KPI, über ihren Namen referenziert werden und somit als Dimension definiert werden.

Als Ergebnis dieser Transformation erhält man das dem zukünftigen Datenbankschema zugrundeliegende Sternschema. Dieses dient als Grundlage für weitere Transformationen.

KettleTransformation

Aufbauend auf den Ergebnissen des SQLTransformators erzeugt die Klasse KettleTransformation die für den ETL benötigten Schritte. Dank der vorgelagerten CEP Engine, kann

der ETL-Prozess vereinheitlicht werden. Die Abfolge der Einzelschritte innerhalb des ETL-Prozesses entspricht hierbei den in Kapitel 5.1.6 beschriebenen Teilschritten.

Es handelt sich bei einem Kettle-Schema um ein sehr umfangreiches XML-Schema, für welches keine offiziellen XML-Schema-Definition existieren. Da für das gewünschte Ergebnis nur ein Bruchteil der von Kettle bereitgestellten Funktionalität benötigt wird, ist die Anzahl der für die KettleTransformation verfügbaren Teilschritte stark eingeschränkt. Eine Erweiterung um weitere Teilschritt-Typen ist jederzeit möglich.

Momentan stehen folgende Teilschritte als Implementierung zur Verfügung:

- Auslesen von XML-Dateien.
- Umwandeln von Timestamps in den Datentyp *Date*. Dies wird benötigt um Zeitdimensionen anzulegen.
- Umwandeln von Timestamps in die einzelnen Elemente einer Zeitdimension.
- Generierung von Buckets und Durchführung der benötigten Berechnungen.
- Combination-Lookup Schritte, zur Verarbeitung von Dimensionen in der Datenbank.
- Schreiben der Fakten in die Datenbank.

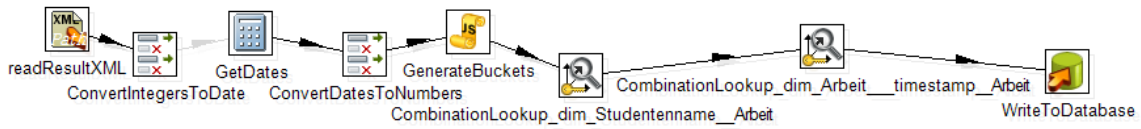

Abbildung 6.4.: Beispiel für einen erzeugten ETL-Prozess, dargestellt in Spoon (eigene Darstellung)

Das hierdurch erzeugte Script kann unter Einsatz des Pentaho-Tools Spoon [pen12] jederzeit erweitert und verfeinert werden (Abbildung 6.4).

MondrianTransformation

Auch die MondrianTransfomation basiert auf den Ergebnissen des SQLTransformators. Die durch das Sternschema definierten Fakten und Dimensionen werden, analog zum in Kapitel 5.1.4 vorgestellten Vorgehen, in Dimensionen und Measures umgesetzt.

Dank der bereits vorliegenden Trennung von Fakten und Dimensionen können Dimensionen direkt erzeugt werden. Hierbei wird pro Dimension lediglich eine einzelne Hierarchie-Ebene erzeugt. Die Anzahl der Level sowie deren Reihenfolge, entspricht der Definition aus der Konfigurationsdatei. Liegt keine Definition in der Konfigurationsdatei vor, so wird für jedes Element der Dimensionstabelle ein Level erzeugt.

Für jeden existierenden Fakt werden Measures angelegt. Anzahl und Aggregation der einzelnen Measures werden mit Hilfe der Konfiguration aufgelöst. Es existieren zwei Möglichkeiten, Measures in der Konfigurationsdatei zu definieren: Allgemeine Measures, im Tag *allMeasures* werden für alle Fakten erzeugt. Hierin finden sich meist Measures mit den Aggregatoren *Summe* und *Count*, da diese in nahezu jedem Kontext benötigt werden. Die zweite Möglichkeit stellen spezialisierte Measures dar. Diese werden unter Berücksichtigung des Datentyps und weiteren optionalen Parametern erzeugt. Ein Beispiel für die Definition von Measures in einer Konfigurationsdatei befindet sich im Anhang A.2.

6.3.3. Change Management

Da auch bei Veränderungen der ProGoalML bereits existierende Daten weiter genutzt werden sollen, müssen die zugehörigen Kerndokumente ebenfalls angepasst werden. Dies ist im Falle des OLAP-Schemas sowie des ETL-Schemas ohne zusätzlichen Aufwand lösbar: Aus der veränderten ProGoalML wird wie bisher eine SQL-Transformation erzeugt. Mit Hilfe dieser kann eine neue Version des gewünschten Schemas erzeugt werden, welches das alte Schema ersetzt. Dies ist möglich, da beide Kerndokumente selbst keine Werte speichern bzw. bereitstellen.

Die Datenbank hingegen muss den neuen Gegebenheiten angepasst werden. Mögliche Probleme sowie passende Lösungsansätze wurden bereits in Kapitel 5.2 diskutiert.

Um diese Lösungsansätze anwenden zu können müssen zwei Voraussetzungen erfüllt sein:

1. Es müssen die alte sowie die neue Version der ProGoalML vorliegen und eingelesen werden.
2. Aus beiden Versionen müssen SQL-Transformationen erzeugt werden.

Die erzeugten SQLTransformationen werden mittels der SQLComparator-Klasse verglichen. Hierbei werden Informationen zu fehlenden und hinzugefügten Tabellen und Spalten gesammelt. Zusätzlich werden veränderte Datentypen und Veränderungen in Formeln erfasst. Diese Daten werden unter Berücksichtigung der bereits besprochenen Problemen und folgenden Lösungsansätzen verarbeitet:

- Messwerte, KPIs und Ziele, welche in der ProGoalML gelöscht werden, werden auch in der Datenbank gelöscht. Es wird davon ausgegangen, dass vor der Ausführung des Scripts ein Backup der Datenbank erzeugt wird.
- Wird der Datentyp eines erfassten Wertes verändert, so wird versucht bereits vorhandene Werte in den neuen Datentyp zu konvertieren. Ein Genauigkeitsverlust der Werte kann hierbei auftreten. Ist dies nicht möglich, so werden die alten Werte gelöscht und durch Null-Werte ersetzt.
- Wird ein Messwert, eine KPI oder ein Ziel umbenannt, so werden vorhandene Werte gelöscht und entsprechende neue Strukturen angelegt. Bereits vorhandene Werte gehen hierbei verloren.

- Wird die Berechnungsgrundlage eines Ziels oder einer KPI verändert, so wird keine Veränderung vorgenommen. Sollte sich jedoch gleichzeitig der Datentyp verändern, so werden bereits vorhandene Werte in einer zusätzlichen Spalte gesichert. Die Werte werden, sofern möglich, auf Basis der neuen Formel erneut berechnet und in der neuen Struktur zur Verfügung gestellt. Hierdurch wird verhindert, dass bereits erfasste Werte verloren gehen.

Als Ergebnis erhält man ein Datenbank-Script, welches die vorhandene Datenbank an die neuen Gegebenheiten anpasst.

6.4. Deployment – ADWDeployment

Das ADWDeployment Projekt ermöglicht die Bereitstellung der mittels ADWCore erzeugten Kerndokumente im Kontext des Pentaho BI-Servers. Hierfür wird folgendes Vorgehen genutzt:

1. Sichern der Informationen in der Datenbank, sofern dies in der Konfigurationsdatei definiert wurde.
2. Erstellen der nötigen Datenbankstrukturen bzw. verändern dieser.
3. Bereitstellen des Mondrian-Schemas.
4. Bereitstellen des ETL-Prozesses in Form der Kettle-Schemata.

Als zentraler Ausgangspunkt hierfür dient die DeploymentFactory, welche sämtlichen Funktionen gebündelt zur Verfügung stellt. Da im vorliegenden Anwendungsfall eine MySQL Datenbank genutzt werden soll, welche nicht in den Pentaho BI-Server integriert ist, wird dies auch auf Entwicklungsebene berücksichtigt:

MySQL Im Paket MySQL werden Klassen und Methoden zur Verfügung gestellt, welche zur Datenbankverarbeitung genutzt werden können. Neben einer Backup-Funktion, können auch die durch ADWCore erzeugten Datenbankscripte verarbeitet werden. Für die Kommunikation mit der Datenbank wird der offizielle MySQL Connector/J JDBC Treiber genutzt. Das Backup wird mittels mysqldump erzeugt [Ora12].

Pentaho Innerhalb des Pentaho Pakets werden Methoden und Klassen bereitgestellt, mit deren Hilfe Mondrian- und Kettle-Schemata in den BI-Server integriert werden können. Im Zuge dieses Deployment-Prozesses werden zusätzlich sogenannte xActions erzeugt. Mit diesen Dateien kann der ETL-Prozess über einen Web-Service ausgeführt werden. Da einige Informationen in der Kettle internen HSQLDB [The12] gespeichert werden, benötigt dieses Paket die entsprechenden JDBC Treiber.

In der Kombination mit der Komponente ADWCore ist man somit in der Lage, aus einer ProGoalML einen gegebenen Pentaho BI-Server automatisch zu konfigurieren und über die erzeugten ETL-Prozesse mit Daten zu befüllen. Einige zusätzliche Informationen müssen hierfür mit Hilfe einer Konfigurationsdatei bereitgestellt werden, wodurch lediglich ein

semiautomatischer Ansatz umgesetzt werden kann. Durch die Nutzung der in ADWCore integrierten Standardkonfiguration kann die Komponente jedoch automtisiert genutzt werden.

7. Zusammenfassung und Ausblick

7.1. Zusammenfassung

Ziel dieser Arbeit war die (semi-)automatische Erzeugung eines Data Warehouse im Kontext des APro-Projektes. Ausgehend von einer in einem Vorprozess generierten ProGoalML wurden Methoden beschrieben, mit deren Hilfe drei Kerndokumente (Datenbankschema, OLAP-Schema, ETL-Prozesse) automatisiert erzeugt werden können. Die Bereitstellung dieser Kerndokumente durch einen Pentaho BI-Server ermöglichte schließlich das gewünschte Ergebnis zu erzielen.

In Kapitel 2 wurden zunächst Basis-Technologien dargestellt, welche für die Umsetzung und das Verständnis von Data Warehouses benötigt werden. Unter Anderem wurde ein Überblick über ETL, eine Einführung in OLAP sowie eine Einführung in das Sternschema gegeben.

Das zu erzeugende agile Data Warehouse sollte in einen Gesamtprozess eingebunden werden, dessen Ziel es ist die Bereitstellung einer Business Intelligence Lösung als SAAS-Lösung zu ermöglichen. Dieser Ansatz sorgt für diverse Erleichterungen und Einschränkungen, wie die Verkürzung des nötigen ETL-Prozesses und wurde in Kapitel 3 vorgestellt.

Die Erstellung eines Data Warehouses ist ein aufwändiger und zeitraubender Prozess. Daher gibt es diverse Versuche und Ansätze diesen Prozess zu beschleunigen bzw. zu automatisieren. Die zwei grundlegenden Herangehensweisen sowie verwandte Arbeiten wurden in Kapitel 4 diskutiert. Ausgehend von den bis zu diesem Zeitpunkt bereitgestellten Informationen wurden erste Entwurfsentscheidungen, betreffend eines automatischen Ansatzes sowie in Bezug auf die Grobstruktur, getroffen.

Auf Basis dieser Vorarbeiten wurde in Kapitel 5 ein allgemeines Konzept erarbeitet, das die Erstellung von Kerndokumenten (5.1) sowie Anforderungen des Change-Managements (5.2) umfasst.

Die Erstellung von Kerndokumenten beinhaltet Herausforderungen und Lösungsansätze, die berücksichtigt werden müssen. Es wird die mögliche Einteilung eingehender Werte in Fakten und Dimensionen innerhalb der Datenbank ebenso dargestellt, wie das Übersetzen dieser Informationen in Measures und Dimensionen innerhalb des OLAP-Schemas. Besondere Aufmerksamkeit wird hierbei der Möglichkeit zur Erzeugung von datentypunabhängigen Unterteilungsstrukturen gewidmet. In diesem Kontext werden deshalb die Beschreibung von Buckets sowie deren Einsatzmöglichkeiten diskutiert.

Während die erzeugten Kerndokumente eine Initialisierung eines Data Warehouses ermöglichen, liegt die eigentliche Herausforderung in der agilen Gestaltung der Wartungsarbeiten.

Mit Hilfe unterschiedlicher Änderungsszenarios versucht man diesen Herausforderungen in Kapitel 5.2 gerecht zu werden. Neben der Erweiterung um zusätzliche Werte sowie Entfernen bisher gesammelter Informationen werden auch Veränderungen des vorhandenen Systems diskutiert und Lösungen für diese einzelnen Problemfelder vorgeschlagen.

Für die Umsetzung wurden zwei aufeinander aufbauende Java-Projekte erzeugt. Das ADWCore-Projekt ermöglicht die Erzeugung der benötigten Kerndokumente sowie des Change-Managements. Hierfür werden die in Kapitel 5 erarbeiteten Konzepte und Problemlösungen genutzt. Unterstützt wird dieser Prozess von einer Konfigurationsdatei.

In den SAAS-Kontext integriert werden diese Dokumente mit dem Projekt ADWDeployment. Dieses stellt die erzeugten Kerndokumente auf Basis eines Pentaho BI-Servers und einer MySQL-Datenbank in Form eines konfigurierten, betriebsbereite Data Warehouse zur Verfügung.

Die Implementierung dieser beiden Komponenten wird im Kapitel 6 beschrieben.

Abschließend wird in diesem Kapitel Einsatz- und Erweiterungsmöglichkeiten des erarbeiteten agilen Data Warehouses gegeben.

7.2. Folgeprozesse

Das vorliegende Konzept zur Erstellung eines agilen Data Warehouses wurde im Kontext einer BI-Lösung entwickelt. Innerhalb einer solchen Lösung stellt ein Data Warehouse nur den Ausgangspunkt für weitere Arbeitsschritte zur Verfügung.

Das Ziel des APro-Projektes besteht in der Optimierung von Geschäftsprozessen. Die durch das Data Warehouse bereitgestellten Informationen werden in nachfolgenden Schritten ausgewertet. Die daraus folgenden Optimierungen des Geschäftsprozesses führen zu einer Anpassung des der ProGoalML zugrundeliegenden Prozesses. Hiermit beginnt der Optimierungszyklus erneut.

Alternativ zur statistischen Auswertung können auch Echtzeitanalyse-Tools an das Data Warehouse angebunden werden. Ein Beispiel hierfür wäre ein durch VisML beschriebenes Dashboard, welches ein Monitoring der anfallenden Daten erlaubt. Hierdurch kann zeitnah und effektiv auf aufgetretene Probleme innerhalb eines Prozesses reagiert werden.

7.3. Optimierung

Die in dieser Arbeit beschriebene Lösung ermöglicht die Erzeugung eines agilen Data Warehouse. Wie man jedoch im Kapitel 5 bereits erkennen kann, werden hierfür diverse Voraussetzungen und Entscheidungen getroffen. Daher sind diverse Optimierungsmöglichkeiten denkbar. Im Folgenden sollen daher einige dieser Möglichkeiten dargestellt werden:

Auflösung der Konfigurationsdatei: Die Konfigurationsdatei dient als externe Informationsquelle. Die hierin zur Verfügung gestellten Daten werden an diversen Stellen benötigt, um Sonderfälle oder kontextabhängige Anpassungen durchzuführen. Eine mögliche Lösung zur Abschaffung dieser Konfigurationsdatei bestände in der Integration dieser Informationen in die ProGoalML. Hierdurch könnten die nötigen Anpassungen schon zum Designzeitpunkt durchgeführt werden.

Alternativ wäre eine Kontextanalyse denkbar. Wie bereits in Kapitel 5.1.3 beschrieben, enthält die ProGoalML auch das BPMN-Model. Auf Basis dieser Daten könnte es möglich sein einen Großteil der in der Konfigurationsdatei enthaltenen Werte zu erhalten.

Erweiterung des Bucket-Systems: Die zur Verfügung stehenden Möglichkeiten zur Bucket-Generierung sind in der aktuellen Version auf wenige Optionen eingeschränkt. Für den produktiven Einsatz als agiles Data Warehouse empfiehlt es sich daher, diese Funktionalität um weitere Operatoren und Bedingungen zu erweitern.

Alternatives Deployment: Mit dem Pentaho BI-Server und der MySQL-Datenbankanbindung werden populäre Lösungen unterstützt. Dennoch wird es auf lange Sicht nötig werden auch alternative BI-Server und Datenbanken zu unterstützen. Hierfür können zusätzliche Deployment-Pakete erzeugt oder das vorhandene Deployment-Paket erweitert werden. Hierbei ist die Trennung in ADWCore und ADWDeployment von Vorteil.

Darüber hinaus ist eine Verfeinerung des Change-Managements, analog zu den Vorschlägen in Kapitel 5.2, denkbar.

7.4. Fazit

Es ist gelungen, eine geeignete Lösung für die Erstellung und Wartung eines agilen Data Warehouse im gegebenen Kontext zu erzeugen. Ausgehend von einer ProGoalML kann mit der bereitgestellten Implementierung innerhalb des APro-Prozesses ein Data Warehouse konfiguriert und gewartet werden. Dabei werden drei unterschiedliche Kerndokumente erzeugt, die eingesetzte MySQL-Datenbank automatisch gesichert und der verwendete Pentaho BI-Server konfiguriert. Auf Veränderungen der ProGoalML wird mit einer Anpassung der Data Warehouse Strukturen reagiert, sodass keine menschliche Interaktion nötig ist. Zusätzliche Informationen können mit Hilfe einer Konfigurationsdatei eingebracht werden. Im Gegensatz zu den untersuchten vorhandenen Ansätzen bietet das erarbeitete Vorgehen eine umfassende Generierung aller benötigten Informationen.

A. Anhang

A.1. Benutzungsanleitung

Im Folgenden wird eine allgemeine Anleitung zur Nutzung der ADW Komponenten gegeben. Hierzu wird zunächst die Installation der benötigten Software behandelt. In Kapitel A.1.2 folgt eine Einleitung in die Nutzung der ADW Komponenten.

Eine Übersicht über die Konfigurationsdatei findet sich in Kapitel A.1.3, welches durch die Beschreibung der Fehlercodes in Kapitel A.1.4 ergänzt wird.

Als Abschluss wird in Kapitel A.1.5 noch ein Überblick über bekannte Probleme, so wie passende Lösungen gegeben.

A.1.1. Installation

In der folgenden Installationsanleitung wird davon ausgegangen, dass es sich beim eingesetzten Betriebssystem um ein Windows XP oder neuer handelt.

Folgende Software muss installiert werden:

- Aktuelles Java Development Kit
- MySQL Server
- Pentaho BI-Server

Java Development Kit

Die aktuelle Version des Java Development Kits findet sich unter http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Starten Sie den Installer und folgen Sie den Anweisungen auf dem Bildschirm. Sie müssen an den Voreinstellungen nichts ändern.

Ist das JDK installiert, muss es noch in die Umgebungsvariablen eingetragen werden. Unter Windows 7 finden sich die Umgebungsvariablen unter Systemsteuerung->System und Sicherheit->System->Erweiterte Systemeinstellungen-> Umgebungsvariablen.

Nun muss eine neue Systemvariable namens JAVA_HOME angelegt werden. Der benötigt Pfad muss zum Installationsverzeichnis des JDK führen. Zusätzlich sollte der Pfad zum /bin Ordner des JDK in die PATH Variable eingetragen werden.

Sollten sie bereits eine ältere JDK Version installiert haben, so können sie diese benutzen. Diese sollte jedoch nicht älter sein als JAVA in der Version 5.

MySQL

ADW funktioniert ab der MySQL Version 5 und neuer. Eine aktuelle Version von MySQL erhalten sie unter http://www.mysql.de/downloads/installer/.

Für eine Testumgebung kann die Installation von MySQL mit den Standardeinstellungen durchgeführt werden. Denken sie daran sich das im Laufe dieser Installation angelegte Root-Passwort zu merken oder aufzuschreiben.

Nach einer erfolgreichen Installation muss der Pfad zum /bin Ordner der MySQL-Installation noch in die Path Variable der Umgebungsvariablen aufgenommen werden. Unter Windows 7 finden sich die Umgebungsvariablen unter Systemsteuerung->System und Sicherheit->System->Erweiterte Systemeinstellungen-> Umgebungsvariablen. Sollten sie dies nicht wünschen, so kann ADW dennoch genutzt werden. Jedoch kann der in ADW integrierte Backup-Mechanismus in diesem Fall nicht genutzt werden.

Als letzter Schritt der MySQL-Installation muss nun noch ein neuer Benutzer angelegt werden. Starten sie hierzu ihre MySQL Command Line und geben sie ihr Root-Passwort ein. Führen sie als nächstes folgenden Befehl aus:

```
GRANT ALL PRIVILEGES ON *.* TO 'USERNAME'@'localhost'
IDENTIFIED BY 'PASSWORT' WITH GRANT OPTION;
```

Ersetzen sie hierbei *USERNAME* durch den gewünschten Benutzernamen und *PASSWORT* durch das gewünschte Passwort. Sollte es sich bei ihrer Installation nicht um ein Testsystem handeln, passen sie unbedingt die Rechte dieses Benutzer ihren Bedürfnissen an. Folgende Liste stellt die minimal nötigen Rechte dar:

- Erstellen einer Datenbank.
- Verwalten von Tabellen in dieser Datenbank (Erstellen, Löschen, Modifizieren).
- Aufrufen von mysqldump.

Pentaho

ADW wurde mit der Version 3.9.0 des Pentaho BI-Servers entwickelt. Die zugehörigen Installationsdateien finden sich unter
http://sourceforge.net/projects/pentaho/files/Business%20Intelligence%20Server/3.9.0-stable/ .

Sie erhalten durch den Download ein Archiv, welches neben dem BI-Server auch den benötigten Administrationsbereich enthält. Entpacken sie dieses Archiv an einen beliebigen Ort. Eine weitere Installation ist nicht nötig.

Der BI-Server enthält bereits einen vorkonfigurierten Tomcat-Server, welcher als Web-Server verwendet wird. Starten sie nun zunächst den BI-Server. Dies ist mit der Datei %Installationspfad%\ biserver-ce\ start-pentaho.bat möglich.

Wurde der Server korrekt gestartet, sollten sie unter der Adresse http://localhost:8080 die in Abbildung A.1 dargestellte Login-Seite in ihrem Browser angezeigt bekommen.

Abbildung A.1.: Anmeldemaske des Pentaho BI-Servers

Starten sie nun zusätzlich den Pentaho Adminstrationsbereich. Hierfür kann die Datei %Installationspfad%\ administration-console\ start-pac.bat genutzt werden. Im Anschluss können sie sich auf der Seite http://localhost:8099 mit dem Benutzernamen *admin* und dem Passwort *password* anmelden. Legen sie nun unter Administration->Users entweder einen neuen Benutzer an, oder setzen sie für einen der existierenden Benutzer ein Passwort (Abbildung A.2). Mit diesen Daten können sie sich nun unter http://localhost:8080 anmelden.

A.1.2. Nutzung der ADWDeployment

ADW besteht aus zwei Komponenten: ADWCore und ADWDeployment. ADWCore ist für die Generierung von Datenbankschema, OLAP-Schema und ETL-Prozesse zuständig.

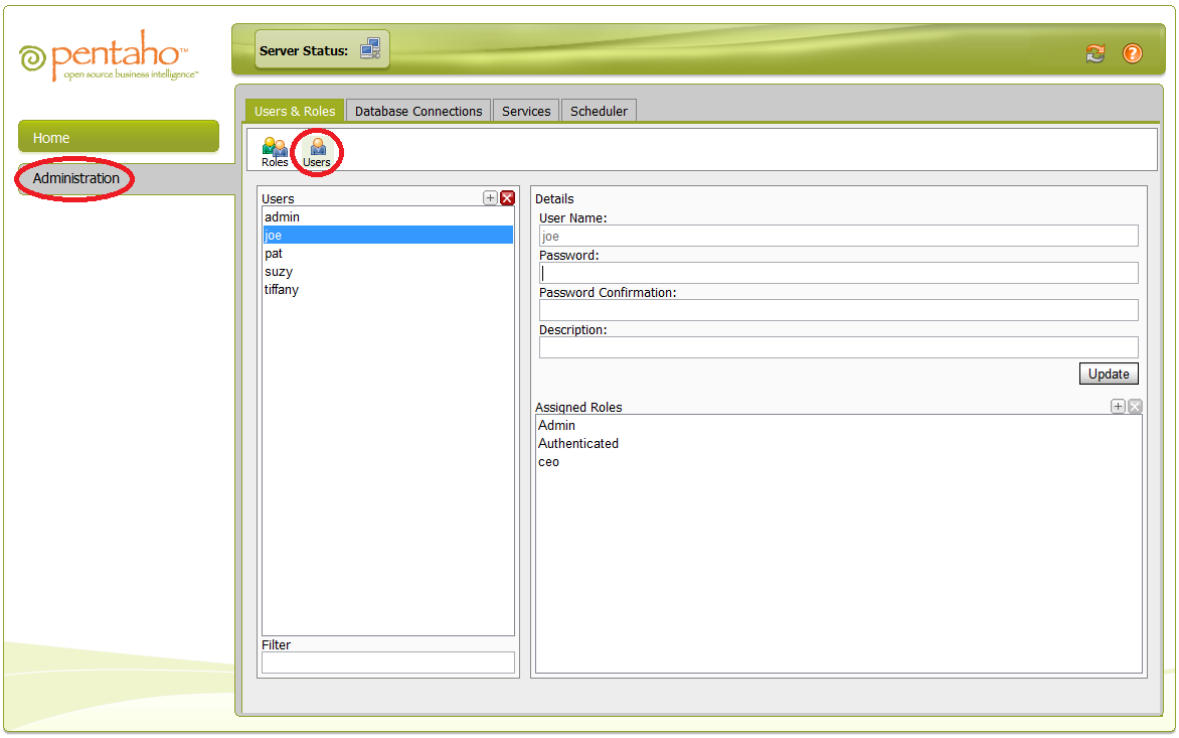

Abbildung A.2.: Einrichten eines Benutzers mit Hilfe der Pentaho Administrationsoberfläche

ADWDeployment ermöglicht die Bereitstellung dieser generierten Dokumente in der Pentaho BI-Server Umgebung.

Den gesamten Generierungs- und Deploymentprozess können sie mit Hilfe der Komponente ADWDeployment anstoßen. Hierfür steht die Klasse DeploymentFactory zur Verfügung. Hierin finden sich folgende Methoden:

setConfigFile(File configFile): Setzt die zu verwendende Konfigurationsdatei. Wird keine Konfigurationsdatei gesetzt, so wird die Standartkonfiguration der ADWCore Komponente genutzt. Mehr Informationen zur Konfigurationsdatei erhalten sie in Kapitel A.1.3.

run(String proGoalMLPath, FilenameFilter filter): Diese Methode startet einen kompletten Deployment-Zyklus. Durch Übergabe eines Pfades, welcher eine oder mehrere ProGoalML-Dateien enthält sowie einem FilenameFilter werden zunächst die benötigten Kerndokumente erzeugt. Mit Hilfe dieser wird im Anschluss der Pentaho BI-Server konfiguriert. Vor Veränderungen an der Datenbank wird ein Datenbank-Backup erzeugt.

executeETLAction(String resultElementName): Mit Hilfe dieser Methode wird versucht, einen der erzeugten ETL-Prozess aufzurufen. ETL-Prozesse werden anhand ihres Namens identifiziert. Dieser entspricht dem Namen der Überdeckungsklasse.

Alle drei Methoden führen im Fall eines Fehlers zu einer ADWException. Diese besitzt neben einer Fehlermeldung auch einen zugehörigen Fehlercode. Einen Überblick über mögliche ADWExceptions und Erklärungen hierzu finden sie in Kapitel A.1.4.

Listing A.1 zeigt einen Beispielaufruf der Komponente ADWDeployment zur Generierung eines Data Warehouses.

```
1  public static void main(String[] args) {
2          String proGoalML = "%\\de.fraunhofer.iao.apro.adwDeployment\\files";
3          FilenameFilter filter = new FilenameFilter() {
4
5                  @Override
6                  public boolean accept(File file, String filename) {
7                          if(filename.startsWith("progoalml")) {
8                                  return true;
9                          }
10                         return false;
11                 }
12         };
13         try {
14                 DeploymentFactory.instance().run(proGoalML, filter);
15         } catch (ADWException e) {
16                 e.printStackTrace();
17         }
18 }
```

Listing A.1: Beispielaufruf der Komponente ADWDeployment

A.1.3. Konfigurationselemente

Die Konfigurationsdatei ist in mehrere Abschnitte unterteilt: *Global*, *Output* sowie *Options*. Im Folgenden werden die einzelnen Elemente dieser Abschnitte aufgelistet sowie ihre Funktion beschrieben.

Global

Der Bereich *Global* enthält allgemeine Informationen zum Loggingverhalten, der zu verwendenden Datenbank sowie dem Pentaho BI-Server. Tabelle A.1 stellt die einzelnen Einstellungsmöglichkeiten detailliert dar.

Output

Innerhalb des *Output*-Bereichs werden Informationen für die Generierung der Einzeldokumente spezifiziert. Informationen zu den einzelnen Punkten bietet Tabelle A.2.

	Beschreibung	Beispiel
Logger	Dieser Bereich beinhaltet Informationen zum Loggingverhalten von ADW	
loggerPath	Speicherort von Logdateien	C://pentaho/adw/logs/
loggerFileName	Name der Logdateien	adw.log
active	Zustand des loggin Mechanismus	true oder false
database	Enthält Informationen zur verwendeten Datenbank.	
name	Name der Datenbank, welche ADW anlegen bzw. verwenden soll.	ADWDataBase
url	URL des Datenbankservers, ohne http://	localhost
username	Name des zu verwendenden Datenbank Benutzers	username
password	Zum Benutzernamen passendes Passwort	password
type	Datenbanktyp	mySQL
engine	Zu verwendende Speicher-Engine	InnoDB
backup	Bestimmt, ob vor Datenbankänderungen ein Backup erstellt werden soll	true oder false
maxBackupCount	Maximale Anzahl von Backupdateien die vorgehalten werden sollen	5
pentaho	Enthält Informationen bezüglich des Pentaho BI-Servers	
path	Pfad zum BI-Server	%dir%\biserver-ce\
solutionFolder	Pfad zum Solution-Ordner. Relativ zum Pfad des BI-Servers	adw\
olapFolder	Pfad zum Ordner, in welchem Olap-Schemata gespeichert werden. Relativ zum Pfad des BI-Servers	adw\olap\
dataSource	Name der innerhalb von Pentaho zu benutzenden DataSource	adw
dbType	Die von Pentaho intern genutzte Datenbank	hsqldb
dbPath	Pfad zur Pentaho internen Datenbank (ohne http://)	localhost:9001/hibernate
dbUserName	Name des zu verwendenden Datenbank-Benutzers	pentaho_user
dbPassword	Zum Benutzernamen passendes Passwort	password

Tabelle A.1.: Global-Tags der Konfigurationsdatei

	Beschreibung	Beispiel
sqlFile	Enthält Informationen, welche zur Generierung des Datenbankscripts benötigt werden	
name	Name des zu erzeugenden Datenbank-Scripts	Adw
targetpath	Zielpfad (relativ zu ADWCore)	C://pentaho/
ending	Dateiendung	.sql
prefixIndex	Der für einen Index zu nutzende Prefix	idx_
prefixDimension	Der für eine Dimensionstabelle zu nutzende Prefix	dim_
prefixFact	Der für eine Faktentabelle zu nutzende Prefix	fact_
kettleFile	Enthält Informationen, welche zur Generierung der ETL-Prozesse benötigt werden	
name	Dateiname des zu erzeugenden ETL-Prozesses	Adw
targetpath	Zielpfad (relativ zu ADWCore)	C://pentaho/
ending	Dateiendung	.ktr
inputPath	Pfad zu den zu verarbeitenden Ergebnisdateien	C://pentaho/input/
inputFileEnding	Endung der zu verarbeitenden Ergebnisdateien	.xml
inputFileNamingRegex	Regular Expression zur Identifizierung von Ergebnisdateien. Wird nur benötigt, falls nur spezielle Dateinamen berücksichtigt werden sollen.	(proGoal)([0-9]+)(.xml)
olapFile	Enthält Informationen zur Generierung des OLAP-Schemas	
name	Dateiname des zu erzeugenden OLAP-Schemas	Adw.mondrian
targetpath	Zielpfad (relativ zu ADWCore)	C://pentaho/
ending	Dateiendung	.xml
schemaName	Name des zu erzeugenden Schemas	ADWSchema
cubeCachingEnabled	Option, das das Cachen eines OLAP-Würfels durch Pentaho ermöglicht	true oder false

Tabelle A.2.: Output-Tags der Konfigurationsdatei

Options

Mit Hilfe des *Options*-Bereichs lassen sich zusätzliche Dimensionen auf Basis von Buckets sowie Informationen zur Erzeugung von Measures spezifizieren.

dimensions

In diesem Bereich können Buckets definiert werden, welche als Dimensionen behandelt werden. Eine Dimension (*dim*) muss hierbei immer einem Datentyp zugeordnet sein. Gültige Datentypen sind: *long*, *int*, *String*, *double*, *float*, *boolean*.

Wird für eine Dimension kein *identifier* angelegt, so wird sie auf jeden Wert eines Ergebnis-Schemas angewendet, welcher dem definierten Datentyp entspricht. Mit Hilfe des *identifier*-Tags lässt sich weiter einschränken, welche Kennwerte aus einem Ergebnis-Schema als Dimension behandelt werden sollen. Hierzu muss der *type* des *identifiers* angegeben werden. Gültige Typen sind *trueTypeIs*, *nameContains* sowie *nameEquals*. Mit *trueTypeIs* lassen sich Datentypen referenzieren, welche in der ProGoalML angegeben werden können, jedoch keinen primitiven Datentyp darstellen. Erlaubt sind also ID und TIMESTAMP.

NameContains prüft, ob der Namen eines Kennwertes den in der Konfiguration gegebenen String enthält. Ist dies der Fall wird der Kennwert als Dimension behandelt. *NameEquals* verfährt auf dieselbe Weise, nur muss der Name des Kennwertes dem String in der Konfiguration entsprechen.

Mithilfe eines *column*-Tags wird ein Level der soeben definierten Dimension definiert. Hierfür muss mittels des *name*-Attributs der Name des Levels sowie mittels des *type*-Attributs der Datentyp des Levels angegeben werden. Gültige Datentypen sind erneut: *long*, *int*, *String*, *double*, *float* und *boolean*.

Innerhalb eines *column*-Tags muss ein *transform* definiert werden. *Transform* kann vom Typ *dateYear*, *dateMonth*, *dateDay*, *dateWeeks*, *dateQuarter*, *substring* oder *charAt* sein. Während *substring* und *charAt* auf Strings angewendet werden können um Teilstrings zu extrahieren, bieten die mit „date“ beginnenden Typen die Möglichkeit zur Auflösung eines Timestamps:

- *substring*: Gibt den ersten Teil eines Strings zurück. Die Länge des Strings entspricht dem gegebenen Wert.
- *charAt*: Extrahiert ein Zeichen an der Stelle X.
- *dateYear*: Erzeugt das Jahr aus einem gegebenen Timestamp.
- *dateMonth*: Erzeugt den Monat aus einem gegebenen Timestamp.
- *dateDay*: Erzeugt den Wochentag aus einem gegebenen Timestamp.
- *dateWeeks*: Erzeugt die Woche aus einem gegebenen Timestamp.
- *dateQuarter*: Erzeugt das Quartal aus einem gegebenen Timestamp.

Die hierbei angegebene Reihenfolge der *column*-Tags entspricht hierbei der Reihenfolge der Levels innerhalb der Hierarchie der beschriebenen Dimension.

Listing A.2 zeigt hierfür ein umfassendes Beispiel. In den Zeilen 61-78 wird beispielsweise eine Zeitdimension auf dem Datentyp long definiert. Zeile 88-93 definieren eine typische Bucket-Dimension für eine Postleitzahl: Zusätzlich zur eigentlichen PLZ wird ein Level für die erste Ziffer der Postleitzahl, zur groben Einordnung in Gebiete, bereitgestellt.

measures

Im Bereich *Measures* werden die in ADWCore zu erzeugenden Measures definiert. Hierbei kann zwischen zwei unterschiedlichen Definitionsarten unterschieden werden. Innerhalb des *allMeasures*-Tags werden Aggregationen definiert, welche auf alle Fakten angewendet werden. Hierin befinden sich im Normalfall Summen- und Count-Aggregationen. Dies wird beispielhaft in Listing A.2 Zeile 96-100 dargestellt.

Innerhalb von *measure*-Tags können zusätzliche Measures auf Basis von Datentypen definiert werden. Dies funktioniert analog zum bereits beschriebenen Mechanismus zur Erstellung von Dimensionen. Der *identifier*-Tag kann hierbei ebenfalls genutzt werden. In den Zeilen 101-105 von Listing A.2 werden zusätzliche Measures für den Datentyp *double* definiert. Diese werden erzeugt, wenn der Name des Kennwertes den String „Savings“ enthält.

Mögliche Aggregationen sind: *sum*, *count*, *min*, *max*, *avg*, *distinct-count*.

A.1.4. Fehler-Codes

Die in den Komponenten ADWCore und ADWDeployment auftretenden Fehler werden abgefangen und als ADWExceptions weitergereicht. Eine ADWException enthält neben einer aussagekräftigen Fehlermeldung einen Fehlercode. Tabelle A.4 gibt einen Überblick über alle existierenden Fehlercodes, ihre Bedeutung sowie Klassen, in welchen diese Fehler auftreten können.

Fehlercode	Beschreibung	Auftreten
100	Es konnte keine ProGoalML gefunden werden.	ADWFactory
110	Fehler beim Erzeugen von Ergebnis-Schemata.	ADWFactory
111	Es wurden keine Ergebnis-Schemata generiert, diese werden jedoch benötigt.	ADWFactory, KettleTransformation
120	Es wurde keine Konfiguration geladen.	ADWFactory
121	Die Konfigurationsdatei konnte nicht gelesen werden.	ConfigFactory
122	Die übergebene Konfigurationsdatei entspricht nicht dem Konfigurationsschema.	ConfigFactory

Tabelle A.3.: Übersicht über Fehlercodes und deren Bedeutung Teil 1

Fehlercode	Beschreibung	Auftreten
300	Die Anzahl der übergebenen Argumente stimmt nicht mit den erwarteten überein.	Paket: kettle
310	Ein nicht unterstützter Datentyp wurde eingelesen.	StepDate
400	Fehler beim Erzeugen der Transformation.	DocumentFactory
500	Ein Feld konnt nicht gefunden werden.	CompareResult
666	Die übergeben Datei besitzt ein nicht unterstütztes Format.	DocumentFactory, TransformationFactory
800	Die Ausführung eines MySQL-Scripts ist fehlgeschlagen.	MySQLDeploy
801	Fehler beim Ausführen eines MySQL-Scripts. Der Name und die Fehlerzeile werden in der Fehlermeldung genannt.	MySQLDeploy
802	Die gegebene Datei konnte nicht gefunden werden. Der Name der Datei wird in der Fehlermeldung genannt.	MySQLDeploy
803	Es konnte keine Verbindung zur Datenbank aufgebaut werden.	SQLConnection
804	Der benötigte SQL-Treiber kann nicht aufgerufen werden.	SQLConnection
805	Die Ausführung eines SQL-Statements ist fehlgeschlagen.	DataSourceHSQLDB
810	Fehler beim Schreiben oder Lesen einer Datei. Der Name der Datei wird in der Fehlermeldung genannt.	Pakete: config, documentHandler.generator, documentHandler.reader, adwDeployment.mysql, adwDeployment.pentaho
811	Ein übergebenes XML-Dokument ist ungültig. Der Name des Schemas wird in der Fehlermeldung genannt.	ConfigFactory
812	Ein Dokument konnte nicht erstellt werden. Der Name des Dokuments wird in der Fehlermeldung genannt.	DocumentFactory
850	Ein Fehler beim Deployment ist aufgetreten. Der fehlerhafte Deployment-Schritt wird namentlich in der Fehlerbeschreibung genannt.	KettleDeploy, MondrianDeploy
851	Das Löschen des Mondrian Cache war nicht möglich.	MondrianDeploy
1000	Ein unbekannter Fehler ist aufgetreten, der nicht näher spezifiziert wurde.	-

Tabelle A.4.: Übersicht über Fehlercodes und deren Bedeutung Teil 2

A.1.5. Bekannte Probleme & Lösungen

Anmeldung als Benutzer ist nicht möglich

Nach der Installation von Pentaho kann es vorkommen, dass die Passwörter der automatisch angelegten Benutzer nicht angelegt werden. In der in Kapitel A.1.1 gegebenen Anleitung wird erklärt, wie sie diese Passwörter neu setzen können.

Es wird kein Datenbankbackup erzeugt

ADW kann die für das Datenbankupdate nötigen Umgebungsvariablen nicht finden. Ergänzen sie die PATH Variable in den Umgebungsvariablen um den Pfad des /bin Ordners ihrer MySQL Installation. Eine detaillierte Anleitung hierfür finden sie in Kapitel A.1.1.

Das neu erstellte OLAP-Schema steht in Pentaho nicht zur Verfügung

Der Mondrian eigene Cache wurde nicht korrekt gelöscht. Dies kann auch von einem Benutzer mit entsprechenden Rechten manuell ausgeführt werden. Melden sie sich hierfür als Benutzer auf http://localhost:8080 an. Nun können sie, wie in Abbildung A.3 dargestellt, über Werkzeuge->Aktualisiere->Mondrian Schema Cache den entsprechenden Cache löschen.

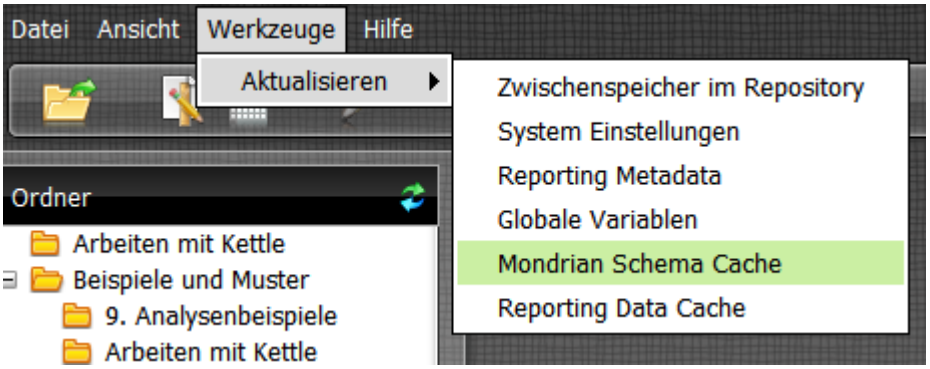

Abbildung A.3.: Manuelles Löschen des Mondrian Cache

A.2. Beispiel für eine Konfigurationsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<ADWConfig>
	<global>
		<logger>
			<loggerPath>.</loggerPath>
			<loggerFileName>log.log</loggerFileName>
			<active>true</active>
		</logger>
		<database>
			<name>databaseName</name>
			<url>localhost</url>
			<username>user</username>
			<password>password</password>
			<port>3306</port>
			<type>mySQL</type>
			<engine>InnoDB</engine>
			<backup>true</backup>
			<backupPath>output/</backupPath>
			<maxBackupCount>3</maxBackupCount>
		</database>
		<pentaho>
			<path>%dir%\pentaho\BIServer\biserver-ce\</path>
			<solutionFolder>adw\</solutionFolder>
			<olapFolder>adw\olap\</olapFolder>
			<dataSource>adw</dataSource>
			<dbType>hsqldb</dbType>
			<dbPath>localhost:9001/hibernate</dbPath>
			<dbUserName>pentaho_user</dbUserName>
			<dbPassword>password</dbPassword>
		</pentaho>
	</global>
	<output>
		<sqlFile>
			<name>Adw</name>
			<targetpath>output/</targetpath>
			<ending>.sql</ending>
			<prefixIndex>idx_</prefixIndex>
			<prefixDimension>dim_</prefixDimension>
			<prefixFact>fact_</prefixFact>
		</sqlFile>
		<kettleFile>
			<name>Adw</name>
			<targetpath>output/</targetpath>
			<ending>.ktr</ending>
			<inputPath>
				%dir%\pentaho\BIServer\biserver-ce\pentaho-solutions\adw\Kettle
			</inputPath>
			<inputFileEnding>.xml</inputFileEnding>
			<inputFileNamingRegExp>*</inputFileNamingRegExp>
		</kettleFile>
		<olapFile>
			<name>Adw.mondrian</name>
			<targetpath>output/</targetpath>
```

```
54                      <ending>.xml</ending>
55                      <schemaName>ADWSchema</schemaName>
56                      <cubeCachingEnabled>true</cubeCachingEnabled>
57              </olapFile>
58      </output>
59      <options>
60              <dimensions>
61                      <dim type="long">
62                              <identifier type="trueTypeIs">timestamp</identifier>
63                              <column name="year" type="TimeYears">
64                                      <transform type="dateYear" />
65                              </column>
66                              <column name="month" type="TimeMonths">
67                                      <transform type="dateMonth" />
68                              </column>
69                              <column name="quarter" type="TimeQuarters">
70                                      <transform type="dateQuarter" />
71                              </column>
72                              <column name="day" type="TimeDay">
73                                      <transform type="dateDay" />
74                              </column>
75                              <column name="week" type="TimeWeeks">
76                                      <transform type="dateWeeks" />
77                              </column>
78                      </dim>
79                      <dim type="int">
80                              <identifier type="nameContains">Param</identifier>
81                              <column name="first2digit" type="int">
82                                      <transform type="substring">2</transform>
83                              </column>
84                              <column name="firstdigit" type="int">
85                                      <transform type="charAt">1</transform>
86                              </column>
87                      </dim>
88                      <dim type="String">
89                              <identifier type="nameEquals">PLZ</identifier>
90                              <column name="firstchar" type="string">
91                                      <transform type="charAt">1</transform>
92                              </column>
93                      </dim>
94              </dimensions>
95              <measures>
96                      <allMeasures>
97                              <aggregator>sum</aggregator>
98                              <aggregator>count</aggregator>
99                              <aggregator>avg</aggregator>
100                     </allMeasures>
101                     <measure type="Double">
102                             <identifier type="nameContains">Savings</identifier>
103                             <aggregator>max</aggregator>
104                             <aggregator>min</aggregator>
105                     </measure>
106             </measures>
107     </options>
```

```
108 </ADWConfig>
```

Listing A.2: Ein Beispiel einer Konfigurationsdatei für ADW-Komponenten

Literaturverzeichnis

[All09] T. Allweyer. *BPMN 2.0 - Business Process Model and Notation*. Books on Demand, 2009.

[BG04] A. Bauer, H. Günzel. *Data-Warehouse-Systeme. Architektur, Entwicklung, Anwendung*. dpunkt Verlag, 2004.

[CGM98] T. Critchlow, M. Ganesh, R. Musick. Automatic Generation of Warehouse Mediators Using an Ontology Engine. In *Proceedings of the 5th International Workshop on Knowledge Represenation Meets Databases (KRDB '98), volume 10. CEUR Workshop Proceedings*, pp. 8–1. 1998.

[DCSW09] U. Dayal, M. Castellanos, A. Simitsis, K. Wilkinson. Data integration flows for business intelligence. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pp. 1–11. ACM, New York, NY, USA, 2009.

[Esp12] EsperTech. Esper – CEP-Engine, 2012. URL http://esper.codehaus.org/. (besucht im Mai 2012).

[GGP09] R. Gabriel, P. Gluchowski, A. Pastwa. *Data Warehouse & Data Mining*. W3L, 2009.

[GHJJ96] E. Gamma, R. Helm, R. Johnson, V. John. *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 1996.

[GSS10] S. Grohe, C. Schlameuß, R. Somme. *Performancevergleich von CEP-Engines*. Master's thesis, Institut für Parallele und Verteilte Systeme Universität Stuttgart, 2010.

[HCS07] S.-M. Huang, T.-H. Chou, J.-L. Seng. Data warehouse enhancement: A semantic cube model approach. *Information Sciences*, 177(11):2238 – 2254, 2007.

[HK07] M. Held, I. Klose. Business Intelligence mit Pentaho. *Heise Software*, 10.12.2007:9, 2007. URL http://www.heise.de/open/artikel/BI-Plattform-224858.html. (besucht im Mai 2012).

[HSB00] K. Hahn, C. Sapia, M. Blaschka. Automatically generating OLAP schemata from conceptual graphical models. In *Proceedings of the 3rd ACM international workshop on Data warehousing and OLAP*, DOLAP '00, pp. 9–16. New York, NY, USA, 2000.

[Hug08] R. Hughes. *Agile Data Warehousing: Delivering World-Class Business Intelligence Systems Using Scrum and XP*. iUniverse.com, Incorporated, 2008.

[KW12] F. Koetter, M. Weidmann. Goal-Oriented Model-Driven Business Process Monitoring using ProGoalML. 2012.

[Lef11] D. Leffingwell. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Addison-Wesley Professional, 1st edition, 2011.

[Leh03] W. Lehnert. *Datenbanktechnologie für Data-Warehouse-Systeme*. dpunkt.verlag, 1. Auflage 2003.

[Li06] X. Li. Building an Agile Data Warehouse: A Proactive Approach to Managing Changes. In *Proc. of the 4th IASTED Intl. Conf.* 2006.

[MnMT09] L. Muñoz, J.-N. Mazón, J. Trujillo. Automatic generation of ETL processes from conceptual models. In *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*, DOLAP '09, pp. 33–40. New York, NY, USA, 2009.

[Nol99] C. Nolan. Manipulate and Query OLAP Data Using ADOMD and Multidimensional Expressions. *Microsoft System Journal*, August 1999, 1999.

[Ora12] Oracle Corporation. MySQL, 2012. URL http://www.mysql.de/. (besucht im Mai 2012).

[ory11] The Oryx Project, 2011. URL http://www.oryx-project.org. (besucht im Mai 2012).

[PC03] F. R. S. Paim, J. F. B. de Castro. DWARF: AN Approach for Requirements Definition and Management of Data Warehouse Systems. RE '03, pp. 75–. IEEE Computer Society, Washington, DC, USA, 2003.

[PD02] C. Phipps, K. C. Davis. Automating data warehouse conceptual schema design and evaluation. In *Design and Management of Data Warehouses*, pp. 23–32. 2002.

[pen12] Pentaho Community Homepage, 2012. URL http://community.pentaho.com/. (besucht im Mai 2012).

[PMT08] J. Pardillo, J.-N. Mazón, J. Trujillo. Model-Driven Metadata for OLAP Cubes from the Conceptual Modelling of Data Warehouses. volume 5182 of *Lecture Notes in Computer Science*, pp. 13–22. Springer Berlin / Heidelberg, 2008.

[PR06] K. Pfetzing, A. Rohde. *Ganzheitliches Projektmanagement*. Versus-Verl., 2006.

[RC02] P. Rob, C. Coronel. *Database Systems Design, Implementation and Management*. Course Technology Press, Boston, MA, United States, 5th edition, 2002.

[SBH00] C. Sapia, M. Blaschka, G. Höfling. Höfling: GraMMi - Using a Standard Repository Management System to build a Generic Modeling Tool. In *In Proc. Hawaii Int. Conference on System Sciences (HICSS00), Maui*, pp. 1–10. 2000.

[SBHD98] C. Sapia, M. Blaschka, G. Höfling, B. Dinter. Extending the E/R Model for the Multidimensional Paradigm, 1998.

[Sim05] A. Simitsis. Mapping conceptual to logical models for ETL processes. In *Proceedings of the 8th ACM international workshop on Data warehousing and OLAP*, DOLAP '05, pp. 67–76. 2005.

[SKA+08] I.-Y. Song, R. Khare, Y. An, S. Lee, S.-P. Kim, J. Kim, Y.-S. Moon. SAMSTAR: An Automatic Tool for Generating Star Schemas from an Entity-Relationship Diagram. volume 5231 of *Lecture Notes in Computer Science*, pp. 522–523. Springer Berlin / Heidelberg, 2008.

[SM11] K. Schweinsberg, H. Messerschmidt. Olap - Gut gewürfelt ist halb entschieden!, 2011. URL http://www.db.informatik.uni-kassel.de/~ks/OLAP-Poster.pdf. (besucht im Mai 2012).

[Solo5] M. D. Solomon. Ensuring A Successful Data Warehouse Initiative. *IS Management*, pp. 26–36, 2005.

[TBC99] N. Tryfona, F. Busborg, J. G. B. Christiansen. starER: A Conceptual Model for Data Warehouse Design. pp. 3–8. 1999.

[The12] The HSQL Development Group. HyperSQL Database, 2012. URL http://hsqldb.org/. (besucht im Mai 2012).

[Tho02] E. Thomsen. *Olap solutions: building multidimensional information systems*. Wiley, 2002.

[TPGS01] J. Trujillo, M. Palomar, J. Gomez, I.-Y. Song. Designing Data Warehouses with OO Conceptual Models. *Computer*, 34(12):66–75, 2001.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(*Philipp Sigloch*)