UNIKASSEL VERSITÄT

SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

MDBDA

Model-driven Big Data Analytics

Masterarbeit

Universität Kassel

Fachbereich Elektrotechnik / Informatik

Fachgebiet Softwaretechnik

vorgelegt von

Marcel Hahn, B.Sc.

Matrikelnummer: 27235476

Erstprüfer	:	Prof. Dr. Albert Zündorf
Zweitprüfer	:	Prof. Dr. Gerd Stumme
Betreuer	:	Stefan Lindel, M.Sc.
Eingereicht am	:	22. April 2014

Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbständig und ohne fremde Hilfe angefertigt zu haben. Die verwendete Literatur und sonstige Hilfsmittel sind vollständig angegeben.

Kassel, 22. April 2014

Inhaltsverzeichnis

1 Motivation und Ziele **3**

2 Grundlagen **4**

2.1 Big Data 4

2.2 Model Driven Engineering 5

2.3 MapReduce 6

2.4 Complex Event Processing 7

3 MDBDA Zielplattform **8**

3.1 Referenzarchitektur und eingesetzte Technologien 10

3.1.1 Apache Hadoop 11

3.1.2 Apache Cassandra 11

3.1.3 Apache Storm 12

3.1.4 Neo4j 13

3.1.5 Apache Kafka 15

3.1.6 Hazelcast 16

4 MDBDA Workflow **17**

5 Tooling **19**

5.1 Eingesetzte Technologien 19

5.1.1 Eclipse Modeling Framework 19

5.1.2 Xcore 20

5.1.3 Graphiti 20

5.1.4 Xpand Template Language 22

5.1.5 Xtend Code Generator 24
5.2 MDBDA Domänenmodell 26
5.3 Graphical Editor 29
5.4 Codegenerierung 32
5.5 MDBDA Pattern 33
5.5.1 Datenorganisation 34
5.5.2 Filter 38
5.5.3 Kombination 39
5.5.4 Berechnung 42
5.5.5 Ressourcen 44

6 Qualitätssicherung 47
6.1 SWTBot 47
6.2 JUnit 48
6.3 MRUnit 49

7 Anwendung 50
7.1 Wordcount 50
7.2 Verknüpfung von Ressourcen 52

8 Related Work 56

9 Fazit und Ausblick 57

10 Anhang 58

1 Motivation und Ziele

Die Verarbeitung großer Datenmengen, sehr heterogener Daten und/oder mit einem hohen Datendurchsatz wird unter dem Begriff Big Data zusammengefasst und ist ein großer Trend in der Informatik und Wirtschaft[1][2]. Der weltweite Big Data Markt soll jährlich um 36% wachsen[3]. Qualifizierte Experten sind in dieser relativ neuen Technologie noch schwer zu finden. Analysen dieser Daten gestalten sich für den Anwender noch recht schwierig, da viel Programmierarbeit in die Erstellung von Analysealgorithmen gesteckt werden muss. Zusätzlich müssen Entwickler dieser Algorithmen sowohl domänenspezifisches Wissen über die Daten, als auch Erfahrungen mit den eingesetzten Technologien und Techniken mitbringen. Im Rahmen dieser Arbeit soll eine Möglichkeit aufgezeigt werden, wie die Erstellung von Auswertungsalgorithmen für Massendaten und heterogene Daten durch den Einsatz eines modellgetriebenen Vorgehens vereinfacht werden kann. Dadurch soll es möglich werden, Analysen mit wenig Wissen über die eingesetzten Technologien und Techniken zu erstellen. Eine Entwicklungsumgebung soll es schließlich einem Domänenexperten erlauben, mithilfe eines graphischen Editors Analysen zu erstellen. Dabei soll er während der Erstellung durch eine stetige Konsistenzanalyse, die durch den definierten Entwicklungsworkflow vorgegebenen ist, gemäß dem Test-first Prinzip unterstützt werden. Der Einsatz eines graphischen Editors bringt dem Experten nicht nur Vorteile bezüglich Übersichtlichkeit der Analysen, sondern bietet auch eine steilere Lernkurve im Vergleich mit einer textuellen Domain-Specific Language (DSL)[4]. Schlussendlich lassen sich die erstellten Diagramme leicht Entscheidungsträgern nahebringen und dienen einer gut verständlichen Dokumentation.

2 Grundlagen

Zum Verständnis der Arbeit wird eine Basis an Fachwissen vorausgesetzt. Aus diesem Grund werden im folgenden Kapitel die grundlegenden Begriffe und Technologien erläutert. Beginnend mit dem Kontext, Big Data, in dem sich diese Arbeit primär ansiedelt, gefolgt von dem Vorgehensmodell, Model-driven engineering, um die angesprochenen Probleme aus dem Abschnitt 1 zu lösen. Gegen Ende dieses Abschnittes werden die beiden Lösungsstrategien, MapReduce und Complex Event Processing, vorgestellt.

2.1 Big Data

Unter dem Begriff Big Data ist nicht nur die Speicherung großer Datenmengen zu verstehen, da sich viele Daten auch in traditionellen relationalen Datenbanken oder Dateiarchiven speichern lassen. Big Data umfasst vielmehr auch die große Vielfalt der Daten, ein hohes Datenaufkommen, sowie große Komplexität. Diese Daten müssen effizient persistiert und verarbeitet werden. Die Abbildung 1, welche die Bitkom 2013 veröffentlicht hat, zeigt eine gute Gesamtübersicht von Big Data. Aufgrund der Vielfalt aller und der Komplexität einzelner Elemente, wird in dieser Arbeit nicht jedes Thema, so zum Beispiel Bildanalytik oder Visualisierung, von Big Data behandelt. Das resultierende Ergebnis wird jedoch ein breites Spektrum abbilden und Möglichkeiten zur Erweiterung bieten.

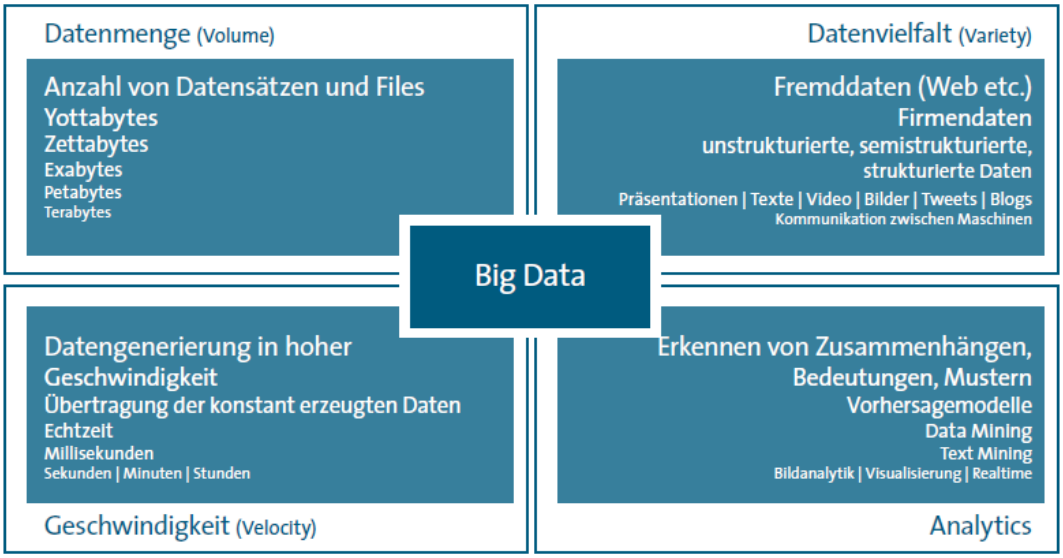

Abbildung 1: Bitkom Big Data Merkmale[5]

2.2 Model Driven Engineering

Model Driven Engineering (MDE)[6][7] ist ein Vorgehensmodell aus der Softwaretechnik, bei dem ein formales Domänenmodell im Vordergrund steht. Struktur und Verhalten der entstehenden Software werden mithilfe dieses Modells abgebildet und der ausführbare Code weitestgehend generiert. Dies bietet mehrere Vorteile gegenüber klassischer Softwareentwicklung. Zum Einen wird das zu lösende Problem auf einem höheren Abstraktionsniveau bearbeitet. Der Entwickler kann dadurch fokussierter an der domänenspezifischen Fragestellung arbeiten ohne von Programmiersprachen oder Framework spezifischen Eigenheiten abgelenkt zu werden. Zum Anderen bietet die Verwendung von formalen Modellen den Vorteil, dass diese Modelle formal verifiziert werden können. Es können im Entwicklungszyklus schon Fehler entdeckt werden, welche sonst nur durch intensives Testen gefunden werden können oder sogar erst im produktiven Einsatz auftreten. Die Übertragung des MDE Paradigmas in die Welt von Big Data Analysen wird vor mir in dieser Arbeit als Model Driven Big Data Analytics (im Folgenden nur noch MDBDA) bezeichnet.

2.3 MapReduce

Im Dezember 2004 veröffentlichten J. Dean und S. Ghemawat von Google ein Papier mit dem Titel „MapReduce: Simplified Data Processing on Large Clusters"[8]. Die Publikation beschreibt ein Programmiermodell für die effektive Verarbeitung großer Datenmengen in einem skalierbaren Cluster. Dazu bedienten sich die Autoren zweier Methoden aus der funktionalen Programmierung. Zum Einen die Map-Funktion, zum Anderen die Reduce-Funktion. In der Map-Phase wird die Funktion verteilt auf jedem Clusterknoten ausgeführt und verarbeitet die lokal gespeicherten Daten, filtert sie und teilt sie in Schlüssel-Wert Paare auf. Beispielsweise werden für eine Indexierung alle Textdokumente lokal gelesen. Jedes Wort aus den Texten wird nach Namen gefiltert. Dieser Name wird als Schlüssel genommen und als Wert eine Eins gesetzt.

Als Ergebnis erhält man folgende Liste auf Knoten (A):

Schlüssel	Wert
Alice	1 ; doc1.txt:23
Alice	1 ; doc1.txt:31
Alice	1 ; doc2.txt:127
Bob	1 ; doc1.txt:42
Bob	1 ; doc3.txt:14
...	...

Als erste Optimierung des in der Publikation beschriebenen Verfahrens wird nach der Map-Phase eine Combinate-Phase angeschlossen, die lokal auf dem Clusterknoten gleiche Schlüssel zusammenfasst.

Bei dieser Phase erhält man folgendes Ergebnis auf Knoten (A):

Schlüssel	Wert
Alice	3 ; doc1.txt:23, doc1.txt:31, doc2.txt:127
Bob	2 ; doc1.txt:42, doc3.txt:14
...	...

Die zweite aus der funktionalen Programmierung stammende Methode ist die Reduce-Funktion. In dieser Phase werden alle Werte zu einem Schlüssel im Cluster zusammengefasst. Beispielsweise ist bei Knoten (A) Alice = 3 und bei Knoten (B) Alice = 39. Nach der Reduce-Phase erhält man schließlich das Endergebnis Alice = 42.

Dieses MapReduce beziehungsweise MapCombinateReduce Verfahren hat sich als de facto Standard für die Verarbeitung großer Datenmengen durchgesetzt. Es gibt mehrere Implementierungen dieses Ansatzes. In dieser Arbeit wird nur auf die Open Source Implementierung Apache Hadoop[9] eingegangen.

2.4 Complex Event Processing

[10]Wie bereits im Abschnitt 2.1 (Big Data) beschrieben, ist unter dem Begriff Big Data nicht nur die Verarbeitung großer Datenmengen gemeint. Ein weiterer wichtiger Punkt ist die Bewältigung vieler zusammenhängender Datenpakete, Events, in einem kurzen Zeitraum. Eingehende Nachrichten werden als Events erfasst und über einen Message Broker an ein Event Processing Netzwerk weitergeleitet. Diese Event Processing Netzwerke bestehen in der Regel aus zustandslosen, zusammengeschalteten Verarbeitungskomponenten. Das Netzwerk überwacht, dass eine Nachricht definitiv verarbeitet wird und, selbst bei Ausfall einzelner Rechner im Netzwerk, nichts verloren geht. Die Zustandslosigkeit der Komponenten ist optional, für die Skalierbarkeit dennoch hilfreich, da dadurch kein gemeinsamer Speicher synchronisiert werden muss.

3 MDBDA Zielplattform

Dieser Abschnitt beschreibt die MDBDA Zielplattform. Zu Beginn werden grob die Rahmenarchitektur und die Anforderungen an die einzelnen Komponenten beschrieben. Auf der Basis dieser Rahmenarchitekturanforderungen werden die in dieser Arbeit für die Zielplattform eingesetzten Technologien kurz erläutert, um im Anschluss eine Referenzimplementierung der MDBDA Zielplattform vorzustellen.

Die in Abbildung 2 gezeigte Rahmenarchitektur berücksichtigt die in Abschnitt 2 (Grundlagen) beschrieben Anforderungen an Big Data Systeme. Eingehende Nachrichten werden von einem Message Broker entgegengenommen. Dieser Message Broker validiert eingehende Nachrichten, übersetzt Encodings und Formate und übernimmt das Routing innerhalb der Plattform. Neben diesen Hauptaufgaben ist der Message Broker für Lastverteilung und die Erhöhung der Robustheit verantwortlich. Für die verschiedenen Arten von Daten wie strukturierte Daten, semi- und unstrukturierte Daten sowie stark relationaler Daten wird jeweils ein spezialisierter Datenspeicher verwendet. Hinzu kommt noch eine sehr schnelle In-Memory Lösung. Diese spezialisierten Speicher sind in der Abbildung 2 blau eingefärbt. Große Datenmassen werden in den jeweiligen Stores, für strukturierte, wie tabellarische oder key-value Paare, semi- und unstrukturierte, wie Extensible Markup Language (XML) und binäre, Daten gespeichert. Diese Massendatenstores müssen leicht skalierbar und fehlertolerant sein. Defekte Rechnerknoten müssen vom System erkannt werden und sich ohne Ausfallzeit austauschen lassen. Zusätzlich muss sich das System an schwankende Lastanforderungen anpassen lassen, indem Knoten hinzu oder entfernt werden können. Die Daten, die

in diese Systeme gespeichert werden, haben die Eigenschaft selten geändert oder gelöscht zu werden. Das Schreiben und Lesen der Daten muss hingegen möglichst schnell erfolgen. Über diesen Speichern liegt eine Verarbeitungskomponente, welche möglichst datennah, möglichst ohne Netzwerkzugriff auf externe Ressourcen, arbeitet. Diese Komponente ist in der Rahmenarchitektur unter Batch Data Processing zu finden. Als Vorgehensmodell für diese Batch ähnliche Verarbeitung wird MapReduce (siehe Abschnitt 2.3) eingesetzt. Für den *Velocity*-Teil der Big Data Merkmale bietet die Rahmenarchitektur den Complex Event Processing Bereich zusammen mit einer In-Memory Speicherlösung. Der In-Memory Speicher ermöglicht es sehr schnell im Hauptspeicher Informationen zu speichern, zu ändern und zu löschen. Der hohe Preis von Hauptspeicher verhindert es noch, die zuvor beschriebenen Massendatenstores durch In-Memory Speicher zu ersetzen.

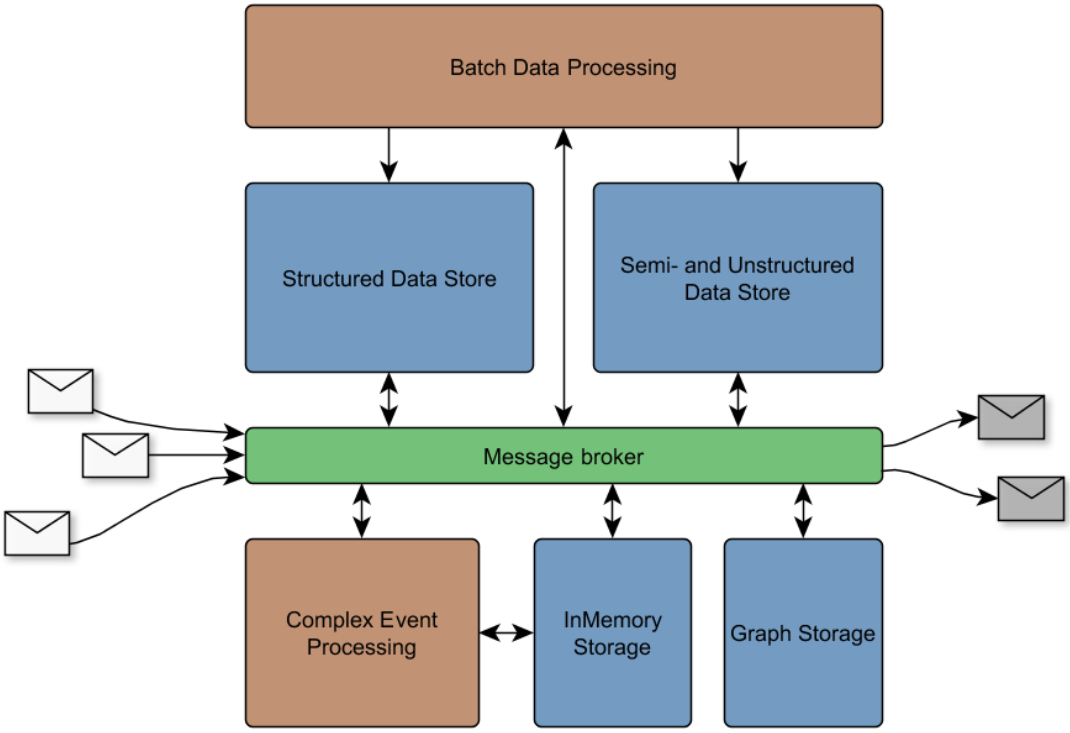

Abbildung 2: Rahmenarchitektur der MDBDA Zielplattform

3.1 Referenzarchitektur und eingesetzte Technologien

In diesem Abschnitt wird die zuvor beschriebene Rahmenarchitektur konkretisiert und die Referenzimplementierung für diese Arbeit beschrieben. Dazu zeigt Abbildung 3 die Referenzarchitektur der MDBDA Zielplattform und die Technologien, welche die verschiedenen Bereiche der Rahmenarchitektur abdecken. Der Vorteil dieser hybriden Architektur wird es sein, dass die Vorteile aller eingesetzten Technologien zum Tragen kommen. Selbst eine noch umfangreichere Architektur mit zusätzlicher klassischer Relational Database Management System (RDBMS)-Lösung[11] ist für den späteren Einsatz möglich, wird in dieser Arbeit jedoch nicht weiter behandelt.

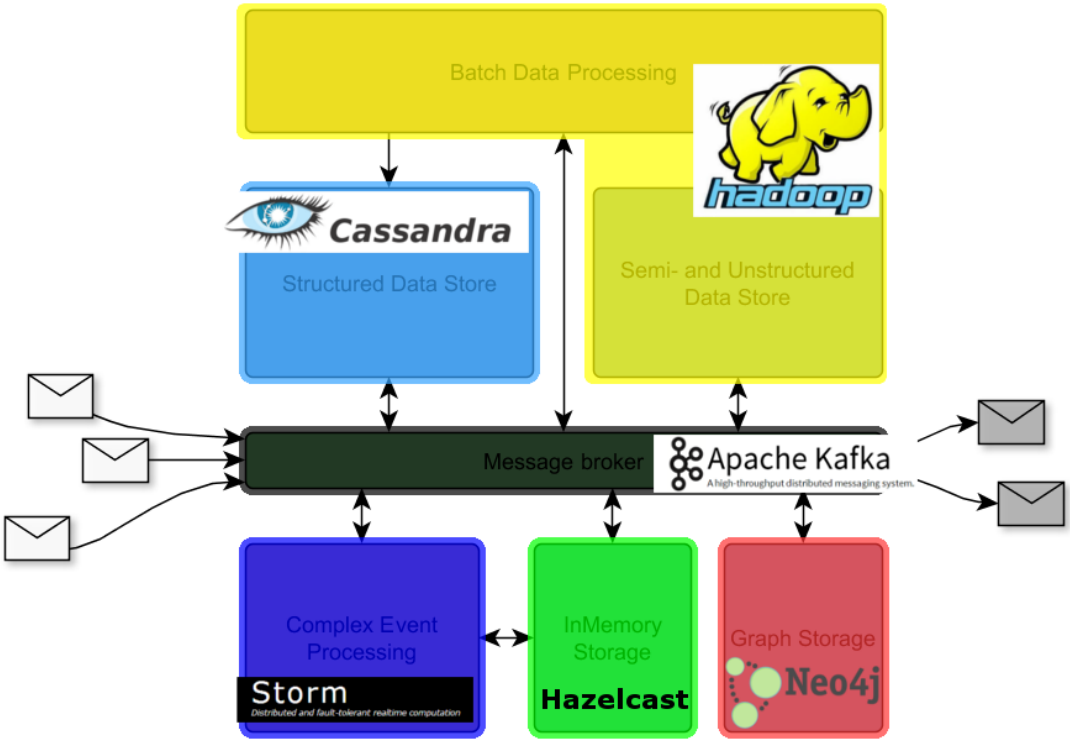

Abbildung 3: Referenzarchitektur der MDBDA Zielplattform

3.1.1 Apache Hadoop

Als Referenzimplementierung für die Batch Data Processing und semi- und unstrukturierte Datenspeicher Komponenten der Rahmenarchitektur wird in dieser Arbeit Apache Hadoop eingesetzt. Die Open Source Implementierung Hadoop [9] [12] [13] hat sich als de-facto-Standard [14] des in Abschnitt 2.3 beschriebenen MapReduce Papers durchgesetzt. Ferner bietet es mit dem Hadoop Distributed File System (HDFS) ein verteiltes Dateisystem, welches es ermöglicht beliebige Binär- oder Textdaten zu speichern. Seit Version 2.2.0[15] erlaubt HDFS den Datenzugriff mittels des *Netork File System* Protokoll nach RFC1813[16]. Damit ist es leicht möglich Altsysteme in die Architektur einzupflegen, indem Logdateien oder ähnliches in HDFS gespeichert werden.

3.1.2 Apache Cassandra

Für den Bereich strukturierte Daten hat sich Cassandra [17] [18] als gute Wahl gezeigt. Cassandra ist eine partiell spaltenorientierte Datenbank, beziehungsweise ein *Wide Column Store* [19]. Das heißt, dass die Daten nicht wie bei klassischen Structured Query Language (SQL) Datenbanken zeilenorientiert, sondern nach Spalten gespeichert werden. Strukturiert werden die Daten in vier oder fünf Dimensionen, entweder $\langle$Keyspace$\rangle \times \langle$ColumnFamily$\rangle \times \langle$RowKey$\rangle \times \langle$Key+Value$\rangle$ oder $\langle$Keyspace$\rangle \times \langle$ColumnFamily$\rangle \times \langle$RowKey$\rangle \times \langle$SuperColumn$\rangle \times \langle$Key+Value$\rangle$. Partiell spaltenorientiert[20, (S. 63 ff.)] ist die Datenbank daher, da die Spaltenorientierung erst in der $\langle$Key+Value$\rangle$. Dimension greift. Es lässt sich innerhalb einer ColumnFamily nicht eine Zeile als Column und eine als SuperColumn speichern. Dahingegen kann sich die Anzahl der Key-Value-Paare in den Zeilen unterscheiden.

3.1.3 Apache Storm

Storm[21][22] ist ein 2011 von Nathan Marz veröffentlichtes Open Source Project zur verteilten und fehlertoleranten Verarbeitung von komplexen Nachrichten, beziehungsweise Events. Marz war bis März 2013 Software Engineer bei Twitter, wo Storm für diverse Echtzeit Analysen [23] eingesetzt wird. Seit September 2013 ist Storm ein Apache Incubator Projekt [21] und steht damit unter der Apache Lizenz, genauso wie Apache Hadoop und Cassandra. Im Storm Kontext werden einige Begrifflichkeiten und Vorhergehensweisen verwendet. Events werden als *Tupel* abgebildet. Ein *Stream* ist eine ungebundene Sequenz von *Tupeln*. Als

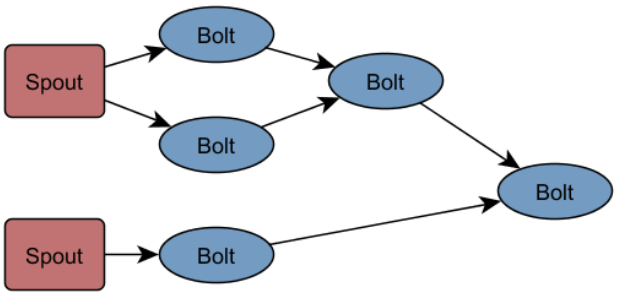

Abbildung 4: Einfache Storm Topologie

Spout wird ein gebundener *Stream* bezeichnet. Ein *Bolt* verarbeitet *Input Streams* und produziert *Output Streams*. Bei der Verarbeitung können beliebige Funktionen, von einfachen Filtern, über Berechnungen und Joins, bis hin zu komplexen Datenbank Operationen, zur Anwendung kommen. Eine *Topologie* ist ein ganzer Workflow, von *Spouts* bis über die Verknüpfung der verwendeten *Bolts*. Abbildung 4 zeigt eine einfache Topologie mit zwei *Spouts* und einigen verknüpften *Bolts*. Im Gegensatz zu einem MapReduce Job hat eine solche *Topologie* kein definiertes Ende und besteht solange, bis sie von Hand beendet wird.

3.1.4 Neo4j

Neo4j[24][25][26] ist eine Open Source Graphdatenbank[27]. Diese Graphdatenbank eignet sich, im Gegensatz zu den anderen vorgestellten Speicherlösungen, wie Cassandra und Hadoop, nicht zur Speicherung von Massendaten[28]. Mithilfe eines Neo4j-Clusters ist es zwar möglich ein Hochverfügbarkeitssystem aufzubauen, eine Skalierbarkeit ist hingegen nicht performant möglich. Die Stärke dieser Datenbank zeigt sich dahingegen bei verknüpften Daten. Algorithmen, beispielsweise zur Wegesuche oder die Analyse von sozialen Netzen, lassen sich dadurch deutlich effizienter ausführen. Die Daten werden bei Neo4j als Knoten und Kanten abgelegt. Ein Knoten kann dabei einen Typen und beliebig viele Attribute, Key-Value Paare, haben. Kanten sind immer gerichtet, sie haben einen Start- und einen Endknoten sowie ein Label. Zusätzlich können sie, wie bei Knoten, attributisiert werden.

```
1 CREATE (Albert:Person {name:"Albert"})  CREATE (Tobias:Person {name:"
    Tobias"})
2 CREATE (Stefan:Person {name:"Stefan"}) CREATE (Nina:Person {name:"Nina"})
3 CREATE (R1338:Room {name:"r1338"}) CREATE (R1339:Room {name:"r1339"})
4 CREATE (Albert)-[:KNOWS]->(Tobias) CREATE (Albert)-[:KNOWS]->(Stefan)
5 ... CREATE (Nina)-[:KNOWS]->(Stefan)
6 CREATE (Albert)-[:IS_IN]->(R1338) CREATE (Tobias)-[:IS_IN]->(R1338)
7 CREATE (Nina)-[:IS_IN]->(R1339) CREATE (Stefan)-[:IS_IN]->(R1339)
```

Listing 1: Daten erzeugen mittels Cypher

Neo4j bring eine einfache Abfragesprache namens Cypher mit. Mit ihr lassen sich einfach komplexe Graphabfragen erstellen. Im Folgenden wird ein kurzes Beispiel gezeigt, in welchem Personen und Raumbeziehungen abgebildet werden. Zu erfahren gilt es, welche Person jemanden kennt, der in einem bestimmten Raum sitzt. Listing 1 zeigt das Cypher Skript um Testdaten zu diesem Beispiel zu erstellen. Das *CREATE* Statement *(Albert:Person {name:"Albert"})* in Zeile 1 sagt aus, dass ein Knoten *Albert* vom Typ *Person* erstellt werden soll mit dem Attribut *name* und dessen Wert *"Albert"*. Listing 2 zeigt die Abfrage, für die zuvor beschriebene Problemstellung. Das *MATCH* Statement beschreibt, dass zwei Knoten vom Typ Person, *p* und *p2*, mit einer Kante von Typ *KNOWS*

```
MATCH (p:Person)-[:KNOWS]->(p2:Person)-[:IS_IN]->(r:Room)
WHERE r.name = "r1339"
RETURN p.name, p2.name, r.name;
```

Listing 2: Daten abfragen mittels Cypher

verbunden sein sollen. Zusätzlich soll es von *p2* aus noch eine Kante von Typ *IS_IN* zu einem Raum geben. Das *WHERE* Statement beschränkt die Abfrage noch weiter indem das Attribut *name* des Raums den Wert *"r1339"* haben soll. Schließlich liefert das *RETURN* Statement eine Ergebnistabelle mit drei Spalten. Abbildung 5 zeigt die Weboberfläche des Neo4j Server, auf welcher das Szenario nachgestellt wurde.

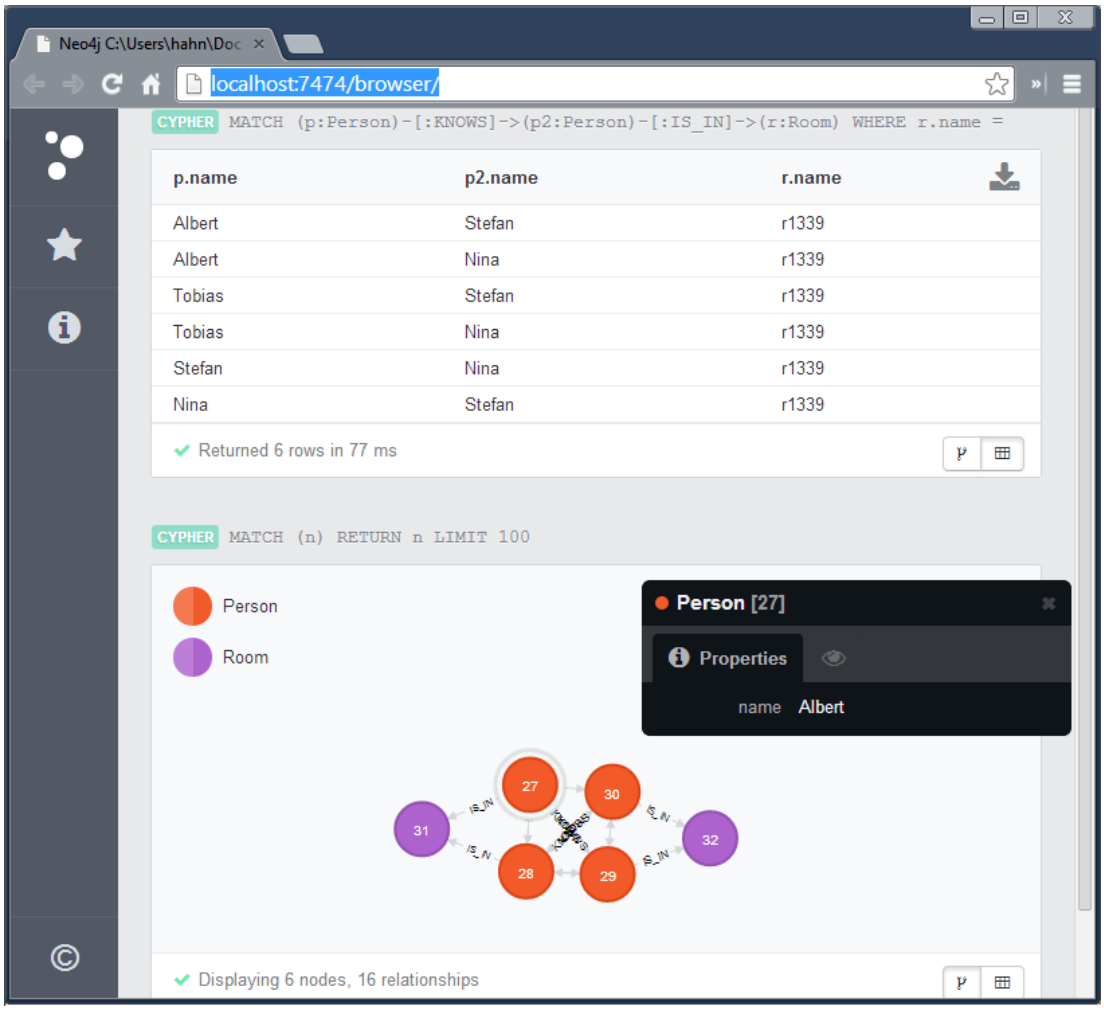

Abbildung 5: Neo4j Cypher Beispiel Weboberfläche

3.1.5 Apache Kafka

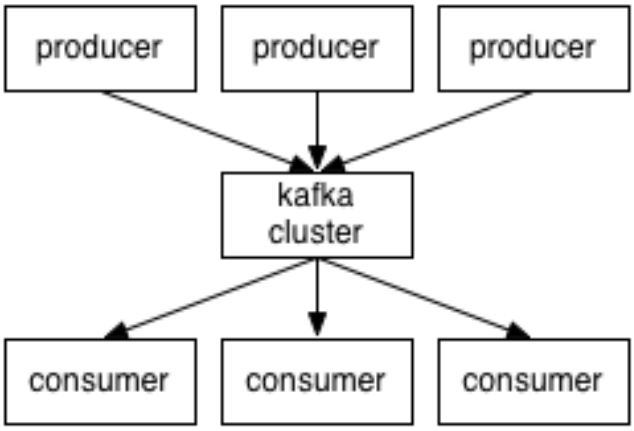

Abbildung 6: Kafka publish-subscribe

Kafka[29][30] ist ein Open Source Message Broker. Er ist dafür ausgelegt, in einem verteilten System einen hohen Nachrichtendurchsatz zu ermöglichen. Durch den Einsatz des publish-subscribe Musters lässt sich das System einfach um weitere Produzenten und Konsumenten erweitern. Es lässt sich dadurch nicht nur leicht um weiter Aufgaben erweitern, sondern auch in Bezug auf die Verarbeitungsgeschwindigkeit optimieren, indem weitere Produzenten oder Konsumenten dynamisch hinzugefügt werden können. Abbildung 6 zeigt dieses Muster. Nachrichten werden bei Kafka einem *topic* zugeordnet. Dieses *topic* verhält sich wie ein Kanal. Ein oder mehrere Produzenten senden ihre Nachrichten an ein *topic*. Konsumenten horchen auf dieses *topic*. Bei einer eingehenden Nachricht, nimmt einer der Konsumenten diese entgegen und verarbeitet sie. Fällt der Konsument bei der Verarbeitung aus, kann das System dies erkennen und die Nachricht einem anderen Konsumenten zur Verfügung stellen.

3.1.6 Hazelcast

Hazelcast[31][32][33] ist eine Open Source Java in-memory Speicherlösung. Sie kann innerhalb der Plattform für verschiedene Aufgaben, von In-Memory Data Processing bis lokalen Datenbank Cache, verwendet werden. Auch wenn Hazelcast deutlich mehr Features bietet, werden in dieser Arbeit nur die verteilten Datenstrukturen verwendet. Der Hauptaufgabenbereich dieser Strukturen ist die Kommunikation und Synchronisation zwischen *Bolts* bei Storm (siehe Abschnitt 3.1.3). Um eine reibungslose Skalierbarkeit der Storm Topologien zu gewährleisten, sollte zwar möglichst auf Synchronisation verzichtet werden, manche Anwendungsfälle machen dies jedoch zwingend erforderlich. Listing 3 zeigt den trivialen Einsatz einer verteilten Hazelcast Datenstruktur.

```
1  import com.hazelcast.config.Config;
2  import com.hazelcast.core.*;
3  import java.util.concurrent.ConcurrentMap;
4
5  public class DistributedMap {
6      public static void main(String[] args) {
7          Config config = new Config();
8          HazelcastInstance h = Hazelcast.newHazelcastInstance(config);
9          ConcurrentMap<String, String> map = h.getMap("my-distributed-map");
10         map.put("key", "value");
11         map.get("key");
12         //Concurrent Map methods
13         map.putIfAbsent("somekey", "somevalue");
14         map.replace("key", "value", "newvalue");
15     }
16 }
```

Listing 3: Hazelcast Distributed Map[31]

In Zeile 9 ist der Methodenaufruf *h.getMap(...)* zum Laden der Datenstruktur zu sehen. Der übergebene Parameter dient der Identifikation der Struktur im verteilten System. Zeile 10 und 11 zeigt das einfache Lesen und Schreiben in die Struktur. Die Methode *putIfAbsent* in Zeile 13 setzt den Wert nur, wenn der Schlüssel noch nicht gesetzt ist. Ansonsten liefert die Methode *null* als Rückgabe. Dahingegen ersetzt die Methode *replace* in Zeile 14 den Wert des Schlüssels nur, wenn er bereits vorhanden ist. Andernfalls liefert die Methode *false* als Rückgabe.

4 MDBDA Workflow

Der MDBDA Workflow beschreibt das Vorgehen eines Domänenexperten bei der Erstellung einer Analyse. Das Verfahren in dem Workflow ist an die Test-first[34] Praxis angelehnt. Das Verfahren sagt aus, dass erst beschrieben werden soll, was bei definierter Eingabeinformation als Ausgabe erwartet wird und wie es überprüft werden soll, bevor das Problem gelöst wird. Bei dem MDBDA Workflow soll dazu, vom Domänenexperten, zu Beginn ein oder mehrere Test Inputs von echten Daten abgeleitet werden. Anhand dieser Testdaten beschreibt der Experte die erwarteten Werte, die seine Analyse liefern soll. Diese Testdaten und erwarteten Ergebnisse werden stets während der Entwicklungszeit überprüft, sobald vom Domänenexperten etwas im MDBDA Editor, welcher im Abschnitt 5 (Tooling) definiert wird, geändert wird. Nachdem der Start, Test Input, und das Ziel, Er-

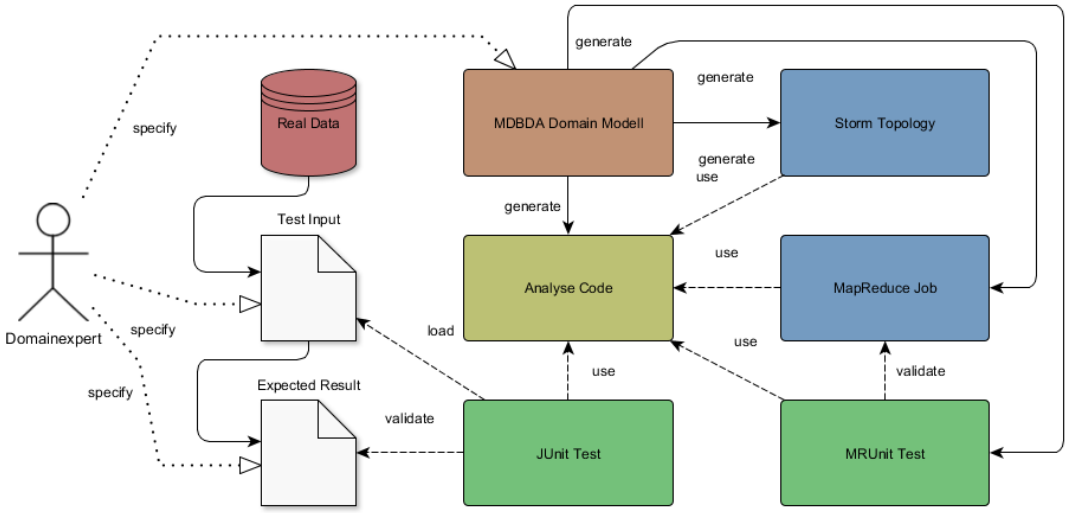

Abbildung 7: MDBDA Workflow

wartungsdaten, definiert wurden, wird der Weg, die Analyse, beschrieben. Dazu

definiert der Experte ein szenariospezifisches MDBDA Domänenmodell, wobei er von einem graphischen MDBDA Editor unterstützt wird. Aus der Beschreibung des Domänenmodells wird automatisch Analyse Java-Quellcode generiert. Zu diesem Datenanalyse Code wird ein Test generiert, welcher lokal den Testinput liest, den generierten Analysecode ausführt und das Ergebnis dieser Analyse mit den erwarteten Ergebnissen abgleicht. Dies wird mithilfe des JUnit Frameworks, welches in Abschnitt 6.2 genau beschrieben wird, durchgeführt. Aus dem Analysecode wird im nächsten Generationsschritt eine Storm Topologie sowie ein MapReduce Job erstellt. Die Topologie sowie der Job sind später auf der Zielplattform lauffähig (siehe Abschnitt 3). Im Falle des MapReduce Jobs wird ein dazugehöriger MRUnit(siehe Abschnitt 6.3) Test generiert. Dieser MRUnit Test führt den MapReduce Job mit den ursprünglichen echten Daten auf der Zielplattform aus. Das Ergebnis wird wiederum mit den erwarteten Daten verglichen. Im Anschluss des MRUnit Test werden alle temporären Test Daten auf der Zielplattform entfernt und das System in den ursprünglichen Zustand versetzt. Mit dem MDBDA Workflow werden zwei weitestgehend unabhängige Big Data Herausforderungen gelöst. Es können Massendaten mithilfe des generierten MapReduce Jobs analysiert werden, als auch Echtzeit Analysen durch die entstandene Storm Topologie vorgenommen werden. Ferner wird durch die Anwendung des MDBDA Workflows die Qualität von erstellten Analysen verbessert, da Fehler, durch das ständige Testen während der Entwicklung, frühzeitig erkannt werden und eine aussagekräftige Dokumentation durch Diagramme der graphische Modellierung entsteht.

5 Tooling

Der Kern dieser Arbeit sind die entwickelten Werkzeuge und Methoden zur Erstellung von modellgetriebenen Big Data Analysen. In diesem Abschnitt wird das Werkzeug vorgestellt, welches bei der Umsetzung des zuvor beschriebenen MDBDA Workflows (siehe Abschnitt 4) hilft. Dieses Werkzeug, im Weiteren nur noch MDBDA Editor genannt, ist innerhalb der Eclipse Kepler Plattform[35] entwickelt worden. Eclipse ist in der Softwaretechnik weit verbreitet [36] und im Bereich Modelling auf dem Stand der Technik. Es ist ohne großen Aufwand erweiterbar, plattformunabhängig und frei unter einer Open Source Lizenz verfügbar.

5.1 Eingesetzte Technologien

Zu Beginn werden in diesem Abschnitt die eingesetzten Technologien kurz beschrieben und ihr Einsatz erläutert, um im Anschluss auf die eigentliche Implementierung des Werkzeugs einzugehen.

5.1.1 Eclipse Modeling Framework

Das Eclipse Modeling Framework (EMF)[37][38] ist ein Framework zur Erzeugung von Java-Quellcode aus Modellbeschreibungen. Der generierte Programmcode erlaubt eine einfache Serialisierung und Deserialisierung von Laufzeitinstanzen. Zusätzlich können grundlegende Baumeditoren generiert werden, welche eine Manipulation des Laufzeitmodells ermöglichen, ohne dafür Programmcode zu

schreiben. Neben der Kernfunktionalität des Frameworks, fiel die Wahl auf EMF auch aufgrund der weiten Verbreitung dieser Technologie und der großen Entwicklergemeinschaft, welche das Projekt um weitere Funktionen erweitert. Beim Quellcode Hostinganbieter GitHub[39] gibt es zu diesem Zeitpunkt alleine 179 Repositories zu den Begriffen EMF und Java[40].

5.1.2 Xcore

Xcore[41] ist eine DSL um Domänenmodelle textuell zu beschreiben. Abbildung 8 zeigt ein einfaches Beispiel in einer Bücherei Domäne. Elemente des Modells werden einfach, ähnlich wie bei Java, als *Class* bezeichnet. Selbst Vererbungen sind entsprechend möglich. Xcore Klassen können sowohl primitive, wie *int* in dem Beispiel, als auch komplexe, wie beispielsweise *String*, Datentypen haben. Referenzen zwischen den Elementen werden mit Hilfe einer Container- oder einer Referenzbeziehung abgebildet. Jedes Element, bis auf das Wurzelelement, muss genau ein Elternelement mittels Containerbeziehung haben. Dadurch entsteht bei der Betrachtung nur der Containerbeziehung ein Baum, welcher einfach als XML serialisiert werden kann. Neben diesen grundlegenden Funktionen, bietet Xcore die Möglichkeit Operationen zu definieren, welche als Methoden im späteren Generat zu finden sind. Dies ermöglicht den Modellcode einfach den individuellen Anforderungen anzupassen.

5.1.3 Graphiti

Graphiti[42][43] ist ein Framework, um Diagrameditoren ohne großen Aufwand für Eclipse zu erstellen. Das Framework wurde bei der SAP AG entwickelt, um leicht einheitliche Tools zu erstellen. Unter Anderem wurden von der SAP AG Ergonomieexperten eingesetzt, um Aussehen und Verhalten der entwickelten Tools zu optimieren. Graphiti wurde 2010 schließlich als Eclipse Open

```
*Library.xcore
package org.example.library

class Library
{
    String name
    contains Book[] books opposite library
    contains Writer[] authors opposite library
}

class Book
{
    String title
    int pages
    container Library library opposite books
    refers Writer[] authors
}

class Writer
{
    String name
    container Library library opposite authors
    refers Book[] books opposite
}
```

authors
Ctrl+Space to show shortest proposals

Abbildung 8: Xcore Beispiel[41]

Source Projekt veröffentlicht und wird dort von der Community gepflegt und weiterentwickelt. Abbildung 9 zeigt einen Klasseneditor, welcher als Beispielanwendung bei Graphiti mitgeliefert wird. Durch den Einsatz des Frameworks wird die Erstellung graphischer Editoren deutlich erleichtert, da Plattform abhängige Technologien und das sehr komplexe Graphical Editing Framework (GEF) [44] [45] verborgen werden und über ein einfaches Java Application Programming Interface (API) programmiert werden kann. Jedoch ermöglicht Graphiti dem Entwickler, im Gegensatz zur Editorgenerierung mit dem Graphical Modeling Framework (GMF)[46][47], auf einfache Art und Weise programmtechnisch Einfluss auf Aussehen und Funktionalität der Editorkomponenten zu nehmen.

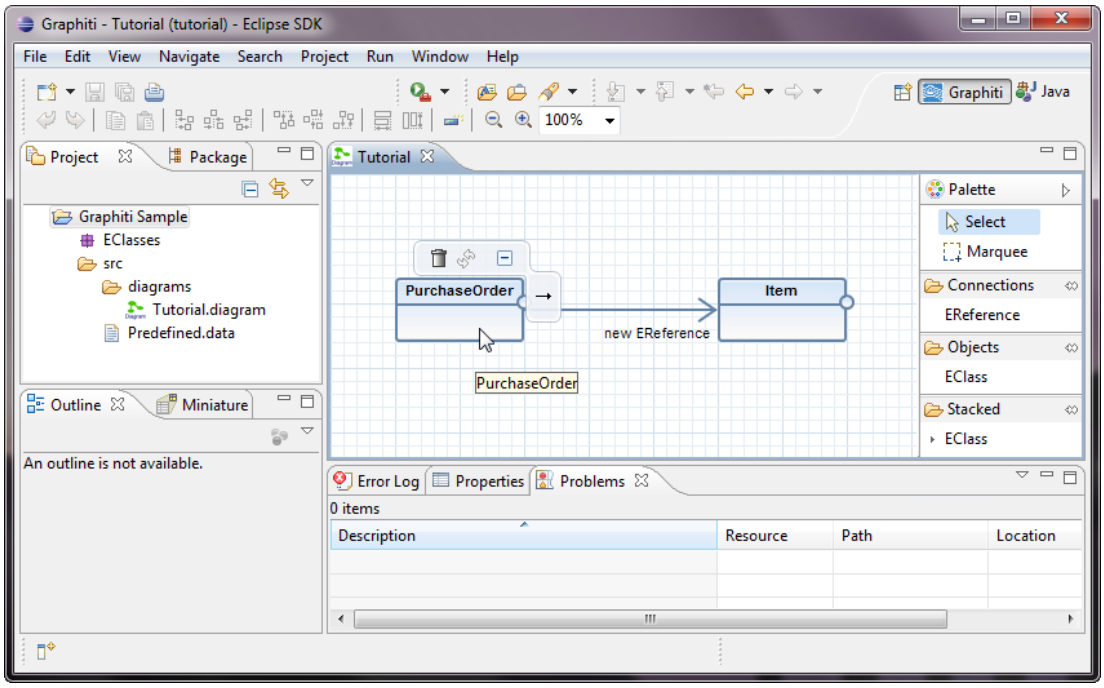

Abbildung 9: Beispiel Graphiti Editor[48]

5.1.4 Xpand Template Language

Xpand[49] ist ein Eclipse Projekt zur „Model nach Text“ Generierung mithilfe einer Templatesprache. Dies wird durch eine Workflow Engine realisiert. Xpand war früher unter openArchitectureWare[50] bekannt und ohne Eclipse lauffähig. In Listing 4 ist die beispielhafte Konfiguration einer Engine gezeigt. In der Variable *modelUri* wird die Uniform Resource Identifier (URI) der EMF Model XML festgelegt, aus der Quellcode generiert werden soll. Die Syntax der Xpand Workflow Engine Sprache erlaubt es Listen Werte mittels eines Gleichzeichens zuzuweisen, optional einem Doppelpunkt folgend zu benennen und innerhalb geschweifter Klammern zu parametrisieren. Dies ist bei Listen ein Hinzufügen zu der Liste, nicht die Neuzuweisung der Liste. Die Variablen *bean* und *component* sind Listen und werden nicht überschrieben. Die in Listing 4 Zeile 6 zu sehenden Zuweisungen bedeuten, dass der Liste *bean* eine Instanz von *XcoreStandaloneSetup* mit dem Namen *xcore* zugewiesen wird und nicht weiter parametrisiert wird. Auf die Namenszuweisung wird in Zeile 8 zurückgegriffen. Die wichtigsten

Anweisungen zur Generierung sind in Zeile 23 und 26. Erstere gibt an, dass das Metamodell aus dem Java Klassenpfad geladen werden soll. Letztere definiert das zur Generierung verwendende Template.

Xpand erlaubt es mittels so genanter *AROUND* Anweisungen aspektorientiert zu programmieren. In einer frühen Version der MDBDA Codegenerierung wurde dieses Feature genutzt um die Generierung erweiterbar zu machen. Weitere Aspekte werden einfach im Workflow angemeldet, wie in Zeile 24 zu sehen. Dies ist jedoch recht umständlich und wird, wie im nächsten Abschnitt zu sehen, nicht mehr verwendet.

```
1  module org.mdbda.codegen.generator
2  import ...
3  var modelUri = "platform:/resource/eg/SimpleFilterJoinSumWorkflow.
       mdbdamodel"
4  var dirSrcGen = "test-src-gen"
5  Workflow {
6    bean = org.eclipse.emf.ecore.xcore.XcoreStandaloneSetup : xcore { }
7    bean = org.eclipse.xtext.mwe.Reader {
8      register = xcore
9    }
10   bean = StandaloneSetup {
11     scanClassPath = true
12     platformUri = "."
13     registerGeneratedEPackage = "org.mdbda.model.ModelPackage"
14   }
15   bean = JavaBeautifier : formatter_java { }
16 /* load model and store it in slot 'model' */
17   component = Reader {
18     uri = modelUri
19     modelSlot = "model"
20   }
21 /* generate code */
22   component = Generator {
23     metaModel = org.eclipse.xtend.type.impl.java.JavaMetaModel{}
24     advices = "template::FilteringPattern::TopTenFilter"
25 ...
26     expand = "template::MDBDATemplate::main FOR model"
27       outlet = {
28         path = dirSrcGen
29       }
30   }
31 }
```

Listing 4: Modeling Workflow Engine

Listing 5 zeigt ein einfaches Xpand Beispiel, bei dem für ein sogenanntes MDBDA-

Diagram Modelobjekt eine neue Datei mit dem Name “JobControl.java“ angelegt wird. Innerhalb der Methode *main(..)* wird für den Rootworkflow des Diagramms, siehe Zeile 14, eine Liste mit beinhalteten Pattern generiert.

```
1  «IMPORT org::mdbda::model»
2  «EXTENSION MDBDAWorkflowHelper»
3
4  «DEFINE main FOR MDBDADiagram»
5    «FILE "JobControl.java"»
6    /**
7      * name = «this.name»
8      * author = «this.author»
9      * version = «this.version»
10     */
11   public static void main(String... args){
12     Configuration conf = new Configuration();
13     //...
14     «EXPAND workflow FOR this.rootWorkflow»
15   }
16   «ENDFILE»
17 «ENDDEFINE»
18
19 «DEFINE workflow FOR Workflow»
20   «FOREACH this.pattern AS p»
21     «EXPAND pattern FOR p»
22   «ENDFOREACH»
23 «ENDDEFINE»
24
25 «DEFINE pattern FOR Workflow»
26   //workflow «this.name» Pattern
27   «FOREACH this.pattern AS p»
28     «IF !containsPattern(p)»
29       //funny stuff
30     «ENDIF»
31   «ENDFOREACH»
32 «ENDDEFINE»
```

Listing 5: Xpand Template

5.1.5 Xtend Code Generator

Xtend beschreibt sich selber mit „JAVA 10, Today!“[51]. Es ist eine Precompile Spracherweiterung, die es ermöglicht, moderne, nicht in Standard Java enthaltene Sprachkonstrukte zu verwenden. Der daraus entstehende Sourcecode ist

Java kompatibel und wird vom Standard Java Compiler in Java Virtual Machine (JVM) Bytecode übersetzt. Listing 6 zeigt exemplarisch die Spracherweiterung der Methodenerweiterung. Die Methodenerweiterung erlaubt es bestehende Typen um Funktionalität zu erweitern, ohne diese zu ändern. In dem Listing ist beispielsweise der Typ String um die Methode toMyAwesomeString(..) erweitert worden. Diese Methode kann beliebig auf Strings aufgerufen werden. Der Precompiler erstellt daraus eine statische Java Methode, welche als Parameter die selben wie die Xtend Methode, in diesem Fall der *input* String, enthält. Der Aufruf in Zeile 4 wird schließlich von *foo.toMyAwesomeString* nach *___toMyAwesomeString(foo);* übersetzt.

```
1  class Foo {
2   def StringExample(){
3     val foo = new String("Xtend")
4     print(foo.toMyAwesomeString)
5   }
6
7   def toMyAwesomeString(String input){
8     'this is awesome: ' + input;
9   }
10 }
```

Listing 6: Xtend Extension Methods

Um Xtend als Codegenerator nutzen zu können, bringt es die Funktion der Template Expressions mit. Innerhalb einer Template Expression kann, in Xpand Template Language ähnlicher Syntax, Quellcode erzeugt werden. Die Xpand Sprache wird somit um die Mächtigkeit einer modernen Programmiersprache erweitert. Listing 7 zeigt eine solche Template Expression und wie sie auf ein Objekt vom Typ *MDBDADiagram* angewendet wird.

```
1 def CharSequence genJobConfiguration(MDBDADiagram diagram){
2   '''
3   class «diagram.name»JobConfiguration{
4     /**
5       * name = «diagram.name»
6       * author = «diagram.author»
7       * version = «diagram.version»
```

```
8      */
9     public static void main(String... args){
10        Configuration conf = new Configuration();
11        //...
12        «FOR pattern : diagram.rootWorkflow.pattern»
13          //«pattern.name»
14        «ENDFOR»
15    }
16  }
17  '''
18 }
```

Listing 7: Xtend Template Expressions

5.2 MDBDA Domänenmodell

Bevor das eigentliche Domänenmodell beschrieben wird, müssen erst die Begrifflichkeiten und Ebenen der Metamodelle festgelegt werden, um eine einheitliche Sprache diesbezüglich zu finden. Abbildung 10 schlüsselt die Ebenen, dessen Bezeichnung in dieser Arbeit und ihren Zusammenhang auf. Um ein langfristig erfolgreiches Projekt zu gewährleisten, muss das System schon bei der Entwicklung auf sich verändernde Anforderungen vorbereitet werden, um später kostenintensive Architekturanpassungen zu vermeiden. Das Domänenmodell des MDBDA Editors muss daher möglichst erweiterbar und flexibel sein. Aus diesem Grund besteht das Modell aus einem Kern und diversen Erweiterungen dieses Modells. Listing 8 zeigt das Kerndomänenmodell, welches mit der Xcore DSL erstellt wurde. Das Wurzelelement des Modells ist immer ein *MDBDADiagram* (siehe Zeile 6). Dieses beinhaltet genau eine Workflow Modellinstanz mit dem Name *rootWorkflow*, eine Liste benötigter Ressourcen und im Diagramm verwendete Datenformatkonfigurationen. Eine Ressource (siehe Zeile 30) kann dabei ein Speichersystem aus der MDBDA Plattform, ein generischer Platzhalter oder ein Analysepattern sein. Speichersysteme aus der MDBDA Plattform können dabei in der aktuellen Version Cassandra, HDFS, Neo4j und Hazelcast sein. Generische Platzhalter sind für eine nachträgliche Konfiguration der Komponente zur Laufzeit ausgelegt

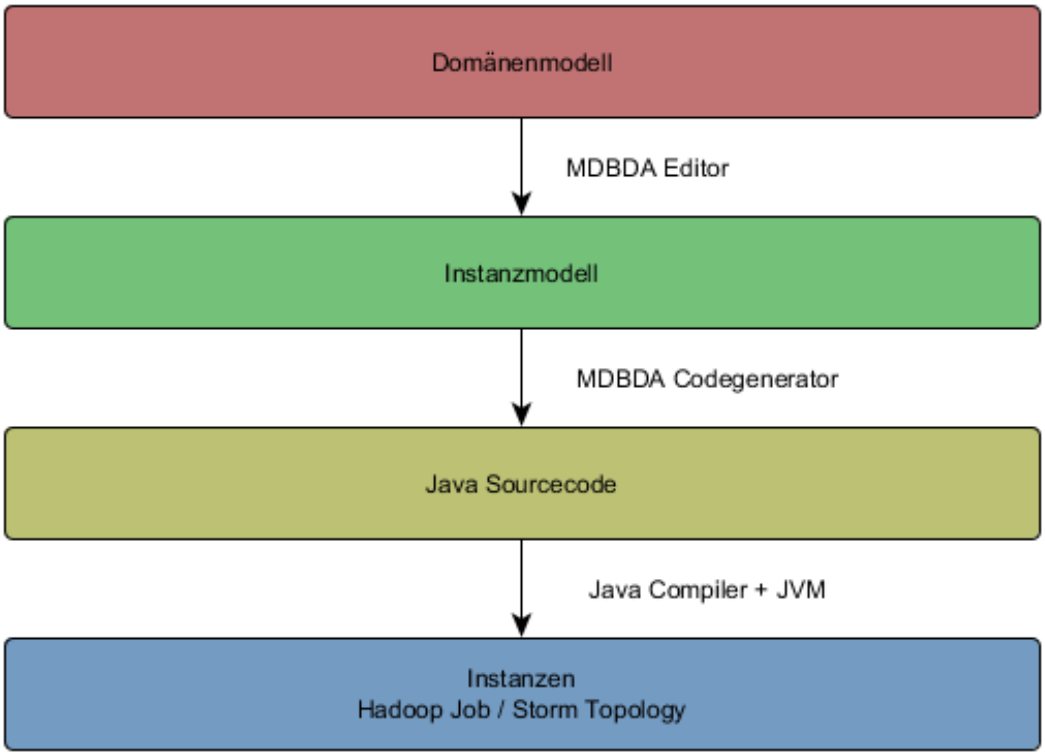

Abbildung 10: Metamodell Ebenen

und ermöglichen eine Wiederverwendbarkeit von Workflows. Analysepattern (siehe Zeile 25) sind schließlich verarbeitende Elemente. Jede Ressource referenziert eine Eingabe- und eine Ausgabekonfiguration, sowie abhängige Ressourcen. Zur Validierung der Kompatibilität zusammengefügter Verarbeitungseinheiten, hier Analysepattern genannt, dienen die Datenformatkonfigurationen (siehe Zeile 40). Neben der eigentlichen Validierung dienen die Konfigurationen bereits bei der Entwicklung neuer Workflows als Hilfe, da ein Analysepattern stets eine Transformation eines Eingabeformats in ein Ausgabeformat durchführt. Im einfachsten Fall kann das System die Ausgabekonfiguration einer Quellressource eigenständig ermitteln, sowie alle Zwischenkonfigurationen durch Transformationen. Im Falle einer Inkompatibilität kann der Entwickler darauf hingewiesen werden. Workflows (siehe Zeile 17) beinhalten Pattern sowie die verwendeten Ressourcen. Aufgrund der Vererbungshierarchie verhält sich ein Workflow wie ein Pattern und ein Pattern wie eine Ressource.

```
@GenModel(...)
package org.mdbda.model

import org.eclipse.emf.common.util.EList

class MDBDADiagram{
  contains Workflow rootWorkflow opposite diagram
  contains Dataformat[] dataformates opposite diagram

  contains Resource[] resources

  String author
  String version
  String name
}

class Workflow extends Pattern{
  contains Pattern[] pattern opposite workflow
  container MDBDADiagram diagram opposite rootWorkflow
  contains Resource[] dataResources
  refers Pattern startPattern
  refers Workflow externalWorkflow
}

class Pattern extends Resource {
  container Workflow workflow opposite pattern
  refers Pattern[] dependsOn
}

class Resource{
  refers Resource[] inputResources
  refers Resource[] outputResources
  String configuration
  String typeId
  String name
  refers Dataformat outputFormat
  refers Dataformat inputFormat
}

class Dataformat{
  container MDBDADiagram diagram opposite dataformates
  String name
  String keyTypeClass
  String valueTypeClass
  String testDataGeneratorClass
}
```

Listing 8: MDBDA Core Domain Model

Im erweiterten Domänenmodell werden Pattern und Ressourcen, durch MDB-DA Editor Plug-ins, zur Instanzmodell Laufzeit erweitert. Ein Beispiel für ein solches Element aus dem erweitertem Modell ist der SimpleMatcher. Dieses Pattern, aus der Kategorie der Filter Pattern, wird verwendet um einfache reguläre Ausdrücke auf die Eingabedaten anzuwenden. Listing 9 zeigt die Factory des SimpleMatchers. Das erzeugte Instanzmodellobjekt ist ein einfaches Pattern aus dem Kerndomänenmodell und unterscheidet sich durch eine andere TypeId (siehe Zeile 16). Dies ermöglicht, dass das Tool während der Laufzeit erweitern werden kann, ohne dass das gesamte Domänenmodell neu generiert werden muss.

```
1  public class CreateSimpleMatcherFeature extends CreatePatternFeature
       implements
2    ICreateFeature {
3   public static String name  = "SimpleMatcher";
4   public static String description =  "Creates a new simple matcher filter
5             pattern";
6
7
8   public CreateSimpleMatcherFeature(IFeatureProvider fp) {
9     super(fp, name, description);
10  }
11
12  @Override
13  public Object[] create(ICreateContext context) {
14
15    Pattern eInst = ModelFactory.eINSTANCE.createPattern();
16    eInst.setTypeId(FilteringPatternTemplateConstatns.SimpleMatcherFilter);
17
18    addToWorkflow(getWorkflow(context), eInst);
19    addGraphicalRepresentation(context, eInst);
20    return new Object[] { eInst };
21  }
22 }
```

Listing 9: SimpleMatcher Factory

5.3 Graphical Editor

Ein wichtiges Element, wenn die Erstellung von Big Data Analysen möglichst intuitiv geschehen soll, ist die Bereitstellung einer guten Entwicklungsumgebung,

welche den Analyseautor optimal bei der Entwicklung unterstützt. In dieser Arbeit wurde dazu ein graphischer Editor mit dem Graphiti Frameworks erstellt (Abschnitt 5.1.3). Abbildung 11 zeigt den erstellten Editor. Mithilfe des Editors wurde eine Beispielanwendung entwickelt, welche zwei Ressourcen, eine Cassandra und eine Neo4j Ressource, erst mittels eines „ReduceSideJoin“ verbindet, diese Zwischenergebnisse numerische addiert, die Summen sortiert und nach „Top 10“ filtert. Das Endergebnis wird schließlich in eine Cassandra Ressource abgespeichert.

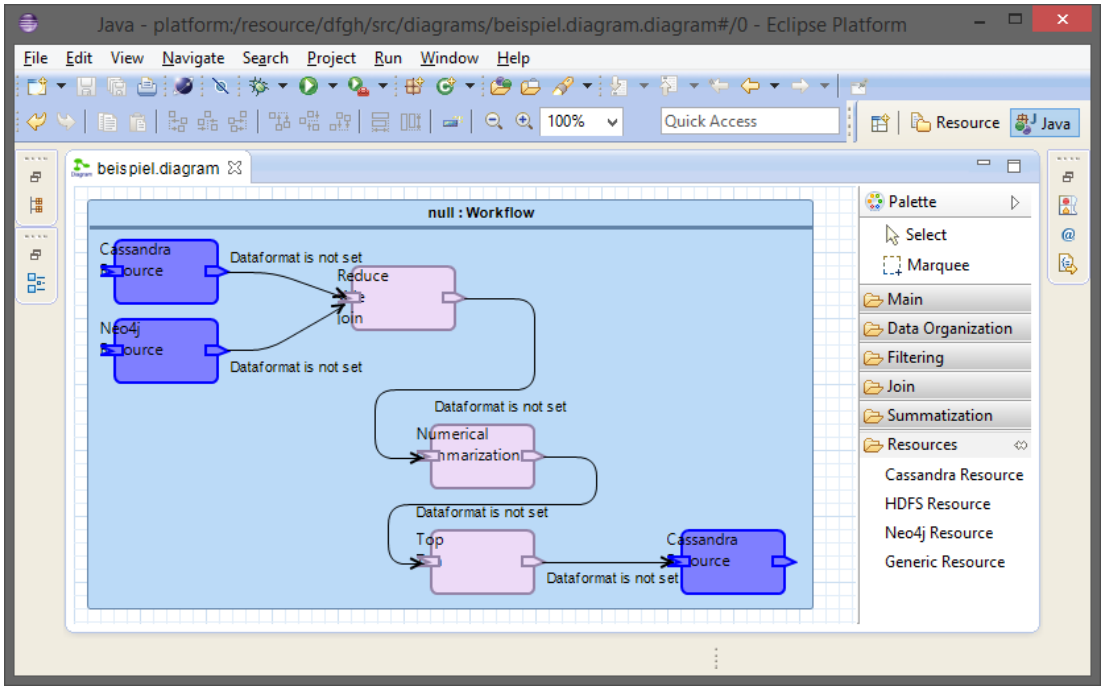

Abbildung 11: MDBDA Editor

Die Funktionsweise des Editors ist möglichst intuitiv gestaltet. Auf der rechten Seite ist stets eine Palette mit allen verfügbaren Analysepattern vorhanden. Die Analysemuster sind in den Kategorien Datenorganisation, Filter, Kombinieren, Berechnen und Ressourcen eingeteilt. In der in dieser Arbeit erstellten Version des Editors sind einige Analysemuster des Buches „MapReduce Design Patterns“ [52], sowie die verschiedene Ressourcen der MDBDA Plattform und eigene Analysemuster enthalten, welche alle im Abschnitt 5.5 (MDBDA Pattern) gesondert beschrieben werden. Mithilfe des MDBDA Editors werden MDBDA Diagram-

me erstellt. Diese werden in zwei Dateien abgespeichert. Zum Einen wird eine „.diagram“ Datei, welche Informationen zur Darstellung im graphischen Editor beinhaltet, abgespeichert. Zum Anderen eine Datei mit der Endung „.mdbdamodel“, welche das Instanzmodell des Domänenmodells als EMF kompatibles XML beinhaltet. Jedes MDBDA Diagramm beinhalten genau ein Workflow. Ein wichtiges Feature des Editors ist die Wiederverwendbarkeit von Workflows. Ein vorhandener Workflow kann dabei wie ein Analysepattern innerhalb eines Workflows eingebunden werden. Input- und Outputressourcen vom Typ *Generic* werden dabei nach außen weitergegeben. Somit können komplexe Anwendungsteile und Unternehmensworkflows wiederverwendet werden. Analysepattern besitzen auf der linken Seite Anker für eingehende, beziehungsweise an der rechten Seite Anker für ausgehende Daten. Der Datenfluss wird über Verbindungen dieser Anker abgebildet. Eine Verbindung ist dabei stets gerichtet, was mithilfe eines Pfeils visualisiert wird. Konfiguriert werden die Analysepattern, nach Selektion durch den Anwender, über eine Eclipse Properties View. In der aktuellen Version des MDBDA Editors wird der Ressource Konfigurationsstring (siehe Listing 8 Zeile 33) über einfaches JavaScript Object Notation (JSON)[53] realisiert.

Der Editor ist, wie bereits das MDBDA Domänenmodell, möglichst modular aufgebaut, um ihn einfach durch Plug-ins erweitern zu können. Ein solches Plug-in muss dabei mindestens folgende Klassen der MDBDA Editor API implementieren: CreatePatternFeature und AddPatternFeature. Bei der *CreatePatternFeature* Klasse muss die *„create(..)“* Methode implementiert werden (siehe Listing 9). In ihr wird eine Modellinstanz eines Analysepattern aus dem MDBDA Domänenmodell, mit einer zusätzlich gesetzten eindeutigen *TypeId*, erstellt. Die Typisierung mithilfe eines Identifikationsstring anstelle der Erweiterung des MDBDA Domänenmodell wurde gewählt, um eine möglichst dynamische und einfache Erweiterbarkeit des Editors zu ermöglichen. Somit ist es auch möglich, unbekannte Analysepattern darzustellen, falls ein Plug-in nicht installiert wurde. Beim AddPatternFeature kann die *„add(..)“* Methode erweitert werden. Sie hat als Rückgabewert ein PictogramElement, ein Interface aus der Graphiti API,

welches die Darstellung des Musters repräsentiert. Das Resultat der Standardimplementierung ist in Abbildung 11 bei „Numerical Summarization“ und „Top 10“ zu sehen.

5.4 Codegenerierung

Mithilfe des zuvor beschriebenen Editors erstellten Instanzmodells, wird mit einem Codegenerator der Java Sourcecode für die eigentliche Ausführung der Algorithmen erzeugt. Der Generierungsprozess wird dabei über ein Kontextmenü in Eclipse beim Rechtsklick auf eine Model Datei aufgerufen. Im Prozess werden zuerst die benötigten Modelle mit Hilfe von EMF eingelesen, um rekursiv für jedes Analysepattern ein MapRecude-Job und eine Storm Topologie zu erstellen. Im Falle von MapReduce werden die erstellten Analysepattern-Jobs mittels eines Jobkontrollers zu Workflow-Jobs orchestriert. Bei der Zusammenstellung des Storm Workflows werden die generierten Bolts einfach zu neuen Topologienen verknüpft. Die erste Version des Codegenerators wurde mithilfe der Xpand Template Sprache (siehe Abschnitt 5.1.4) erstellt. Im Laufe der Entwicklungszeit hat sich jedoch gezeigt, dass sich Xpand nicht gut geeignet ist um einfach zu erweiternde Codegeneratoren zu erstellen. Zum Einen ist die Konfigutation mittels Modeling Workflow Engine 2 (Mwe2) komplex und fehleranfällig, zum Anderen lassen sich die Templates schlecht debuggen. Da der Editor über Plug-ins erweiterbar sein soll, muss auch der Generator entsprechend anpassbar sein. Schlussendlich setzte sich Xtend (siehe Abschnitt 5.1.5) als Technologiewahl durch, da unter anderem zuvor die mit Xpand erstellte Templates mit geringen Zeitaufwand portiert werden konnten. Um den Codegenerator zu erweitern bietet die MDBDA Codegen API die Klasse MDBDACodegenerator und das Interface IPatternTemplate. Beim Interface müssen die Methoden „generateStormBolt(..)“ und „generateMapReducePattern(..)“ implementiert werden. Als Übergabeparameter wird ein Pattern Objekt übergeben. Sie liefern schließlich jeweils einen String zurück welcher Inhalt einer eigenen Java Datei sein wird. Die Auswahl des richtigen Codegenerator Plug-

ins geschieht mithilfe der *TypeId*, welche im Instanzmodell gesetzt wurde. Die Codegenerator Plug-ins müssen sich dazu mit den *TypeIds*, die sie bedienen, bei der statischen Methode *MDBDACodegenerator.addPatternTemplat(String patternId, IPatternTemplate template)* anmelden.

5.5 MDBDA Pattern

Ein MDBDA Pattern oder auch Muster transformiert eingehende in ausgehende Daten. Dabei kann sich das Datenformat ändern. Bei MDBDA wird grob, wie in Formel 1 zu sehen, zwischen vier Datenformaten unterschieden.

$$\begin{array}{rcl} \text{spaltenorientierte} & \subseteq & \text{zeilenorientierte} \\ & \subseteq & \text{hierarchische} \\ & \subset & \text{unstrukturierte} \\ \text{strukturierte Daten} & \longleftarrow & \text{unstrukturierte Daten} \end{array}$$

Formel 1: Strukturierung von Daten

Spaltenorientierte Daten sind zweidimensionale Matrizen, beziehungsweise Tabellen. Ein Beispiel für diese Daten sind Comma-separated values (CSV) Dateien in einem HDFS. Zeilenorientierte Daten sind wie Tabellen, in welchen in jeder Zeile nicht jede Spalte vorhanden ist. Zusätzlich sind Spalten mit Bezeichnern versehen. Somit verhält sich eine Zeile wie eine Liste von Key-Value Paaren. Zeilenorientierte Daten lassen sich zu spaltenorientierten Daten umwandeln indem fehlende Felder einfach leer beziehungsweise mit einem Null-Wert befüllt werden. Hierarchische Daten fügen der flachen Struktur der vorherigen Datenformaten eine Tiefe hinzu. Dadurch lassen sich Baumstrukturen, wie bei XML Dateien darstellen. Hierarchische Daten lassen sich in eine zeilenorientierte Form überführen, indem die Hierarchieinformation in jede Zeile mit übernommen wird. Zu den unstrukturierte Daten zählen alle Datenformate. Es wird keine Einschränkung an Struktur und Format gegeben.

5.5.1 Datenorganisation

Da es sich bei den zu analysierenden Daten um sehr heterogene Daten handeln kann, muss die Grundfunktionalität des MDBDA Editors einige Möglichkeiten zur Neuorganisation dieser Daten bieten, um sie in weiteren Schritten besser verarbeiten zu können.

5.5.1.1 Partitioning

Das Partitioning Muster wird verwendet, um Daten in Kategorien einzuordnen. Beispielsweise werden Messdaten, wie Wert kleiner oder gleich 23, zwischen 23 und 42, sowie größer gleich 42 gruppiert. Dieses Beispiel liefert drei Gruppen von Messwerten. Das Muster lässt sich prinzipiell auf hierarchische und stärker strukturierte Daten anwenden. Die Abbildungen 12 und 13 zeigen jeweils eine schematische Struktur dieser Muster für Hadoop MapReduce und Storm. Eine

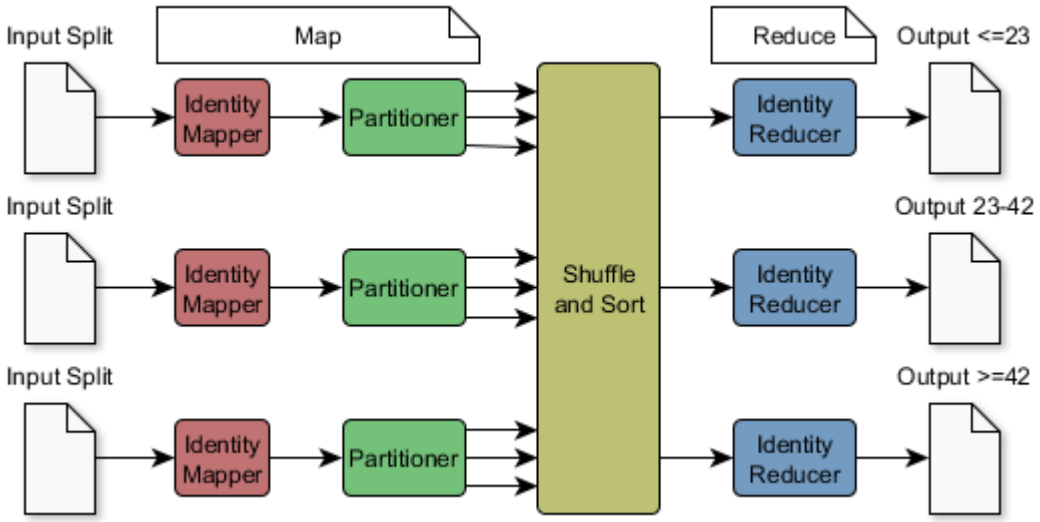

Abbildung 12: Struktur des Partitioning Pattern in MapReduce

beispielhafte Verwendung dieses Musters wäre in einem Onlineshop Szenario, bei dem das Kaufverhalten von Kunden gegenüber verschiedener Warengruppen untersucht werden soll. Das Pattern würde zur Vorsortierung dienen. Dabei können

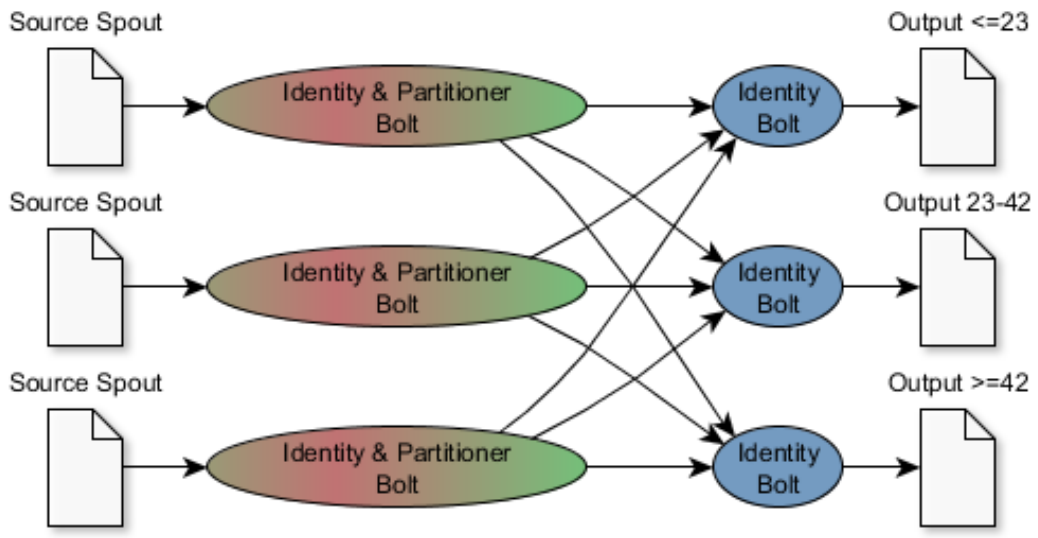

Abbildung 13: Struktur des Partitioning Pattern in Storm

bereits uninteressante Produktgruppen verworfen werden und so die Laufzeit der Analyse reduziert werden.

5.5.1.2 Shuffling

Teilweise ist es für Analysen notwendig, dass Daten unsortiert in einen Algorithmus eingelesen werden müssen, um das Ergebnis nicht zu verfälschen, jedoch aufgrund anwendungsspezifischer Anforderungen beispielsweise zeitlich sortiert oder nach fortlaufender Nummer gespeichert wurden. Das Shuffling Pattern ist ein Organisationsmuster, welches Datensätze nach einer zufälligen Reihenfolge sortiert. Sinnvoll ist das Muster ab zeilenorientierten Daten, kann prinzipiell jedoch auch bei hierarchischen Daten eingesetzt werden.

Es ist unter anderem sehr hilfreich, um Datensätze, wie Nutzerkommentare, zu anonymisieren und so strengen Datenschutzbestimmungen zu begegnen.

5.5.1.3 Structured to Hierarchical

Die Bereitstellung hierarchischer Datenformate wie XML oder JSON bei Webservices ist Grundvoraussetzung für einen guten Service, da sich diese Datenformate deutlich einfacher verwenden und verständlicher dokumentieren lassen. Das Structured-to-Hierarchical-Pattern bietet dafür die Funktionalität, zeilenorientierte oder spaltenorientierte Datensätze in hierarchische Formate zu übersetzen. Dazu werden zwei oder mehrere Datensätze miteinander verknüpft. Abbildung 14

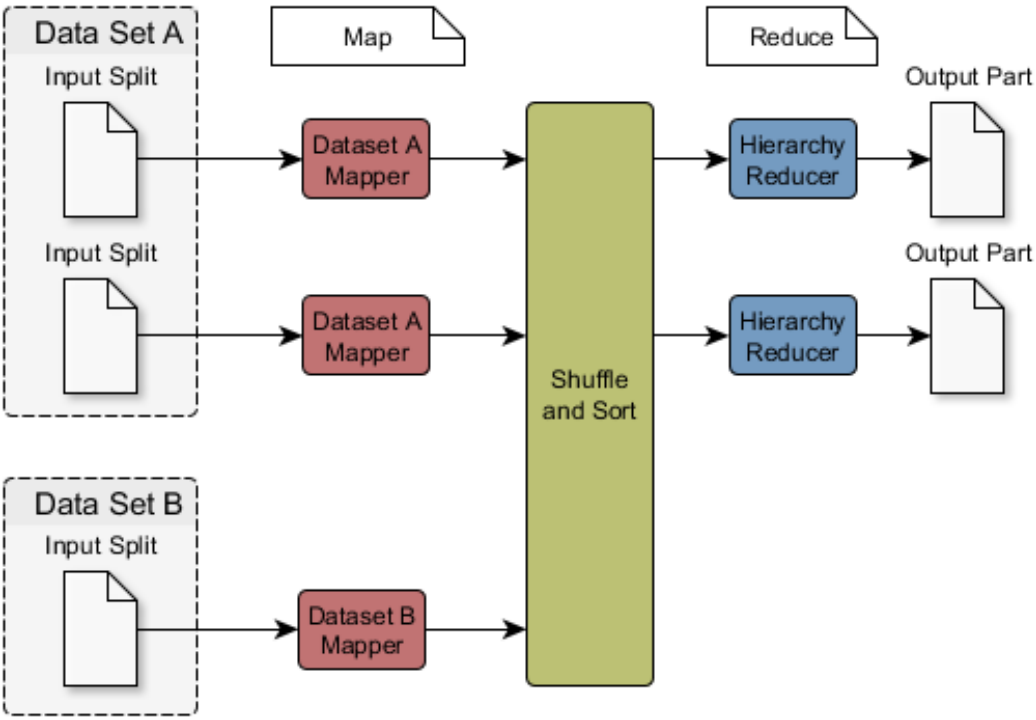

Abbildung 14: Struktur des Structured-to-Hierarchical Pattern in MapReduce

zeigt die schematische Darstellung des Musters für MapReduce. In einer Datenstromumgebung wie Storm macht dieses Pattern wenig Sinn, da ein Datenstrom kein definiertes Ende hat und damit keine hierarchisches Datenformat abgeschlossen werden kann. Daher wird es in dieser Arbeit nicht umgesetzt.

5.5.1.4 Total Order Sorting

Das Total Order Sorting Pattern ist ein wichtiges Muster um Daten verteilt zu sortieren und das exakte Gegenteil des Shuffling Musters. Abbildung 15 zeigt die Struktur des Musters für Hadoop MapReduce. Die Umsetzung für Apache Storm wäre trivial wie beim Partitioning Pattern. Das Ergebnis der Storm Lösung kann jedoch keine totale Ordnung über den gesamten Zeitraum liefern, sondern ist immer auf einen Zeitslot beschränkt. Daher wird die Storm Umsetzung in dieser Arbeit nicht umgesetzt.

Der Sort Mapper setzt bei jedem Datensatz den Wert als Key, nach dem sortiert werden soll. Der restliche Datensatz wird als Value weitergeleitet. Der TotalOrderPartitioner ist Bestandteil der Hadoop-API und partitioniert die Mapper Ausgaben vorsortiert in Gruppen für die Reducer. Während der Shuffle-and-Sort-Phase werden diese Gruppen jeweils zusammengefügt und sortiert. Der Reducer hat nur noch die Aufgabe die Ausgabe zu schreiben.

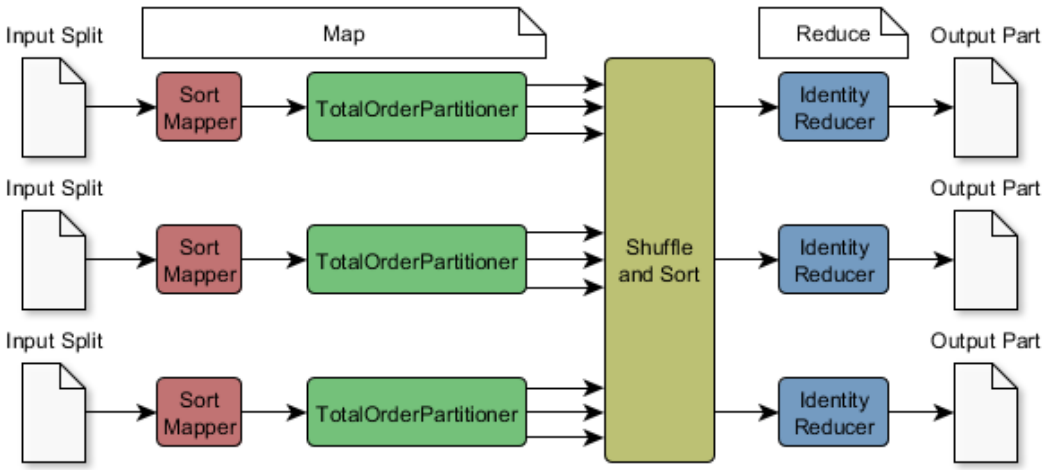

Abbildung 15: Struktur des Total Order Pattern in MapReduce

5.5.2 Filter

Die Laufzeit von Big Data Analysen kann man ohne großen Aufwand optimieren, indem man den Grundsatz "Filter Early and Often"[54] befolgt. Dazu dienen bei MDBDA die folgenden Muster.

5.5.2.1 Top Ten

Der Top Ten Filter ist ein einfaches Muster, um verteilt eine Sortierung und Filterung vorzunehmen. Dazu wird in der Map-Phase eine lokale Top Ten Liste erstellt, welche in der Reduce-Phase zu einer globalen Top Ten vereinigt wird. Dazu benötigt das Muster strukturierte Eingabedaten, wie mindestens hierarchische Daten. Abbildung 16 verdeutlicht noch einmal die Struktur bei MapReduce. Bei Apache Storm macht dieses Muster wenig Sinn, da in einem Stream kein Anfang und Ende definiert ist und damit keine endgültige Top Ten Liste erstellt werden kann. Möglich wäre höchstens einen Zeitslot zu betrachten. Dies wird in dieser Arbeit jedoch nicht weiter behandelt.

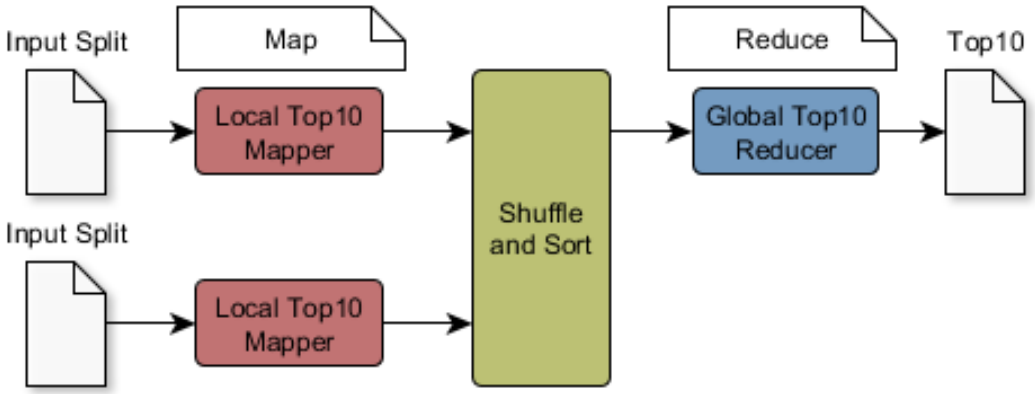

Abbildung 16: Struktur des Top Ten Musters in MapReduce

5.5.2.2 Distinct

Das Distinct Pattern wird dafür verwendet, sehr ähnliche oder gleiche Datensätze zu finden. Dazu bedient sich das Muster einer Eigenheit von MapReduce, Keys zu gruppieren und doppelte Werte zu entfernen. Dazu wird in der Map-Phase der gesamte Datensatz als Key verwendet und dem Value ein Null-Wert zugewiesen. In der Reduce-Phase wird der Datensatz nur noch geschrieben oder weitergeleitet. Das Muster kann prinzipiell mit unstrukturierten Daten arbeiten, wenn die Eingabedaten in eindeutige Teildaten, welche verglichen werden sollen, zerlegt werden können.

5.5.2.3 Simple Matcher

Der Simple Matcher ist einer der Muster in dieser Arbeit, welcher nicht aus dem Pattern Buch[52] abgeleitet ist. Bei dem Matcher wird in der Map-Phase ein regulärer Ausdruck auf jeden Datensatz angewendet und dieser bei Nichterfüllung verworfen. Die Reduce-Phase ist entsprechend der des Distinct Analysemusters. Die Eingabedaten können ebenfalls unstrukturiert sein.

5.5.3 Kombination

Bei der Analyse großer Datenmengen wird oft nach Korrelationen zwischen Daten gesucht. Dazu müssen eine Vielzahl verschiedener Datenquellen miteinander kombiniert und untersucht werden. Um diese Aufgabe zu erfüllen, bietet MDBDA folgende Muster.

5.5.3.1 Cartesian Product

Dieses Muster verbindet zwei Eingabequellen in mindestens zeilenorientiert strukturierter oder strukturierterter Form (siehe Formel 1) mit einem kartesi-

schen Produkt. Jedes Element der ersten Quelle wird mit jedem Element der zweiten Quelle verknüpft. Dies führt zu einem exponentiellen Datenwachstum. Die anfallenden Datenmengen können zu Performanceproblemen führen. Dieses Muster benötigt ebenfalls den Zugriff auf eine Datenhistorie, um sinnvolle Ergebnisse zu liefern. Daraus resultierend gibt es für Storm in dieser Arbeit auch für dieses Analysepattern keine Umsetzung.

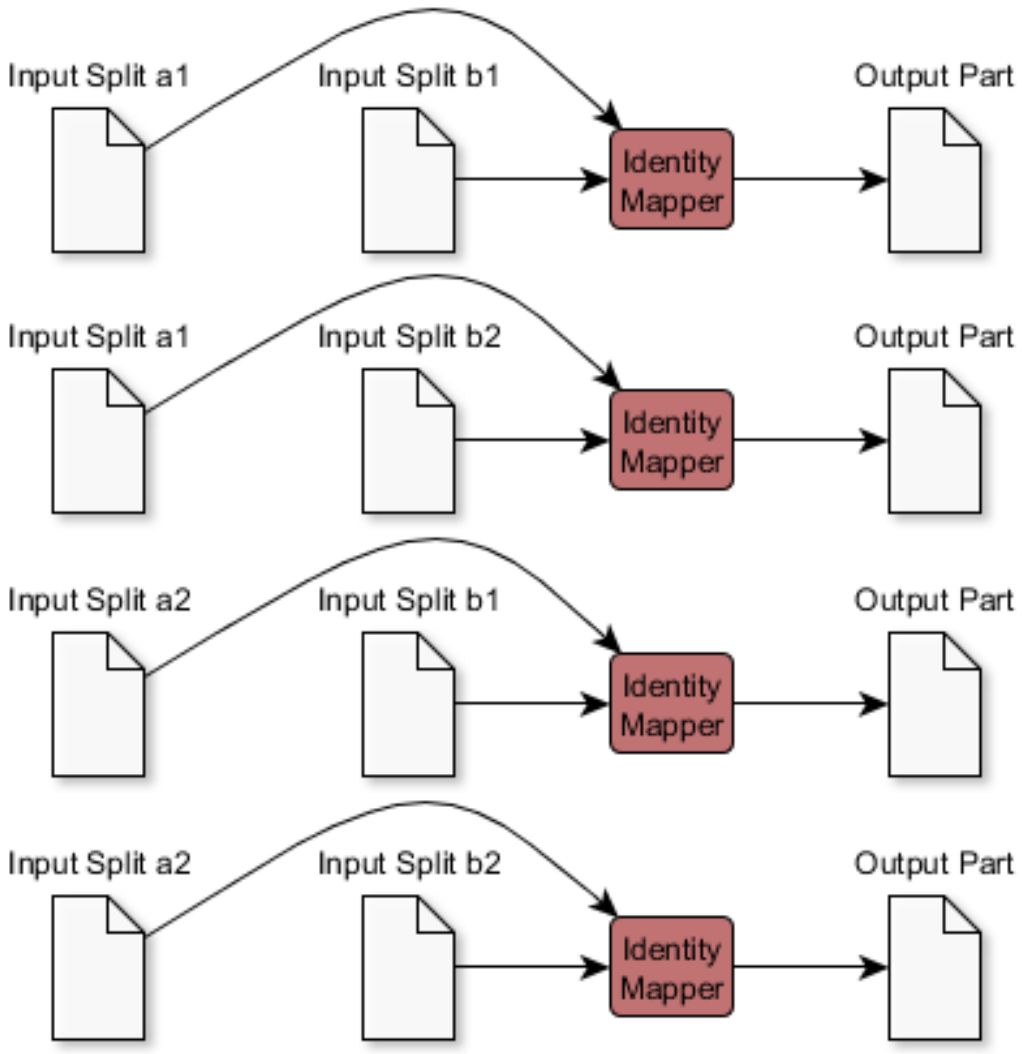

Abbildung 17: Struktur des Cartesian Product Pattern in MapReduce

5.5.3.2 Reduce-side-Join

Das Reduce-side-Join Pattern ist ein mächtiges Muster um große strukturierte, wie mindestens hierarchische, Datensätze miteinander zu verbinden. Die Abbildungen 18 und 19 zeigen jeweils die Struktur des Musters für Apache Hadoop und Apache Storm. Im Join-Mapper werden die Daten zu einem Verknüpfungswert, bei Datenbanken als Primär- oder Fremdschlüssel bekannt, gruppiert. Bei MapReduce dient dieser Verknüpfungswert als Key und die restlichen Daten als Value. Dadurch übernimmt die automatische Shuffle-and-Sort-Phase den größten Teil der Verknüpfungsarbeit. Bei der Storm Topologie dienen die Verknüpfungswerte als Routinginformationen, um die Daten dem richtigen Reducer Bolt zuzusenden. Der Join Reducer dient schließlich dazu die eigentliche Verknüpfung vorzunehmen. Die Daten kommen vorsortiert beim richtigen Reducer an und müssen nur noch verbunden werden.

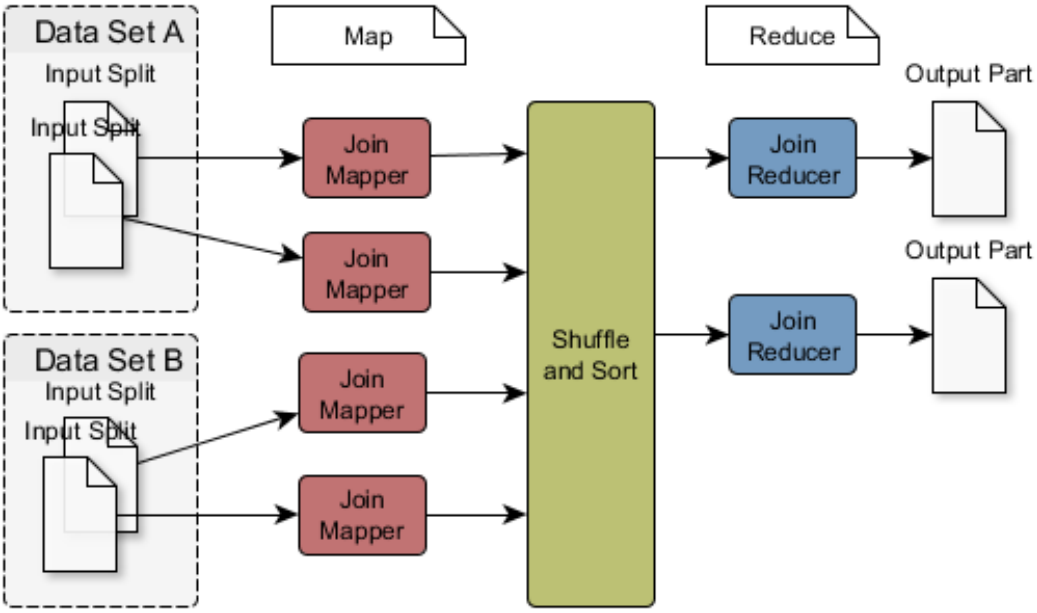

Abbildung 18: Struktur des Reduce Side Join Pattern in MapReduce

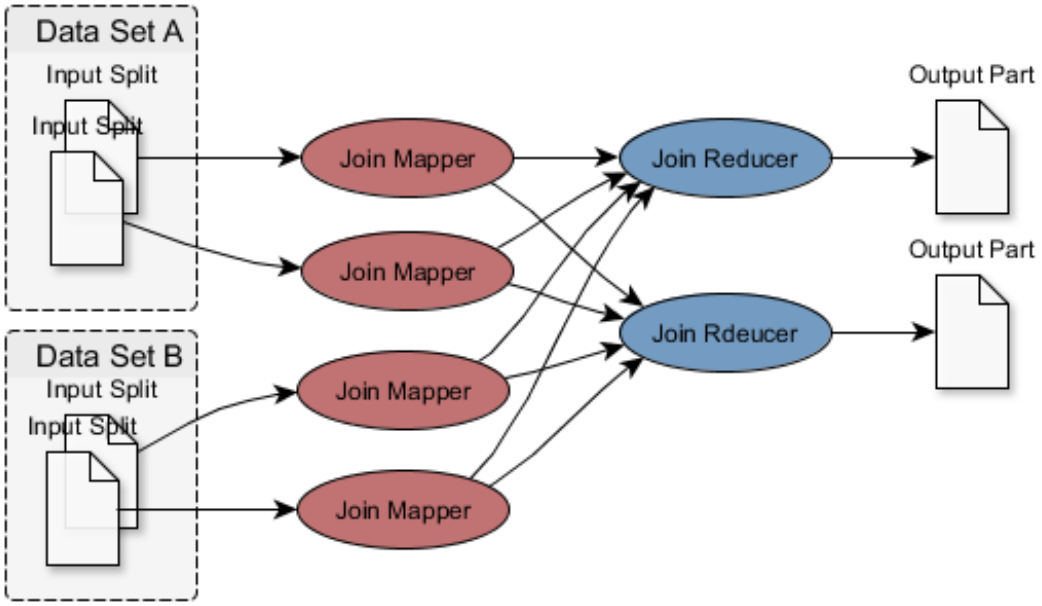

Abbildung 19: Struktur des Reduce Side Join Pattern in Storm

5.5.3.3 Replicated Join

Das Replicated-Join Pattern unterscheidet sich ein wenig vom Reduce-side-Join Muster. Einer der Eingabedatensätze wird dabei komplett zu jedem Mapper kopiert. Daher macht das Muster vor allem Sinn, wenn einer der Datensätze besonders klein ist. Abbildung 20 zeigt die Struktur des Pattern. Solange der zu kopierende Datensatz in den Hauptspeicher passt, kann das Muster für die Storm Umgebung wie das Reduce-side-Join Pattern umgesetzt werden.

5.5.4 Berechnung

Nachdem die Daten organisiert, gefiltert und verknüpft wurden, sollen die nächsten Muster bei der Berechnung helfen.

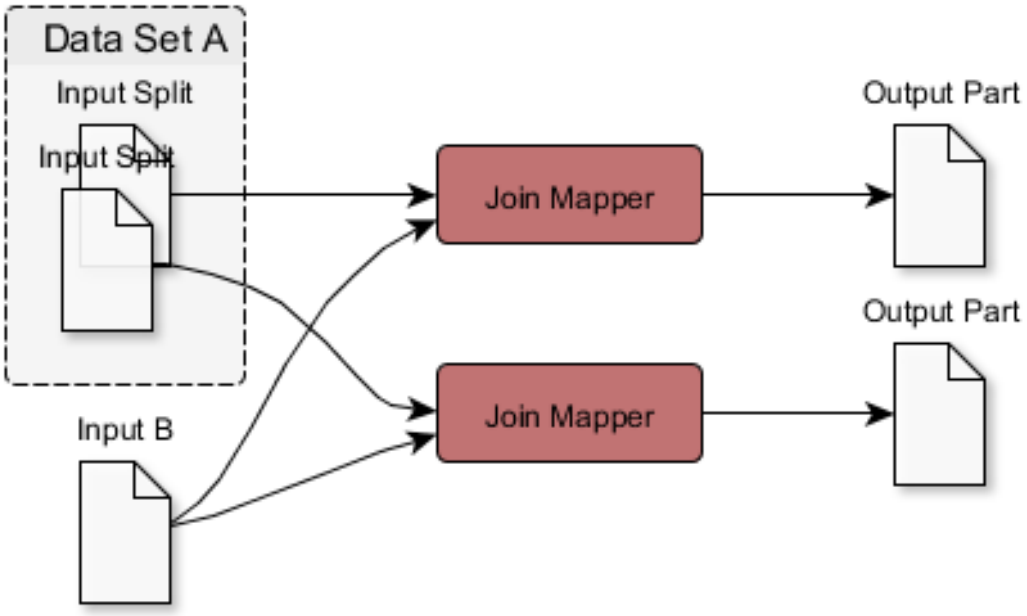

Abbildung 20: Struktur des Replicated Join Pattern

5.5.4.1 Numerical summarization

Das Nummerical-summarization-Pattern ist ein Muster, um einfache Berechnungen auf großen Datenmengen durchzuführen. Die Eingabedaten müssen strukturiert, mindestens zeilenorientiert, vorliegen. Die Berechnungsfunktion muss dabei assoziativ und kommutativ sein. Die Ausführungsreihenfolge und Verkettungsreihenfolge müssen somit beliebig sein. Ist dies der Fall, kann die Berechnung mit einer Map-Combine-Reduce Struktur, wie in Abbildung 21 zu sehen, durchgeführt werden. Der Mapper gruppiert die lokalen Daten nach Schlüssel und legt die zu berechnenden Werte fest. Der Combiner berechnet das lokale Ergebnis mit derselben Funktion wie der Reducer. Es handelt sich somit um einen lokalen Reducer. Der eigentliche Reducer verknüpft schließlich die lokalen Zwischenergebnisse zum Endergebnis. Durch den Combine Schritt wird eine Menge an zu übertragender Daten gespart, dieser setzt jedoch die zuvor beschriebenen Eigenschaften an die Berechnungsfunktion voraus.

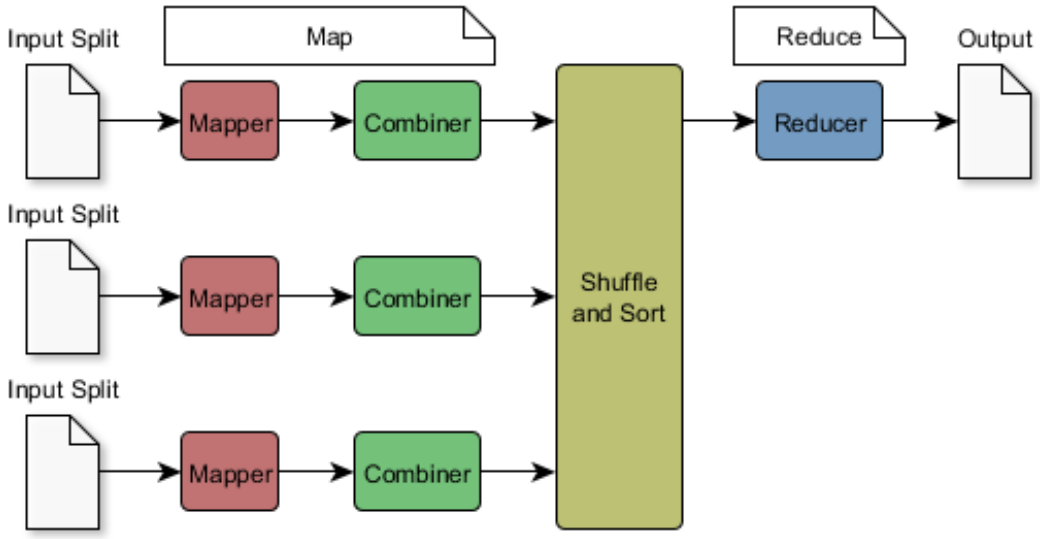

Abbildung 21: Map-Combine-Reduce Struktur

5.5.4.2 Custom Calculation

Um die Einschränkungen aus dem vorherigen Berechnungsmuster zu beseitigen, wurde dieses Pattern hinzugefügt. Dabei handelt es sich streng genommen nicht um ein eigenes Muster, vielmehr wird es ermöglicht, auf die einzelnen Komponenten der Map-Combine-Reduce Struktur des vorherigen Musters Einfluss zu nehmen. Somit ist es möglich beliebig mächtige Berechnungen mit diesem Tool zu erstellen.

5.5.5 Ressourcen

Um Daten zu laden und zu speichern werden Komponenten benötigt, die diese Funktionalität transparent abstrahieren. Dazu wurden die Ressourcenpattern dem MDBDA Editor hinzugefügt, welche in diesem Abschnitt weiter erläutert werden.

5.5.5.1 Cassandra

Cassandra ist eine skalierbare Datenbank, wie in Abschnitt 3.1.2 vorgestellt. Sie wird in der MDBDA Referenzarchitektur (siehe Abbildung 3) verwendet, um große strukturierte Datenmengen zu speichern. In der ersten Version des MDBDA Tools wird das Lesen von ganzen ColumnFamilies unterstützt. Auf dieser Ebene verhält sich das Cassandra Datenmodell wie ein Speicher zeilenorientierter Daten. Später soll die begrenzte Möglichkeit von hierarchischen Daten hinzugefügt werden. Jedoch ist dies bei Cassandra nur mit Einschränkungen bezüglich Hierarchietiefe und Baumstruktur zu ermöglichen. Daher wird dies in dieser Arbeit nicht weiter betrachtet. Das Schreiben der Daten ist sowohl in der MapReduce als auch in der Storm Umgebung voll für zeilenorientierte Daten unterstützt. Bei MapReduce wird dabei auch die Apache Hadoop API verwendet, für Storm wird das GitHub Projekt storm-cassandra [55] eingesetzt.

5.5.5.2 HDFS

Das Hadoop Distributed File System (siehe Abschnitt 3.1.1) wird eingesetzt, um semistrukturierte und unstrukturierte Daten zu speichern. Dazu lassen sich mithilfe dieses Musters in der MapReduce Umgebung ganze Ordner und verteilt gespeicherte Daten einlesen. In der Storm Umgebung lassen sich in der aktuellen MDBDA Version nur einzelne Dateien laden, um beispielsweise Konfigurationen oder ähnliches mittels HDFS zu verwalten. Die Implementierung zur Verarbeitung ganzer Ordnerstrukturen wurde nicht umgesetzt, da Analyse vieler Dateien mittels MapReduce deutlich performanter ist. Das Schreiben funktioniert ähnlich wie bei einer Cassandra Ressource. Beim Lesen als auch beim Schreiben werden unstrukturierte Daten voll unterstützt.

5.5.5.3 Neo4j

Neo4j, die Graphdatenbank aus Abschnitt 3.1.4, eignet sich besonders gut um relationale Daten zu speichern. So bietet sie sich vor allem an, um beispielsweise Navigation oder soziale Netzwerk Anwendungen abzudecken. Neo4j bietet die Cypher Query Language um Abfragen zu erstellen. Als Resultat wird eine spaltenorientierte Tabelle geliefert, welche als Datenquelle für MapReduce und Storm dient. Zum Schreiben von zeilenorientierten Daten nach Neo4j wird ebenfalls die Abfragesprache verwendet.

5.5.5.4 Hazelcast

Für die In-Memory Lösung Hazelcast aus Abschnitt 3.1.6, wird Lesen und Schreiben für zeilenorientierte Daten voll unterstützt. In der Konfiguration der Ressource muss mindestens der Typ der Hazelcast Datenstruktur und der *HazelcastKey* angegeben werden.

5.5.5.5 Generic

Das Generic Ressource Pattern wird dafür eingesetzt, um Workflows für eine spätere Wiederverwendbarkeit vorzubereiten. Es bietet die Möglichkeit Anforderungen, wie die Struktur der Daten, an eingehende beziehungsweise ausgehende Daten anzugeben, welche bei Wiederverwendung erfüllt werden müssen. Die Ressource kann sowohl als Eingabe als auch als Ausgabe Ressource verwendet werden. Wird ein Workflow mit einem solcher Muster wiederverwendet, kann auch dieses Workflow Analysepattern Daten liefern oder entgegennehmen.

6 Qualitätssicherung

Die Sicherung der Qualität war bei der Erstellung dieser Anwendung ein wichtiges Thema und wurde von Anfang an im Entwicklungsprozess berücksichtigt. So wurde versucht, nicht nur bei den entstehenden Generaten die Qualität durch generierte dokumentierende Szenarios mit graphischen Diagrammen und Tests dieser Szenarios zu überprüfen (MRUnit), sondern auch durch das ständige Testen des Editors mithilfe eines automatisierten Oberflächentests (SWTBot) Fehler frühzeitig zu erkennen. Der folgende Abschnitt beschreibt die Umsetzung und die eingesetzten Technologien.

6.1 SWTBot

Um eine qualitativ hochwertige Software zu erstellen, ist es notwendig Qualitätsmerkmale frühzeitig im Entwicklungsprozess zu definieren und die Einhaltung dieser Merkmale ständig zu überwachen. In der Softwaretechnik hat sich das Prinzip der automatisierten Tests dafür durchgesetzt. Dabei wird ein Testszenario mit Startsituation, Aktion und erwartete Endsituation beschrieben und mit einem sogenannten Unit-Test umgesetzt. Dieser Unit-Test ist ein ausführbarer Test, welcher ständig überprüft werden kann. Da Graphical User Interface (GUI) Anwendungen meist eine Interaktion mit einem Nutzer voraussetzt, ist hier ein automatisierter Unit-Test nicht ohne weiteres möglich. Um diese Lücke zu schließen, gibt es das Eclipse Projekt SWTBot[56]. Mithilfe dieses Open-Source Projektes lassen sich für Eclipse Anwendungen wie den MDBDA Editor Nutzerinteraktionen simulieren, sowie Darstellung und Inhalt von graphischen Komponenten

überprüfen.

Für das MDBDA Projekt wurden dazu einige Nutzerszenarien erstellt. Grundlegende Funktionen werden durch Szenarien wie „Alice legt neues MDBDA Projekt an und erstellt ein leeres Diagramm.“ abgebildet und durch einen SWTBot Unit Test validiert. Zusätzlich wurden ein paar komplexere Szenarien erstellt, welche den kompletten MDBDA Workflow für Beispielprojekte abdecken. Abbildung 22 zeigt die durch dieses Vorgehen erreichte Testabdeckung, welche mit Eclipse Tool EclEmma [57] gemessen wurde, und kann einen Überblick zur Qualität des Editors geben.

Variables | Coverage | Console

Merged (18.04.2014 19:29:07)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
org.mdbda.diagrameditor	90,2 %	4.877	528	5.405
src	89,0 %	3.858	475	4.333
org.mdbda.diagrameditor.utils	89,5 %	681	80	761
org.mdbda.diagrameditor.features.resources	93,3 %	609	44	653
org.mdbda.diagrameditor.features	84,8 %	478	86	564
org.mdbda.diagrameditor.pictogramelements	76,6 %	376	115	491
org.mdbda.diagrameditor.features.pattern.datac	98,6 %	363	5	368
org.mdbda.diagrameditor.diagram	95,9 %	257	11	268
org.mdbda.diagrameditor.features.pattern	95,1 %	253	13	266
org.mdbda.diagrameditor.features.pattern.filteri	98,8 %	241	3	244
org.mdbda.diagrameditor.features.pattern.join	98,8 %	241	3	244
org.mdbda.diagrameditor.features.pattern.sumr	98,9 %	179	2	181
org.mdbda.diagrameditor.features.updateFeatur	56,1 %	88	69	157
org.mdbda.diagrameditor.wizard	88,5 %	77	10	87
org.mdbda.diagrameditor	71,4 %	15	6	21
org.mdbda.diagrameditor.action	0,0 %	0	28	28
test	95,1 %	1.019	53	1.072
org.mdbda.xcore	21,9 %	2.107	7.510	9.617
src-gen	51,3 %	2.062	1.957	4.019
src	24,6 %	30	92	122
src-edit	1,0 %	15	1.495	1.510
src-editor	0,0 %	0	3.966	3.966
org.mdbda.codegen	43,9 %	1.278	1.631	2.909
xtend-gen	44,4 %	1.139	1.427	2.566
src	58,7 %	132	93	225

Abbildung 22: Testabdeckung

6.2 JUnit

Um lange Testlaufzeiten zu vermeiden ist es notwendig, Programmteile testen zu können, ohne die ganze MDBDA Plattform als Abhängigkeit zu haben. Dazu wird in dieser Arbeit auf JUnit[58], ein Framework für automatisierte Test

unter Java, gesetzt. Im Generationsprozess wird für jedes MDBDA Pattern ein oder mehrere Tests, je nach Anzahl der Komponenten beziehungsweise Phasen, erstellt, wenn Testdaten, erwartete Endergebnisse und gegebenenfalls notwendige Zwischenergebnisse vorhanden sind. Diese Tests werden jedes mal ausgeführt, wenn sich etwas an der Musterkonfiguration ändert, um so diese Konfiguration zu validieren.

6.3 MRUnit

Nachdem die Qualität des Editors und der einzelnen Komponenten gesichert ist, muss es dem Domänenexperten ermöglicht werden qualitativ hochwertige Analysen zu erstellen. Es ist nicht nur wichtig, dass jede einzelne Komponente für sich korrekt funktioniert, sondern auch das Zusammenspiel aller Teile miteinander getestet wird. Dazu wird in dieser Arbeit das Apache MRUnit ™[59] eingesetzt. Dieses Framework führt die Analyse mit definierten Testdaten, wie sie mit dem MDBDA Workflow (siehe Abschnitt 4) aufgrund des Test-First-Prinzip erstellt wurden, auf der MDBDA Plattform (siehe Abschnitt 3) aus.

7 Anwendung

In diesem Abschnitt werden einige Anwendungsbeispiele gezeigt, die den Nutzen von modellgetriebener Entwicklung von Big Data Anwendungen und den Umgang mit dem MDBDA Editor verdeutlicht.

7.1 Wordcount

Das Standardbeispiel bei Map Reduce Frameworks ist immer das Wordcount Beispiel. In einem großen Text soll die Häufigkeit der Wörter gezählt werden. Dazu soll ein verteilt gespeicherter Text aus einem Hadoop File System geladen, Wörter gezählt und Wörter mitsamt der Summe in einer Cassandra Datenbank gespeichert werden. Der MDBDA Workflow besagt, dass zuerst aus realen Daten, in diesem Fall ein natürlichsprachlicher Text, Testdaten abgeleitet werden und das daraus erwartete Ergebnis. Listing 10 zeigt einen beispielhaften Text von Wikipedia. Anhand dieser Testdaten wurden Erwartungswerte, beziehungsweise Testergebnisse, erstellt. Tabelle 3 zeigt die Ergebnistabelle. Nachdem Start und Ziel definiert wurden, wurde mithilfe des MDBDA Editor der Wordcount Workflow modelliert. Abbildung 23 zeigt den Workflow. Dieser besteht aus drei Pattern. Das erste, eine HDFS Ressource, dient zum Einlesen von Texten. Die Konfiguration wird auf Textinput gestellt und das HDFS Verzeichnis, aus dem die Texte geladen werden sollen, wird angegeben. Die Ressource liefert als Ausgabe Paare der Form *Dateiname:TextInhalt*, wobei der Dateiname als Key dient. Das zweite Muster ist das Custom Calculation Pattern (siehe Abschnitt 5.5.4.2). Ein einfaches Numerical summarization Muster (siehe Abschnitt 5.5.4.1) reicht nicht

```
Die Geschichte, die allen Versionen zugrunde liegt, sind die Abenteuer
des Arthur Dent, eines Durchschnittsengländers, der mit knapper Not
und mit Hilfe seines Freundes Ford Prefect der totalen Zerstörung des
Planeten Erde durch eine außerirdische Rasse namens Vogonen entgeht.
Zu Arthurs Verblüffung erweist sich sein Freund als außerirdischer
Besucher, der die Erde zu Recherchezwecken wegen eines galaktischen
Nachschlagewerks bereiste und zu dessen Unglück sich für einige Jahre
keine Weiterreisemöglichkeit ergeben hatte. Als die Flotte der
Vogonen auftaucht, um die Erde zwecks Baus einer galaktischen
Hyperraum-Expressroute zu zerstören, nutzt Ford die Gelegenheit und
bringt sich und Arthur mittels "Subraum-Äther-Winker" (Sub-Etha-Sens-O-
Matik, eine Art elektronischer Daumen) sozusagen per Anhalter an Bord
eines der Vogonenraumschiffe.
```

Listing 10: Wordcount Testinput

die	7	Arthur	2	sind	1	Prefect	1	Verblüffung	1
der	5	Ford	2	Abenteuer	1	totalen	1	erweist	1
zu	4	als	2	durch	1	Zerstörung	1	sein	1
und	4	Galaktischen	2	Dent	1	Planeten	1	Freund	1
sich	3	Arthurs	1	Durchsch..	1	Auße..sche	1	Besucher	1
eines	3	Geschichte	1	knapper	1	Auße..scher	1	Recherch..	1
Erde	3	allen	1	Not	1	Rasse	1	wegen	1
eine	2	Versionen	1	Hilfe	1	Namens	1	Nachsch..	1
mit	2	zugrunde	1	seines	1	Vogonen	1	bereiste	1
des	2	liegt	1	Freundes	1	entgeht	1	...	1

Tabelle 3: Wordcount erwartete Ausgabe

aus, da die Eingabedaten nicht strukturiert sind, was eine Voraussetzung des Numerical summarization Muster ist. In der Konfiguration des Custom Calculation Musters müssen die Eingabedaten in der Map-Phase erst Strukturiert werden, indem das Value, in diesem Fall der Text Inhalt, nach Leerzeichen aufgeteilt wird. Nach der Strukturierung erhält man eine Liste mit Wörtern. Diese Liste wird zu einer Tabelle erweitert, indem eine weitere Spalte hinzugefügt wird. Jede Zelle in dieser neuen Spalte erhält als Wert eine Eins. Eine weitere Spalte wird eingefügt und mit dem ursprünglichen Key, dem Dateinamen, gefüllt. Die Spalte mit Wörtern dient am Ende der Map-Phase als Keys und die anderen beiden als dazugehöriges Value-Paar. Jedes Wort hat nach der Map-Phase als Wert eine eins und die Ursprungsdatei. Die Combine-Phase und die Reduce-Phase werden gleich

konfiguriert. Die 3-Tupel *(Wort:1:DateiName)*, werden nach Wörtern gruppiert und zusammengefasst. Gab es das Wort beispielsweise drei mal im Text ist das neue Tupel *(Wort:3:DateiName)*. Nach der Reduce-Phase erhält man schließlich eine Tabelle mit Wörter und das dazugehörige Vorkommen. Schlussendlich wird

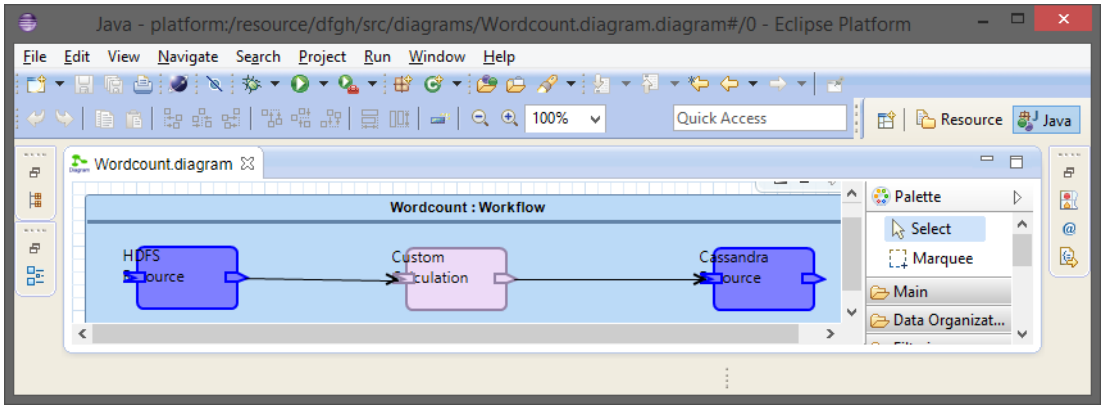

Abbildung 23: Wordcount Workflow

die Cassandra Ressource zur Speicherung der Ergebnisse konfiguriert. Dazu wird eine ColumnFamily als Ziel angegeben und die Daten so parametrisiert, dass der *DateiName* als RowKey, das *Word* als Key und die *Summe* als Value fungiert.

7.2 Verknüpfung von Ressourcen

Die MDBDA Plattform (siehe Abschnitt 3) bietet eine Vielzahl verschiedener Speichermöglichkeiten. Diese sollen in diesem Beispiel miteinander verknüpft werden. Das Szenario beschreibt ein einfaches Problem in einem Customer-Relationship-Management (CRM) System. In einer Neo4j Datenbank sind Beziehungsverknüpfungen, wie *ist_Kollege_von*, zwischen Personen abgebildet. Sämtliche Firmen EMails werden als Text in einer HDFS Ressource archiviert. Der Dateiname der archivierten EMail ist nach der Struktur *Sender:Emfänger:timestamp* aufgebaut. Die Datei beinhaltet die EMails im Textformat nach RFC2822 [60]. Die Analyse soll dabei helfen, EMails zu finden, die von einem Kollegen einer Person ebenfalls in dem Zeitraum geschrieben wurden. Das Ergebnis soll in ein JSON Format gebracht werden und nach HDFS geschrieben werden. Abbildung

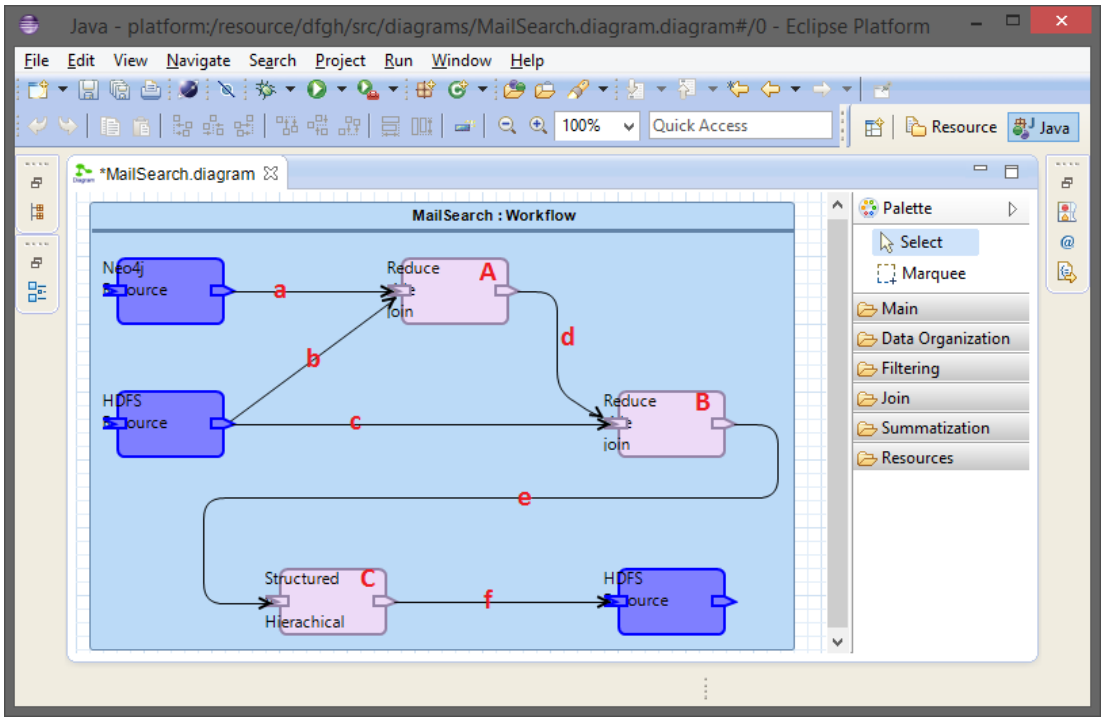

Abbildung 24: CRM EMail Workflow

24 zeigt den Workflow für diese Analyse. Neben den Ein- und Ausgaberessourcen besteht dieser aus drei Mustern, in der Abbildung mit **A**, **B** und **C** markiert, sowie sechs Verknüpfungen **a** bis **f**. Diese Elemente werden im folgenden kurz beschrieben. Die erste Ressource, eine für Neo4j Datenbanken, stellt die zuvor beschrieben Beziehungsverknüpfungen in der tabellarischer Form [EMail Person | EMail Kollege] über die Verknüpfung **a** bereit. Ein Beispieldatensatz ist in Tabelle 4 zu sehen. Die zweite Eingaberessource stellt über die Verknüpfung **b** und **c**

Person	**kollegeVon(Person)**
Road.Runner@acme.com	Wile.E.Coyote@acme.com
Road.Runner@acme.com	Foghorn.Leghorn@acme.com
Road.Runner@acme.com	Elmer.Fudd@acme.com
Foghorn.Leghorn@acme.com	Road.Runner@acme.com
Wile.E.Coyote@acme.com	Road.Runner@acme.com

Tabelle 4: Kombinieren Beispiel: CRM Beziehungsverknüpfungen

jeweils die Daten in der Form [Dateiname | *NULL*] bereit. Der Dateiname ist, wie zuvor beschrieben, aus einer Sender- , Empfängeradresse sowie einem Timestamp

aufgebaut. Mit Beispieldaten in Tabelle 5 aufgeführt. Das Reduce Side Join Mus-

From	**To**	**Timestamp**
Road.Runner@acme.com	mh@hahnpro.com	ts1
Foghorn.Leghorn@acme.com	mh@hahnpro.com	ts2
Elmer.Fudd@acme.com	info@hahnpro.com	ts3

Tabelle 5: Kombinieren Beispiel: EMail Archiv

ter **A** (siehe Abschnitt 5.5.3.2) wird so konfiguriert, dass zum einen die *EMail Person* aus Beziehung **a**, und zum anderen die Teilinformation *Senderadresse* aus dem *Dateiname* aus Verknüpfung **b**, als Key der Map-Phase hervorgeht. Ein beispielhaftes Map Ergebnis ist in Tabelle 6 zu finden. Nach der Map-Phase werden

Person (KEY)	**kollegeVon(Person)**	
Road.Runner@acme.com	Wile.E.Coyote@..	
Road.Runner@acme.com	Foghorn.Leghorn@..	
Road.Runner@acme.com	Elmer.Fudd@..	
Wile.E.Coyote@acme.com	Road.Runner@..	
Foghorn.Leghorn@acme.com	Road.Runner@..	
From (KEY)	**To**	**Timestamp**
Road.Runner@acme.com	mh@hahnpro.com	ts1
Wile.E.Coyote@acme.com	mh@hahnpro.com	ts2
Foghorn.Leghorn@acme.com	info@hahnpro.com	ts3

Tabelle 6: Map A

die Keys in der Reduce-Phase miteinander verknüpft. In der Beispielanwendung entsteht nun eine Liste mit EMails die gesendet wurden und potentiellen Kollegen, welche auch eine EMail gesendet haben könnten. Zu betrachten ist dieses Ergebnis in Tabelle 7. Mithilfe des Reduce Side Join Musters **B** soll diese Liste potentieller

From=Person	**kollegeVon(..)**	**To**	**Timestamp**
Road.Runner@acme.com	Wile.E.Coyote@..	mh@..	ts1
Road.Runner@acme.com	Foghorn.Leghorn@..	mh@..	ts1
Road.Runner@acme.com	Elmer.Fudd@..	mh@..	ts1
Wile.E.Coyote@acme.com	Road.Runner@..	mh@..	ts2
Foghorn.Leghorn@acme.com	Road.Runner@..	info@..	ts3

Tabelle 7: Kombinieren Beispiel: Reduce A

EMails von Kollegen mit den tatsächlich gesendeten EMails abgeglichen werden.

Dazu wird das Muster so konfiguriert, dass in der Map-Phase der Key eingehender Daten aus der Verknüpfung **d** mit *Kollege:To* und Daten von Verknüpfung **c** mit *From:To* zusammengesetzt wird (siehe Tabelle 8). In der Reduce-Phase wird

From:To (KEY)	**Timestamp**	
Road.Runner@..:mh@..	ts1	
Wile.E.Coyote@..:mh@..	ts2	
Foghorn.Leghorn@..:info@..	ts3	
kollegeVon(..):To (KEY)	**From=Person**	**Timestamp**
Wile.E.Coyote@..:mh@..	Road.Runner@acme.com	ts1
Foghorn.Leghorn@..:mh@..	Road.Runner@acme.com	ts1
Elmer.Fudd@..:mh@..	Road.Runner@acme.com	ts1
Road.Runner@..:mh@..	Wile.E.Coyote@acme.com	ts2
Road.Runner@..:info@..	Foghorn.Leghorn@..	ts3

Tabelle 8: Kombinieren Beispiel: Map B

schließlich der Schnitt gebildet mit wirklichen EMails von Kollegen (siehe Tabelle 9). Das Ergebnis dieser Reduce-Phase wird über die Verknüpfung **e** an das letzte

From:To	**org. TS**	**alternativ From**	**alt. TS**
Road.Runner@..:mh@..	t1	Wile.E.Coyote@..	t2
Wile.E.Coyote@..:mh@..	t2	Road.Runner@..	t1

Tabelle 9: Kombinieren Beispiel: Reduce B

Muster **C** weitergeleitet. Schlussendlich wird mithilfe des Structured to Hierarchical Musters **C** ein hierarchischer JSON String gebildet. Dazu werden in der Map-Phase die Daten der Key *From:To:org.TS*, was der ursprüngliche Dateiname aus dem Archiv war, und der Value *alternativFrom:To:alt.TS* vorverarbeitet. In der Reduce-Phase werden die Keys gruppiert und in einer Baumstruktur, wobei der Key jeweils das Elternelement und die Values jeweils die Kinder sind, nach JSON formatiert. Das Endergebnis wird am Ende nach einer HDFS Ressource geschrieben.

8 Related Work

In diesem Abschnitt werden vergleichbare Arbeiten vorgestellt. Mit dem MDBDA Editor lassen sich am besten Extract, Transform, Load (ETL) Tools aus dem Data Warehouse[61] Bereich vergleichen. Der ETL Prozess beschreibt die Extraktion von Daten aus verschiedenen Quelldatenbanken, die Transformation dieser Daten und das anschließende Laden dieser Daten in eine andere Datenbank. Zwei Open Source Vertreter dieser ETL Tools sind Talend[62] und Pentaho[63]. Beide bieten einen graphischen Editor um Datenanalysen zu erstellen. Ebenfalls werden seit einiger Zeit Big Data Lösungen unterstützt, um beispielsweise MapReduce Analysen zu erstellen. Jedoch ist die Echtzeitanalyse von Datenstreams, wie bei MDBDA nicht vorgesehen. Talend bietet zwar die Unterstützung zur Konfiguration von Routing und Datentransformation von Webservices und ähnliches, jedoch mit einem anderen Anspruch, als die Echtzeitdatenanalyse mit Kombination von MapReduce Analysen bei MDBDA. Auch der modellgetriebene Ansatz steht bei den vorgestellten Tools nicht so im Vordergrund wie bei MDBDA. Zwar gibt es Ansätze zur Modellierung von ETL Prozessen [64] [65], jedoch wird auch hier nicht auf die Echtzeitanalyse oder auf den Big Data Schwerpunkt eingegangen.

9 Fazit und Ausblick

Während der Entwicklung von MDBDA hat sich gezeigt, dass die ursprüngliche Idee, *"einfach"* MapReduce Analysemuster für Echtzeitanalysen zu übersetzen und zu verwenden nicht immer Sinn macht, da teilweise viele historische Daten benötigt werden. Daher werden einzelne Muster in der aktuellen Version noch nicht unterstützt. Für diese muss noch ein Workaround gefunden werden und geprüft werden, ob diese im Echtzeitanwendungsfall benötigt werden oder weiterhin weggelassen werden können.

Mit der Version 2.0 von Apache Hadoop wurde das klassische MapReduce Muster überarbeitet und YARN, als neues MapReduce eingeführt. YARN bietet die Möglichkeit der Ausführung von nicht nur MapReduce Anwendungen auf der verteilten Hadoop Plattform. Damit könnte man auch die Funktionalität der Storm Plattform in YARN nachbilden und so die Administration einer Anwendung sparen. Diese Möglichkeit, der Generierung von YARN Anwendungen zur Echtzeitanalyse, gilt es noch zu untersuchen.

Bei der aktuellen Version des MDBDA Editors werden die Analysemuster hauptsächlich in ein JSON Format konfiguriert. Dies ist keine optimale Lösung, da der Anspruch, Big Data Analysen einfach für Nichtinformatiker zu erstellen, so nicht komplett erfüllt werden kann. Schöner wäre eine eigene DSL zur Konfiguration oder sogar eine StoryBoard[66] ähnliche Lösung, welche das Zeichnen der Konfiguration, wie bei der Verknüpfung der Analysemuster, erlaubt.

10 Anhang

Abkürzungsverzeichnis

ACME A Company that Manufactures/Makes Everything

API Application Programming Interface

CSV Comma-separated values

CRM Customer-Relationship-Management

DSL Domain-Specific Language

EMF Eclipse Modeling Framework

ETL Extract, Transform, Load

GEF Graphical Editing Framework

GMF Graphical Modeling Framework

GUI Graphical User Interface

HDFS Hadoop Distributed File System

JSON JavaScript Object Notation

JVM Java Virtual Machine

MDE Model Driven Engineering

Mwe2 Modeling Workflow Engine 2

RDBMS Relational Database Management System

SQL Structured Query Language

URI Uniform Resource Identifier

XML Extensible Markup Language

Abbildungsverzeichnis

1 Bitkom Big Data Merkmale[5] 5
2 Rahmenarchitektur der MDBDA Zielplattform 9
3 Referenzarchitektur der MDBDA Zielplattform 10
4 Einfache Storm Topologie 12
5 Neo4j Cypher Beispiel Weboberfläche 14
6 Kafka publish-subscribe 15
7 MDBDA Workflow 17
8 Xcore Beispiel[41] 21
9 Beispiel Graphiti Editor[48] 22
10 Metamodell Ebenen 27
11 MDBDA Editor 30
12 Struktur des Partitioning Pattern in MapReduce 34
13 Struktur des Partitioning Pattern in Storm 35
14 Struktur des Structured-to-Hierarchical Pattern in MapReduce . . 36
15 Struktur des Total Order Pattern in MapReduce 37
16 Struktur des Top Ten Musters in MapReduce 38
17 Struktur des Cartesian Product Pattern in MapReduce 40
18 Struktur des Reduce Side Join Pattern in MapReduce 41
19 Struktur des Reduce Side Join Pattern in Storm 42
20 Struktur des Replicated Join Pattern 43
21 Map-Combine-Reduce Struktur 44
22 Testabdeckung 48
23 Wordcount Workflow 52
24 CRM EMail Workflow 53

Listings

1 Daten erzeugen mittels Cypher 13
2 Daten abfragen mittels Cypher 14
3 Hazelcast Distributed Map[31] 16
4 Modeling Workflow Engine 23
5 Xpand Template 24
6 Xtend Extension Methods 25
7 Xtend Template Expressions 25
8 MDBDA Core Domain Model 28
9 SimpleMatcher Factory 29
10 Wordcount Testinput 51

Tabellenverzeichnis

3 Wordcount erwartete Ausgabe 51
4 Kombinieren Beispiel: CRM Beziehungsverknüpfungen 53
5 Kombinieren Beispiel: EMail Archiv 54
6 Map A 54
7 Kombinieren Beispiel: Reduce A 54
8 Kombinieren Beispiel: Map B 55
9 Kombinieren Beispiel: Reduce B 55

Literatur

[1] "Wer hebt das Datengold?," *Die Zeit*, vol. N° 2, Jan 2013.

[2] "Big Data," *Frankfurter Allgemeine Zeitung*, vol. Nr. 141, Jun 2013.

[3] "Big Data Special," in *Venture Capital Magazin*, vol. Nr. 1/2, pp. 14 – 25, Jan 2014.

[4] H. Hoffmann, "Malt mehr!," in *Java Magazin 03.10*.

[5] "Leitfaden Big Data im Praxiseinsatz." Verfügbar Online `http://www.bitkom.org/de/publikationen/38337_73446.aspx` zuletzt besucht am 8. März 2014, 2013.

[6] S. Kent, "Model Driven Engineering," in *Proceedings of the Third International Conference on Integrated Formal Methods*, IFM '02, (London, UK, UK), pp. 286–298, Springer-Verlag, 2002.

[7] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[8] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, Jan. 2008.

[9] "Apache Hadoop." Website. Verfügbar Online `http://hadoop.apache.org/` zuletzt besucht am 9. März 2014.

[10] M. Eckert and F. Bry, "Complex Event Processing (CEP)," *Informatik-Spektrum*, vol. 32, no. 2, pp. 163–167, 2009.

[11] D. Schmid and S. Igel, "Hadoop im Unternehmenseinsatz," in *Java Magazin 07.12*.

[12] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, 2012.

[13] *Java Magazin - Apache Hadoop*, vol. Vol. 11, Nov. 2011.

[14] "Apache Hadoop 2: die neue Generation des Big-Data-Frameworks." Website. Verfügbar Online `http://heise.de/-1980205` zuletzt besucht am 9. März 2014.

[15] "Apache hadoop releases."

[16] "RFC1813 - NFS Version 3 Protocol Specification." Website. Verfügbar Online `http://tools.ietf.org/html/rfc1813` zuletzt besucht am 10. März 2014.

[17] "Apache Cassandra." Website. Verfügbar Online `http://cassandra.apache.org/` zuletzt besucht am 10. März 2014.

[18] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 35–40, Apr. 2010.

[19] K. Spichale and E. Wolff, "NoSQL: Konzepte und Prinzipien," in *Java Magazin 04.12*.

[20] S. Edlich, A. Friedland, J. Hampe, and B. Brauer, *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. Hanser Fachbuchverlag, 10 2010.

[21] "Apache Storm." Website. Verfügbar Online `http://incubator.apache.org/projects/storm.html` zuletzt besucht am 8. März 2014.

[22] K. Spichale, "Verarbeitung komplexer Ereignisse und Datenströme," in *JavaMagazin 03.13*.

[23] "Github Storm Powerd By." Website, 2013. Verfügbar Online `https://github.com/nathanmarz/storm/wiki/Powered-By/` zuletzt besucht am 8. März 2014.

[24] "Neo4j." Website. Verfügbar Online `http://www.neo4j.org/` zuletzt besucht am 9. März 2014.

[25] P. Neubauer, "Neo4j," in *Java Magazin 07.12*.

[26] M. Hunger, "Neo4j hands-on," in *Java Magazin 01.14*, pp. 82 – 89.

[27] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. O'Reilly Media, Incorporated, 2013.

[28] M. Hunger, "Produktiv mit Graphen," in *Java Magazin 02.14*.

[29] "Apache Kafka." Website. Verfügbar Online `http://kafka.apache.org/` zuletzt besucht am 10. März 2014.

[30] N. Garg, *Apache Kafka*. EBL-Schweitzer, Packt Publishing, 2013.

[31] "Hazelcast." Website. Verfügbar Online `http://www.hazelcast.org/` zuletzt besucht am 10. März 2014.

[32] M. Ebbing and C. Lambert, "Hazelcast," in *JavaMagazin 11.10*.

[33] M. Johns, *Getting Started with Hazelcast*. Packt Publishing Ltd.

[34] K. Beck, *Test-Driven Development by Example*. The Addison-Wesley Signature Series, Addison-Wesley, 2003.

[35] The Eclipse Foundation, "Eclipse." `http://www.eclipse.org/`, 2007.

[36] A. Neumann, "10 Jahre Eclipse: Konsolidierung des Java-IDE-Markts." Website. Verfügbar Online `http://heise.de/-1370644` zuletzt besucht am 12. März 2014.

[37] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*. Boston, MA: Addison-Wesley, 2 ed., 2009.

[38] J. von Pilgrim, "Agile MDA mit EMF," in *Eclipse Magazin 02.05*, pp. 78 – 84.

[39] "GitHub." Website. Verfügbar Online `https://github.com/` zuletzt besucht am 28. März 2014.

[40] "github emf repositories."

[41] "Eclipse Xcore Wiki." Website. Verfügbar Online `https://wiki.eclipse.org/Xcore` zuletzt besucht am 12. März 2014.

[42] "Eclipse Graphiti." Website. Verfügbar Online `http://www.eclipse.org/graphiti/` zuletzt besucht am 12. März 2014.

[43] M. Wenz, "Graphiti," in *Eclipse Magazin 01.11*.

[44] "Eclipse Graphical Editing Framework (GEF)." Website. Verfügbar Online `http://www.eclipse.org/gef/` zuletzt besucht am 12. März 2014.

[45] B. Bokowski and F. Gerhardt, "Maßgeschneiderte grafische Editoren mit dem Graphical Editing Framework (GEF)," in *Eclipse Magazin 02.05*, pp. 85 – 90.

[46] "Eclipse Graphical Modeling Framework (GMF)." Website. Verfügbar Online `http://wiki.eclipse.org/Graphical_Modeling_Framework` zuletzt besucht am 12. März 2014.

[47] A. Schwartz, "Grafische Editoren für Domänenmodelle generieren mit GMF," in *Eclipse Magazin 04.06*, pp. 63 – 67.

[48] "Graphiti Developer Guide." Website. Verfügbar Online `http://help.eclipse.org/kepler/topic/org.eclipse.graphiti.doc/resources/docu/gfw/scenario-description.htm?cp=27_1_0` zuletzt besucht am 12. März 2014.

[49] "Xpand." Website. Verfügbar Online `http://wiki.eclipse.org/Xpand` zuletzt besucht am 15. März 2014.

[50] B. Kolb and M. Völter, "openArchitectureWare und Eclipse," in *Eclipse Magazin 03.05*.

[51] The Eclipse Foundation, "Xtend." `https://www.eclipse.org/xtend/`.

[52] D. Miner and A. Shook, *Mapreduce Design Patterns Building Effective Algorithms and Analytics for Hadoop and Other Systems*. Oreilly & Associates Inc, 2012.

[53] "JSON." `http://www.json.org`.

[54] "Graphiti Developer Guide." Website. Verfügbar Online `http://pig.apache.org/docs/r0.12.0/perf.html#filter` zuletzt besucht am 20. März 2014.

[55] "Storm Cassandra Integration." Verfügbar Online `https://github.com/hmsonline/storm-cassandra` zuletzt besucht am 19. März 2014, 2014.

[56] "SWTBot."

[57] B. Muskalla, "EclEmma," in *Eclipse Magazin 03.07*, pp. 56 – 57.

[58] K. Beck, *JUnit: kurz & gut*. O'Reillys Taschenbibliothek, O'Reilly Vlg. GmbH & Company, 2005.

[59] "Apache MRUnit ™." Website. Verfügbar Online `http://mrunit.apache.org/` zuletzt besucht am 20. März 2014.

[60] P. Resnick, "RFC2822 - Internet Message Format." `https://tools.ietf.org/html/rfc2822`, apr 2001. accessed: 2014-03-22.

[61] P. Ponniah, *Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals*. Wiley, 2004.

[62] "talend Big Data." Website. Verfügbar Online `http://de.talend.com/products/big-data` zuletzt besucht am 24. März 2014.

[63] "pentaho Big Data." Website. Verfügbar Online `http://www.pentahobigdata.com/` zuletzt besucht am 24. März 2014.

[64] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "Conceptual Modeling for ETL Processes," in *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '02, (New York, NY, USA), pp. 14–21, ACM, 2002.

[65] J. Trujillo and S. Luján-Mora, "A UML Based Approach for Modeling ETL Processes in Data Warehouses," in *Conceptual Modeling - ER 2003* (I.-Y. Song, S. Liddle, T.-W. Ling, and P. Scheuermann, eds.), vol. 2813 of *Lecture Notes in Computer Science*, pp. 307–320, Springer Berlin Heidelberg, 2003.

[66] T. Fischer, J. Niere, L. Torunski, and A. Zündorf, "Story diagrams: A new graph rewrite language based on the unified modeling language and java," in *Theory and Application of Graph Transformations*, pp. 296–309, Springer Berlin Heidelberg, 2000.