Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jens Ellenberg

Event Stream Processing mit ESPER unter Einsatz von Datamining Verfahren

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jens Ellenberg

Event Stream Processing mit ESPER unter Einsatz von Datamining Verfahren

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Michael Neitzke
Zweitgutachter : Prof. Dr. sc. pol. Wolfgang Gerken

Abgegeben am 18. Juni 2009

Jens Ellenberg

Thema der Bachelorarbeit

Event Stream Processing mit ESPER unter Einsatz von Datamining Verfahren

Stichworte

Event, Event Stream Processing, Datamining Algorithmen, Esper

Kurzzusammenfassung

In dieser Arbeit wurde untersucht, welche Möglichkeiten Esper bietet, um Datamining Algorithmen auf Event Streams umzusetzen. Dabei wurden als Ausgangspunkt bestehende Anwendungsmodelle verwendet. Für diese Anwendungsmodelle wurden Vertreter von Datamining Algorithmen für Event Streams gesucht. Diese Datamining Algorithmen wurden dahingehend analysiert, inwieweit sie sich für die Umsetzung mit Esper eignen. Aufgrund dieser Erkenntnisse wurde anschließend eine Architektur entwickelt, in der Esper eingesetzt wird, um Datamining Algorithmen auf Event Streams umzusetzen. Diese Architektur wurde mit einem Prototypen verdeutlicht.

Jens Ellenberg

Title of the paper

Event Stream Processing with ESPER using Datamining Technics

Keywords

Event, Event Stream Processing, Datamining Algorithmen, Esper

Abstract

This work examined which possibilities are given, using Esper to implement datamining algorithms on event streams. Therefore, existing application models have been used and exponents of datamining algorithms for event streams were searched for these application models. The datamining algorithms have been analyzed on their adequacy for implementing, using Esper. Concerning this technical expertise, an architecture has been created subsequently, in which Esper is used to implement dataming algorithms on event streams. This architecture was clarified by implementing a prototype.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AWSOM	Arbitrary Window Stream mOdeling Method
CF	Cluster Feature Vector
CFT	Cluster Feature Vector Temporal
CSP	Complex Stream Processing
DBMS	DatenBank Management System
DOM	Document Object Model
DWT	Diskrete Wavelet-Transformation
EPL	Event Processing Language
ESP	Event Stream Processing
GPS	Global Positioning System
ID3	Iterative Dichotomiser 3
JDK	Java Development Kit
P-Tree	Peano Count Tree
POJO	Plain Old Java Object
SQL	Structured Query Language
VFDT	Very Fast Decision Tree learner
XML	Extensible Markup Language

1. Einleitung

1.1. Motivation

Wir leben im Zeitalter der Informationen, die ein wertvolles Gut darstellen. Sie werden in allen Bereichen des Lebens erzeugt und gespeichert. Einige Beispiele, in denen Informationen anfallen, sind: Die Börse, Internetseiten, Funkantennen oder die Telekommunikation.

Es gab 18 exabytes (exa = 10^{18}) neu generierte Informationen im Jahr 2002 [5]. Wahrscheinlich ist, dass sich diese Zahl bis heute erhöht hat und in Zukunft auch noch weiter wachsen wird. Bei anfallenden Daten kommt es nicht zwingend auf die einzelne Information an. Es kann viel wichtiger sein, die Daten als Ganzes zu betrachten und aus den Zusammenhängen weitere Informationen zu gewinnen. Beispielsweise könnte ein aufgefangener Funkspruch zuwenig aussagen. Dem gegenüber könnten erst viele Funksprüche über einen Zeitraum eine Kommunikation ergeben, die von Interesse ist. Das bedeutet, dass man in bestimmten Fällen sehr viele Informationen analysieren und verarbeiten muss, bevor man sie effektiv nutzen kann.

Daten haben immer einen zeitlichen Ursprung. Dieser Ursprung ist jener Zeitpunkt, an dem sie das erste mal gesammelt und gespeichert werden. An manchen Stellen werden die Daten in einem stetigen Strom gesammelt. Die empfangenen Signale von einer Antenne, die zur Überwachung genutzt wird, werden als Datenobjekte abgespeichert. Jedes Mal, wenn diese Antenne Funksprüche eines Flugzeugpiloten oder Signale von einem Handy auffängt, entsteht ein neues Datenobjekt. Nicht jedes dieser Datenobjekte ist für den Nutzer wichtig. Die Antenne empfängt demnach auch Funksprüche, die unwichtig sind, oder Signale, die nichts mit der eigentlichen Überwachung oder der späteren Auswertung zu tun haben. Daher ist es sinnvoll, nicht alle anfallenden Daten abzuspeichern. Eine bessere Methode ist es, direkt auf dem eingehenden Datenstrom zu arbeiten und nur die wichtigen Daten zu nutzen. Dadurch ergeben sich verschiedene Vorteile. Die zu bearbeitenden Daten müssen nicht erst gespeichert werden, die benötigte Hardware reduziert sich und die Geschwindigkeit der Bearbeitung wird erhöht.

Es ist notwendig, die Daten zeitgerecht nach ihrem Eingang zu bearbeitet, um anschließend folgerichtig auf die erzielten Erkenntnisse reagieren zu können. Diese eingehenden Datenobjekte sind dabei atomare, unmittelbare Ereignisse an einem Punkt in der Zeit und

entsprechen damit der Definition eines Events aus [7]. Daraus folgt, dass jedes Datenobjekt als ein Event interpretiert werden kann. Jedes dieser Events kann eine Verarbeitungskette zur Analyse anstoßen. Am Ende der Verarbeitungskette können weitere Aktionen gestartet werden. Zum Beispiel könnte nach dem Auffangen eines bestimmten Frequenzmusters eine Überwachung gestartet werden.

Um weitere Informationen aus dem Strom der Events (Event Stream) zu gewinnen, gibt es verschiedene Datamining-Techniken. Diese Techniken sind überwiegend für statische Datenmengen entwickelt worden. Auf Event Streams kann man nicht die gleichen Algorithmen anwenden, die man auch auf einer Datenbank anwendet, da bei Event Streams kein Zugriff auf den vollen Umfang der Daten besteht und die Daten sich ständig ändern [3]. Daher ist es notwendig, die Datamining Algorithmen diesen Umständen speziell anzupassen [13].

Event Stream Processing unterscheidet sich hauptsächlich in zwei Dingen von den Abfragen an eine Datenbank. Das erste ist, dass der Sprachumfang erweitert werden muss. Eine wichtige Erweiterung ist, dass Events in Abhängigkeit zueinander gesetzt werden können. Diese Unterstützung bezieht sich auf die Definition relevanter Events, die Strategie mit der die Events ausgewählt werden und die Kriterien, mit denen die Abfrage beendet wird. Zum zweiten ist die Effizienz der Abfragen neu zu bewerten, wenn sie auf Streams arbeiten. Zum Beispiel ist der Join zwischen zwei Streams ein völlig anderer Prozess als der Join zweier Tabellen [2].

Für die Umsetzung der oben genannten Punkte bietet das Tool Esper (Kapitel: 2.2) folgende Voraussetzungen:

- Esper ist eine Laufzeitumgebung für Event Stream Processing.
- Es unterstützt die Kleenesche Hülle auf Streams.
- Mit Esper können Abfragen auf Event Streams durchgeführt werden.
- Die Architektur von Esper unterstützt eine hohe Anzahl eingehender Events pro Sekunde.
- Mit dem Ergebnis der Abfragen kann dann entsprechend den Anforderungen weiter verfahren werden.

Es bietet sich daher an, die Umsetzung von Datamining Algorithmen auf Event Stream Processing mit Esper zu untersuchen.

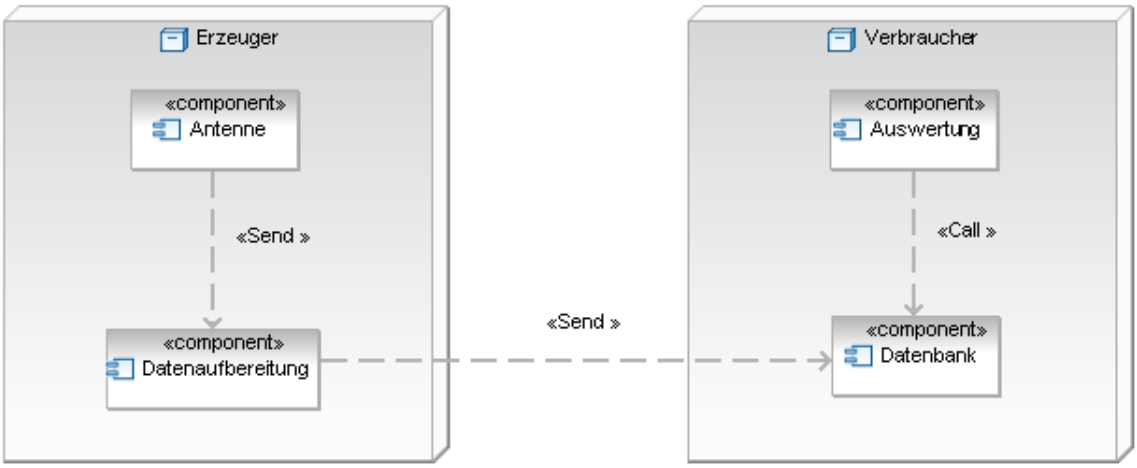

Abbildung 1.1.: Erzeuger und Verbraucher mit einer Datenbank

1.2. Ziel der Arbeit

Die Arbeit ordnet sich in dem Bereich der Informationsgewinnung ein. In diesem Bereich liegen häufig viele Informationen aus einem bestimmten Kontext vor. Aus dem Zusammenhang der vorliegenden Daten, die eine externe Einheit generiert, sollen neue Informationen gewonnen werden. Diese externe Einheit erzeugt ständig sehr viele Daten und diese stehen in einem Stream zur Verfügung. Der Erzeuger der Daten könnte zum Beispiel eine Funkantenne sein. An der Antenne werden ständig neue Signale aufgefangen. Jedes aufgefangene Signal selbst liefert zu wenig Informationen. Daher ist es notwendig innerhalb der Signale nach weiteren Informationen zu suchen oder den gewonnenen Signalen Metainformationen hinzuzufügen um eine weiterführende Analyse zu ermöglichen.

In verschiedenen bestehenden Anwendungen werden alle erzeugten Daten vor der Verarbeitung in eine Datenbank geschrieben (siehe Abbildung 1.1).

Diese Daten werden nach dem Speichern wieder aus der Datenbank gelesen und durch eine Auswertung analysiert. Dieses Verfahren kann bei einem hohen Datenaufkommen sehr aufwändig sein. Beispielsweise empfängt eine Funkantenne[1] pro Tag im Schnitt ca. 32 Millionen Signale. Das sind 350 Signale pro Sekunde. Um die Menge der Informationen zu bewältigen, ist eine effizientere Vorgehensweise notwendig, da das bisherige Verfahren verschiedene Grenzen hat.

- Die Geschwindigkeit, mit der eine Datenbank Lese- und Schreiboperationen durchführen kann, ist begrenzt. Daraus folgt, dass die Ressourcen, um die erzeugten Daten zu

[1] Funkantenne aus einem Projekt der Firma Plath

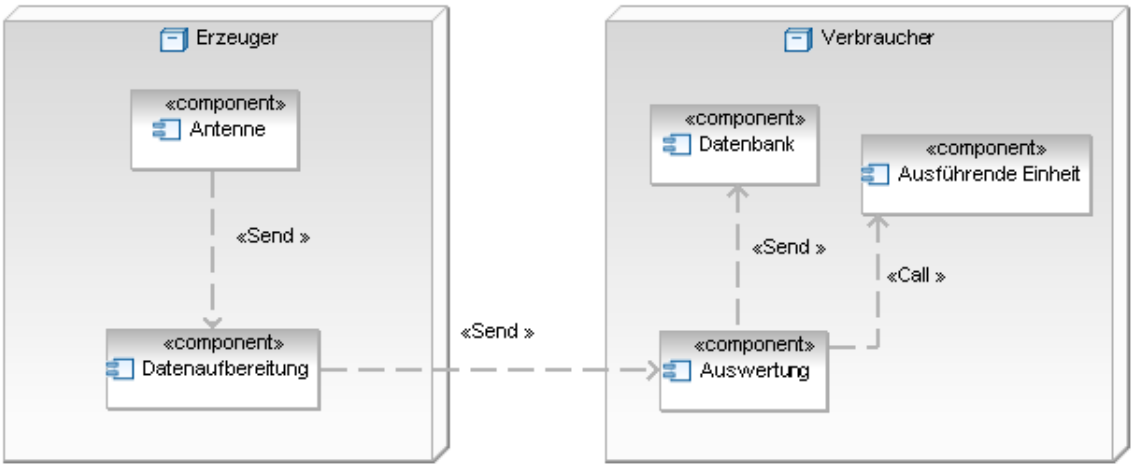

Abbildung 1.2.: Erzeuger und Verbraucher mit CEP

speichern und während dessen diese Daten zur Analyse bereit zu stellen, begrenzt sind.

- In einigen Anwendungen, in denen das Verfahren eingesetzt wird, ist es erforderlich, innerhalb weniger Minuten nach dem Auftreten eines Musters Aktionen durchzuführen. Dies bedeutet, dass die Analyse der Daten aktuell und immer aktiv sein muss.

Der Fokus dieser Anwendungen liegt auf dem Finden der Informationen. Die Menge der Daten macht das Lesen und Schreiben in einem langsamen Speicher und damit die Nutzung eines Datenbank Management Systems sehr aufwändig. In dieser Arbeit soll nun ein anderer Ansatz diskutiert werden. Es ist besser, direkt auf dem Datenstrom zu arbeiten, um komplexe Muster auf einem Datenstrom zu suchen.

Um auf neue Erkenntnisse zeitgerecht reagieren zu können, werden die eingehenden Daten als Events interpretiert. Dieses hat zwei Gründe. Bei der Arbeit auf dem Datenstrom stehen fortlaufend neue Daten zur Verfügung und die Verarbeitung der Daten muss ständig aktiv sein. Der zweite Grund ist, dass die Ergebnisse in annähernd Echtzeit zur Verfügung stehen sollen. Diese Art der Datenverarbeitung wird Event Stream Processing (ESP) oder Complex Stream Processing (CSP) genannt (siehe Abbildung 1.2).

Für die Arbeit auf Event Streams bietet sich das Tool Esper an. Mit Esper kann man Abfragen auf einem Event Stream definieren. Die Ergebnisse werden von Esper unmittelbar zur Verfügung gestellt.

Der Ausgangspunkt für die Umsetzung von Datamining auf Event Streams sind bestehende Anwendungsmodelle. In diesen Anwendungen wurden auf einer Datenbank Datamining Algorithmen eingesetzt.

Für den neuen Ansatz, der Arbeit auf einem Event Stream, müssen die Datamining Algorithmen analysiert werden. Event Streams unterscheiden sich zu den Datenbanken in den folgenden Eigenschaften:

- Die Daten können nur einmal gelesen werden.
- Der Speicher für die Daten ist auf den Arbeitsspeicher begrenzt.
- Es sind theoretisch unendlich viele Daten.
- Die Ergebnisse sollen in annähernd Echtzeit zur Verfügung stehen.
- Die Analyse muss auf jedes Event direkt reagieren.

Diese Eigenschaften müssen berücksichtigt werden, wenn man Datamining Algorithmen auf Event Streams anwenden will. Es werden in dieser Arbeit Event Stream Algorithmen analysiert und festgestellt, wie sie sich in Bezug auf die Anwendungsmodelle und allgemein für die Umsetzung mit Esper eignen.

In dieser Arbeit wird dargestellt, was nötig ist, um Datamining Algorithmen mit Esper umzusetzen. Außerdem wird untersucht, welche Möglichkeiten das Tool bei der Umsetzung der Algorithmen bietet. In dieser Arbeit wird eine Architektur mit Esper für Datamining konzipiert. Dieses Konzept soll allgemein zeigen, welche Möglichkeiten bestehen mit Esper Datamining Algorithmen umzusetzen.

Um die Architektur zu verdeutlichen, wird das erarbeitete Konzept prototypisch implementiert. Dabei wird für ein Anwendungsmodell ein geeigneter Algorithmus mit Esper umgesetzt. Das Programm mit Esper wird in Java implementiert.

1.3. Aufbau der Arbeit

Diese Arbeit beginnt mit einer Darstellung der Grundlagen. (Kapitel 2). Zu den Grundlagen gehört, was unter Event Stream Processing zu verstehen ist und welche Eigenschaften Esper hat. Desweiteren werden Anwendungsmodelle aus der Praxis beschrieben, damit der Leser eine bessere Vorstellung des Anwendungsgebietes bekommt. Diese Anwendungsmodelle werden im weiteren Verlauf der Arbeit zur Suche und Analyse der Algorithmen herangezogen.

Um Datamining auf Event Streams durchführen zu können sind besondere Algorithmen erforderlich. Daher ist es nötig Algorithmen zu finden, die den besonderen Ansprüchen genügen. (Kapitel 3). Anschließend, wird in diesem Kapitel analysiert, wie sich diese Algorithmen für die Anwendungsmodelle und für Esper eignen.

Aus den gewonnenen Erkenntnissen der Analyse wird nachfolgend eine Architektur erstellt, die Esper für Datamining auf Event Streams einsetzt. (Kapitel 4). Es wird zunächst eine allgemeine Übersicht gegeben. Anschließend werden die einzelnen Komponenten der Architektur erläutert und das System als Ganzes beschrieben.

Um die Architektur zu verdeutlichen, wird diese in einer prototypischen Implementierung umgesetzt. (Kapitel 5).

Abschließend werden die wesentlichen Inhalte dieser Arbeit noch einmal zusammengefasst und es wird ein Ausblick gegeben, welche Untersuchungen auf diese Arbeit folgen können (Kapitel 6).

2. Grundlagen

Das folgende Kapitel beschreibt den Begriff „Event Stream Processing“ genauer. Event Stream Processing ist die Kombination von „Event Processing“ und „Stream Processing“. Daher werden erst diese beiden Konzepte dargestellt, bevor „Event Stream Processing“ beschrieben wird. Danach wird das Tool Esper erläutert. Es werden die verschiedenen Möglichkeiten von Esper aufgezeigt. Es wird gezeigt, warum Esper sich sehr gut für die Arbeit mit Event Streams eignet. Nachdem der Begriff „Event Stream Processing“ und das Tool Esper beschrieben wurden, werden Anwendungsmodelle für Event Stream Processing aus der Praxis dargestellt.

2.1. Event Stream Processing

2.1.1. Beschreibung von Event Processing

Oft ist es wichtig, bei dem Auftreten eines Ereignisses direkt informiert zu werden. Nach dem Auftreten des Ereignisses ist es manchmal zusätzlich nötig andere Prozesse einzuleiten. So ein Ereignis wird im Englischen als Event bezeichnen. In [7] sind Events atomare Vorkommnisse an einem Punkt in der Zeit. Nach [8] ist die Definition eines Events keine einfache Angelegenheit. Es gibt im allgemeinen Sprachgebrauch Events, die das physische Vorkommnis selbst sind. Der Funkspruch eines Piloten wäre zum Beispiel so ein Event oder ein Datenpaket eines Handygesprächs. Wenn man die technische Repräsentation des Vorkommnis meint, dann wird ebenfalls von Events gesprochen. In diesem Fall ist das Event ein Datensatz in einem Informationssystem. In dieser Arbeit werden die beiden Bedeutungen synonym verwendet [15].
Eine Möglichkeit auf Events zu reagieren, ist, sie in einer Datenbank zu speichern. Vor oder nach dem Abspeichern kann man dann mit einem Trigger oder einer Abfrage auf den Daten überprüfen, ob eine bestimmte Situation eingetreten ist. Datenbanken sind aber nicht in der Lage, automatisiert das „Monitoring“ oder den „Trigger“ gut umzusetzen. Daher ist es notwendig andere Methoden zu nutzen, um auf bestimmte Events korrekt zu reagieren [7].

2.1.2. Beschreibung von Stream Processing

Datenströme sind an vielen Stellen in der Informationsverarbeitung zu finden. Die Erzeugung der Daten erfolgt dabei nicht in Intervallen aus vielen Daten, die parallel erzeugt werden, sondern die Daten werden nacheinander in einem stetigen Strom erzeugt. Eine Funkantenne empfängt immer nur ein Signalpaket an einem Zeitpunkt. Es werden aber viele Signalpakete pro Sekunde empfangen. Diese Signalpakete weisen in manchen Anwendungen Zusammenhänge auf und in anderen wiederum nicht. Das bedeutet, dass der Inhalt der Signale jedes mal eine neue Information darstellen kann, die sich nicht auf die letzte Information bezieht. Zum Beispiel ist ein Funkspruch eines Piloten jedes Mal eine eigenständige Information. Es kann sich bei einem Datenpaket auch um eine Kette von Informationen handeln, die jeweils mit den schon empfangenen Daten in Zusammenhang stehen. Signale von einem Handy zum Beispiel sind ein Gespräch, das sich aus vielen Datenpaketen zusammen setzt. Jedes Datenpaket gehört zu einer Kette, die als gesamtes in ihrer Reihenfolge betrachtet werden muss.
Die traditionelle Art des Datenmanagements sind Datenbanken. In Datenbanken ist es üblich, dass nur wenige neue Daten in die Datenbank geschrieben werden (write), Daten verändert werden (update) oder Daten gelöscht werden (delete). Vielmehr kommt es häufig zu einer Abfrage der Daten (read). Dieses ist in den oben genannten Stream Beispielen nicht der Fall. Im Stream Processing gibt es sehr viele write- / update- / delete-Operationen und nur wenige read-Operationen.
Eine weitere Besonderheit in Streams ist, dass nur ein kleiner Teil der Informationen benötigt wird. Am Beispiel der Handys sind die Gespräche nur dann wichtig, wenn sie als verdächtig eingestuft werden. Es könnte aber auch sein, dass man nach einem bestimmten Muster innerhalb der Gespräche sucht. In diesem Fall sind nur die Daten interessant, die dem gesuchten Muster entsprechen. Alle anderen Gespräche brauchen nicht abgespeichert werden. Vergleiche auch [3].

2.1.3. Event Stream Processing

Event Stream Processing ist die Vereinigung der Eigenschaften von Stream Processing und Event Processing. Es gibt einen Stream von Events, von dem jedes eingehende Event eine Mustererkennung auslöst, auf die entsprechend der Vorgaben reagiert wird. Event Stream Processing hat die folgenden Eigenschaften:

- Es liegen Events in einem ständigen Strom vor.
- Der Strom der Events ist nicht regelmäßig, sondern die Frequenz der ankommenden Events verändert sich ständig.

- Die eingehenden Events müssen nicht geordnet sein. Das heißt, dass die Events nicht immer in der Reihenfolge bei dem System ankommen, in der sie erzeugt wurden. Die Reihenfolge, die nötig ist, um auf einer Menge von Events Muster zu erkennen, muss explizit erstellt werden.
- Die Menge der Events ist so groß, dass es sehr aufwändig wäre, sie in einer Datenbank abzuspeichern.
- Auf diesem Event Stream sollen Muster erkannt werden.
- Sobald ein Muster erkannt ist, müssen weitere Maßnahmen ergriffen werden. Hier gibt es verschiedene Möglichkeiten warum nicht alle Events abgespeichert werden sollen.
  - Die Events selbst sind nicht wichtig, sondern nur die Muster. Nach dem Erkennen der Muster werden die Events verworfen und es werden die entsprechenden Maßnahmen eingeleitet, sobald ein Muster erkannt wird.
  - Events sollen gespeichert werden, aber nur dann, wenn sie dem gesuchten Muster entsprechen.
- Jedes ankommende Event könnte dazu beitragen, dass ein Muster erkannt wird. Daher muss die Mustererkennung jedes Mal durchgeführt werden, wenn ein neues Event eintrifft.

2.2. ESPER

2.2.1. Allgemeine Beschreibung

Esper [17] ist ein Tool, das ESP bzw. CSP unterstützt. Man kann mit Esper Muster innerhalb eines Event Streams suchen. Vereinfacht ausgedrückt ist die Arbeitsweise von Esper genau entgegengesetzt der einer normalen Datenbank. Es werden nicht Daten gespeichert und dann Abfragen über diese Daten geschickt, sondern es werden die Abfragen gespeichert und die Daten werden über diese Abfragen geschickt. Da jede Ankunft der Daten eine Auswertung der gespeicherten Abfragen auslöst, werden die Daten als Events betrachtet. So werden Abfragen auf einem Event Stream ermöglicht.
Die Abfragen werden in einer Event Processing Language (EPL) definiert. Um auf einem Event Stream Abfragen zu erstellen, muss es möglich sein, Sequenzen auszudrücken. Außerdem müssen Kleenesche Hülle auf Streams, Negation und komplexe Prädikate unterstützt werden [2].

Mit der Unterstützung der Kleeneschen Hülle ist hier der besondere Bezug auf Event Streams gemeint. Der Unterschied zu Kleeneschen Hülle von regulären Ausdrücken besteht in den folgenden Punkten.

Definition relevanter Events: Die Auswahl der Events wird unter anderem durch Filter spezifiziert, mit denen die Events in Bezug auf ihre Werte ausgewählt werden. Es könnten aber auch die Werte eines Events mit Werten vorheriger Events verglichen werden. Weiterhin ist es möglich, die Werte des aktuellen Events mit einer Berechnung aus den Werten vergangener Events zu verglichen.

Strategie, nach der die Events ausgewählt werden: In einem Stream aus relevanten und irrelevanten Events ist es manchmal nötig, nur die Events auszuwählen, die auf dem Stream aufeinander folgen. Andererseits sollen in anderen Fällen irrelevante Events ausgeblendet werden, während die relevanten Events nach wie vor nicht verworfen werden.

Kriterien, nach denen die Suche beendet wird: Dadurch, dass ein Event Stream unendlich ist und man so auch unendlich viele irrelevante Events ausblenden kann, ergibt sich bei der Definition des Terminierung der Abfrage auch Unterschiede zu normalen regulären Ausdrücken.

Diese Funktionen werden in der EPL von Esper umgesetzt. Diese EPL ist eine speziell für Esper entworfene Sprache. Esper gibt es einer Variante für .Net und für Java. Ich habe mich für die Sprache Java entschieden, da die Java-Variante, laut der Website [17], am weitesten verbreitet ist.

Das Herzstück von Esper ist die Laufzeitumgebung, an die die einlaufenden Events übergeben werden. Die Laufzeitumgebung übernimmt die Verwaltung der Abfragen und führt auch die Suche auf dem Event Stream aus. Es ist möglich mehrere Event Streams zu verwalten und pro Event Stream mehrere Abfragen zu definieren. Die Definition einer Abfrage kann auch während des laufenden Betriebs erfolgen. Aus den Events des eingehenden Streams, die die Kriterien einer Abfrage erfüllen, können neue Event Streams erzeugt werden. So lässt sich eine Analyse modular aufbauen oder man kann entsprechende Zwischenschritte der Analyse dokumentieren. In der Abbildung 2.1) bekommen die Querys Eins und Zwei den eingehenden Event Stream und speisen beide ihre Ergebnisse in einen neuen Stream. Dieser Stream ist die Eingabe für die Query Drei, die wiederum ihre Ergebnisse an die Querys Vier, Fünf und Sechs weiterleitet. Die Querys werden von Listenern oder Subscribern belauscht, wenn es nötig ist, dass ihre Ergebnisse weiter verarbeitet werden sollen.

Bisher ist noch nicht definiert, was geschehen soll, wenn eine Abfrage Events oder Event-Muster erkannt hat. Diese Aufgabe übernehmen Listener oder Subscriber. In der Laufzeitumgebung werden die erkannten Events übergeben. Die Listener oder Subscriber sind dann für alle weiteren Maßnahmen zuständig. Es können mehrere Listener / Subscriber auf einer

Abfrage definiert werden. Somit ist es auch möglich, verschiedene Maßnahmen durchzuführen, wenn eine Abfrage ein passendes Event findet. Der Unterschied zwischen Listenern und Subscribern besteht darin, dass Listener universal einsetzbar sind, aber eine schlechtere Performance haben. Subscriber können hingegen nur verwendet werden, wenn die Objekte als Plain Old Java Object (POJO) vorliegen und nur ein Subscriber pro Abfrage notwendig ist. Subscriber haben hingegen eine bessere Performance als Listener, da an sie die Objekte direkt und typisiert übergeben werden können.

Weiterhin bietet Esper auch die Möglichkeit, über Variablen die Ausgabe dynamisch zu gestalten. Mit den Variablen kann man zur Laufzeit die Parameter in der Abfrage verändern. So ist es möglich, die Abfragen den aktuellen Gegebenheiten dynamisch anzupassen. Es ist auch möglich, die Events zu verändern. Eine einfache Möglichkeit ist hier, nicht alle Variablen des eingehenden Events zu übernehmen. Man kann aber auch die Ergebnisse aus Berechnungen den Events als Variable hinzufügen.

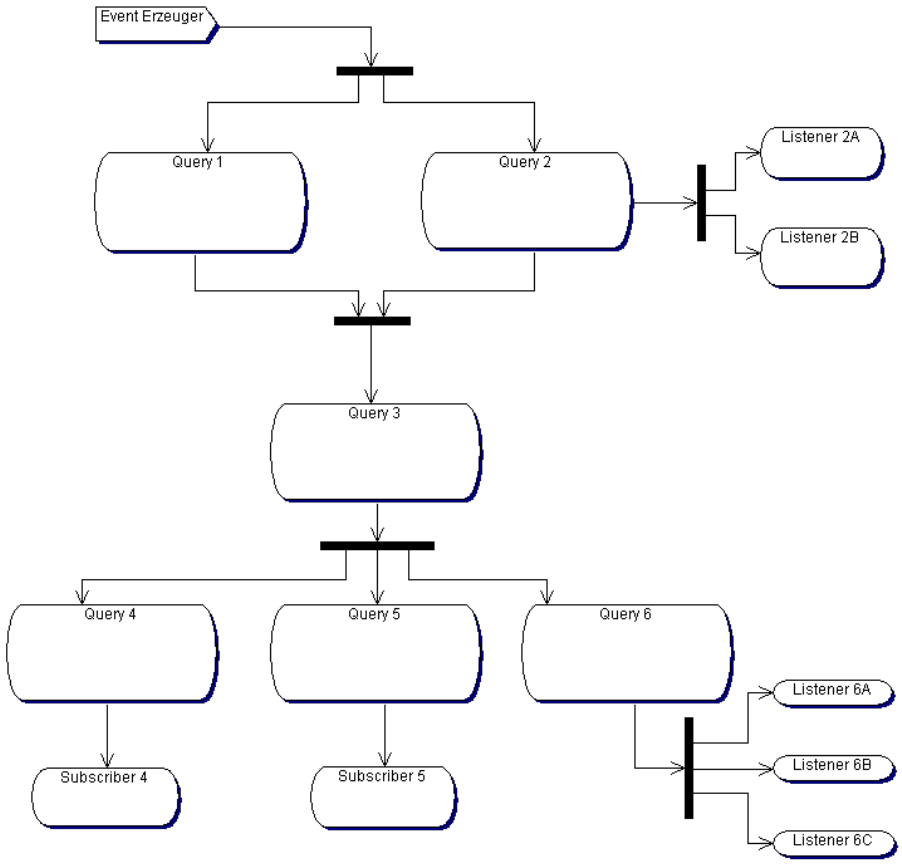

Abbildung 2.1.: Querys in Esper

2.2.2. Repräsentation von Events

Events, die an die Laufzeitumgebung weitergegeben werden, müssen in einem bestimmten Format vorliegen. In der Java-Variante von Esper gibt es drei verschiedene Darstellungen für Events. Beispiele sind im Anhang B.1 zu finden.

- Plain Old-Java Objects (POJO)
  Durch die in den Konventionen beschriebenen Methoden wird von Esper auf die Attribute der Java Objekte zugegriffen. Java-Klassen, die den Konventionen nicht folgen, werden auch als Event betrachtet. Es müssen dann aber bestimmte Einstellungen vorgenommen werden, wenn von den Konventionen abgewichen wird. Die Attribute der POJO können primitive Datentypen, wiederum andere POJO´s oder Arrays enthalten. In den Abfragen von Esper ist es möglich auf diese geschachtelten Attribute direkt zuzugreifen. Es wird von Esper außerdem auch die Polymorphie von Java unterstützt. Beispiel: B.1.1
- java.util.Map
  Hier ist die Darstellung ähnlich wie bei den POJO´s, nur dass die Attribute des Events als Key-Value-Beziehung in einer Map vorliegen. In den Values können andere Objekte, geschachtelte weitere Maps und Arrays enthalten sein. Auf diese tiefere Verschachtelung kann, wie auch bei den POJO´s, in beliebiger Tiefe zugegriffen werden. Auch hier gibt es die Polymorphie aus Java. Beispiel: B.1.2
- org.w3c.dom.Node
  Die Repräsentation kann auch in Extensible Markup Language (XML) als Document Object Model (DOM) vorliegen. Auch hier kann die Struktur der XML Node geschachtelt werden. Um an die tiefer gelegenen Elemente in der Struktur zu gelangen, setzt Esper die Ausdrücke in XPath um. Zum Beispiel wird aus der Abfrage „anItem.aSequence[0]“ die X Path Notation „/event/anItem/aSequence [position()=1]“ Beispiel: B.1.3

2.2.3. Übersicht der Funktionen in Esper

Der Aufbau einer Abfrage in der EPL von Esper orientiert sich an den Abfragen aus der Structured Query Language (SQL).

```
1 select itemName, price from OrderEvent where price < 100
```

Hier sind „price“ und „itemName“ Attribute des Events „OrderEvent“. Vor dem „select“ kann durch die Schlüsselwörter „insert into“ ein neuer Stream erzeugt werden. Die Einschränkungen der Events mit dem Ausdruck „price < 100“ hinter dem Schlüsselwort „where“ sind optional.

Des weiteren stellt Esper die folgenden Funktionen zur Verfügung.

- logische Operatoren (und / oder / nicht)
- Operatoren, die sich auf die Reihenfolge der eingehenden Events beziehen.
- Operatoren, die sich auf eine Zeit beziehen, zu denen die Events ankommen. Diese zeitbezogenen Operatoren können sich auf ein zeitliches Intervall oder einen Zeitpunkt beziehen.
- Definition der Bereiche, in denen die Events für die Abfrage gültig sind
  - Gleitendes Fenster: Zeit, Längen, sortiert, ansteigend, zeitorientiert, externe Zeit Dieses Fenster bewegt sich kontinuierlich immer weiter. Bei dem Fester mit einer definierten Länge fällt das letzte Event heraus, wenn ein neues eintrifft. Bei dem Fenster mit einer festen Zeit wird immer der gewählte Zeitraum von „Jetzt“ in die Vergangenheit betrachtet.
  - Fallendes Fenster: Zeit, Länge und mehrfach ausgelöstes Fenster. Hierbei handelt es sich um Fenster, die aneinandergereiht werden. Für Fenster mit einer festen Anzahl von Events, zum Beispiel zehn, werden die ersten zehn Events betrachtet. Nach dem zehnten Event wird mit dem nächsten Event ein neues Fenster begonnen. Dann beziehen sich die Abfragen nur noch auf die nächsten zehn Events.
  - Es gibt verschiedene Möglichkeiten mehrere Event Streams zu einem Stream zu vereinen. Diese Möglichkeiten sind: Gruppierung, Ansammlung, Sortierung, Filterung und Zusammenführung. Das bedeutet, dass verschiedene Event Streams zu einem Strom vereint werden können.
- Esper ist Multithreadsave und erlaubt, dass unabhängige Esper Engines in der Java VM parallel arbeiten.

Neben den oben aufgeführten Funktionen gibt es noch viele weitere Funktionen, die in Esper zur Verfügung stehen. Alle Funktionen von Esper im Detail zu erklären würde hier zu weit führen.

Um diese Abfragen zu erstellen gibt es in Esper zwei Möglichkeiten. Die erste ist, die Abfrage als String zu erstellen und diesen String dann der Laufzeitumgebung zu übergeben. Bei der Angabe des Events ist es wichtig den Pfad zur Position anzugeben, an der das Event

definiert wird. Die zweite Möglichkeit ist eine objektorientierte Repräsentation der Abfragen. In diesem Fall ist die Abfrage ein Objekt und dieses Objekt stellt Methoden zur Verfügung, durch die die Abfragen erstellt werden können. Man kann aus der String-Repräsentation auf das „Abfrage“-Objekt konvertieren und umgekehrt aus dem Objekt einen String generieren. Alle erstellten Abfragen können von der Laufzeitumgebung abgerufen und verändert werden. Beispiel: B.1.4

2.3. Beschreibung von Anwendungsmodellen

2.3.1. Flugdatenerfassung

Allgemeine Beschreibung

In diesem System wird der Funkverkehr von Flugzeugen aufgefangen. Dabei kann es sich um eine Kommunikation zwischen den einzelnen Flugzeugen handeln oder es ist ein Gespräch zwischen dem Flugzeug und einer Bodenstation. Die Daten werden als Datenpakete von den Peilstationen an die weiterverarbeitende Software übergeben. Die Empfangsstation ist eine empfangende Einheit, ohne weitere Elemente zur Datenverarbeitung. Daher können die Daten des Funkverkehrs von der Empfangsstation nicht unterschieden werden. Die Datenverarbeitung findet an einer zentralen Stelle statt, an der Signale von verschiedenen Antennen zusammenlaufen. Alles, was der richtigen Frequenz entspricht und zu dem entsprechenden Muster passt, wird zu einem Datenpaket weiterverarbeitet. Daher kommen die Daten in einem ständigen, unsortierten Strom bei der weiterverarbeitenden Software an. Es werden ca. 500 Datenpakete pro Sekunde erwartet, die dann verarbeitet werden müssen.

Die zu verarbeitenden Daten

Die eingehenden Signale werden von einem Modul bearbeitet. In diesem Modul wird eine XML-Struktur aus diesen Signalen befüllt. Diese XML-Daten werden dann an die Datananalyse weitergereicht. Innerhalb eines XML-Datensatzes gibt es die folgenden Attribute. Die Reihenfolge innerhalb der XML-Node ist dabei nicht festgelegt.

- Identifikationsnummer
- [weitere Merkmale[1]]
- Ortungsdaten

[1] Hier wurden Merkmale ausgelassen, um die Komplexität zu reduzieren. Die Merkmale haben keine Auswirkungen auf die Analyse

- Liste von:
  - Zeit
  - Absender
  - Liste von:
    - Empfänger
  - Inhalt

Eine Beispiel-Node befindet sich im Anhang A.1

Je nach Qualität des empfangenen Signals, kann es sein, dass manche Attribute nicht mit einem Wert gefüllt sind.

Ziel der Analyse

Das Ziel der Analyse ist es, alle Daten zu entfernen, die für eine weitere Analyse nicht geeignet sind und Daten zu gruppieren. Dass Daten für eine weitere Analyse nicht geeignet sind, kann daran liegen, dass wichtige Datenteile fehlen. Es kann aber auch sein, dass Daten von einer Antenne doppelt aufgefangen werden. Diese fehlerhaften Daten müssen ausgefiltert werden. Um eine weitere Analyse zu ermöglichen, werden die empfangenen Datenpakete in Gruppen eingeteilt. Zu einer Gruppe gehören zwei Gesprächspartner, die miteinander kommunizieren.

Regeln für die Analyse

Dem Nutzer stehen zwei Möglichkeiten zur Verfügung, mit denen die Daten bearbeitet werden können. Es steht ihm frei in welcher Reihenfolge er diese Bearbeitung vornimmt.

Bei der **Reduktion** werden die folgenden Schritte durchgeführt:

1. Wenn in den Datenpaketen die Felder Empfänger und Absender nicht gefüllt sind, werden diese Datenpakete nicht weitergeleitet.
2. Datenpakete mit mehr als einer Informationseinheit und unterschiedlichen Absendern oder Empfängern innerhalb der Informationseinheiten werden nicht weitergeleitet.

3. Wenn verschiedene Datenpakete in einem Zeitfenster die gleichen Absender, Empfänger und Inhaltsdaten haben, werden diese, dem ersten Paket folgenden, Pakete nicht weitergeleitet. Die Inhalte der Ortungsdaten der nicht weitergeleiteten Pakete müssen dem ersten Datenpaket (das zur Weiterleitung bestimmt ist) hinzugefügt werden. Das Zeitfenster, in dem die Datenpakete verglichen werden, ist vom Nutzer wählbar. Ein denkbares Zeitfenster wären zehn Sekunden.

Bei der **Gruppierung** werden folgende Schritte durchgeführt:

1. Zur Vereinfachung des Programmablaufes werden die Datenpakete, die mehr als eine Informationseinheit haben, auf die Größe Eins reduziert. Nur der erste Eintrag der Liste bleibt erhalten. Diese Vereinfachung könnte bei der Weiterentwicklung des Programms entfallen.

2. Wenn bei zwei Datenpaketen ein Empfänger einem Sender direkt antwortet, werden diese Datenpakete einer Gruppe zugeordnet. Bei dem ersten Paket muss der Empfänger mit dem Sender des zweiten übereinstimmen und der Sender des ersten Pakets muss mit dem Empfänger des zweiten Pakets übereinstimmen.

   Zum Beispiel:
   Es gibt die Teilnehmer A, B und C und es wird die folgende Kommunikation empfangen:

   1. A an B
   2. B an C
   3. B an A

   In diesem Beispiel gehören die Funksprüche 1) und 3) in eine Gruppe und 2) in eine weitere Gruppe.

   Gruppennummern werden ab dem Start der Bearbeitung mit Eins initialisiert und dann für jede Gruppe um Eins inkrementiert. Datenpakete, bei denen es keine Antwort innerhalb einer festgesetzten Zeit gibt, haben die Gruppengröße Eins. Die Zeit, innerhalb der die Gesprächspartner einander zugeordnet werden, ist vom Nutzer wählbar. Ein denkbares Zeitfenster wären 15 Minuten.

2.3.2. Telefondatenanalyse

Allgemeine Beschreibung

Es werden von Antennen die Funksignale von Mobilfunkgeräten (Handys) aufgefangen. Ein Handy stellt den Kontakt mit einer Sendestation her. Über diese Sendestation werden nach

dem ersten Kontakt alle Telefonanrufe des Handys geleitet. Dies geschieht, solange das Handy im Empfangsbereich dieser Sendestation bleibt. Die Anrufe teilen sich in mehrere Datenpakete auf. Die Datenpakete, die zwischen dem Handy und der Station ausgetauscht werden, kann auch jede andere Antenne empfangen, die sich innerhalb der Reichweite der Signale befindet. Diese Antennen stehen an verschiedenen Orten und empfangen viele Funksignale. Aus diesen Funksignalen sollen nun weitere Informationen extrahiert werden.

Die zu verarbeitenden Daten

Die Funksignale liegen als Java Objekte vor, die in eine Map umgewandelt werden. In dieser Map sind die Keys und Values Strings. Die Daten bestehen aus:

- Daten über Eigenschaften des Funkspruchs
- Global Positioning System (GPS)-Daten
- Nummern der Teilnehmer

Eine genaue Beschreibung der Daten ist im Anhang A.2 nachzulesen.

Ziel der Analyse

Es gibt verschiedene Ziele, die man in diesem Anwendungsgebiet verfolgen kann. Im Folgenden werden drei Beispiele angegeben.

1. In einem Gebiet, das überwacht werden soll, gibt es verschiedene Zonen. Jede dieser Zonen hat einen bestimmten Sicherheitsstatus. Der Status zeigt an, wie gefährlich es ist, sich durch dieses Gebiet zu bewegen. Das gesamte Gebiet wird ständig überwacht und der Sicherheitsstatus wird entsprechend dieser Überwachung angepasst. Eine Art der Überwachung ist das Auffangen von Handysignalen. Ziel der Analyse ist es herauszufinden, wie aktiv (= unsicher) eine bestimmte Zone ist. Dazu ist es nötig Bereiche zu erkennen, in denen es hohe Aktivitäten von Handys gibt. Diese Ansammlungen von Handys und die dazugehörigen Gespräche weisen dann auf mögliche Lager in einem ländlichen Gebiet hin, in dem sich eine Gruppe von Personen aufhält.
2. Es sollen durch die Analyse von Handygesprächen Gefahren aufgedeckt werden. Bei der Überwachung der Handysignale werden sehr viele Handygespräche aufgefangen. Ein Problem bei der Analyse einer so hohen Datenflut ist, dass man nicht weiß, wo man mit der Analyse beginnen soll. Daher werden die eingehenden Daten in Klassen eingestuft, zu welchem Grad die Anrufe einen verdächtigen Inhalt haben könnten. Ziel der Analyse ist es also, die vorliegenden Daten in Kategorien einzuordnen. Je höher

ein Anruf eingestuft wird desto verdächtiger ist er. Durch diese Einstufung ist es einfacher mit einem sinnvollen (wahrscheinlich gefährlichen) Anruf die Analyse der Daten zu beginnen.

3. Eine Vorstufe der oben (Beispiel 2) genannten Einstufung der Daten ist, herauszufinden, welche Handynummern eine hohe Aktivität aufweisen. Bei dieser Analyse sind nur die Datenpakete von diesen besonderen Nummern interessant, da bei einer hohen Aktivität ein höherer Verdacht besteht. Diese Gespräche sollen gespeichert werden und die Handynummern sollen für eine weiterführende Analyse bereitgestellt werden.

Vorgaben zur Analyse

zu Bsp 1: Hier geht es darum festzustellen, wo es Ansammlungen von Gesprächen gibt. Die folgenden Parameter werden in die Berechnung miteinbezogen:

Geometrischer Abstand: Der Abstand wird über die mitgelieferten GPS-Daten ermittelt. Es ist der Abstand der einzelnen Gesprächsteilnehmer zueinander.

Häufigkeit der Nutzung pro Stunde: Handys, die innerhalb einer Stunde sehr aktiv sind, haben eine höhere Gewichtung. Alle Gespräche werden in einer Gruppierung zusammengefasst.

Die letzten 48 Stunden: Die betrachteten Events werden innerhalb eines fließenden Fensters von 48 Stunden betrachtet. Gespräche, die älter sind, verfälschen die Analyse, da sich die Sicherheitslage ständig ändert.

zu Bsp 2: Die folgenden Faktoren fließen in die Einteilung der Klassen ein:

Häufige Nutzung: Telefonnummern sind verdächtig, wenn sie innerhalb von 36 Stunden sehr stark genutzt werden.

Nutzung in verdächtigen Gebieten: Eine weitere Eigenschaft, die ein Handy verdächtig erscheinen lässt, ist eine Nutzung in einem als verdächtig eingestuftem Gebiet.

Kontakt mit verdächtigen Nummern: Es ist verdächtig, wenn das Ziel des Anrufes eine schon bekannte eingestufte Nummer ist.

Inhalt hat eine nutzbare Menge an Daten: Eine sinnvolle Analyse ist nur möglich, wenn die Inhalte der Nutzdaten für eine Analyse ausreichen. Hier wird nicht der Inhalt analysiert, sondern die Menge der Nutzdaten. Wenn die Gesprächsdaten des Funksignals zu gering sind, ist es nicht wichtig dieses Gespräch in eine spätere Analyse einfließen zu lassen.

zu Bsp 3: Hier müssen die Häufigkeiten überwacht werden. Es ist das Ziel herauszufinden, welche Handys sehr aktiv sind. Alle Handys, die ein gewisses Maß an Aktivität überschreiten, sind interessant. Diese Informationen sollen für spätere Analysen zur Verfügung stehen. Bei dieser Analyse ist nicht die Komplexität das Problem, sondern der Zeitraum. Der betrachtete Zeitraum umfasst hier vierzehn oder mehr Tage.

3. Analyse von Event Stream Algorithmen

Im Nachfolgenden werden ausgewählte Datamining Algorithmen untersucht. Für die Auswahl der Algorithmen ist zu beachten, dass beim Event Stream Processing die Ressourcen begrenzt sind. Daher müssen spezielle Algorithmen gefunden werden, die für dieses Anwendungsgebiet geeignet sind. Als Grundlage für die Bewertung der gefundenen Algorithmen werden die Ein- und Ausgabedaten definiert und das Event Stream Processing am Beispiel der Anwendungsmodelle beschrieben. Dann wird untersucht, inwieweit sich die Algorithmen für die gegebenen Anwendungsmodelle eignen. Am Ende dieses Kapitels werden die Algorithmen in einem allgemeinen Kontext bewertet.

3.1. Beschreibung relevanter Event Stream Algorithmen

3.1.1. Beschreibung der Ausgangssituation

Bei den Anwendungsmodellen aus Abschnitt 2.3 geht es darum, Informationen aus den ankommenden Daten zu gewinnen oder ihnen weitere Informationen hinzuzufügen. In den bestehenden Lösungen werden Datenbanken eingesetzt, um alle Daten zu speichern. Erst nach dem Speichern der Daten wird das Datamining auf den Daten durchgeführt.

Aufgrund der eingeschränkten Ressourcen können die Datamining-Verfahren der Datenbank-Anwendung nicht für das Event Stream Processing einsetzten werden. Im Folgenden werden nun Datamining Algorithmen genauer untersucht, um festzustellen, wie sich diese Algorithmen für die Anwendungsmodelle eignen.

Um Informationen aus einem Datenstrom zu gewinnen, gibt es viele verschiedene Algorithmen. In [16] werden die folgenden Bereiche des Datamining genannt:

- **Konzept/Klassen-Beschreibung: Charakterisierungen und Unterscheidungen** Daten können durch Konzepte oder Klassen beschrieben werden. Um Daten bestimmten Konzepten/Klassen zuzuordnen gibt es die Möglichkeit den besonderen Charakter eines Konzeptes zu definieren oder die Unterschiede zu beschreiben, die

zwischen den verschiedenen Konzepten bestehen. Die Unterschiede werden durch einen Experten festgelegt.

- **Frequenzmuster, Verbindungen und Verknüpfungen** Hierbei werden Muster und Zusammenhänge innerhalb der Daten festgestellt. Dabei kann es sich um Muster innerhalb einzelner Daten oder Muster auf einer Gruppe von Daten handeln. Die Muster werden mit einer Wahrscheinlichkeit angegeben, mit der sie auftreten. Eine Person hat beispielsweise das Alter von 20 - 29 Jahren und bezieht ein Gehalt von 2000 - 3000 Euro, dann kauft diese Person zu 70% Unterhaltungselektronik.
- **Classification und Vorhersage** Bei der Klassifikation soll ein Modell oder eine Funktion gefunden werden, um Daten einzuschätzen, die noch nicht klassifiziert sind. Hierfür wird aus einem Satz von Trainingsdaten ein Modell erstellt, das anschließend zur Analyse von weiteren Daten eingesetzt wird. In diesem Fall gibt es keinen Experten im Gegensatz zur Konzept/Klassen Beschreibung.
- **Ausreißer-Analyse** In vielen Datamining Algorithmen werden Ausreißer als Sonderfälle abgetan und ignoriert. In manchen Anwendungen sind gerade diese Ausreißer interessant. Bei der Ausreißer-Analyse sollen die Datensätze gefunden werden, die sich von der Masse absetzen.
- **Clustering** Unter Clustering versteht man die Einteilung der Daten in Gruppen. Es gibt bei Clustering keine Trainingsdaten, da beim Clustering die Klassen selbst nicht bekannt sind. Ziel der Analyse ist es, die Grenzen zwischen den Daten zu finden. Dabei sollen die Gemeinsamkeiten innerhalb der Klasse maximiert werden und die Gemeinsamkeiten zwischen den Klassen minimiert.
- **Evolutionsanalyse** Bei der Evolutionsanalyse werden Algorithmen aus den vorher genannten Bereichen eingesetzt. Es ist bei der Evolutionsanalyse nicht das Ergebnis zu einem bestimmten Zeitpunkt interessant, vielmehr liegt der Fokus auf der Veränderung der Ergebnisse über einem bestimmten Zeitraum.

In diesem sehr breit gefächerten Bereich des Datamining sollen Algorithmen gefunden werden, welche die besondere Situation des Event Stream Processing berücksichtigen.

Eine Untersuchung der Anwendungsmodelle hat ergeben, dass die folgenden Operationen durchgeführt werden.

Zusammenlegen von Events in Gruppen: Es werden Events in Gruppen eingeteilt. Die Einteilung erfolgt aufgrund von Gemeinsamkeiten in den Attributen der Events.

Einteilung von Signalen in Gruppen: Eingehende Signale sollen auf bekannte Gruppen verteilt werden. Diese Einteilung erfolgt nach vorher festgelegten Regeln

Erkennung von Signalen, die am häufigsten Auftreten: Signale, die aus der Menge durch ihr häufiges Auftreten heraus stechen, sollen identifiziert werden.

Aus diesen Beschreibungen ergeben sich die folgenden Kategorien für die Suche nach Event Stream Algorithmen:

- Clustering
- Classification
- Frequency Counting, als spezielle Form der Evolutions- und Ausreißeranalyse

Diese Bereiche des Datamining und Vertreter der Event Stream Algorithmen werden im Nachfolgenden beschrieben.

3.1.2. Clustering

Bei der Einteilung der Daten in Gruppen geht es um die Frage, wann ein Datenobjekt der einen bzw. der anderen Gruppe zugesprochen wird. Algorithmen für diese Art des Datamining nennt man Clusteringalgorithmen.
Beim Clustering will man Untergruppen innerhalb einer übergeordneten Gruppe finden. Die übergeordnete Gruppe kann dabei die Menge aller Daten sein, es kann sich dabei auch um eine Teilmenge der Daten handeln, zum Beispiel eine Gruppe von Daten, die bei einem vorherigen Clusteringverfahren zusammengefasst wurde. Es werden beim Clustering die Objekte zusammengefasst, die innerhalb eines bestimmten Abstandes zueinander liegen. Bei diesem Abstand kann es sich um eine Distanz von Vektoren handeln oder um ein anderes Maß für die Eigenschaften des Datenobjekts. Es ist nun entscheidend, ob man die Anzahl der Cluster von vornherein festlegt oder der Algorithmus die Anzahl der Cluster selbst bestimmen soll. In diesem Fall legt man einen maximalen Abstand fest, in dem alle Daten zueinander liegen.
Beim Erstellen der Cluster kommt es darauf an festzustellen, wo die Grenzen zwischen den einzelnen Clustergruppen verlaufen. Der Algorithmus muss diese Grenzen mit Hilfe der Abstandsfunktion finden.
Für die Lösung des Problems gibt es zwei Verfahren, das hierarchische und das partitionierende Clusteringverfahren. Bei den hierarchischen Verfahren werden alle Objekte als ein Cluster mit der Größe Eins angesehen. Dann werden die Cluster nach und nach zusammengefasst, bis die Anzahl der Cluster der gegebenen Anzahl der Cluster entspricht oder alle Datenobjekte innerhalb der gegebenen Distanz liegen. Bei den partitionierenden Clusteringverfahren werden alle Objekte als ein einzelnes, riesiges Cluster aufgefasst und dieses Cluster wird nach und nach geteilt. Siehe hierzu auch [14].

Das Konzept des Event Stream Processing ist, dass die Events nach und nach eintreffen. Das bedeutet, dass in einem bestehenden Clustersystem diese neuen Events hinzugefügt werden müssen. Daher wurde bei der Suche das partitionierende Clustering nicht weiter verfolgt. Ein Vertreter des hierarchischen Clustering auf Streams wird im Folgenden beschrieben.

Framework for Clustering Evolving Data Streams

Das „Framework for Clustering Evolving Data Streams“ [1] ist ein Algorithmus, der für das Clustern einer hohen Datenflut ausgelegt ist. Um die Daten den Clustern zuzuordnen, wird das Verfahren in zwei Schritte aufgeteilt.

Online Abschnitt: Es werden Micro-Cluster erzeugt. Die Micro-Cluster werden unabhängig von externen Angaben gebildet.

Offline Abschnitt: In dem Offline Abschnitt wird das eigentliche Clustering nach den Vorgaben des Nutzers durchgeführt. Für das Clustering wird der Stand der Micro-Cluster aus dem Online Abschnitt genutzt. Aus den Micro Clustern werden nun Macro-Cluster gebildet.

Der Online-Abschnitt wird mit einem k-Means-Algorithmus[1] initialisiert. Bei der Initialisierung werden so viele Cluster gebildet, wie es der Speicher erlaubt. Dieser Online-Abschnitt geht automatisiert von statten. Die Anzahl der Cluster bleiben auf diesem Maximalwert festgesetzt. Die Anzahl der Cluster wird wahrscheinlich viel kleiner sein, als die Anzahl aller eingehenden Datensätze, aber größer als die Anzahl der endgültig geforderten Cluster. Jeder neue Datensatz wird einem bestehenden Cluster zugeordnet oder, wenn ein neuer Cluster erstellt werden muss, werden zwei andere Cluster zusammengeführt. Dadurch bleibt die Anzahl der Cluster immer gleich.

Ein Micro-Cluster ist eine zeitlich begrenzte Erweiterung des Cluster Feature Vector (CF) (Abgekürzt: Cluster Feature Vector Temporal (CFT)) [21]. Der CF ist eine Zusammenfassung von Kennzahlen.
$CF = \left\{ N, \vec{S}, SS \right\}$

N Anzahl der Punkte im Cluster

$\vec{S}$ Summe der Punkte

SS Die Quadratsumme der Punkte

[1] siehe Glossar

Beim Clustering gibt es verschiedene Möglichkeiten die Distanz zwischen zwei Objekten zu bestimmen.

$$\vec{X_0} = \frac{\sum_{t=1}^{N} \vec{X_t}}{N} \tag{3.1}$$

$$R = \left(\frac{\sum_{i=1}^{N} (\vec{X_i} - \vec{X_0})^2}{N}\right)^{\frac{1}{2}} \tag{3.2}$$

$$D = \left(\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (\vec{X_i} - \vec{X_j})^2}{N(N-1)}\right)^{\frac{1}{2}} \tag{3.3}$$

$\vec{X_0}$ Zentrum des Clusters

R ist die durchschnittliche Distanz der Punkte zum Zentrum

D ist die durchschnittliche paarweise Distanz innerhalb des Clusters

Aus den oben genannten Kennzahlen des CF kann man die verschiedenen Distanzen erzeugen. Außerdem ist es möglich, die CF untereinander zu addieren. Daher wird im Online Abschnitt der CF als Grundlage genommen. Dem CF wird hier die zeitliche Komponente hinzugefügt.
Ein Micro-Cluster besteht somit aus dem Tupel: CFT = $\left\{\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n\right\}$

$\overline{CF2^x}$ Je Dimension die Datenwerte zum Quadrat zur Summe

$\overline{CF1^x}$ Je Dimension die Datenwerte zur Summe

$CF2^t$ Summe der Zeitstempel zum Quadrat

$CF1^t$ Summe der Zeitstempel

n Anzahl der Datenwerte

Um über die Vergangenheit Aussagen treffen zu können, werden die Micro-Cluster als Momentaufnahmen abgespeichert. Die Momentaufnahmen der Micro-Cluster werden mit logarithmisch abnehmender Rasterung gespeichert. Momentaufnahmen, die nicht mehr im Raster liegen, werden gelöscht. Das Muster zur Abspeicherung der Momentaufnahmen ist in [1] genau beschrieben.

Durch den Online Abschnitt ergibt sich ein extrem reduzierter Speicherbedarf für die Analysen im zweiten Schritt. Der zweite Schritt wird von einem Analysten durchgeführt, der auch den zeitlichen Horizont der Bearbeitung fest legt. Neben der Erstellung von Clustern ist durch die abgespeicherten Momentaufnahmen eine Analyse des Verlaufs der Daten möglich.

3.1.3. Classification

Classification bedeutet das Erlernen einer Funktion, die Daten auf verschiedene vordefinierte Klassen abbildet [11]. Ein Classification Algorithmus durchläuft in der Regel die folgenden Schritte:

1. Erlernen der Funktion mit Trainingsdaten
2. Überprüfen / Testen der erlernten Funktion
3. Anwenden der Funktion auf neue Daten
4. Optional: Anpassen der Funktion während der Anwendung

Formal ausgedrückt bedeutet Classification die Umsetzung der Funktion $\Phi : D \times C \rightarrow \{T, F\}$, wobei D die Menge der zu klassifizierenden Objekte ist und C die Menge der Kategorien ist. Das Ergebnis dieser Funktion $\Phi(x, c)$ ist T (True) wenn $x \in D$ zu der Kategorie $c \in C$ gehört und F (False) wenn das nicht der Fall ist.
Ein Problem bei der Classification ist die exakte Zuordnung eines Objektes zu einer Klasse. In vielen Fällen ist die Entscheidung subjektiv. Das Problem besteht in den verschiedenen Kontexten, in denen eine Information beurteilt werden soll.
Ein Beispiel ist die Analyse von Telefondaten. Ein Telefongespräch von einem Handy, das in Hamburg aufgefangen worden ist, kann der Gruppe „Hamburger Gespräche“ zugeordnet werden. Dieses Handy gehört nun einem Amerikaner aus New York, somit wäre auch eine Zuordnung zu „New Yorker Gespräche“ möglich.
Bei der Classification gibt es die Unterscheidung zwischen Single Label und Multi Label. Bei Single Label gehört ein Objekt immer zu einer Klasse und nur zu dieser Klasse. In dem Beispiel oben ist es entweder ein „New Yorker Gespräch“ oder ein „Hamburger Gespräch“, aber nicht beides. Das Gespräch muss aber in eine der beiden Kategorien eingeordnet werden.
Eine spezielle Form der Single Label Classification ist die Binary Classification. Bei dieser Form gibt es immer zwei Möglichkeiten der Klassifizierung. Bei diesen Möglichkeiten ist die eine Klasse das Gegenteil der anderen Klasse.
Zum Beispiel „Hamburger Gespräche“ und „Nicht Hamburger Gespräche“. Bei der Multi Label Classification ist es erlaubt, dem Handy beide Klassen zuzuweisen, damit das Gespräch bei weiteren Analysen über Hamburg auftaucht und bei Analysen über New York auch zur Verfügung steht.
Es ist bei Multi Label auch erlaubt dem Gespräch keine Kategorie zuzuteilen. Dieses könnte nötig sein, wenn die Informationen für die Klassifizierung nicht vorliegen, oder es gefordert ist, dass nur eindeutige Klassifizierungen durchgeführt werden sollen. Siehe hierzu auch [20].

Die erlernte Funktion beim Classification kann durch verschiedene Systeme dargestellt werden.

- Entscheidungsbaum
- Support Vector Maschine
- künstliches neuronales Netz

ein Entscheidungsbaum zeichnet sich dadurch aus, dass er die Daten in verschiedene Streams aufteilt. Diese Aufteilung erfolgt durch Überprüfung der Attributswerte der Daten. Dieses Vorgehen stimmt mit den Möglichkeiten überein, die Esper zur Verfügung stellt. Daher wurde speziell nach Event Stream Algorithmen gesucht, die einen Entscheidungsbaum nutzen, um die erlernte Funktion darzustellen.

Im nachfolgenden Abschnitt werden Classification Algorithmen beschrieben, die sich für Streams eignen. Bei der Arbeit auf Streams sind die knappen Ressourcen im Algorithmus zu beachten. Um dieser Einschränkung gerecht zu werden, beschränken sich die nachfolgenden Algorithmen auf einen Teil aller möglichen Anwendungsgebiete. Durch diese Beschränkung ist es möglich die Berechnungen effektiver durchzuführen und mit den reduzierten Ressourcen auszukommen. Da es in den Anwendungsmodellen um Frequenzen und geografische Daten geht, wurden aus diesen beiden Bereichen Algorithmen analysiert. Zum Schluss wird noch ein Algorithmus beschrieben, der Heuristiken nutzt, um mit den begrenzten Ressourcen auszukommen.

Algorithmus für die Klassifikation von Frequenzen

Arbitrary Window Stream mOdeling Method (AWSOM) ist ein Algorithmus für die automatische Erkennung von Mustern in Sensordaten. Dieser Algorithmus nutzt eine bestimmte Darstellung der Daten, um das erhöhte Datenaufkommen zu bewältigen.

Der Algorithmus wurde mit Blick auf die folgenden Ziele entwickelt:

- Es soll keine menschliche Komponente für die Analyse nötig sein
- Es sollen periodische Muster erkannt werden
- Die Daten sollen den Algorithmus nur einmal durchlaufen
- Der Speicher ist begrenzt
- Es sollen einfache aber starke Muster erkannt werden
- Es soll jederzeit eine Vorhersage möglich sein

Zur Darstellung der Events des Sensors wird die Diskrete Wavelet-Transformation (DWT)[2] genutzt. Mit dieser Transformation ist es möglich Berechnungen schnell durchzuführen [19]. Die Darstellung der Daten als Wavelet reduziert die Vielfalt der Anwendungsgebiete dieses Algorithmus auf das Datamining in Frequenzen.

Der Algorithmus unterstützt lange Zeiträume, in denen Events eingehen um auf Muster überprüft zu werden. Dieses wird vom Algorithmus erreicht, indem er nur einmal die Events durchläuft und die Berechnungen mit Wavelets sehr effektiv durchgeführt werden können. Die langen Zeiträume können auch ein endloser Stream sein. Daher ist der Algorithmus für die Anwendungsbeispiele geeignet.

Eine weitere Maßnahme, um die Effizienz des Algorithmus zu steigern, ist die Reduktion der relevanten Merkmale auf eine fest vorbestimmte Anzahl. Durch diese Reduktion wird in kauf genommen, dass einige wenige Events nicht richtig zugeordnet werden können. Die Geschwindigkeit der Zuordnung wird durch diese Maßnahme erhöht. Der Algorithmus ist robust gegenüber einer kurzfristigen Datenflut. So eine Datenflut kann auftreten, wenn viele Signale an der Antenne oder den Antennen eintreffen.

Die Schwerpunkte des Algorithmus sind:

Repräsentation der Information: Da der Speicher im Event Stream Processing begrenzt ist, werden DWT´s eingesetzt.

Struktur der Korrelation: In den Wavelets lassen sich durch einfache Vergleiche Muster erkennen. Es ist außerdem möglich mit den Wavelets Muster über Zeiträume hinweg zu erkennen und die Sensordatenflut zu bewältigen.

Modell für die Korrelation: Um verschiedene Korrelationen in der Wavelet Domäne zu finden, wird in dem Algorithmus ein lineares Regressionsmodell genutzt.

AWSOM ist durch das Wavelet als Datenrepräsentation nur für das Erkennen von Mustern auf Frequenzen ausgelegt. Das schränkt die Einsetzbarkeit des Algorithmus stark ein. Innerhalb seiner Domaine sind die Zeit und Speicherkomplexität des Algorithmus sehr gut im Vergleich zu Clusteringverfahren, die nicht auf Streams arbeiten.

Algorithmus für Klassifikation räumlicher Daten

Events, die sich auf die Darstellung eines Raumes oder von Räumlichkeiten beziehen, sind in der Regel sehr groß. Aus diesem Grund lassen sich diese Daten nur schwer klassifizieren. Durch die Darstellung der Daten in einem Peano Count Tree (P-Tree)[3] ist es möglich

[2] siehe Glossar
[3] siehe Glossar

mit diesen Events schnelle Berechnungen durchzuführen. Dadurch wird das Erzeugen des Klassifizierers vereinfacht.

Durch den besonderen Aufbau eines P-Tree´s können die P-Tree´s mit den Operationen UND, ODER und NICHT sehr schnell kombiniert werden.
Eine weitere Verbesserung ist eine Glättung der Daten. Dabei werden die Events in dem P-Tree durch Reduzierung von 1 auf 0 und von n-1 auf n geglättet. Das Bedeutet, dass in einem Knoten der zweiten oder dritten Ebene, in dem nur ein Kind eine Eins aufweist, alle Kinder nicht mehr angegeben werden.

Für die Classification wird der Iterative Dichotomiser 3 (ID3)-Algorithmus[4] genutzt [4]. Aufgrund der speziellen Eventstruktur, kann mit dem ID3 Algorithmus der Entscheidungsbaum erstellt werden, da die Berechnungen eine geringe Zeit-Komplexität aufweisen. Der Entscheidungsbaum, der durch den ID3 Algorithmus erzeugt wird, entspricht wiederum dem Event Stream Processing. Weiterführende Informationen sind in [9] enthalten.

Very Fast Decision Tree Learner

Der Hoeffding Tree ist ein Algorithmus für ein Lernverfahren eines Entscheidungsbaums. In dem Algorithmus werden die Daten sequenziell gelesen und direkt verarbeitet. Dadurch ist dieses Lernverfahren speziell für die Arbeit auf einem Datenstrom geeignet. Very Fast Decision Tree learner (VFDT) ist eine Verbesserung des Hoeffding Trees.

Das Hauptproblem ist die Wahl des Spaltungsattributes über einem Knoten. Hierzu werden eine bestimmte Anzahl von Beispielen für eine Auswertung benötigt. Dabei wird mit den bisher eingelesenen Beispielen der Informationsgehalt berechnet, um das Spaltungsattribut auszuwählen. Um die genaue Anzahl der notwendigen Beispiele zu bestimmen, wird der Hoeffding-bound genutzt.

R Menge alle Variablen $\subset \mathbb{N}$

r Variable $\in R$

n Anzahl der bisherigen Beobachtungen mit der Variable

$\overline{r}$ Informationsgehalt von r

Der Hoeffding-bound besagt, dass mit einer Wahrscheinlichkeit von $1-\delta$ der echte Informationsgehalt von r höchstens um ϵ abweicht, wobei

$$\epsilon = \sqrt{\frac{|R|^2 \ln(1/\delta)}{2n}}$$

[4] siehe Glossar

So kann für ein gegebenes δ der benötigt Informationsgehalt einer Variable bestimmt werden, damit diese als Spaltungsattribut ausgewählt werden kann. Dieses vorgehen unterscheidet sich zum ID3 Algorithmus dadurch, dass hier nicht bekannt ist, wie viele Beispiele in der Zukunft noch eingelesen werden. Das Spaltungsattribut muss mit den bisherigen Informationen ausgewählt werden, auch wenn es sich in der Zukunft zeigen sollte, dass diese Wahl nicht optimal war.

Im Anhang B.2 ist der Pseudocode für den Hoeffding Tree aufgeführt.

Verbesserungen des Hoeffding Trees sind:

- Wenn zwei Attribute eine sehr ähnlichen Informationsgehalt haben, kann eins der beiden Attribute ausgelassen werden.
- Die neue Berechnung des Informationsgehalts nimmt viel Zeit in Anspruch. Daher kann ein Minimum an verarbeiteten Knoten festgelegt werden, bis der Informationsgehalt neu erstellt wird.
- Sollte der Speicher zur Berechnung nicht ausreichen, werden die schlechtesten Blätter des Baumes entfernt, um Platz für neue Blätter zu schaffen.
- Schlechte Attribute können ebenfalls entfernt werden.
- Der Baum kann mit einem konventionell erstellten Baum initialisiert werden. Hierfür eignet sich beispielsweise der ID3 Algorithmus. Dadurch wird die Anfangsphase des Algorithmus verbessert.
- Sollten die Daten zu langsam ankommen, können schon durchlaufene Datensätze erneut eingelesen werden, um die Genauigkeit des Baums zu erhöhen.

Dieses wird detailliert in [10] ausgeführt.

3.1.4. Frequency Counting

Beim Frequency Counting sollen die Häufigkeiten von Daten festgestellt werden. Hierbei ist es in der Regel uninteressant, Daten mit geringer Häufigkeit zu erfassen. Vielmehr will man nur die Daten erkennen, die eine gewisse Anzahl übersteigen. In einer SQL-Abfrage würde der Code so aussehen:

```
1 SELECT R.EventType, COUNT(*) FROM R GROUP BY R.EventType
2         HAVING COUNT(*) > s |R|
```

- R ist der Stream
- Eventtyp ist das gesuchte Event

- s ist die vorgegebene Häufigkeit, ab der die Daten ausgegeben werden sollen
- $|R|$ ist die Länge des Streams, der bisher erfasst wurde

Beim Frequency Counting auf Streams bestehen verschiedene Unterschiede zu den herkömmlichen Ansätzen.
Bei Streams muss der Algorithmus mit nur einem Durchlauf der Daten auskommen. Das Resultat wird innerhalb einer kurzen Zeit erwartet.

Approximate Frequency Counts

In diesem Algorithmus wird der besondere Umstand der Streams beachtet.

- Es werden die Daten nur einmal durchlaufen
- Der Speicherbedarf ist so gering wie möglich
- Ein Ergebnis steht zu jedem Zeitpunkt zur Verfügung

Der Algorithmus sucht nun alle Elemente in einem Datenstrom, die häufiger vorkommen, als eine gegebene Schranke. Um den Speicherbedarf zu reduzieren und es zu ermöglichen mit nur einem Durchlauf die entsprechenden Elemente zu identifizieren, wird eine Toleranz angegeben, um die sich die Schranke nach unten verschieben kann. Wenn zum Beispiel Elemente gefunden werden sollen, die 0,1 % der bisher betrachteten Daten ausmachen und die Toleranz soll 0,01 % betragen, dann werden alle Elemente ausgegeben, die mindestens 0,1 % aller bisher betrachteten Elemente ausmachen und es werden keine Elemente ausgegeben, die 0,09 % oder einen kleineren Anteil an der Menge haben. Alle Elemente mit einem Anteil zwischen 0,09 % und 0,1 % könnten ausgegeben werden, es ist aber unsicher aufgrund der Einschränkungen des Algorithmus. Ob diese Toleranz akzeptabel ist, muss bei jeder Anwendung überprüft werden.

Das bedeutet, dass der Algorithmus folgende Zusicherungen machen kann:

1. Alle gefundenen Elemente sind sicher auch Elemente, die die gegebene Grenze überschreiten. Oder anders ausgedrückt, es gibt keine falschen Negative.
2. Kein Element, dessen Frequenz unter der gegebenen Schranke abzüglich der Toleranz liegt, wird ausgegeben. Anders ausgedrückt bedeutet es, dass es keine falschen positiven Elemente aus dem Bereich unter der Schranke gibt.
3. Die ermittelte Frequenz der ausgegebenen (erkannten) Elemente ist nie zu hoch angegeben und beträgt nie weniger als die festgelegte untere Schranke .

Siehe hierzu auch [18].

3.2. Definition der Eingabe und Ausgabedaten

Um die gefundenen Algorithmen bewerten zu können, müssen die Vor- und Nachbedingungen klar definiert sein.

Vorbedingung: Art, Struktur und Aufkommen der Daten, die beim Event Stream Processing ankommen.

Nachbedingung: Art, Struktur und Aufkommen der Daten, die nach dem Durchlaufen des Event Stream Processing ausgegeben werden.

3.2.1. Eingabedaten

Die eingehenden Daten haben folgende Eigenschaften:

Events treten mit hoher Frequenz auf: Eine hohe Frequenz der ankommenden Events bedeutet hier, dass über 100.000 Events in der Sekunde eintreffen können.

Events haben eine unregelmäßiger Dichte: Eine unregelmäßige Dichte ist dann der Fall, wenn die Anzahl der eintreffenden Events pro Sekunde schwankt. Diese Schwankungen hängen davon ab, wie die Events erzeugt werden. Wenn bei den Funkdaten keine Flugzeuge in der Luft sind, dann gibt es auch keine Daten zu empfangen. Gibt es viele Flugzeuge in der Luft, dann gibt es viele Funksprüche und viele Events.

Events sind typisiert: Eine wichtige Voraussetzung für das Gewinnen von Informationen auf Events ist eine Typisierung der Events. Bei der Analyse der Events muss auf die entsprechenden Attribute zugegriffen werden. Die Attribute müssen daher vor der Analyse fest stehen. Dieses ist aber nur möglich, wenn die Events typisiert sind.

Events sind unvollständig: Auch wenn die Events typisiert sind, müssen die Attribute des Events nicht vollständig gefüllt sein. Es kann sein, dass ein Teil der Attribute keinen Wert aufweist. Wenn eine Funkantenne ein unvollständiges Signal aufgefangen hat, dann ist zum Beispiel das Attribut „Sender“ leer.

3.2.2. Ausgabedaten

Das Ergebnis von Esper und dem Mining-Prozess hat unterschiedliche Ausprägungen. Die folgenden Eigenschaften können, müssen aber nicht auf jede Anwendung zutreffen:

Es gibt keine Ausgabedaten: Es müssen nicht immer Daten bei einem Mining-Prozess entstehen. Es kann auch sein, dass nur weitere Prozesse aufgrund der Ergebnisse angestoßen werden. Auch wenn der Fokus auf den folgenden Prozessen liegt, ist es in der Regel so, dass die Events, die dieses Vorgehen ausgelöst haben, protokolliert werden.

Hohe Frequenz und unregelmäßige Dichte: Wenn es zu einer Ausgabe kommt, werden die Daten mit der gleichen Frequenz und Dichte ausgegeben, wie sie bei der Analyse eintreffen. Das liegt an den Echtzeiteigenschaften des Event Stream Processing. Durch jedes Event, das die Analyse auslöst, kann es zu einer Ausgabe kommen.

Die Events wurden um Metainformationen erweitert: Eine andere mögliche Art der Ausgabe ist eine Teilmenge der eingehenden Events. Zum Beispiel bei der Classification von Events ist nur eine bestimmte Klasse von Bedeutung. Dann werden alle Events dieser bestimmten Klasse gespeichert. Die Speicherung erfolgt mit den entsprechenden Metainformationen. Diese Metainformationen können als zusätzliche Werte dem Event hinzugefügt sein oder die Informationen werden durch den Speicherort wiedergegeben. Der Speicherort könnte ein Ordner auf der Festplatte sein oder eine bestimmte Tabelle in einer Datenbank.

Die Events haben weniger offene Attribute: Die Events sind in dem Maße vollständig, wie der Mining-Prozess es vorgeschrieben hat. Die Events, in denen wichtige Informationen fehlten, können in dem Mining-Prozess nicht verarbeitet werden. Eine Möglichkeit ist, diese Events vorher zu filtern. Danach werden diese Events auch nicht mehr ausgegeben. Eine andere Möglichkeit ist es, die Events mit Standardwerten zu versehen. Dadurch stehen die Events für die Analyse zur Verfügung und habe in der Ausgabe mehr befüllte Attribute.

Angepasste Attribute: Durch Esper lassen sich die Bezeichnungen und Inhalte der Attribute den Vorgaben des Nutzers anpassen. Die Bezeichnungen können sprechender werden oder es könnten Extremwerte begrenzt werden.

3.3. Beschreibung von Event Stream Processing am Beispiel der Anwendungsmodelle

3.3.1. Flugdatenerfassung

Die Eigenschaften des Event Stream Processing werden in diesem Modell in den folgenden Punken deutlich:

- Es sind primitive Events, da sich die eingehenden Events auf jeweils einen bestimmten Funkspruch beziehen.
- Jedes Event bezieht sich auf einen bestimmten Zeitpunkt und nicht auf ein Zeitintervall.
- Die Events sind typisiert, da alle Attributtypen bekannt sind und von einer vorgeschalteten Einheit generiert werden.
- Es ist möglich, dass nicht alle Attribute gefüllt sind. Dies geschieht, wenn die empfangenen Signale verstümmelt von der Antenne empfangen werden.
- Die Events kommen in einem ständigen Strom an. Die dichte des Stroms richtet sich nach den Signalen, die an der Antenne aufgefangen werden. Daher schwankt die Anzahl der Events pro Sekunde.

In diesem Modell ist das Zeitfenster, auf dem die Vorverarbeitung operiert, sehr unterschiedlich. Bei der Reduktion werden nur ca. zehn Sekunden betrachtet. Das bedeutet eine Menge von ca. 5000 Events, auf der jede Abfrage läuft. Bei der Gruppierung sind es aber schon ca. 15 Minuten. Daraus ergibt sich eine Eventmenge von 450.000 Events die sich ständig ändern und auf denen laufend operiert werden muss.

Bei der Flugdatenerfassung werden zwei verschiedene Schritte durchgeführt.

Reduktion: Bei diesem Schritte handelt es sich um eine Filterung der Daten. Es werden unvollständige Events, die sich für eine weitere Analyse nicht eignen, entfernt und es werden doppelte Events gelöscht.

Gruppierung: Hier werden Events mit Meta-Daten versehen. Events, die innerhalb eines vorher definierten Zeitfensters auftreten und eine bestimmte Gemeinsamkeit aufweisen, werden als zusammengehörig markiert.

Diesen beiden Schritte sind mit Suchoperationen auf einer Datenbank zu vergleichen. Daher ist bei diesem Anwendungsmodell kein zusätzlicher Aufwand durch einen Datenmining Algorithmus erforderlich. Der Datenstrom muss hier als Datenbank aufgefasst werden, auf dem die Abfragen definiert werden. Es kann daher, bei diesem Anwendungsmodell, sehr gut verdeutlicht werden, wie Esper arbeitet.

3.3.2. Telefondatenanalyse

Die Eigenschaften des Event Stream Processing werden in diesem Modell in den folgenden Punken deutlich:

- Jedes Event ist genau ein Funksignal, man spricht von primitiven Events.
- Diese Funksignale beziehen sich auf einen Punkt in der Zeit .

- Die Typisierung der Events liegt vor.
  - Jeder Wert hat eine Bezeichnung, über welchen auf ihn zugegriffen werden kann. (Key-Value-Map)
  - Alle Werte liegen als String vor.
  - Sollte es nötig sein, mit den Werten zu rechnen, dann müssen die Strings in andere Typen (Integer, Double) umgewandelt werden.
- Es besteht die Möglichkeit, dass die Signale unvollständig empfangen werden und dadurch die Attribute der Events nicht immer belegt sind.
- Die Events kommen in einem ständigen Strom an (eine dieser Funkantennen empfängt pro Tag im Schnitt ca. 32 Millionen Signale, das sind 350 Signale pro Sekunde).
- Die Dichte der Events ist unterschiedlich, je nach dem wie die Signale an den Antennen empfangen werden.

Hier geht es um ein weites Feld des Datamining.

Bsp.1 In diesem Beispiel geht es darum, die aufgefangenen Events in Zonen oder Cluster einzuteilen. Bei der Einteilung werden auch zeitliche Aspekte berücksichtigt, die den Daten hinzugefügt werden müssen.

Bsp.2 Bei dieser Einteilung werden die Handygespräche selbst analysiert und in Klassen eingeteilt. Neben bekannten externen Daten werden auch Daten zur Analyse herangezogen, die während der Laufzeit dynamisch erzeugt werden müssen.

Bsp.3 Hier wird das Frequency Counting benötigt. Es sollen in einem Event Stream die Events (Handys) gefunden werden, die eine bestimmte Schwelle (Aktivität) überschreiten.

3.4. Bewertung der Event Stream Algorithmen in Bezug auf die Anwendungsmodelle

Im Folgenden wird für die Event Stream Algorithmen untersucht, inwieweit sie sich für die Anwendungsmodelle eignen. Wie im letzten Kapitel 3.3 beschrieben, sind für die Flugdatenerfassung keine besonderen Datamining Algorithmen notwendig. Daher werden bei dieser Bewertung nur die Beispiele der Telefondatenanalyse betrachtet. Bei der Bewertung der Algorithmen werden die folgenden Punkte berücksichtigt:

- benötigtes Format der Eingabe

- was zur Umsetzung des Algorithmus nötig ist
- in welchem Maß die Anforderungen mit dem Algorithmus umgesetzt werden können
- welche Rolle Esper bei der Umsetzung des Data Mining Algorithmus hat

Es handelt sich in den angegebenen Beispielen 2.3 um jeweils einen bestimmten Bereich des Datamining.

Beispiel 1: gehört in den Bereich des Clustering

Beispiel 2: gehört in den Bereich der Classification

Beispiel 3: gehört in den Bereich des Frequency Counting

Jeder Algorithmus wird im Folgenden mit dem dazugehörigen Beispiel bewertet.

3.4.1. Clustering

Algorithmus: Framework for Clustering Evolving Data Streams[1]

Allgemein: Der in Abschnitt 3.1.2 beschriebene Algorithmus fasst die eingehenden Events zu „Micro-Clustern“ zusammen. Durch diese Zusammenfassung gehen Informationen verloren. Es lassen sich nicht mehr ideale Cluster-Grenzen finden. Die Grenzen sind jedoch relativ nahe an der idealen Grenze, da die „Micro-Cluster“ eine hohe Auflösung haben. Um Cluster zu verschiedenen Zeitpunkten zu erstellen, werden Momentaufnahmen der „Micro-Cluster“ abgespeichert. Die Momentaufnahmen werden immer seltener, je weiter sie in der Vergangenheit liegen.

Eingabe: Die Eingangsdaten dieses Algorithmus sind Berechnungen aus den Attributen der Events. Der Algorithmus nutzt nur diese aus den Berechnungen entstehenden Kennzahlen, die als Cluster Feature Vector dargestellt werden.

Umsetzung: Nachdem die Events in Cluster Feature Vectoren umgewandelt wurden, müssen die Vektoren in die bestehenden Micro-Cluster eingefügt werden. Die Berechnungen zum Einfügen der Vektoren sind gut umzusetzen, da es sich um einfache Arithmetik handelt. Die Hauptaufgabe liegt bei diesem Algorithmus darin, die Datenstrukturen für die Operationen zur Verfügung zu stellen. Für die Erstellung der Macro-Cluster sollte ein geeignetes Tool zur Verfügung gestellt werden. Hierfür kann der Aufwand beliebig komplex gewählt werden.

Anforderungen: Ein Problem ist die Zweiteilung des Ablaufes. Bei der Bildung der geforderten Cluster muss das Ergebnis manuell gebildet werden. Da die Micro-Cluster aber ständig aktuell sind, könnte dieses Problem durch weitere Software unterstützt werden. In dieser Software wird das Wissen implementiert, wie Cluster zu bilden sind und der Anwender ist nur noch für die Eingabe der Parameter verantwortlich. Durch diese Maßnahme muss nicht ein Experte die Analyse vornehmen, sondern es kann ein eingewiesener Nutzer die Ergebnisse erstellen.

Esper: Esper eignet sich sehr gut für die Umwandlung der eingehenden Events in Cluster Feature Vectoren. Nachdem ein Event eingetroffen ist, kann durch Esper auch das Einfügen in die Datenstruktur initialisiert werden. Somit eignet sich Esper, die Vorverarbeitung dieses Algorithmus umzusetzen. Der Algorithmus selbst muss in einem externen Modul implementiert werden.

3.4.2. Classification

Algorithmus für die Klassifikation von Frequenzen[19]

Allgemein: Dieser Algorithmus löst das Problem der geringen Ressourcen durch eine Beschränkung des Anwendungsgebietes auf Frequenzdaten, die in ein Wavelet umgewandelt werden. Durch diese Umwandlung können Funktionen zur Mustererkennung in Folgen von Events effektiv umgesetzt werden.

Eingabe: Die Eingabedaten müssen Frequenzen sein, die dann in Wavelets umgewandelt werden. Diese Umwandlung gehört zur Vorverarbeitung der Daten.

Umsetzung: Eingehende Daten müssen in das richtige Format transferiert werden. Die Hauptaufgabe ist die Umsetzung des Korrelationsmodelles, wie es in [19] beschrieben ist.

Anforderungen: Mit diesem Algorithmus können innerhalb von Frequenzen Muster definiert werden, durch die die Frequenzen klassifiziert werden können. In dem Anwendungsbeispiel werden Frequenzen untersucht. Die Klassifizierung der Daten wird nicht nur durch die Frequenzen bestimmt, sondern noch durch weitere Faktoren. Daher können die Anforderungen durch diesen Algorithmus nicht vollständig erfüllt werden.

Esper: Mit Esper kann die Umwandlung der Events in Wavelets vorgenommen werden. Dieses ist mit den Möglichkeiten von Esper einfach Umzusetzen, da nach dem Auftreten eines Events Java Code ausgeführt werden kann. Die Muster, die durch den Algorithmus erkannt werden, sollen mit dem Algorithmus automatisch erzeugt werden. Dieses ist mit Esper nicht umsetzbar. Daher eignet sich Esper hier nur für die Vor- und Nachbereitung der Events.

Algorithmus für Klassifikation räumlicher Daten [9]

Allgemein: Es wird das Problem der geringen Ressourcen durch eine Beschränkung des Anwendungsgebietes auf Rasterdaten reduziert. Diese Rasterdaten müssen in einen Peano Count Tree transformiert werden. Nach der Transformation wird eine Nachbearbeitung der Daten vorgenommen. Diese Daten werden dann an den ID3 Algorithmus weitergeleitet.

Eingabe: Die eingehenden Daten müssen sich auf ein Raster reduzieren lassen, das durch binäre Werte dargestellt werden kann. Beispiele für diese Art der Daten sind Bilder oder Landkarten.

Umsetzung: Es ist nötig, die Daten in die geforderte Datenstruktur zu überführen. Nach dieser Vorverarbeitung müssen die Daten einen Entscheidungsbaum durchlaufen, der von einem Clustering Algorithmus erstellt worden ist. Hier wurde der ID3 gewählt. Der Fokus liegt bei diesem Algorithmus auf der Datenstruktur und den Operationen, die mit der Datenstruktur möglich sind.

Anforderungen: Die Angaben der Muster werden bei diesem Algorithmus auf den räumlichen Daten spezifiziert. Es werden räumliche Daten in den Anforderungen dargestellt. Die beschriebenen Muster erstrecken sich aber über weitere Daten. Diese Daten müsste man in weiteren Dimensionen über den geografischen Daten darstellen. Dieses ist aber nicht möglich, da einige Daten nicht im direkten Kontakt mit einer geografischen Position stehen.

Esper: Diese Transformation kann direkt in Esper umgesetzt werden. Nach dieser Umsetzung sind weitere Schritte nötig, in denen die Daten angepasst werden. Auch diese Anpassung kann durch Esper direkt ausgeführt werden, wenn ein Event auftritt. Der Entscheidungsbaum, kann mit Esper umgesetzt werden.

Algorithmus: Very Fast Decision Tree Learner [10]

Allgemein: Bei diesem Algorithmus wird ein Online Lernverfahren für das Erstellen des Entscheidungsbaums genutzt. Das Spaltungsattribut wird nicht perfekt gewählt, sondern es liegt innerhalb einer bestimmten Wahrscheinlichkeit eines bestimmten Intervalls des perfekten Spaltungsattributes. Der Algorithmus zum Erstellen des Entscheidungsbaums ist eine Verbesserung des Hoeffding Tree Algorithmus. Die Trainingsdaten sind hier ein Datenstrom.

Eingabe: Die Eingehenden Daten können beliebig sein. Sie müssen aber den unter 3.2 definierten Eigenschaften genügen.

Umsetzung: Bei diesem Algorithmus muss hauptsächlich das Verfahren zum Erstellen des Entscheidungsbaumes umgesetzt werden. Es müssen Heuristiken implementiert werden, die für die jeweiligen Attribute die Wahrscheinlichkeit berechnen.

Anforderungen: Dieser Algorithmus lässt Attribute aus unterschiedlichen Anwendungsgebieten zu. Die eingehenden Daten, sowie die Trainingsdaten, können als Datenstrom vorliegen. Der Entscheidungsbaum kann jederzeit verbessert werden. Dass das Spaltungsattribute möglicherweise schlecht gewählt wurde, ist ein Nachteil dieses Algorithmus. Die Wahrscheinlichkeit für eine ungünstige Wahl ist fest definiert, und somit ein kalkulierbares Risiko. In dem gegebenen Beispiel ist es nicht erforderlich, dass die Spaltungsattribute optimal gewählt werden. Daher erfüllt dieser Algorithmus die Anforderungen.

Esper: Der erzeugte Entscheidungsbaum hat eine einfache Struktur und die Attribute der Events behalten ihre Struktur. Daher eignet sich Esper sehr gut den Entscheidungsbaum umzusetzen, nachdem er von dem Algorithmus erstellt wurde.

3.4.3. Frequency Counting

Algorithmus: Approximate Frequency Counts [18]

Allgemein: Mit diesem Algorithmus werden alle Events gefunden, die eine bestimmte Häufigkeit im Verhältnis zu allen bisher erfassten Daten überschreiten. Da es nicht möglich ist alle Daten im Arbeitsspeicher zu behalten, gibt es eine Schranke für eine Fehlertoleranz von falschen positiven Treffern.

Eingabe: Es gibt keine expliziten Einschränkungen für die eingehenden Daten. Für Attribute, die einen sehr großen oder unendlichen Definitionsbereich haben, sollte dieser Wertebereich auf eine überschaubare Anzahl von Teilmengen reduziert werden. Werte aus der gleichen Teilmenge gelten dann als gleich.

Umsetzung: Der Fokus der Anwendung liegt auf dem Algorithmus, mit dem die Einhaltung der Fehlertolleranz gewährleistet wir. Eine besondere Vor oder Nachbereitung der Daten ist bei diesem Algorithmus nicht nötig.

Anforderungen: Es werden von dem Algorithmus keine falschen negativen Elemente gefunden und die falschen Positiven bleiben in einem fest definiertem Rahmen. Die Angaben zur Frequenz sind auch mit einer Fehlertolleranz behaftet. Diese Fehlertolleranz ist die gleich, der Fehlertolleranz für den Rahmen innerhalb der die falschen positiven Elemente liegen. Eine Fehlertolleranz ist in dem gegebenen Anwendungsmodell vertretbar. Daher ist dieser Algorithmus für das Anwendungsbeispiel geeignet.

Esper: Esper ist für die Umsetzung des Algorithmus selbst nicht nötig. Esper kann zur Vorverarbeitung der Daten eingesetzt werden. Aus einem unendlichen Wertebereich (Reale Zahlen) kann Esper Gruppen bilden. Es können zum Beispiel aus einer Größenangabe für Menschen in Metern die Gruppen Klein ([0.6, 1.65[), Mittel ([1.65,1.72[) und Groß ([1.72,3.15]) gebildet werden. Außerdem kann Esper ungültige oder unerwünschte Angaben entfernen. In diesem Beispiel sind es die Menschen unter 0,6 Meter oder über 3,15 Meter

3.5. Bewertung der Event Stream Algorithmen allgemein

Im nachfolgenden Abschnitt sollen die gefundenen Algorithmen allgemein bewertet werden. Es wird untersucht welche Eigenheiten die verschiedenen Bereiche des Datamining aufweisen und wie die Konzepte von Esper zu diesen Eigenheiten passen.

3.5.1. Clustering

Die besonderen Eigenschaften des Clustering sind:

- Für das Erstellen der Cluster sind die Informationen aller bisher aufgetretenen Daten notwendig.
- Es ist keine Initialisierungsphase notwendig.
- Der Zeitpunkt an dem die Cluster erstellt werden ist wichtig. Je nach Zeitpunkt gibt es unterschiedliche Datenmengen, die für den Algorithmus relevant sind.

Diesen Punkten stehen folgende Eigenschaften des Event Stream Processing gegenüber:

- Die Menge der Daten ist zu groß um sie alle gleichzeitig im Speicher bereitzustellen.
- Der Menge der Daten werden ständig neue Events hinzugefügt. Das bedeutet, dass sich die Menge der Daten laufend verändert.

Esper eignet sich beim Clustering um die eingehenden Events auf die Informationen zu reduzieren, die für den Algorithmus wichtig sind. Da beim Clustering alle bisherigen Daten wichtig sind und diese nicht alle in den Arbeitsspeicher passen, ist eine Anpassung der Daten immer wichtig.
Bei der Umsetzung des Algorithmus können die Eigenschaften von Esper nicht genutzt werden. Beim Clustering werden Datensätze verglichen und entsprechend dieser Informationen eingeteilt. Für diese Aufgabe sind alle bisherigen Events relevant. Daher kann Esper dieses Problem nicht lösen, da für Esper nur der Arbeitsspeicher zur Verfügung steht.

3.5.2. Classification

Die besonderen Eigenschaften des Classification sind:

- Es gibt eine Initialisierungsphase, die vor dem klassifizieren neuer eingehender Events erfolgen muss.
- Für das Klassifizieren der Events sind nur die Informationen aus der Initialisierungsphase notwendig.
- Der Zeitpunkt, an dem die Events klassifiziert werden, ist unwichtig.

Eigenschaften des Event Stream Processing, die diesen Punkten gegenüberstehen, sind:

- Beim Event Stream Processing gibt es keine Initialisierungsphase.
- Die Events können in einer hohen Frequenz auftreten, dieses reduziert die Ressourcen, die für die Bearbeitung zur Verfügung stehen.

Die Voraussetzungen für diese Art des Datamining ist, dass es Trainingsdaten gibt, nach denen alle weiteren Daten eingeteilt werden. Wenn diese Voraussetzungen gegeben sind, ist das Problem der Rechenaufwand für das Erstellen der Funktion für die Klassifizierung. Dieses Problem wird von den gefundenen Algorithmen entweder durch die Reduzierung des Anwendungsgebietes oder durch Anwendung von Heuristiken gelöst. Die Reduzierung des Anwendungsgebietes hat den Vorteil, dass der Algorithmus weiterhin die beste mögliche Funktion zur Klassifizierung erstellt. Es ist nur selten möglich den Algorithmus einzusetzten. Wenn Heuristiken eingesetzt werden, erzeugt der Algorithmus nicht immer die beste Funktion. Diese Maßnahme reduziert die möglichen Anwendungsgebiete auf Problemfelder, bei denen eine fehlerhafte Klassifizierung zulässig ist. Um den reduzierten Ressourcen des Event Stream Processing Rechnung zu tragen, schränken die gefundenen Algorithmen die Vielfalt der Anwendungsgebiete ein.

Die eingehenden Events müssen entweder in ein neues Format transformiert werden oder es müssen statistische Berechnungen durchgeführt werden wie es in Kapitel 3.1.3 beschrieben wurde. Dieses kann von Esper durchgeführt werden.
Die Trainingsphase stellt eine Besonderheit da. Die Erstellung der Klassifizierungsfunktion kann nicht durch Esper dargestellt werden. Wenn auch die Trainingsdaten als Stream vorliegen, kann Esper für die Vor- und Nachbereitung dieser Daten genutzt werden. Wenn die Klassifizierungsfunktion in Form eines Entscheidungsbaumes vorliegt, ist Esper geeignet, um die Klassifizierung umzusetzen. Das Durchlaufen eines Entscheidungsbaumes entspricht der Grundidee von Esper.

> In Esper werden Abfragen gespeichert und Daten werden über diese Abfragen gestrichen.

In diesem Fall kann der Dataminingvorgang vollständig von Esper umgesetzt werden. Wird sich bei der Klassifizierung für eine andere Umsetzung der Klassifizierungsfunktion entschieden, muss auch hier ein externes Modul eingesetzt werden.

3.5.3. Frequency Counting

Die besonderen Eigenschaften von Frequency Counting sind:

- Das Muster aller bisherigen Events ist relevant
- Es gibt keine Initialisierungsphase
- Äquivalenz der Events muss definiert sein

Diesen Punkten stehen die Eigenschaften des Event Stream Processing gegenüber:

- Die Menge der Daten ist zu groß um sie alle gleichzeitig im Speicher bereitzustellen, auch wenn es nur die Muster der Daten sind.
- Der Menge der Daten werden ständig neue Events hinzugefügt. Das bedeutet, dass sich die Menge der Daten laufend verändert.

Bei dem Frequency Counting sollen bestimmte Events gefunden werden, die besonders häufig auftreten. Dabei stellen sich zwei Probleme. Das erste Problem ist die Definition der Gleichheit. Dieses muss vor dem Ablauf feststehen und kann sich auch mit der Zeit ändern. Das zweite Problem besteht darin, dass alle Events relevant sind. Selbst ein Event, dass nur 0,1% des Gesamtvolumens ausmacht, kann dazu beitragen, dass später der Schwellwert von 20% überschritten wird.
Das erste Problem kann gelöst werden, indem im Vorfeld die Events so verändert werden, dass sie einfach zu vergleichen sind. Bei dem gefundenen Algorithmus wird das zweite Problem durch stochasitische Berechnungen gelöst. Bei Frequenzzählung wird ein Unsicherheitsfaktor zugelassen, durch den der Aufwand reduziert wird.

Esper eignet sich um die Änderungen vor dem Algorithmus durchzuführen. Durch diese Vorarbeit ist der Algorithmus einfacher zu implementieren. Außerdem ist es mit Esper möglich, die Auswertung des Algorithmus zu aktivieren, sobald ein Event auftritt. Durch Esper kann die Hauptaufgabe des Algorithmus nicht umgesetzt aber unterstützt werden.

4. Konzeption der Architektur

In diesem Kapitel wird eine Architektur konzipiert, mit der die Möglichkeiten von Esper im Event Stream Processing eingesetzt werden. Die Architektur bezieht sich auf die Analysen im vorherigen Kapitel (3.4 und 3.5). Dort wurde festgestellt, wie sich Esper eignet Datamining Algorithmen im Event Stream Processing umzusetzten.

4.1. Allgemeiner Aufbau der Architektur

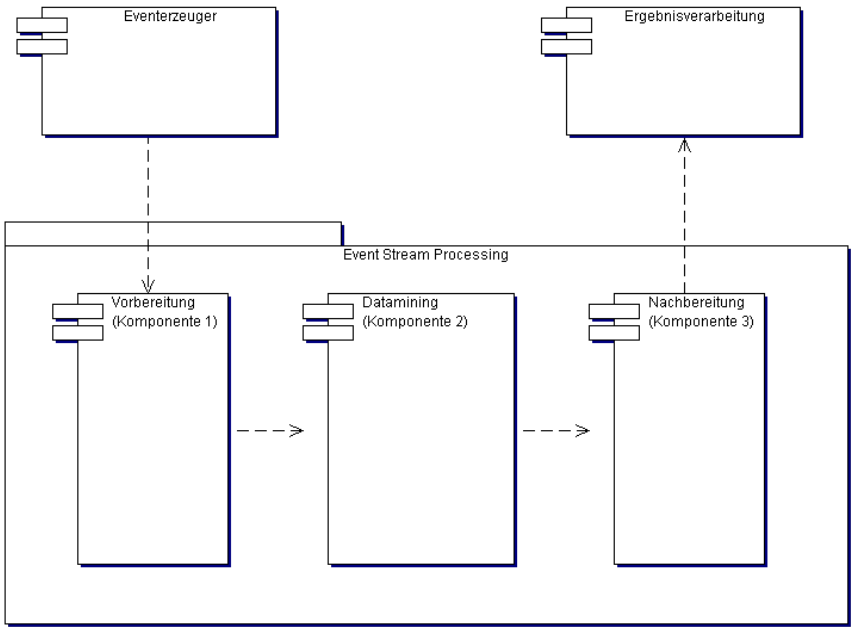

Abbildung 4.1.: Datamining im ESP

In den Analysen in Kapitel 3.5 wurde deutlich, dass nahezu jeder Datamining Algorithmus eine Vorverarbeitung der Events benötigt. Daher wird in dieser Architektur ein eigenes Modul vor den Datamining-Abschnitt gelegt. Nachdem die Events das Datamining durchlaufen haben, ist eine Nachbearbeitung der Events notwendig. Diese Nachbearbeitung kann aus einer

einfachen Weiterleitung bestehen oder aus einem erneuten Umformen der Events und dem Abspeichern in einer Datenbank. Bei jedem Anwendungsmodell ist es notwendig die Ergebnisse des Datamining Moduls weiter zu verarbeiten. Diese Weiterverarbeitung übernimmt die Nachbereitung.

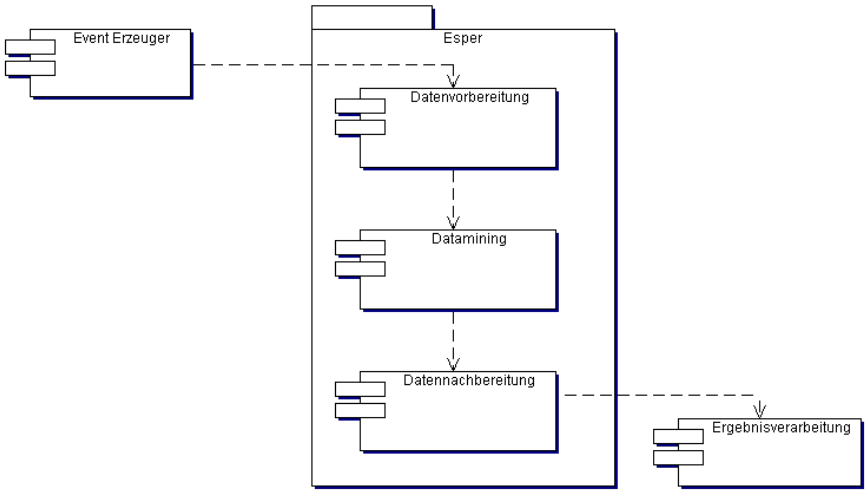

Abbildung 4.2.: Dataming mit Esper

Bei der Analyse ist außerdem festgestellt worden, dass in manchen Fällen Esper den gesamten Datamining Prozess darstellen kann und in anderen Fällen nur die Vor- und Nachbereitung der Events für Esper geeignet ist. Diese beiden Varianten werden im Folgenden als „Dataminig mit Esper“ 4.2 und als „Dataming ohne Esper“ 4.3 unterschieden. Letzteres bedeutet nicht, dass Esper im gesamten Vorgang nicht eingesetzt werden kann. „Datamining ohne Esper“ soll deutlich machen, dass der Datamining-Vorgang und nur der Datamining-Vorgang nicht mit Esper umgesetzt werden kann.

4.2. Definition der Komponenten im Event Stream Processing

In diesem Abschnitt werden die notwendigen Komponenten für das Datamining auf Event Streams beschrieben.

Esper bietet folgende Möglichkeiten:

Filter: Events, können nach vielfältigen Kriterien gefiltert werden.

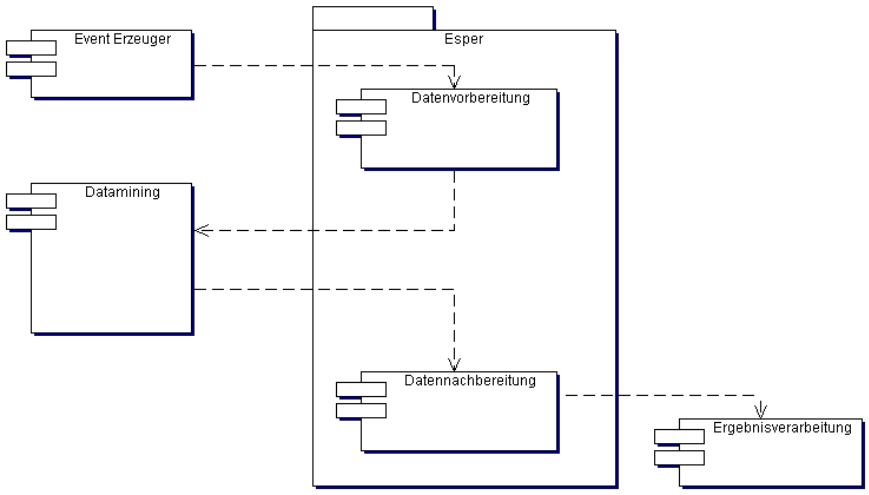

Abbildung 4.3.: Dataming ohne Esper

- nach dem Zeitpunkt des Auftretens
- nach Inhalt (z.B. durch Mengenoperationen größer als)
- nach Berechnungen mit Werten aus dem Event
- im Verhältnis zu anderen Events (z.B. durch die Definition eines Musters)

Zusammenfassen oder Neuordnung von Inhalten: Die Attribute der Events können durch Esper umbenannt oder gelöscht werden. Es ist auch möglich, neue Attribute hinzuzufügen. Diese neuen Attribute können Zähler-Werte oder Berechnungen der bestehenden Attribute beinhalten.

Suche nach Daten: Wenn ein fester Regelsatz für die Abfragen auf den Events vorliegt, kann mit Esper der Event Stream durchsucht werden. Hierfür werden nach dem Regelsatz die Abfragen definiert und die Events werden dann über diese Abfragen gestrichen.

4.2.1. Übersicht

Wie in Kapitel 3.2 dargestellt, können nicht alle Events direkt an den Algorithmus übergeben werden. Unvollständige Events müssen ausgefiltert oder verändert werden. Dieses ist mit den Möglichkeiten von Esper sehr gut umzusetzen.

In Kapitel 3.5 wurde beschrieben, dass Algorithmen mit den reduzierten Ressourcen des Event Stream Processing auskommen müssen. Dieses Problem lösen die Algorithmen durch eine besondere Datenstruktur oder durch eine besondere Darstellung der Daten. Events, die der Definition 3.2 entsprechen, müssen in einem zweiten Schritt in eine neue Datenstruktur oder in eine neue Art der Darstellung überführt werden. Für diese Punkte der Vorbereitung werden folgende Komponenten benötigt:

Komponente 1a: Filter Entfernen/Vervollständigen von Events mit Esper

Komponente 1b: Vorverarbeitung Bearbeiten/Verändern von Events mit Esper

Nachdem die Events entsprechend den Anforderungen des Algorithmus angepasst wurden, können sie nun dem Algorithmus übergeben werden. Der Algorithmus kann in der Regel (siehe Abschnitt 4.2.3) nicht durch Esper dargestellt werden.

Komponente 2: Datamining Umsetzung des Datamining Algorithmus

Durch den Datamining Algorithmus können verschiedene Aktionen durchgeführt worden sein.

- Es wurden neue Informationen erzeugt
- Es wurden Metadaten den Events hinzugefügt
- Die Struktur der Events wurde verändert
- Der Inhalt der Events wurde verändert

Nach diesen Operationen gibt es verschiedene Aktionen, die als nächstes folgen.

Komponente 3a: Nachbearbeitung Die Events befinden sich nicht in dem geforderten Format und müssen angepasst werden. Für diese Aufgabe eignet sich erneut Esper sehr gut.

Komponente 3b: Trigger Ein weiteres Modul wird durch den Algorithmus aktiviert.

Komponente 3c: Speicher Die Events sollen in einer Datenbank gespeichert werden.

Komponente 3d: Weiterleitung Die Events sollen einem weiteren Datamining Algorithmus übergeben werden.

Diese Komponenten können alle nach der Datamining Komponente folgen. Es kann aber auch nur eine Komponente nötig sein. Welche Komponenten gewählt werden, hängt von der Anwendung ab.

4.2.2. Datenvorbereitung

Filter

Es werden nur die Events weitergeleitet, die den Vorbedingungen des verwandten Algorithmus genügen. Daher ist es notwendig die eingehenden Events zu Filtern. Ein Attribut eines Events kann aufgrund einer Störung der Funksignale leer sein. Dieses Event würde das Ergebnis des Algorithmus verfälschen, wenn es weitergeleitet wird.
Eine Möglichkeit ist, das Event aus dem Event Stream zu entfernen. Es besteht hier eine fest definierte Abfrage und die Daten werden über diese Abfrage gestrichen. Alle erkannten Events sollen zur nächsten Komponente weitergeleitet werden. Dieses kann durch Esper mit einer entsprechenden Abfrage durchgeführt werden.
Es gibt weiterhin die Möglichkeit den Events, bei denen Werte fehlen, default-Werte zu setzen. Wenn zum Beispiel Frequenzen größer 10 kHz gesucht werden und die Frequenzangabe fehlt, könnte in dem Event eine Frequenz von 0 Hz gesetzt werden. Dann ist das Event nicht verloren und steht für andere Auswertungen noch zur Verfügung. Hierfür muss der Event Stream nach Events mit fehlerhaftem oder nicht vorhandenem Frequenzwert durchsucht werden. Bei diesen Events wird in der Ausgabe die Frequenz durch den default-Wert ersetzt. Dieses wäre in Esper eine weitere Abfrage, die ohne Probleme neben der ersten aktiv sein kann.

Vorverarbeitung

In diesem Abschnitt geht es nicht um die Inhalte der Events, sondern um deren Struktur. Die eingehenden Events liegen bei Esper, das mit Java umgesetzt wird, als POJO, Map oder XML Node vor (2.2). Diese Struktur ist in der Regel nicht die, die für einen Datamining Algorithmus notwendig ist. In den analysierten Algorithmen aus dem Kapitel 3.1 sind sehr unterschiedliche Berechnungen notwendig. Zwei Beispiele:

- Beim „Framework for Clustering Evolving Data Streams“ müssen Kennzahlen aus den Events berechnet werden.
- Beim „Algorithmus für die Klassifikation von Frequenzen“ müssen die eingehenden Frequenzwerte in ein Wavelet transformiert werden.

Die notwendigen Berechnungen müssen für jedes eingehende Event durchgeführt werden, bevor es dem Algorithmus zur Verfügung gestellt wird. Jedes von Esper erkannte Event wird von einem Listener oder Subscriber erfasst (2.2). In den Listener/Subscriber kann vor der Weiterleitung der Events noch Javacode ausgeführt werden. Somit eignen sich diese Bausteine von Esper sehr gut um diese Komponente umzusetzen.

4.2.3. Datamining

In der Miningkomponente wird der Hauptteil der Arbeit durchgeführt. Der Miningkomponente werden die Events übergeben. Aus diesen Events werden weitere Informationen gewonnen. Die gewonnenen Informationen beinhalten:

Globale Informationen: Diese Informationen beziehen sich auf die gesamten Daten und werden gesondert gespeichert. Hier sind die statistischen Informationen aus dem Klassifikationsprozess gemeint, zum Beispiel die Anzahl der Daten pro Klasse oder Cluster. Es können aber auch Ergebnisse aus Datamining Prozessen sein, die in dieser Arbeit nicht untersucht worden sind, wie zum Beispiel Untersuchungen von Frequenzmustern, Verbindungen und Verknüpfungen.

Zeitpunkte für Aktionen: Es gibt Fälle in denen die Ergebnisse des Datamining nicht eine Menge von Daten sind, sondern es kommt nur auf den Zeitpunkt an, an dem ein Ergebnis fest steht. Zu diesem Zeitpunkt wird dann eine weiterführende Aktion auslöst. In dem Anwendungsmodell der Telefondatenanalyse in Abschnitt 2.3.2 Beispiel 2, könnte das Erkennen eines Telefongesprächs mit einem kritischen Sicherheitsstatus den Abbruch einer Mission in dem überwachten Gebiet ausgelösen.

Meta Informationen: Diese Informationen werden den bestehenden Daten hinzugefügt. Durch diese zusätzlichen Informationen lassen sich weiterführende Analysen leichter durchführen. Zum Beispiel wenn die Datensätze einer bestimmten Klasse oder einem Cluster zugeordnet werden.

In Esper werden verschiedene Möglichkeiten zur Filterung von Event Streams bereitgestellt. Außerdem gibt es die Möglichkeit, in den Events Methoden zu implementieren, die Berechnungen durchführen. Alle Problemstellungen, die sich auf das Filtern von Daten und einfache Berechnungen reduzieren lassen, werden im Folgenden als „Datamining mit Esper“. Die Komplexeren Lösungsansätze werden als „Datamining ohne Esper“ bezeichnet. In dem Lösungsansatz „Datamining ohne Esper“ kann Esper in der Datenvorbereitung oder Datennachbereitung eingesetzt werden, auch wenn die Komponente Datamining nicht mit Esper implementiert wird.

Datamining mit Esper

Esper bietet die Möglichkeit aus dem Ergebnis einer Abfrage einen neuen Datenstrom zu generieren und auf diesem Datenstrom erneut Abfragen zu definieren. Mit Esper ist es möglich, wie oben beschrieben, Javacode in dem Listener oder Subscriber auszuführen. Dadurch können für jedes Event verschiedene Berechnungen durchgeführt werden.
Durch die Möglichkeiten der Abfrage kann ein Entscheidungsbaum zur Klassifikation von

Events ohne weitere Software erstellt werden. Die Voraussetzung hierfür ist, dass dieser Entscheidungsbaum feststeht und nicht zu komplex ist. Wo die Grenze der Komplexität genau ist, hängt von dem Anwendungsmodell ab. Laut der Homepage von Esper [17] sind 500.000 Events pro Sekunde mit 1000 Abfragen möglich auf einer CPU mit 2 GHz. Esper stellt einen objektorientierten Ansatz zur Verfügung, um Abfragen zu generieren. So ist es möglich einen Entscheidungsbaum durch geschachtelte Abfragen in Esper darzustellen. Wenn Datamining-Algorithmen mit Esper umgesetzt werden soll, dann muss der Algorithmus die folgenden Punkte erfüllen:

- Für den Mining-Prozess dürfen nicht alle bisher gelesenen Events relevant sein
- Die Berechnungen pro Event dürfen nicht zu komplex sein (kein hoher Rechenaufwand)
- Die eingehenden Events müssen sich als POJO, Map oder XML Node darstellen lassen

Datamining ohne Esper

Wenn die oben genannten Voraussetzungen nicht gegeben sind, ist es nicht möglich Datamining durch Esper zu implementieren. In diesem Fall muss für die eingehenden Daten eine Schnittstelle für Esper bereitgestellt werden. Über diese Schnittstelle kommuniziert Esper mit dem Datamining-Modul.
Vorhandene Datamining-Algorithmen sind möglicherweise nicht für das Event Stream Processing optimiert. Um die Performance des Algorithmus zu verbessern, kann es hilfreich sein, die Events in einem Puffer zu sammeln und blockweise zu verarbeiten. So muss der Datamining Prozess nicht bei jedem, sondern bei jedem n-ten Event angestoßen werden. Durch diese blockweise Verarbeitung, wird die herkömmliche Arbeit des Datamining nachgeahmt. Ob diese Maßnahme sinnvoll ist, muss in jedem Fall neu entschieden werden.

4.2.4. Datennachbereitung

Den Abschluss des Event Stream Processing bilden die Komponenten:

- Nachbearbeitung
- Trigger
- Speicher
- Weiterleitung

Nachbearbeitung

Die Ergebnisse der Datamining-Komponente sind in der Regel nicht in dem Format, das für einen Trigger, die Speicherung oder die Weiterleitung gefordert ist. Die Daten haben bei der Ausgabe das Format, das für den Datamining-Prozess erforderlich war. Die Ergebnisse müssen nun in ihrer Struktur verändert werden und der gewünschten Darstellung angepasst werden.
Diese Nachbearbeitung kann von Esper übernommen werden. Es müssen Aktionen für jedes Ergebnis durchgeführt werden und diese Aktionen müsse direkt durchgeführt werden, bevor die Daten weiter verarbeitet werden können.

Trigger

Ein Trigger ist eine relativ kleine Komponente. Sie stellt eine Verbindung zu anderen Programmodulen da, die gestartet werden sollen wenn ein bestimmtes Ereignis eintritt. Die Trigger-Komponente überwacht die Ergebnisse der Datamining-Komponente und startet über eine Schnittstelle die Funktion des Moduls, sobald der Trigger ausgelöst wird.
Dieser Trigger kann von Esper umgesetzt werden. Hier werden die Ergebnisse der Datamining-Komponente als Events betrachtet und diese Events werden nach festgesetzten Regeln gefiltert.

Speicher

Diese Komponente ist für die Sicherung der Ergebnisse zuständig. Da die Events möglicherweise schneller auftreten, als das DatenBank Management System (DBMS) eine Speicherung zulässt, muss hier eine Queue eingerichtet werden. Die Queue vermeidet, dass Daten verloren gehen oder die Verarbeitung ins stocken kommt. Bei der Gestaltung der Datenbank ist auch die Geschwindigkeit der Schreiboperation aus dem oben genannten Grund ein wichtiger Faktor.
Diese Komponente kann mit Esper umgesetzt werden. Für jedes Ergebnis muss eine bestimmte Aktion ausgeführt werden. Die Anpassung der Ergebnisse an die Struktur der Tabellen lässt sich in den Listenern oder Subscribern implementieren.

Weiterleitung

Diese Komponente stellt die Verbindung zu einem externen Modul da. Alle Events werden zu diesem Modul weitergeleitet. Bei der Weiterleitungskomponente muss die Schnittstelle des Moduls bekannt sein.

Die Weiterleitung lässt sich in einem der oben genannten Komponenten zusätzlich in den Listener oder Subscriber von Esper einbauen.

4.3. Relation der Komponenten im Event Stream Processing

Im folgenden Abschnitt wird der Ablauf des Event Stream Processing mit Esper genauer beschrieben. Es wird darauf eingegangen, wie die Komponenten aus dem letzten Abschnitt zusammenhängen und wie der Datenfluss zwischen diesen Komponenten geregelt ist. Anschließend wird im Detail beschrieben, welche Möglichkeiten Esper bei der Umsetzung der Komponenten bereitstellt.

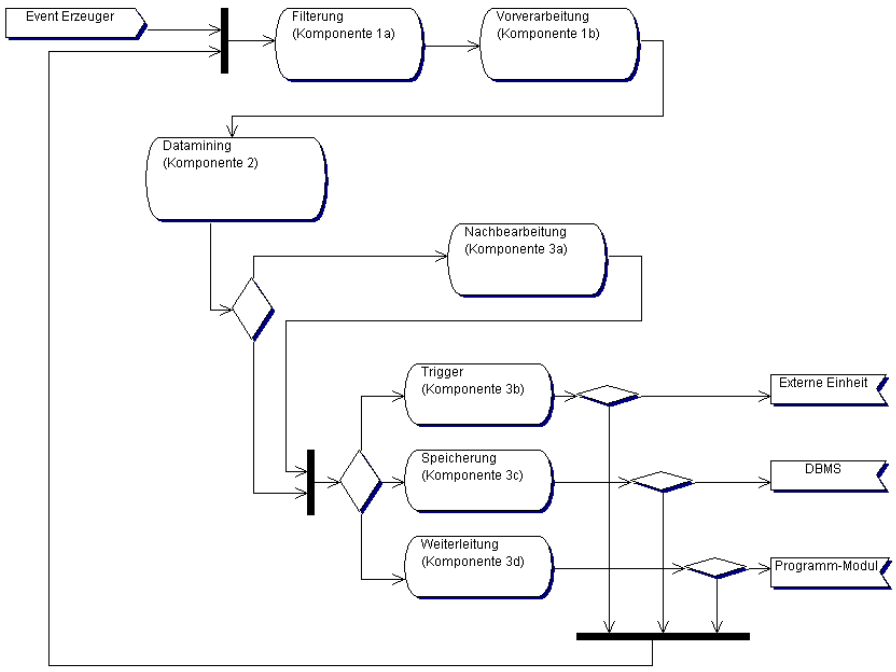

Abbildung 4.4.: Aktivitätsdiagramm: Datamining

4.3.1. Ablauf

Das UML Diagramm 4.4 zeigt, wie die Aktionen in diesem Konzept zusammenhängen. Die Aktionen spiegeln die Komponenten aus Abschnitt 4.2 wider.

Von der Komponente 1a (Filter) werden die Events das erste Mal gelesen. An dieser Stelle ist ein Puffer sinnvoll. Die Events kommen mit einer schwankenden Frequenz an. Es können mehr Events eintreffen, als der Rechner verarbeiten kann. Diese Events müssen für Perioden gepuffert werden, in denen die Frequenz der eingehenden Events geringer ist. Vor dem Filter gibt es nur einen Event Stream.

Durch den Filter kann der Event Stream deutlich weniger Events aufweisen oder alle Events, die beim Filter eingegangen sind. Was genau zutrifft, hängt sehr von den definierten Filtern und der Qualität der Events ab. Bei einfachen Filtern und einer guten Qualität der Events werden kaum Events ausgefiltert. Bei einer schlechten Qualität der Events oder strengen Filtern kann sich der Durchfluss der Events stark reduzieren. Dieses ist bei jeder Anwendung unterschiedlich. Für die Überprüfung der Performance muss beachtet werden, dass möglicherweise kein Event ausgefiltert wird und der Event Stream in voller Stärke bei der nächsten Komponente eintrifft.

Der Event Stream zur Datamining-Komponente kann verschiedene Ausprägungen haben.

Datamining mit Esper: In diesem Fall wird innerhalb von Esper ein neuer Datenstrom erzeugt, für den die Abfragen der Datamining-Komponente erzeugt werden. Es sind keine zusätzlichen Puffer oder Abhängigkeiten nötig.

Datamining ohne Esper: Da die Events an ein Modul außerhalb der Esper Umgebung geleitet werden müssen, ist auch hier ein Puffer sinnvoll. Von der Komponente 1b: Vorverarbeitung muss eine Schnittstelle zur Komponente 2 (Datamining) implementiert werden.

Nach der Datamining-Komponente gibt es verschiedene Möglichkeiten wie der Event Stream weiterverläuft. Die Komponente 3a (Nachbearbeitung) ist optional. In manchen Anwendungen müssen die Events in eine neue Struktur überführt werden. Die Nachbearbeitung kann sehr gut von Esper dargestellt werden. Bei einer externen Datamining-Komponente muss darauf geachtet werden, dass die Ergebnisse in einem Format vorliegen, das von Esper verarbeitet werden kann (siehe Abschnitt: 2.2.2). Wenn die Ergebnisse nicht dem Format entsprechen, müssen sie entsprechend konvertiert werden.
Die Komponenten 3b, 3c und 3d bilden das Ende des Event Stream Processing. Hier können eine oder mehrere Komponenten notwendig sein. Jede Anwendung benötigt eine individuelle Zusammenstellung dieser Komponenten. Die Aufgaben dieser Komponenten sind sehr simpel, daher müssen sie nicht mit Esper implementiert werden, wenn nur diese eine Aufgabe (zum Beispiel: Datenweiterleitung) umgesetzt werden soll. Esper bietet sich aber für eine Implementierung an, wenn Esper in der vorhergehenden Komponente Verwendung findet. Das heißt, wenn die Komponente Datamining oder die Komponente Nachbearbeitung mit Esper umgesetzt wird. In diesen Fällen ist Esper schon im Einsatz und kann die Aufgaben der Komponenten 3b bis 3d auch umsetzen.

4.3.2. Einsatz von Esper

Esper bietet zwei wesentliche Bausteine.

Abfrage: Alle Events werden durch die Abfrage weitergeleitet. Folgende Operationen können mit der Abfrage durchgeführt werden:

- Es stehen vielfältige Möglichkeiten zur Filterung des Event Streams zur Verfügung.
- Die Attribute des Events können neu strukturiert und umbenannt werden.
- Es können weitere Attribute hinzugefügt werden, wie zum Beispiel Zähler.
- Es kann ein neuer Event Stream erzeugt werden.

Listener/Subscriber: Diesem Baustein werden die erkannten Events aus der Abfrage übergeben. In diesen Bausteinen können weiter Aktionen durchgeführt werden.:

- Es kann Javacode ausgeführt werden.
- Die Attribute der Events können beliebig verändert werden
- Es können weitere Programmteile angesprochen werden
- Esgibt zwei verschiedene Umsetzungen des Event Stream Processing mit Esper:
  - Datamining mit Esper
  - Datamining ohne Esper

Beim Datamining mit Esper müssen die Events für die Komponente 3 (Datamining) in einem Format vorliegen, das Esper verarbeiten kann. Daher darf die Struktur der Events in der Komponente 2 (Vorverarbeitung) nicht zu sehr verändert werden. Da die Abfrage in Esper auch eine Anpassung der Attribute der Events anbietet, kann man hier die Komponenten 1a und 1b zusammenlegen. Anschließend wird von Esper ein neuer Event Stream erzeugt und im nächsten Schritt kann direkt mit dem Datamining begonnen werden.
Nach der Komponente 2 gibt es die Möglichkeit, einen weiteren Event Stream zu erzeugen und durch eine weitere Abfrage die Komponente 3a (Nachbearbeitung) zu implementieren. Eine oder mehrere der Komponenten 3b - 3d können von Esper in einem Listener/Subscriber dargestellt werden. Diesen Komponenten werden die Events vom Datamining oder von der Nachbearbeitung übergeben.

Beim Datamining ohne Esper bildet die Abfrage von Esper die Komponente 1a (Filter) und die Listener/Subscriber die Komponente 1b (Vorverarbeitung). Im Listener/Subscriber kann auch die Schnittstelle zur Datamining Komponente implementiert werden.
Da die Komponente 2 nicht mit Esper umgesetzt wird kommt es darauf an, ob noch eine

Nachbearbeitung der Ergebnisse notwendig ist. Diese Nachbearbeitung kann am besten durch den Abfragebaustein durchgeführt werden, wenn die Ergebnisse das richtige Format haben. Wenn dies der Fall ist, werden die Komponenten 3b - 3d in einem Listener/Subscriber von Esper umgesetzt. Dieser Baustein bekommt die erkannten Events der Nachbearbeitung übergeben und kann sie direkt weiterverarbeiten.
Wenn keine Nachbearbeitung notwendig ist, macht der erneute Einsatz von Esper keinen Sinn. In diesem Fall ist am besten ein eigenes Modul zu implementieren, das die Funktionen der Komponenten 3b - 3d übernimmt.

4.4. Beschreibung der Vorzüge und Grenzen dieses Konzepts

In diesem Kapitel werden die Möglichkeiten der Architektur beschrieben. Es wird herausgestellt für was sich die Architektur besonders eignet. Desweiteren wird gezeigt, welche Eigenschaften des Datamining im Event Stream Processing sich mit dieser Architektur umsetzten lassen. Abschließend werden die Grenzen der Architektur aufgezeigt. Es wird dargestellt, was sich mit der Architektur nicht umsetzen lässt.

4.4.1. Vorzüge

Mit Esper lassen sich sehr gut die eingehenden und ausgehenden Events bearbeiten. Diese Bearbeitung der Events findet sich in den Komponenten 1a und 1b, sowie 3a, 3b, 3c und 3d (Abschnitt: 4.2) wieder. Die Datenvorbereitung und -nachbereitung können mit Esper einfach umgesetzt werden. Die Umsetzung ist aufgrund der Eigenschaften von Esper(Filtern und Bearbeiten großer Mengen von Events in nahezu Echtzeit) effektiv und entspricht vollständig dem Konzept des Event Stream Processing.

Der Aufbau der Architektur ist modular. Dadurch können Änderungen vorgenommen werden ohne die gesamte Struktur zu beeinflussen. Für den Ablauf des Event Stream Processing ist Esper direkt verantwortlich. Das bedeutet, dass man hierfür keine weiteren Konzepte oder Module erstellen muss. Esper nimmt die eingehenden Events an, startet die Datamining Komponente und leitet die nachbearbeiteten Ergebnisse an die entsprechenden Komponenten weiter.

In bestimmten Fällen kann mit Esper sogar die Datamining-Komponente selbst umgesetzt werden. In diesen Fällen kommen alle Vorzüge des Event Stream Processing mit Esper zur Geltung.

4.4.2. Grenzen

Die passenden Algorithmen für das vorliegende Anwendungsmodell zu finden ist ein grundsätzliches Problem des Datamining im Event Stream Processing. Alle Algorithmen beschränken die Vielfalt der Anwendungsgebiete für die sie eingesetzt werden können. Daher gibt es nur wenige Algorithmen für eine spezielle Anwendung.

Wenn ein passender Algorithmus gefunden wurde, lässt dieser sich in der Regel nicht mit Esper darstellen. Esper ist ein Tool, das im weitesten Sinne Filter zur Verfügung stellt. Diese Filter-Funktion ist sehr komplex und mächtig, aber es lassen sich nur in Ausnahmefällen Datamining-Algorithmen mit diesen Filtern umsetzten. Daraus folgt, dass der Algorithmus in einem eigenen Modul umgesetzt werden muss.

Esper nimmt dem Programmierer viele Aufgaben ab. Das Tool Esper ist ein zu großer Aufwand für die vorliegende Anwendung, wenn die Aufgaben in den Komponenten Datenvorbereitung und Datennachbereitung nur sehr klein sind. In diesen Fällen kann eine direkte Programmierung der Komponenten besser sein.

5. Prototypische Implementierung

Im Folgenden wird die prototypische Umsetzung eines Datamining Algorithmus mit Esper beschrieben. Bei der Analyse der Algorithmen wurde festgestellt, dass sich die Umsetzung des Entscheidungsbaums eines Classification Algorithmus am besten für Esper eignet. Daher wurde für die Umsetzung des Prototyp ein Algorithmus aus dieser Kategorie gewählt. Bei der Bewertung der Algorithmen in Bezug auf die Anwendungsmodelle, zeigte sich, dass der Hoeffdingtree-Algorithmus die Anforderungen der Anwendungsmodelle am besten erfüllt. Daher wird mit dem Prototyp dieser Algorithmus implementiert.

5.1. Anwendungsfälle des Prototyp

Das Classification teilt sich auf in eine Initialisierungsphase und in die Phase des Datamining. Daher werden diese beiden Phasen auch im Prototyp getrennt behandelt. In der ersten Phase wird der Entscheidungsbaum erstellt und in der zweiten Phase laufen die Events durch den Entscheidungsbaum bis zur Entscheidung an einem Blatt des Baumes. Es werden nun die Anwendungsfälle beschrieben, die der Prototyp umsetzen soll.

In der Initialisierungsphase gelten die folgenden Anwendungsfälle:

Hoeffdingtree erstellen: Es wird ein Hoeffdingtree erstellt. Dieser Heoffdingtree bekommt alle Attribute mit ihren möglichen Values übergeben. Zu diesen Attributen gehört auch das Entscheidungsattribut und dessen Values.

Trainings Event an Hoeffdingtree senden: Es werden Trainings-Events an den Hoeffdingtree übergeben. Der Tree sortiert das Trainings-Event ein und erzeugt gegebenenfalls neue Knoten.

Esper-Umgebung erzeugen: Es wird ein Wrapper für Esper erzeugt, mit dem Vorverarbeitung, Classification durch Entscheidungsbäume und Nachbearbeitung an Esper weitergereicht werden.

Vorverarbeitung des Datamining setzten: Es werden nur Events an den Entscheidungsbaum weitergereicht, deren Values valide sind.

Entscheidungsbaum in eine Abfragestruktur umwandeln: Ein Entscheidungsbaum wird in eine Abfragestruktur von Esper umgewandelt.

Nachverarbeitung des Datamining setzen: Events, für die eine Entscheidung getroffen wurde, sollen von einem Programm nachbereitet werden. Dieses Programm soll vom Nutzer individuell gestaltet werden können.

In der Phase des Datamining ist der folgende Anwendungsfall umzusetzen:

Entscheidung für ein Event finden: Für ein Event wird durch Esper eine Entscheidung getroffen. Je nach Initialisierung erfolgen optional eine Vor- und Nachbearbeitung.

Um das Programm allgemein zu halten, wird eine abstrakte Klasse für den Entscheidungsbaum und ein Interface für die Nachbearbeitung definiert.

Entscheidungsbaum: Die abstrakte Klasse gibt die Methoden vor, die für eine Konvertierung in eine Abfragestruktur notwendig sind. Dadurch kann die Konvertierung auch durch einen anderen Entscheidungsbaum als dem Hoeffdingtree durchgeführt werden.

Nachbearbeitung: In der Architektur wurde dargelegt, dass die Nachbearbeitung durch die Listener von Esper umgesetzt werden sollte. Da aber jedes Anwendungsmodell eine individuelle Nachbearbeitung benötigt wird das Interface DecisionUpdateListener definiert. Dadurch ist es möglich, eine individuelle Umsetzung der Nachbearbeitung sicherzustellen. Mit der Anwendung des Factory Pattern kann in Esper dann ein Objekt erzeugt werden, das dieses Interface implementiert.

Eine detaillierte Beschreibung der Anwendungsfälle kann im Anhang C nachgelesen werden.

5.2. Aufbau des Prototyp

Die Umsetztung der Anwendungsfälle erfolgt in zwei Klassen. Die erste Klasse ist für die Erstellung des Hoeffdingtree zuständing. Die zweite Klasse ist eine Wrapper-Klasse, die die folgenden Funktionen übernimmt:

- Vorverarbeitung in Esper erzeugen
- Entscheidungsbaum konvertieren
- Nachverarbeitung in Esper setzen
- Events der Esper-Laufzeitumgebung übergeben

5.2.1. Erstellen des Entscheidungsbaums

Es wurden zwei Klassen implementiert die einen Entscheidungsbaum erzeugen. Den Aufbau kann man auch in der Abbildung 5.1 nachvollziehen.

SimpleDTree In dieser Klasse wird der Baum manuell erzeugt, in dem Wurzeln und Blätter frei gesetzt werden. Diese Klasse wurde implementiert, um Esper mit einem eigenen Entscheidungsbaum testen zu können.

Hoeffdingtree Diese Klasse setzt den Hoeffdingtree-Algorithmus um. Der Baum baut sich auf, indem Trainings-Events mit der Methode setExample(anExample) an den Baum übergeben werden.

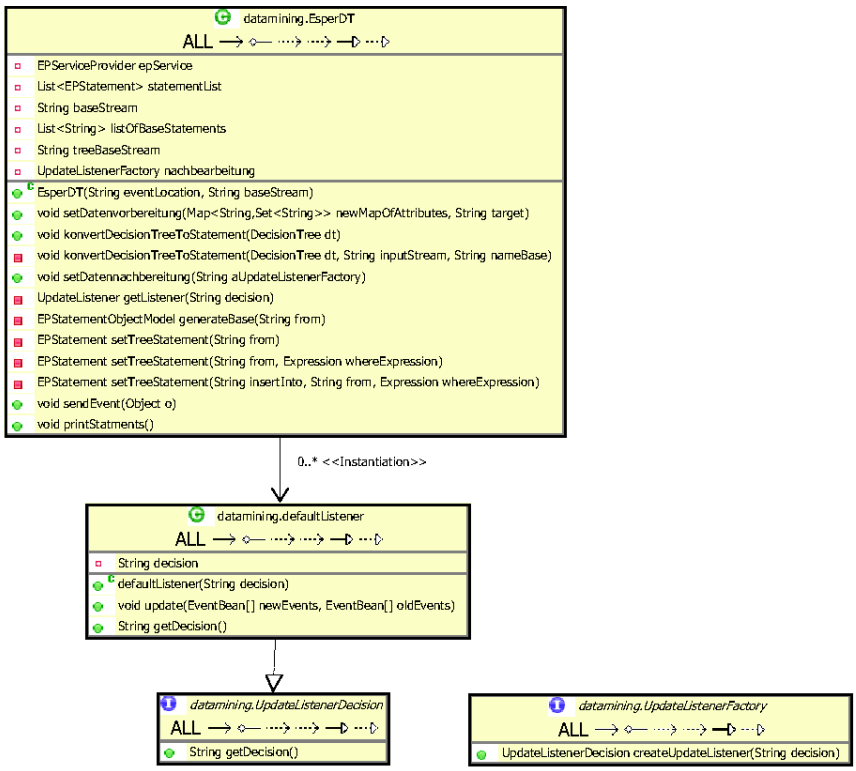

Abbildung 5.1.: Klassendiagramm Entscheidungsbaum

Die Klasse Hoeffdingtree wird mit den folgenden drei Werten initialisiert:

propability Hier wird eine Wahrscheinlichkeit angegeben, mit der das falsche Spaltungsattribut ausgewählt werden könnte.

targetAttribute Mit diesem Wert wird das Entscheidungsattribut identifiziert.

mapOfAttributesValues Mit dieser Map werden dem Hoeffdingtree alle Attribute mit ihren möglichen Values übergeben. Dieses ist nötig, um die Berechnungen anstellen zu könnnen.

5.2.2. Erstellen der Esperumgebung

Die Esperumgebung sorgt dafür, dass die Datenvorbereitung, Konvertierung des Entscheidungsbaums und Datenachbereitung mit Esper dargestellt werden.

Dafür wurden die folgenden Methoden implementiert:

setVorverarbeitung(aMap<String,List<String») Mit dieser Methode wir vor dem Entscheidungsbaum eine Abfrage erzeugt, die nur valide Events an den Entscheidungsbaum weiterreicht.

konvertDecisionTreeToStatement(aDecisionTree) Diese Methode konvertiert den Entscheidungsbaum in eine Abfragestruktur von Esper.

setNachverarbeitung(aUpdateListenerFactory) Mit der Factory wird es der Umgebung ermöglicht individuelle UpdateListener zu erzeugen. An die UpdateListener werden die Events übergeben, die in der Abfagestruktur an einem Blatt angekommen sind.

Der Aufbau der Umgebung kann auch der Abbildung 5.2 entnommen werden.

Die Esper-Umgebung benötigt bei der Initialisierung den Ort, an dem die Klasse des Events definiert ist. Diese Angabe ist für die Laufzeitumgebung von Esper notwendig.

Nachdem die Initialisierungsphase für das Classification abgeschlossen ist, kann mit der Methode setEvent (anEvent) ein Event an Esper übergeben werden.

setEvent(anEvent) Mit dieser Methode werden die Events an Esper übergeben und die Auswertung wird angestoßen.

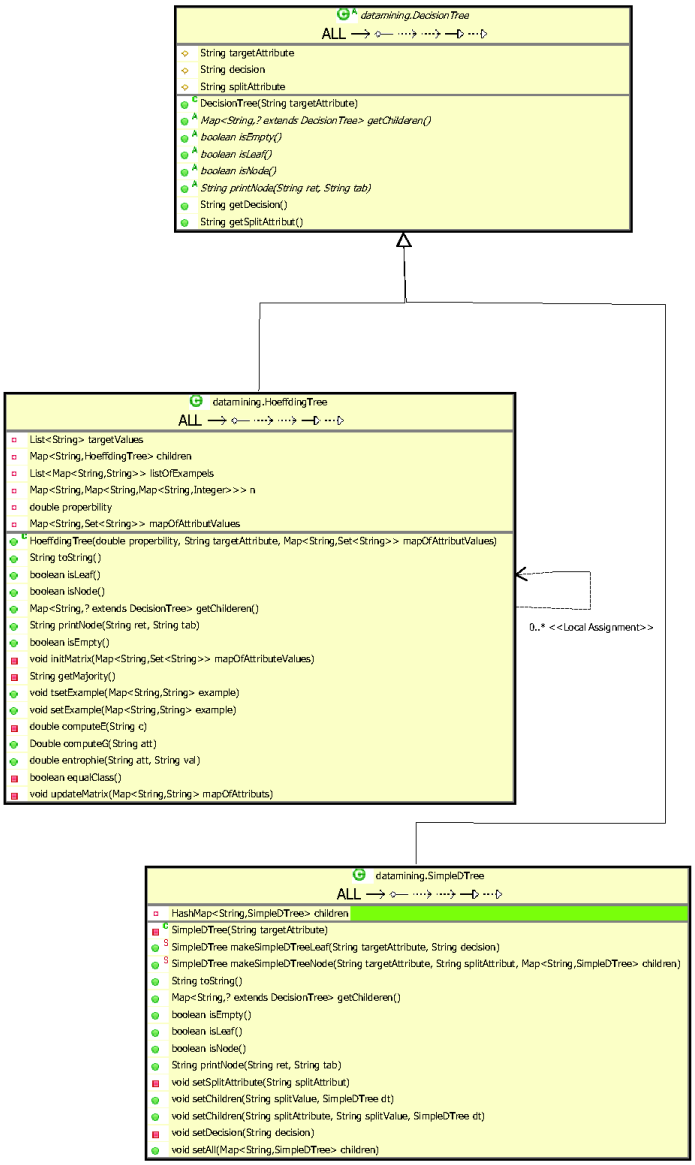

Abbildung 5.2.: Klassendiagramm des Wrapper für Esper

5.2.3. Ablauf

Der Ablauf gliedert sich in die folgenden Punkte die auch dem Sequenzdiagramm 5.3 zu entnehmen sind.

1. Erzeugen des Hoeffdingtree
2. Senden von Trainings-Events (so oft wie nötig)
3. Erzeugen der Esper-Umgebung
4. Setzen der Vorverarbeitung (optional)
5. Konvertieren des Entscheidungsbaums
6. Setzen der Nachbereitung (optional)
7. Senden von Events (so oft wie nötig)

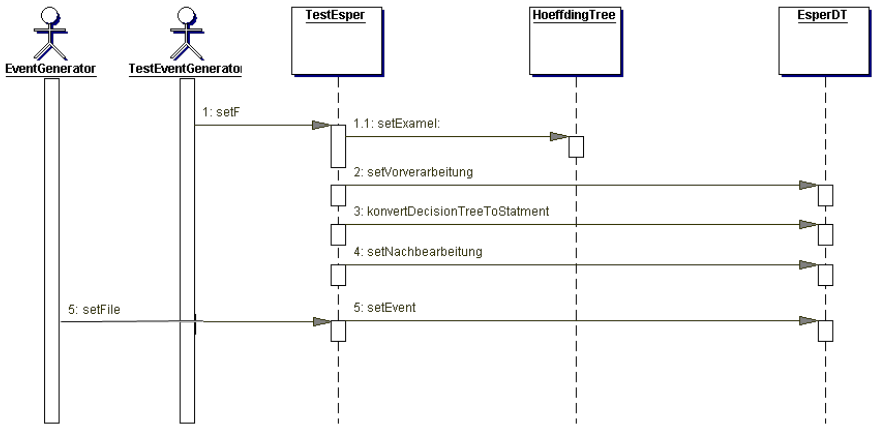

Abbildung 5.3.: Sequenzdiagramm des Prototypen

5.3. Eingesetzte Technologien

Zu den eingesetzten Technologien gehören neben der Programmiersprache und den Tools zur Erstellung der Software auch die Bibliotheken, die im Programm Anwendung finden.

5.3.1. Tool

- Eclipse (Ganymede)

5.3.2. Sprachen

- Java
- EPL (von Esper)

5.3.3. Bibliotheken

Neben dem Java Development Kit (JDK) 1.6.0_12 wurden die folgenden Bibliotheken eingesetzt:

- esper-3.0.0.jar
- esperio-3.0.0.jar
- cglib-nodep-2.2.jar
- commons-logging-1.1.1.jar
- log4j-1.2.15.jar
- antlr-runtime-3.1.1.jar

5.4. Zusammenfassung zum Prototyp

In dem Prototyp wurden alle Architekturelemente umgesetzt. Der Hoeffdingtree-Algorithmus erzeugt den Entscheidungsbaum in der Initialisierungsphase und der Wrapper setzt die Komponenten Datenvorbereitung, Datamining und Datennachbereitung um.

Datenvorbereitung: In dem Prototyp wurde die Komponente Filter umgesetzt, die alle Daten auf Validität überprüft.

Datamining: Ein innerer Knoten des Entscheidungsbaums durch eine Abfrage umgesetzt worden. Die Weiterleitung an den nächsten Knoten erfolgte dann mit dem Erzeugen eines neue Event Streams. Die Blätter des Entscheidungsbaums stellen die Updatelistener da.

Nachbereitung: Die Nachbereitung kann nicht automatisiert erzeugt werden, daher wird diese durch eine Factory dem Programm übergeben. Sollte keine Nachbearbeitung gesetzt werden, steht ein default-Listener zur Verfügung.

6. Zusammenfassung und Ausblick

6.1. Zusammenfassung

In dieser Arbeit wurden verschiedene Algorithmen vorgestellt, die sich für das Datamining auf Event Streams eignen. Die Algorithmen kamen aus den Bereichen Clustering, Classification und Frequency Counting. Besonderheiten des Event Stream Processing sind die reduzierten Ressourcen, mit denen die Datamining-Algorithmen auskommen müssen. Alle gefundenen Algorithmen lösen dieses Problem, indem sie die Vielfalt der Anwendungsgebiete, für die sie eingesetzt werden können, reduzieren.

Die Darstellungsform der Events ist für die Algorithmen von großer Bedeutung. Daher ist es in der Regel notwendig die eingehenden Events an die Anforderungen des Algorithmus anzupassen. Diese Anpassung der Events wurde in dieser Arbeit in der Komponente Datenvorbereitung zusammengefasst.

Je nach Anwendungsmodell gibt es für die Weiterverarbeitung der Ergebnisse unterschiedliche Möglichkeiten. Bei der Weiterverarbeitung handelt es sich um Weiterleitung oder Speicherung der Ergebnisse und das Triggern anderer Prozesse. Wenn die Komponenten, an die die Ergebnisse weitergereicht werden sollen, ein besonderes Format der Daten benötigen, dann ist noch eine Nachbearbeitung der Ergebnisse nötig. Diese Aufgaben sind in der Komponente Datennachbereitung untergebracht.

Bestimmte Algorithmen benötigen eine Initialisierungsphase (zum Beispiel: Classification) und andere Algorithmen können direkt die eingehenden Events bearbeiten (zum Beispiel: Clustering). Dieser Umstand muss bei der Erstellung des Programms gesondert berücksichtigt werden, da das Event Stream Processing selbst keine Initialisierungsphase hat.

Esper eignet sich gut, um die Datenvorbereitung und Datennachbereitung umzusetzen. Es ist aber nur in besonderen Fällen möglich die Datamining Komponente mit Esper zu implementieren. Ein Beispiel für einen solchen Sonderfall ist der Entscheidungsbaum bei Classification Algorithmen. Da aber beim Classification eine Initialisierungsphase notwendig ist, die nicht mit Esper umsetzbar ist, wird auch hier ein gesondertes Modul neben Esper benötigt.

Dieses zusammengefasst bedeutet, dass Esper eine gute Hilfe für die Umsetzung von Datamining-Algorithmen auf Event Streams ist. Mit Esper können nur in Außnahmefällen alle notwendigen Schritte des Algorithmus umgesetzt werden.

Wenn die Aufgaben in den Komponenten Datenvorbereitung und Datennachbereitung gering sind und Esper das Datamining nicht darstellen kann, kann das Tool ein zu großer Aufwand für die Anwendung sein.

6.2. Ausblick

In dieser Arbeit wurde ein Konzept vorgestellt, mit dem Datamining auf Event Streams umgesetzt werden kann. In weiteren Arbeiten könnte untersucht werden, wie sich die Algorithmen unter Last verhalten. Es gilt zu klären, wie viele Events pro Sekunde verarbeitet werden können und wie hoch die Schwankungen in der Frequenz ankommender Events sein darf.

Ein weiterer Punkt ist, welche Komponente (Datenvorbereitung, Datamining, Datennachbereitung) die Leistung des Prozessors am stärksten beansprucht. Ein wichtiger Punkt bei der Betrachtung der Leistungsfähigkeit ist vermutlich die Größe und Geschwindigkeit des RAM-Speichers. Wenn leistungsstarker RAM-Speicher zur Verfügung steht, könnte Esper auch mit größeren oder komplexeren Event-Mengen umgehen.

Es wurden in dieser Arbeit die Möglichkeiten von Esper vorgestellt, Entscheidungsbäume direkt umzusetzen. Das Feld des Datamining ist sehr groß, es ist durchaus vorstellbar, dass sich Esper noch für andere Datamining-Konzepte eignet, die in dieser Arbeit nicht untersucht wurden. Hier gilt es in weiteren Analysen diese Datamining-Konzepte zu identifizieren. Denkbar wären andere Arten des Classification, welche nicht den Entscheidungsbaum nutzen, um die Entscheidungsfunktion darzustellen.

Neben Esper gibt es noch andere Tools, die Event Stream Processing unterstützen. Man kann in folgenden Arbeiten diese Tools untersuchen und sie mit Esper vergleichen. Es ist auch eine Kombination verschiedener Tools denkbar, die sich in ihren Funktionen ergänzen um Datamining auf Event Streams umzusetzen.

Literaturverzeichnis

[1] AGGARWAL, C. C. ; HAN, J. ; WANG, J. ; YU, P. S.: A framework for clustering evolving data streams. In: *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, VLDB Endowment, 2003. – ISBN 0–12–722442–4, S. 81–92

[2] AGRAWAL, J. ; DIAO, Y. ; GYLLSTROM, D. ; IMMERMAN, N. : Efficient pattern matching over event streams. In: *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–102–6, S. 147–160

[3] BABU, S. ; WIDOM, J. : Continuous queries over data streams. In: *SIGMOD Rec.* 30 (2001), Nr. 3, S. 109–120. http://dx.doi.org/http://doi.acm.org/10.1145/603867.603884. – DOI http://doi.acm.org/10.1145/603867.603884. – ISSN 0163–5808

[4] BREIMAN, L. ; FRIEDMAN, J. H. ; OLSHEN, R. A. ; STONE, C. J.: *Classification and Regression Trees*. Belmont, California, U.S.A. : Wadsworth Publishing Company, 1984 (Statistics/Probability Series)

[5] CALIFORNIA, U. of: *How Much Information? 2003*. http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/execsum.htm#summary, Oktober 2003

[6] CHAKRAVARTHY, S. ; MISHRA, D. : Snoop: an expressive event specification language for active databases. In: *Data Knowl. Eng.* 14 (1994), Nr. 1, S. 1–26. http://dx.doi.org/http://dx.doi.org/10.1016/0169-023X(94)90006-X. – DOI http://dx.doi.org/10.1016/0169–023X(94)90006–X. – ISSN 0169–023X

[7] CHAKRAVARTHY, S. ; ADAIKKALAVAN, R. : Events and streams: harnessing and unleashing their synergy! In: *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–090–6, S. 1–12

[8] DAVID LUCKHAM, R. S.: *Event Processing Glossary - Version 1.1*. http://complexevents.com/wp-content/uploads/2008/08/epts-glossary-v11.pdf, Oktober 2008

[9] DING, Q. ; DING, Q. ; PERRIZO, W. : Decision tree classification of spatial data streams using Peano Count Trees. In: *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*. New York, NY, USA : ACM, 2002. – ISBN 1–58113–445–2, S. 413–417

[10] DOMINGOS, P. ; HULTEN, G. : Mining high-speed data streams. In: *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : ACM, 2000. – ISBN 1–58113–233–6, S. 71–80

[11] FAYYAD, U. ; PIATETSKY-SHAPIRO, G. ; SMYTH, P. : From data mining to knowledge discovery in databases. In: *AI Magazine* 17 (1996), S. 37–54

[12] FLANAGAN, D. : *Java in a Nutshell*. http://docstore.mik.ua/orelly/java-ent/jnut/ch06_02.htm, November 1999

[13] GABER, M. M. ; ZASLAVSKY, A. ; KRISHNASWAMY, S. : Mining data streams: a review. In: *SIGMOD Rec.* 34 (2005), Nr. 2, S. 18–26. http://dx.doi.org/http://doi.acm.org/10.1145/1083784.1083789. – DOI http://doi.acm.org/10.1145/1083784.1083789. – ISSN 0163–5808

[14] GARSON, G. D.: *Cluster Analysis*. http://faculty.chass.ncsu.edu/garson/PA765/cluster.htm, Oktober 2008

[15] GRAZIOLI, D. ; PASQUALI, E. : A scalable grid based application platform for high volumes of transactional event driven processes. In: *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–090–6, S. 211–219

[16] HAN, J. : *Data Mining: Concepts and Techniques*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2005. – ISBN 1558609016

[17] INC., E. : *EsperTech*. http://esper.codehaus.org/, Oktober 2008

[18] MANKU, G. S. ; MOTWANI, R. : Approximate frequency counts over data streams. In: *In VLDB*, 2002, S. 346–357

[19] PAPADIMITRIOU, S. ; BROCKWELL, A. ; FALOUTSOS, C. : Adaptive, hands-off stream mining. In: *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, VLDB Endowment, 2003. – ISBN 0–12–722442–4, S. 560–571

[20] SEBASTIANI, F. : Text categorization. In: *Text Mining and its Applications to Intelligence, CRM and Knowledge Management*, WIT Press, 2005, S. 109–129

[21] ZHANG, T. ; RAMAKRISHNAN, R. ; LIVNY, M. : BIRCH: an efficient data clustering method for very large databases. In: *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 1996. – ISBN 0–89791–794–4, S. 103–114

A. Beispiele

A.1. XML Node

XML Node Funkverkehr von Flugzeugen

```
1
2  <?xml version="1.0" encoding="ISO-8859-1"?>
3  <emissions>
4    <emission>
5      <job_id>3012</job_id>
6      <jobpos_id>10268</jobpos_id>
7
8      <emi_id>5349631503</emi_id>
9      <emi_sband>0</emi_sband>
10
11     <netcode>VTTEST099</netcode>
12
13     <emi_start>2006-06-22 02:18:30.019</emi_start>
14     <emi_status>0</emi_status>
15     <emi_ende>2006-06-22 02:18:36.57</emi_ende>
16
17 <!-- weitere Merkmale wurden hier
18       für den Erhalt der Übersicht ausgelasen -->
19
20     <files></files>
21
22     <ortungen>
23       <ortung>
24         <latitude>1.054425</latitude>
25         <longitude>-0.54409033</longitude>
26         <zeitpunkt>2006-06-19 16:58:09.66</zeitpunkt>
27       </ortung>
28     </ortungen>
```

```
29
30      <sse_products>
31        <sse_product>
32          <startzeit>2006-06-19 16:57:52.912</startzeit>
33          <decoder_ausgabe>
34            <ALE class="ALE" mode="MilStd␣188-141␣A">
35
36              <infounit channel-nr="1">
37                <content>
38                  <tag>[AMD]</tag>
39                  <text>CCCCPWO1NNNN</text>
40                </content>
41
42                <from>
43                  <callsign>UKE303</callsign>
44                  <tag>TIS</tag>
45                </from>
46
47                <rawdata>
48                  16:57:52:912[TO ][CRO][TIS][UKE303]
49                  [AMD]CCCCPWO1NNNN
50                </rawdata>
51
52                <time>2006-06-19T16:57:52.912</time>
53
54                <to>
55                  <callsign>CRO</callsign>
56                  <tag>TO</tag>
57                </to>
58              </infounit>
59
60              <infounit channel-nr="1">
61                <from>
62                  <callsign>CRO</callsign>
63                  <tag>TIS</tag>
64                </from>
65
66                <rawdata>
67                  16:58:04:212[TO ][UKE303][TIS][CRO]
68                </rawdata>
```

```
69
70                <time>2006-06-19T16:58:04.212</time>
71
72                <to>
73                  <callsign>UKE303</callsign>
74                  <tag>TO</tag>
75                </to>
76              </infounit>
77
78              <infounit channel-nr="1">
79                <from>
80                  <callsign>UKE303</callsign>
81                  <tag>TIS</tag>
82                </from>
83
84                <rawdata>
85                  16:58:08:467[SND][][TIS][UKE303]
86                </rawdata>
87
88                <time>2006-06-19T16:58:08.467</time>
89
90                <to>
91                  <callsign/>
92                  <tag>SND</tag>
93                </to>
94              </infounit>
95            </ALE>
96          </decoder_ausgabe>
97        </sse_product>
98      </sse_products>
99    </emission>
100 </emissions>
```

A.2. Rach Daten

Telefonnummer Integer

Art der Nummer Integer

Signalqualität Integer

Priorität des Anrufs Boolean

Id Char(20)

Leistung des Signals Integer

Zeitstempel UTC Time in dem Format “YY-MM-DD hh:mm:ss“

Relais Station Id Integer

gps Zeitstempel Integer

gps aktuell Boolean

gps Latetude -90.0 - 90.0 oder leer

gps Longitude -180.0 - 180.0 oder leer

angerufene Nummer Integer

Kanal Integer

Teilnehmer Id Integer

B. Code

B.1. Esper Beispielcode

Um die Beispiele einfach und verständlich zu halten, sind keine Handy- oder Funksprüche zugrunde gelegt worden. In den Beispielen wird immer eine Bestellung (OrderEvent) mit einem Namen (itemName) und einem Preis (price) verwendet.

B.1.1. POJO

Beispiel POJO

```
1  public class OrderEvent {
2      private String itemName;
3      private double price;
4
5      public OrderEvent(String itemName, double price) {
6          this.itemName = itemName;
7          this.price = price; }
8
9      public String getItemName() {
10         return itemName; }
11
12     public double getPrice() {
13         return price; }
14 }
```

Übergabe des POJO an die Laufzeitumgebung

```
1 OrderEvent event = new OrderEvent("shirt", 74.50);
2 epService.getEPRuntime().sendEvent(event);
```

B.1.2. Map

Beispiel Map und die Übergabe der Map an die Laufzeitumgebung

```
Map event = new HashMap();
event.put("shirt", itemName);
event.put("74.50", price);
epRuntime.sendEvent(event, "OrderEvent");
```

B.1.3. Node

Beispiel XML Node

```
<?xml version="1.0" encoding="UTF-8"?>
<OrderEvent xmlns="OrderEventSchema">
  <itemName> shirt </itemName>
  <price> 74.50 </price>
</OrderEvent>
```

Übergabe der XML Node an die Laufzeitumgebung

```
InputSource source =
        new InputSource(new StringReader(xml));
Document doc =
        builderFactory.newDocumentBuilder().parse(source);
epService.getEPRuntime().sendEvent(doc);
```

B.1.4. Abfragen

Beispiel: Abfrage String (mit einem POJO)

```
String expression =
        "select itemName, price from
de.pathName.OrderEvent where price < 100";

// übergabe an die Runtimeumgebung
EPStatement statement =
        epService.getEPAdministrator().createEPL(expression);
```

Beispiel: Abfrage Objekt (mit einem POJO)

```
1  EPStatementObjectModel model = new EPStatementObjectModel();
2  model.setSelectClause(SelectClause.create()
3      .add("itemName").add("price"));
4  model.setFromClause(
5          FromClause.create(
6                  FilterStream.create("de.pathName.OrderEvent")));
7  model.setWhereClause(Expressions.lt("price", 100));
8
9  // übergabe an die Runtimeumgebung
10 EPStatement statement =
11         epService.getEPAdministrator().create(model);
```

B.2. Hoeffding Tree Pseudocode

Der Hoeffdingtree-Algorithmus benötigt die folgenden Eingaben:

S Eine Sequenz von Beispielen

X Ein Set der Attribute

G Eine Funktion für die Bewertung der Attribute, um das Teilungsattribut zu bestimmen.

δ 1-δ ist die Wahrscheinlichkeit, dass das richtige Attribut gewählt wurde

for all class y_k **do**
  for all Wert x_{ij} von jedem Attribut $X_i \in X$ **do**
    Sei $n_{ijk}(l_1) = 0$
  end for
end for
for all Beispiel (x, y_k) in S **do**
  Sortiere (x, y) in ein Blatt l in HT ein
  for all x_{ij} in x so das $X_i \in X_l$ **do**
    Increment $n_{ijk}(l)$
  end for
  Markiere l mit der mit der häufigsten Klassen der bisherigen Beispiele
  if Die Beispiele in l gehören nicht alle der selben Klasse an **then**
    for all Attribut $X_i \in X_l - \{X_0\}$ **do**
      Berechne $\overline{G_l}(X_i)$ mit den Zählern $n_{ijk}(l)$
    end for
    Sei X_a das Attribut mit dem höchsten $\overline{G_l}$
    Sei X_b das Attribut mit dem zweit höchsten $\overline{G_l}$
    Berechne ϵ mit Gleichung 1
    if $\overline{G_l}(X_a) - \overline{G_l}(X_b) > \epsilon$ AND $Xa \neq X_0$ **then**
      Ersetzte l durch einen internen Knoten der in X_a trennt
      for all Zweig der Trennung **do**
        Füge ein neues Blatt l_m hinzu
        Sei $X_m = X - \{X_a\}$
        Sei $\overline{G_m}(X_0)$ das $\overline{G}$ die Klasse mit der höchsten Frequenz in l_m
        for all Klasse y_k und Wert x_{ij} aller Attribute $X_i \in X_m - \{X_0\}$ **do**
          Sei $n_{ijk}(l_m) = 0$
        end for
      end for
    end if
  end if
end for

Procedure 1: HoeffdingTree (S,X,G,δ)

B.3. ID3 Algorithmus

Der ID3-Algorithmus benötigt die folgenden Punkte:

T Eine Liste von Beispielen

Eine Funktion um das informativste Merkmal zu bestimmen

if alle Elemente aus T (Daten) gehören zu einer Klasse **then**
  Konstruiere ein Blatt mit der Klasse als Bezeichner
else
  wähle das informativste Merkmal x_i
  for all Werte von Merkmal x_i **do**
    Konstruiere Teilbäume rekursiv mit den entsprechenden Teilmengen als Daten
  end for
  Konstruiere einen Baumknoten mit Bezeichner x_i und hänge alle erzeugten Teilbäume an
end if

Procedure 2: ID3Algorithmus (T)

C. Anwendungsfälle des Prototyp

C.1. Hoeffdingtree erstellen

Name: Hoeffdingtree erstellen

Kurzbeschreibung: Es wird ein Objekt erzeugt, mit dem nach dem Hoeffdingtree Algorithmus ein Entscheidungsbaum erzeugt werden kann.

beteiligte Akteure: Anwender

Hauptablauf: Der Anwender initialisiert ein Objekt das den Heoffdingtree Algorithmus umsetzt und das von der abstrakten Klasse Deciciontree abgeleitet sein muss.Die Abstrakte Klasse schreibt die folgenden Methoden vor:

- isEmpty();
- isLeaf();
- isNode();
- printNode();
- getChilderen();
- getDecision()
- getSplitAttribut()

Das Objekt bekommt eine Map mit allen Attributen und ihren möglichen Values übergeben. Zu diesen Attributen gehören auch das Entscheidungsattribut mit seinen möglichen Values.

andere Abläufe: keine

Auslöser: Initialisierung des Datamining

Erweiterungspunkte: keine

Vorbedingungen: Es existiert eine Map in der die alle Attribute mit ihren möglichen Values als Liste abgelegt sind. (inklusive des Entscheidungsattributes)

Nachbedingungen: keine

nicht-funktionale Anforderungen: keine

Autor: Jens Ellenberg

Datum: 20.05.2009

Status: abgeschlossen

Version: 1.0

C.2. Trainings Event an Hoeffdingtree senden

Name: Trainings-Event an einen Hoeffdingtree senden

Kurzbeschreibung: Es wird ein Trainings-Event an eine Hoeffdingtree-Implementierung übergeben. Die Implementierung verarbeitet das Event.

beteiligte Akteure: Trainings-Event-Generator

Hauptablauf: Ein Trainings-Event wird an eine Hoeffdingtree-Implementierung übergeben. In dem Tree wird das Event in die bestehende Baumstruktur einsortiert. Anschließend überprüft der Algorithmus, ob das Blatt geteilt werden muss. Sollte dies der Fall sein, wird ein Spaltungs-Attribut gesetzt. Anschließend werden neue Blätter erzeugt und diese dem Knoten hinzugefügt. Die Trainingsdaten des Knoten werden an die Kinder weitergeleitet.

andere Abläufe: keine

Auslöser: senden des Training-Events

Erweiterungspunkte: keine

Vorbedingungen: Es wurde ein Hoeffdingtree-Objekt erzeugt und dieses Objekt wurde mit den richtigen Attributen initialisiert.

Nachbedingungen: keine

nicht-funktionale Anforderungen: keine

Autor: Jens Ellenberg

Datum: 20.05.2009

Status: abgeschlossen

Version: 1.0

C.3. Esper-Umgebung erzeugen

Name: Esper-Umgebung erzeugen

Kurzbeschreibung: Es wird eine Wraper für Esper erzeugt, der Datenvorverarbeitung, einen Entscheidungsbaum und Datennachbearbeitung mit Esper erstellt.

beteiligte Akteure: Nutzer

Hauptablauf: Es wird ein Objekt der Klasse EsperDT erzeugt. Dieses Klasse stellt die folgenden Methoden zur Verfügung:

- setDatenvorverarbeitung(anExpersion)
- konvertDesicionTreeToStatment(aDecisionTree)
- setDatennachbearbeitung(aUpdateListenerFaktory)
- sendEvents(anEvent)
- printStatment()

Die Klasse wird mit dem Ort und dem Klassennamen der Events initialisiert.

andere Abläufe: keine

Auslöser: Nutzer

Erweiterungspunkte: keine

Vorbedingungen: Es existiert eine POJO-Repräsentation der Events, die diesem Objekt übergeben werden sollen. Die Position der POJO Klasse muss ebenfalls bekannt sein.

Nachbedingungen: keine

nicht-funktionale Anforderungen: keine

Autor: Jens Ellenberg

Datum: 20.05.2009

Status: abgeschlossen

Version: 1.0

C.4. Vorverarbeitung des Datamining setzten

Name: Vorverarbeitung des Datamining setzten

Kurzbeschreibung: Es wird eine Map mit allen Attributen und ihren möglichen Values dem Esper-Wraper übergeben. Dieser erstellt eine Abfrage, die eine Validierung der eingehenden Events durchführt.

beteiligte Akteure: Nutzer

Hauptablauf: Eine Map mit allen Attributen und ihren möglichen Values wird mit der Methode setVorverarbeitung an das Objekt der Klasse EsperDT übergeben. Es wird eine Abfrage erzeugt, die nur Events zulässt, die in ihren Attributen ein Value haben, das in der übergebenen Map vorhanden ist. Sollte das Entscheidungsattribut in dieser Map existieren, wird es nicht beachtet. Die Reihenfolge, in der Vorverarbeitung setzen, Umwandeln des Entscheidungsbaums und Nachbearbeitung setzen durchgeführt werden, soll unerheblich sein.

andere Abläufe: keine

Auslöser: Nutzer

Erweiterungspunkte: keine

Vorbedingungen: Es existiert eine Map, in der alle Attribute mit ihren möglichen Values als Liste abgelegt sind.

Nachbedingungen : keine

nicht-funktionale Anforderungen: keine

Autor: Jens Ellenberg

Datum: 20.05.2009

Status: abgeschlossen

Version: 1.0

C.5. Entscheidungsbaum in eine Abfragestruktur umwandeln

Name: Entscheidungsbaum in eine Abfragestruktur umwandeln

Kurzbeschreibung: Es wird ein Entscheidungsbaum, der von der abstrakten Klasse DecisionTree abgeleitet ist, an den Esper-Wrapper übergeben.

beteiligte Akteure: Nutzer

Hauptablauf: Der Esper-Wrapper konvertiert den Entscheidungsbaum in eine Abfragestruktur von Esper. Die Reihenfolge, in der Vorverarbeitung setzen, Umwandeln des Entscheidungsbaums und Nachbearbeitung setzen durchgeführt werden, soll unerheblich sein.

andere Abläufe: keine

Auslöser: Nutzer

Erweiterungspunkte: keine

Vorbedingungen: Es wurde ein Esper-Wrapper erzeugt und mit dem richtigen Event initialisiert. Außerdem wurde ein Entscheidungsbaum erzeugt, der von der abstrakten Klasse DecisionTree abgeleitet ist.

Nachbedingungen : keine

nicht-funktionale Anforderungen: keine

Autor: Jens Ellenberg

Datum: 20.05.2009

Status: abgeschlossen

Version: 1.0

C.6. Nachverarbeitung des Datamining setzen

Name: Nachverarbeitung des Datamining setzen

Kurzbeschreibung: Es wird eine Factory für eine individuelle Implementierung eines DecisionUpdateListener an den Esper Wraper übergeben.

beteiligte Akteure: Nutzer

Hauptablauf: Dem Esper Warper wird eine Factory für Klassen übergeben, die das Interface DecisionUpdateLitener implementieren. Das Interface stellt die folgenden Methoden zur Verfügung:

- update(newEvents, oldEvents)
- getDecision()

Dem Listener werden alle Events übergeben, die ein Blatt des Entscheidungsbaums erreicht haben. Der Listener übernimmt die Weiterverarbeitung der Events. Die Reihenfolge, in der Vorverarbeitung setzen, Umwandeln des Entscheidungsbaums und Nachbearbeitung setzen durchgeführt werden, soll unerheblich sein.

andere Abläufe: keine

Auslöser: Nutzer

Erweiterungspunkte:

Vorbedingungen:

Nachbedingungen:

nicht-funktionale Anforderungen:

Autor: Jens Ellenberg

Datum: 20.05.2009

Status: abgeschlossen

Version: 1.0

C.7. Entscheidung für ein Event finden

Name: Entscheidung für ein Event finden

Kurzbeschreibung: Es wird ein Event, für das noch keine Entscheidung getroffen wurde, an den Esper-Wrapper übergeben und mit Esper wird eine Entscheidung getroffen.

beteiligte Akteure: Event-Generator

Hauptablauf: Ein Event wird an Esper übergeben. Für dieses Event sucht Esper mit der Abfragestruktur eine Entscheidung. Sollte eine Vorverarbeitung gesetzt sein, muss das Event zuerst diese durchlaufen, bevor es der Abfrage übergeben wird. Sollte eine Nachbearbeitung gesetzt sein, dann wird das Event dieser Nachbearbeitung übergeben. Sollte das nicht der Fall sein, nutzt der Warpper eine Default-Implementierung, in der das Event mit der Entscheidung auf der Konsole ausgegeben werden.

andere Abläufe: keine

Auslöser: Event-Generator

Erweiterungspunkte: keine

Vorbedingungen: Es wurde ein Esper-Wrapper für dieses Event erzeugt und dem Wrapper wurde ein Entscheidungsbaum zur Konvertierung übergeben.

Nachbedingungen: Wenn eine Nachbereitung der Events gesetzt wurde, müssen die Schnittstellen in dieser Nachbereitung genutzt werden, zur Verfügung stehen.

nicht-funktionale Anforderungen: keine

Autor: Jens Ellenberg

Datum: 20.05.2009

Status: abgeschlossen

Version: 1.0

Index

Glossar

Datamining
: Datamining ist das Durchsuchen von Datenmengen nach zusätzlichen Informationen. Diese zusätzlichen Informationen sind in der Regel Muster innerhalb der Daten.

Diskrete Wavelet-Transformation
: Die Diskrete Wavelet-Transformation ist eine Zeit-Frequenz Transformation, die diskret durchgeführt wird. Die Frequenzen eines Signals werden in einem Fenster dargestellt. Durch die Skalierung des Fensters ergibt sich, wie durch Modulation, eine Frequenzverschiebung. Allerdings wird gleichzeitig mit einer Frequenzerhöhung die Zeitdauer („Breite“ im Zeitbereich) des Fensters verringert. Dadurch ergibt sich bei höheren Frequenzen eine bessere zeitliche Auflösung. Bei niedrigen Frequenzen wird die Frequenzauflösung besser, dafür wird die Zeitauflösung schlechter.

Echtzeit
: Etwas findet in „Echtzeit“ statt, wenn es keinen wesentlichen Verzug zwischen dem Start einer Aktion und dem Ergebnis dieser Aktion gibt. (Anmerkung: Es gibt verschiedene Abstufungen des Begriffs „Echtzeit“, die sich darauf beziehen, wie hart der Begriff „wesentlich“ ausgelegt wird. Diese Abstufungen sind in dieser Arbeit nicht relevant.)

Esper
: Esper ist ein Tool, mit dem Abfragen auf Event Streams einfach und effektiv umgesetzt werden können. Esper gibt es als Bibliotheken für Java und .Net [17](siehe auch Kapitel: 2.2).

Event
: Events sind atomar, sie passieren entweder vollständig oder gar nicht. Außerdem können Events in zwei Klassen aufgeteilt werden: primitiv und kombiniert. Primitive Events treten zu einem definierten Zeitpunkt auf. An diesem Zeitpunkt werden sie initiiert und wieder terminiert. Kombinierte Events sind zwei Events die zusammengehören und sich auf ein Intervall beziehen. Das erste Event ist das initiierende und das letzte Event das terminierende. Beide Events werden erst mit dem terminierenden Event erkannt, da sie erst dann vollständig sind. Diese Art der Eventerkennung wird „Point-Based Semantic“ genannt.[6]

Event Processing Language
Eine EPL ist eine SQL-artige Sprache, die um weitere Funktionen erweitert ist.

Event Stream
Das ist ein Stream aus Events

Iterative Dichotomiser 3
Es ist ein Algorithmus, der zur Erzeugung eines Entscheidungsbaums genutzt wird. Pseudocode: B.3

k-Means-Algorithmus
Bei diesem Verfahren werden ähnliche Objekte mit einer Abstandsfunktion einander zugeordnet. Die Anzahl der Zentren (k) ist dabei vorgegeben.

Kleenesche Hülle
Die Kleensche Hülle ist die Menge aller Wörter, die sich aus den Symbolen eines Alphabets erzeugen lassen. Das leere Wort gehört dabei zur Kleenschen Hülle dazu.

Laufzeitumgebung
Ein Programm, das einen Service anbietet. Dieses Programm kann über Schnittstellen angesprochen werden, um den Service zu nutzten.

Listener
Listener ist ein Konzept in Esper, das auf einer Abfrage lauscht und die Events übergeben bekommt, die zu einem Treffer der Abfrage geführt haben. Sie werden in der Laufzeitumgebung an eine Abfrage gebunden. Im Gegensatz zu Subscriber können mehrere Listener an eine Abfrage gebunden werden aber die Events werden nicht typisiert übergeben.

Metainformationen
Metainformationen sind zusätzliches Wissen, welches sich auf den Inhalt von Informationen bezieht. Zum Beispiel das Wissen, dass ein Funkspruch zu einer Kette von Funksprüchen gehört, die zusammengefasst ein Gespräch ergeben.

Multithreadsave
Multithreadsave bedeutet, dass verschiedene Threads zeitgleich auf einem Computer ausgeführt werden können, ohne dass diese Threads in einen Konflikt geraten.

Peano Count Tree
Ein gloPTree ist ein Baum, in dem mehrdimensionale binäre Daten gespeichert werden. Diese Daten können verlustfrei reduziert werden. Ein Datenfeld (in der 2. Dimension), wird in vier Felder (2 x 2) große Kacheln aufgeteilt. Jede Kachel, in der untersten

Ebene des Baumes, wird zu einem Knoten zusammengefasst. Somit hat jeder Knoten vier Kinder. In den Knoten eines P-Trees werden die Anzahl der binären Einsen der Kinder gespeichert. Ein Knoten, der vier Blätter hat und in jedem Blatt eine Eins stehen hat, besitzt also den Wert Vier. Jede höhere Ebende des P-Tree hat vier Blätter einer tieferen Ebene. Das Datenfeld wird um leere Kacheln erweitert, wenn das Datenfeld nicht in ein logarithmisches Raster passt. Vollständige (nur Einser) oder leere Zweige (nur Nullen) können entfernt werden, da ihr Aufbau und Inhalt bekannt sind.

Plain Old Java Object
POJO´s sind Java-Klassen, die den JavaBean-Konventionen [12] folgen.

Stream
Stream ist das englische Wort für Strom. Hier ist es als eine Aneinanderreihung von Objekten gemeint. Diese Objekte werden in der Regel von einem Modul erzeugt und werden einem weiteren Modul übergeben. Dieses erzeugen und übergeben geschieht kontinuierlich über einen bestimmten Zeitabschnitt. Dadurch entsteht die Aneinanderreihung der Objekte, welche als Stream bezeichnet wird.

Subscriber
Subscriber sind ein Konzept in Esper, das auf einer Abfrage lauscht und die Events übergeben bekommt, die zu einem Treffer der Abfrage geführt haben. Sie werden in der Laufzeitumgebung an eine Abfrage gebunden. Im Gegensatz zu Listener kann maximal ein Subsriber an eine Abfrage gebunden werden aber die Events werden typisiert übergeben.

Trigger
Ein Trigger ist ein Auslöser für eine Anwendung. Beispielsweise soll eine aufwändige Überwachung eines Gebietes erst erfolgen, wenn verdächtige Handygespräche in diesem Gebiet entdeckt werden. Das Entdecken dieser verdächtigen Gespräche wäre ein Trigger für die Überwachung.

Wavelet
Mit einem Wavelet werden Fequenzen üben einen Zeitraum dargestellt. Dabei ist die Skalierung der Zeitachse von der Frequenz abhängig.

Wrapper
Ein Wrapper ist eine Programmiertechnik um eine bestimmte Klasse zu nutzen und ihr zusätzliche Funktionen zu geben oder die bestehenden Methoden zu erweitern.

XPath
Die XML Path Language (XPath) ist eine vom W3-Konsortium entwickelte Abfragesprache, um Teile eines XML-Dokumentes zu adressieren.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungs- und Studienordnung des Bachelorstudiengangs Angewandte Informatik an der Hochschule für Angewandte Wissenschaften Hamburg, vom 22. November 2001, geändert am 07. Dezember 2004, nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 18. Juni 2009

Ort, Datum

Unterschrift