

Masterarbeit

**Eine adaptiv konfigurierbare Web-Umgebung zur visuellen
Spezifikation von Logfile-Analysen**

Malte Wessel
Matrikelnummer: 2240808



Abteilung Informatik und angewandte Kognitionswissenschaft
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

11. Juli 2016

Betreuer:
Prof. Dr. H. U. Hoppe
Tilman Göhnert Tobias Hecking

Inhaltsverzeichnis

1. Einleitung	1
1.1. Motivation	1
1.2. Aufgabenstellung	2
1.3. Aufbau der Arbeit	2
2. Grundlagen	3
2.1. Data-Mining & Wissensentdeckung	3
2.1.1. Sequential Pattern Mining	5
2.1.2. Logfiles und Datenformate	7
2.2. Workflow-basierte Analyse-Plattformen	9
2.2.1. Visuelle Sprachen	10
2.2.2. Verwandte Arbeiten	11
2.2.2.1. RapidMiner	11
2.2.2.2. Konstanz Information Miner	13
2.3. Analytics Workbench	14
2.3.1. Benutzeroberfläche	15
2.3.2. Architektur	17
2.3.3. Analyse-Komponenten	18
2.3.4. Logile-Analysen	19
2.3.4.1. Activity Stream Filter	19
2.3.4.2. Sequential Pattern Mining	20
2.4. Technologische Grundlagen	20
2.4.1. Single-page Application	21
2.4.2. React-Framework	22
2.4.3. Self-adaptive Software	24
3. Ansatz	27
3.1. Anforderungsanalyse	27
3.1.1. Analyse Module	27
3.1.2. Schwach strukturierte Daten	29
3.1.3. Anforderungen	29
3.2. Adaptive Konfiguration	30
3.2.1. Meta-Daten	30
3.2.2. Erzeugung von Meta-Daten durch Module	32
3.2.3. Erzeugung von Meta-Daten durch Workflow	32
3.2.4. Transformation von Meta-Daten	33
3.2.5. Adaption der Benutzeroberflächen	36
3.2.6. Zusammenfassung	37
3.3. Analyse-Komponenten	37
3.3.1. Activity Stream Filter	38
3.3.2. Sequential Pattern Mining	40

Inhaltsverzeichnis

3.4.	Arbeitsumgebung	42
3.4.1.	Workflow-Spezifikation	42
3.4.2.	Konfiguration, Ergebnisse und Verwaltung	44
4.	Implementierung	49
4.1.	Frontend	49
4.1.1.	Komponenten-Hierarchie	49
4.1.2.	Datenfluss & Kommunikation	52
4.1.3.	Workflow-Durchführung	54
4.1.4.	Adaptive Konfiguration	55
4.2.	Analyse-Module	58
4.2.1.	Meta-Analyse Modul	58
4.2.2.	Activity Stream Filter Modul	60
4.2.3.	Sequential Pattern Mining Modul	62
5.	Anwendungsbeispiel	65
5.1.	Selektion der Daten	65
5.2.	Bereinigung der Daten	66
5.3.	Transformation und Analyse	67
5.4.	Evaluation und Exploration	68
5.5.	Zusammenfassung	70
6.	Zusammenfassung und Ausblick	71
A.	Eidesstattliche Erklärung	73
B.	Quellcode der Software	75
	Literaturverzeichnis	77

Abbildungsverzeichnis

2.1.	Sequenz-Datenbank (links) und dazugehörige häufige Muster (rechts)	6
2.2.	Input-Format für den CM-SPAM-Algorithmus	7
2.3.	Beispiel Activity Stream Eintrag	8
2.4.	Snap! (BYOB), visuelle Programmiersprache	10
2.5.	RapidMiner Studio	12
2.6.	Konstanz Information Miner	13
2.7.	Benutzeroberfläche der Analytics Workbench	15
2.8.	Architektur der Analytics Workbench (vgl. [Rem15])	16
2.9.	Filter Description	18
2.10.	Activity Stream Filter Modul	19
2.11.	Sequential Pattern Mining Modul	20
2.12.	React-Komponenten	23
2.13.	Redux-Beispiel	25
2.14.	Architektur mit React und Redux	25
2.15.	Konzept für adaptive Benutzeroberflächen	26
3.1.	Beispiel für Meta-Daten für Activity Streams	31
3.2.	Ansatz 1: Erzeugung von Meta-Daten durch Module selber	32
3.3.	Ansatz 2: Erzeugung von Meta-Daten durch unabhängigen Workflow	33
3.4.	Ansatz 3: Erweiterung um Meta-Ebene	34
3.5.	Meta-Daten-Transformationsfunktion Activity Stream Filter	36
3.6.	Meta-Daten-Transformationsfunktion Sequential Pattern Mining	36
3.7.	Konzept für die Benutzeroberfläche des Activity Stream Filter Moduls	39
3.8.	Konzept für die Benutzeroberfläche des Sequential Pattern Mining Moduls	41
3.9.	Konzept für die Benutzeroberfläche der Workbench	43
3.10.	Konzept für die Benutzeroberfläche der Workbench, Hinzufügen von Modulen	45
3.11.	Konzept für die Benutzeroberfläche der Workbench, Modul-Konfiguration	47
4.1.	Architektur des Frontends	50
4.2.	Neue Benutzeroberfläche der Workbench	51
4.3.	React-Komponenten	53
4.4.	Ablauf einer Workflow-Durchführung	55
4.5.	Module-Status, von links nach rechts: Standard, Warten, Durchführen, Fertig	55
4.6.	Oberflächenbeschreibung und Oberflächen-Transformationsfunktion des Sequential Pattern Mining Moduls	56
4.7.	Adaptive Konfiguration	57
4.8.	Ausschnitt des Ergebnisses einer Meta-Daten-Analyse	59
4.9.	Vereinfachte Implementierung der matchesFilter-Funktion	60
4.10.	Oberflächenbeschreibung und Oberflächen-Transformationsfunktion des Activity Stream Filter Moduls	61

Abbildungsverzeichnis

4.11. Implementierung der Übersetzung von Activity Stream Einträgen in eine Sequenz-Datenbank (vereinfacht)	62
5.1. Datenselektion durch Direct Uploader und Blick in die Meta-Daten	65
5.2. Adaptive Konfiguration des Activity Stream Filter Moduls	66
5.3. Konfiguration des Sequential Pattern Mining Moduls	68
5.4. Ergebnisse der Analyse	69
5.5. Erweiterung des Workflows, zur Gegenüberstellung von Nutzergruppen . .	69

1. Einleitung

1.1. Motivation

Durch die weite Verbreitung von Internetzugängen und einer stetigen Weiterentwicklung von Webtechnologien, sind Webplattformen allgegenwärtig. Neben herkömmlichen textbasierten Webseiten können auch komplexe Anwendungen über das Internet ausgeliefert und genutzt werden. Wikis, soziale Netzwerke und E-Learning Plattformen sind nur einige Beispiele für webbasierte Anwendungen. Ein Vorteil von Webplattformen ist, dass Änderungen der zugrundeliegenden Software nicht wie bei Desktop-Anwendungen als Update heruntergeladen werden müssen, sondern beim erneuten Aufruf der Anwendung automatisch mitgeliefert werden. So ist es möglich, Webplattformen im laufenden Betrieb zu analysieren und entsprechend der Erkenntnisse weiterzuentwickeln. Zentral bei der Verbesserung von Webplattformen und insbesondere dessen grafischer Benutzeroberflächen ist das Wissen, wie Nutzer mit dem System interagieren. Beim Start einer neuen Webplattform ist beispielsweise nicht bekannt, welche Funktionen für Nutzer besonders relevant sind und welche Pfade innerhalb der Anwendung genutzt werden. Für die Analyse von Webplattformen kommen dabei verschiedene Techniken zur Anwendung. Häufige Nutzungsmuster lassen sich mit Logfile-Analysen mittels Sequential Pattern Mining untersuchen. Dazu eignen sich Analyse-Plattformen wie die Analytics Workbench, die von der Collide Research Group an der Universität Duisburg-Essen entwickelt wurde. Initial wurde die Workbench entwickelt, um Forschern bei der Analyse von Netzwerken zu unterstützen [GHHH13]. Durch die offene Architektur sind mittlerweile aber zahlreiche Analyse-Techniken möglich, sodass die Workbench auch zur Analyse von Logfiles aus Webplattformen genutzt wird. Für die Anwendung dieser Analyseverfahren ergeben sich allerdings einige Herausforderungen. Dazu gehört die allgemeine Vorverarbeitung der erhobenen Daten, wie das Filtern auf Basis beteiligter Nutzer oder interessanter Zeiträume, die konkrete Vorverarbeitung der Daten für die Analyse, wie die Einteilung in Sequenzen und die Übersetzung zwischen den Eingabedaten und einem für die Verarbeitung durch einen Algorithmus geeigneten Format. Schließlich muss die Analyse-Plattform eine flexible Spezifikation des Analyse-Workflows ermöglichen. Um die Aktivitäten von Nutzern aufzuzeichnen, kommen häufig schwach strukturierte Datenformate wie Activity Streams zum Einsatz. Problematisch dabei ist, dass die Beschaffenheit dieser Daten variieren kann [SAN12]. Sollen die Daten beispielsweise anhand bestimmter Merkmale gefiltert werden, so muss die Analyse-Plattform in der Lage sein, diese Merkmale zu erkennen und dem Nutzer entsprechende Konfigurationsmöglichkeiten bereitzustellen. Eine Adaption der Konfigurationsmöglichkeiten an die zu verarbeitenden Daten ist also notwendig für eine explorative Analyse von Logfiles und Activity Streams mittels Sequential Pattern Mining.

1. Einleitung

1.2. Aufgabenstellung

Das Ziel dieser Arbeit ist die Entwicklung adaptiv konfigurierbarer Analysekomponenten zur Analyse von Activity Streams mittels Sequential Pattern Mining sowie die Umsetzung einer Web-Umgebung zur Spezifikation und Konfiguration von auf diesen Komponenten basierenden Analyse-Workflows. Die Komponenten sollen in eine bestehende, auf einem Multi-Agentensystem aufbauende Analyse-Umgebung integriert werden. Dabei soll die bestehende Umgebung angepasst oder ersetzt werden. Die Arbeit umfasst folgende Aufgaben:

- Erarbeitung der Grundlagen zum Thema Logfile-Analyse, insbesondere Sequential Pattern Mining
- Erarbeitung der technischen Grundlagen zur Darstellung und Manipulation visueller Sprachen zur Spezifikation von Workflows mittels Web-Technologien
- Konzept für flexibel konfigurierbare Analysekomponenten, die die Analyse von Activity Streams mittels Sequential Pattern Mining unter Verwendung der SPMF-Bibliothek erlauben
- Konzept für eine flexibel konfigurierbare auf Web-Technologien aufbauende visuelle Konfigurationsumgebung für Analyse-Workflows
- Implementierung der konzipierten Analysekomponenten
- Implementierung der konzipierten Konfigurationsumgebung inklusive der visuellen Darstellung der Analysekomponenten
- Nachweis der Funktion der entwickelten Komponenten anhand eines Anwendungsbeispiels und der Analyse von Beispieldaten

1.3. Aufbau der Arbeit

In Kapitel 2 werden zunächst die für die vorliegende Arbeit notwendigen Grundlagen erarbeitet. Dazu gehören verschiedene Data-Mining-Verfahren und insbesondere das Sequential Pattern Mining. Danach werden die Analytics Workbench und weitere Workflow-basierte Analyse-Plattformen vorgestellt. Am Ende des Kapitels werden technologische Grundlagen, die für die Implementierung wichtig sind, vorgestellt. In Kapitel 3 wird zunächst eine Anforderungsanalyse durchgeführt. Dazu werden Probleme mit der aktuellen Version der Analytics Workbench im Hinblick auf Logfile-Analysen mittels Sequential Pattern Mining besprochen und auf dieser Grundlage Anforderungen definiert. Im Anschluss folgt ein umfangreicher Ansatz zur Lösung der festgestellten Probleme. In Kapitel 4 erfolgt die Dokumentation der Implementierung der neuen Version der Analytics Workbench. In einem Anwendungsbeispiel in Kapitel 5 wird schließlich die Funktionalität nachgewiesen. Ein abschließendes Fazit dieser Arbeit und zukünftige Entwicklungsmöglichkeiten werden in Kapitel 6 gegeben.

2. Grundlagen

In dem folgenden Kapitel sollen zunächst die für die vorliegende Arbeit notwendigen Grundlagen erarbeitet werden. Abschnitt 2.1 gibt zunächst eine Übersicht zum Thema Data-Mining & Wissensentdeckung. Dazu werden Data-Mining und insbesondere Sequential Pattern Mining Verfahren vorgestellt und eingeordnet. Am Ende des Abschnitts werden Logfiles und wichtige Datenformate wie Activity Streams behandelt. Abschnitt 2.2 stellt Plattformen zur Anwendung der Data-Mining Verfahren und deren Charakteristika vor. Dem gegenüber steht die Analytics Workbench, die zentral für die vorliegende Arbeit ist und in Abschnitt 2.3 behandelt wird. Schließlich werden in Abschnitt 2.4 technologische Grundlagen für die Implementierung webbasierter Anwendungen erarbeitet.

2.1. Data-Mining & Wissensentdeckung

Durch das World Wide Web und die Digitalisierung von Industrie, Wissenschaft, Technik, Medizin und vielen anderen Lebensbereichen, stehen riesige Mengen an Daten zur Verfügung, aus denen neues Wissen gewonnen werden kann. Die effektive Analyse dieser Daten stellt eine Herausforderung dar und ist seit mehreren Dekaden Gegenstand der Forschung [HPK11]. Data-Mining ist ein interdisziplinärer Bereich der Informatik und umfasst die Anwendung verschiedener statistischer Verfahren zur Wissensentdeckung in großen Datenbeständen. Dabei kommen Methoden aus verschiedenen wissenschaftlichen Feldern wie der Künstlichen Intelligenz, Maschinelles Lernen, Statistik und Datenbanksystemen zur Anwendung [CEF+06]. Der Begriff *Data-Mining* ist allerdings irreführend, da mit Data-Mining nicht die Extraktion (Mining) von Daten, sondern die Extraktion von Mustern und Wissen aus Datenbeständen, gemeint ist. Data-Mining wird in der Literatur als Teil der Wissensentdeckung in Datenbanken (*Knowledge Discovery in Databases*, KDD) gesehen, der sich auf die Anwendung von Analyse-Verfahren und Algorithmen beschränkt [FPSS96]. KDD bezeichnet im Gegensatz zum Data-Mining den Gesamtprozess zur Wissensentdeckung in Datenbeständen, wozu auch die Vorbereitungen der Analyse, die Transformation der Daten und die Evaluierung der Ergebnisse gehören. Fayyad et al. definieren KDD als den nicht-triviale Prozess zur Entdeckung gültiger, neuer, potenziell nützlicher und verständlicher Muster in Datenbeständen [FPSS96]. Gemäß Fayyad et al. lässt sich dieser Prozess in fünf Abschnitte aufteilen:

1. Selection

Der erste Schritt des KDD-Prozesses umfasst die Lokalisierung geeigneter Datenquellen und die anschließende Erhebung und Auswahl der zu analysierenden Datensätze.

2. Pre-processing

Die erhobenen Daten werden dann in einem Vorverarbeitungs-Schritt bereinigt.

2. Grundlagen

Dazu können nicht benötigte Datensätze herausgefiltert werden oder fehlende Werte in den Daten ergänzt werden. Auch dient dieser Schritt der Beseitigung von Ausreißern, Rauschen und Inkonsistenzen in den Daten.

3. Transformation

Für die Analyse der Daten müssen diese nun in ein für das Analyse-Verfahren geeignetes Format überführt werden.

4. Data Mining

Nun erfolgt der eigentliche Schritt der Analyse. Hier kommen Data-Mining Algorithmen zum Einsatz, die Muster in der Datenbasis finden sollen.

5. Evaluation

Der letzte Schritt umfasst die Interpretation und Bewertung der gewonnenen Erkenntnisse hinsichtlich ihrer Interessantheit und Bedeutung.

Der Prozess kann mehrere Iteration umfassen, bei denen bisher erzeugtes Wissen wieder in den Prozess integriert werden kann, um zusätzliche oder genauere Ergebnisse zu gewinnen.

Data-Mining-Verfahren zeichnen sich durch ihren explorativen Charakter aus. Im Gegensatz zu statistischen Verfahren sind die meisten Data-Mining-Verfahren nicht hypothesengetrieben, d.h es wird nicht versucht Theorien anhand von Daten zu verifizieren. Zentral ist die Aufdeckung neuer Muster und Beziehungen [KW00]. Je nach Problemstellung kommen verschiedene Verfahren und Algorithmen zum Einsatz. Data-Mining-Verfahren lassen sich grob in sechs Klassen aufteilen [HPK11]:

- **Ausreißererkennung**

Verfahren zur Ausreißererkennung werden verwendet um untypische und auffällige Datensätze zu identifizieren z.B. zur Erkennung von betrügerischen Kreditkartentransaktionen.

- **Assoziationsanalyse**

Assoziationsanalysen sind Methoden zur Entdeckung interessanter Zusammenhänge zwischen Variablen in großen Datenbeständen. Diese Verfahren kommen vor allem im Rahmen der sogenannten Warenkorbanalyse zum Einsatz, bei der Zusammenhänge zwischen gekauften Artikeln untersucht werden.

- **Clusteranalyse**

Das Ziel von Verfahren zur Clusteranalyse ist die Einteilung einer Menge von Objekten in Gruppen möglichst ähnlicher Objekte. Anwendung finden die Clusteranalysen vor allem in der Marktforschung.

- **Klassifikation**

Ziel von Klassifikation-Verfahren ist das Einordnen von Objekten in vorgegebene Klassen. Dazu gehört z.B. die Klassifikation von E-Mails auf Grundlage bestimmter Merkmale in Spam oder Nicht-Spam.

- **Regressionsanalyse**

Regressionsanalysen kommen vor allem in der Statistik zum Einsatz und dienen der Identifizierung von Beziehungen zwischen (mehreren) abhängigen und unabhängigen Variablen.

- **Zusammenfassung**

Verfahren zur automatischen Zusammenfassung reduzieren Text-Dokumente auf ihre wesentlichen Inhalte und werden vor allem von Suchmaschinen genutzt.

Wie die Auflistung oben impliziert, finden Data-Mining-Verfahren in vielen Bereichen Anwendung. Unternehmen, Banken und Versicherungen nutzen diese Verfahren, um potentielle Kunden auszumachen oder Marktentwicklungen vorherzusagen. In der Medizin können die Verfahren zur automatischen Diagnostik genutzt werden. Im Kontext des World Wide Webs wird die Anwendung von Data-Mining-Verfahren als Web-Mining bezeichnet. Dabei wird zwischen Web-Structure-Mining, Web-Content-Mining und Web-Usage-Mining unterschieden [Mob06]. Das Ziel von Web-Structure-Mining ist die Analyse von Beziehungen zwischen Webseiten, die über Links miteinander verbunden sind. Hier kommen vor allen Analyse-Verfahren aus dem Gebiet der Graphentheorie zum Einsatz. Web-Content-Mining befasst sich mit der Wissensgenerierung aus unstrukturierten Inhalten aus dem Web und findet oft im Gebiet des Information Retrieval Anwendung. Zentral für die vorliegende Arbeit ist das Web-Usage-Mining. Web-Usage-Mining bezeichnet die Anwendung von Data-Mining-Verfahren zur Entdeckung und Analyse von Nutzungsmustern aus Daten, die bei der Nutzung von Webseiten und Webanwendungen erzeugt wurden. Auf Grundlage dieser Erkenntnisse können Marketing-Strategien entwickelt und evaluiert, personalisierte Inhalte angeboten oder die Funktionalität einer Webseite bzw. Webanwendung verbessert werden [Mob06]. Web-Usage-Mining umfasst verschiedene Problemstellungen und Verfahren. Mittels Cluster-Analysen können Nutzer (aber auch Inhalte) in homogene Gruppen eingeteilt werden. Diese kommen häufig zur Nutzersegmentierung im E-Commerce-Bereich zur Anwendung, um personalisierte Inhalte oder Waren anzubieten [Mob06]. Assoziations- und Korrelationsanalysen werden genutzt, um Inhalte oder Waren zu identifizieren, die häufig zusammen besucht oder gekauft wurden. Für die Extraktion von Navigations- und Nutzungsmustern aus Logfiles kommen Sequential Pattern Mining Algorithmen zum Einsatz. Diese Form der Analyse ist zentral für die vorliegende Arbeit und wird im folgenden Abschnitt näher behandelt.

2.1.1. Sequential Pattern Mining

Sequential Pattern Mining bezeichnet Verfahren mit denen statistisch relevante Muster in großen Mengen an Sequenzen gefunden werden können [ME10]. Die Verfahren finden in einer Vielzahl von Bereichen Anwendungen, in denen diskrete zeitbezogene Daten erhoben werden. Abzugrenzen ist Sequential Pattern Mining von dem verwandten Frequent Itemset Mining. Frequent Itemset Mining Algorithmen arbeiten auf Grundlagen von Transaktions-Datenbanken. Eine Transaktion ist eine ungeordnete Liste von Elementen. Das Ziel hier ist Elemente zu finden, die häufig zusammen in Transaktionen vorkommen. Die zeitliche Dimension wird dabei nicht berücksichtigt. Sequential Pattern Mining Algorithmen hingegen arbeiten auf Grundlage von Sequenz-Datenbanken. Eine Sequenz ist eine geordnete Liste von Transaktionen. Ziel ist es, Sub-Sequenzen zu finden, die häufig in der gesamten Menge an Sequenzen vorkommen. Hier ist die zeitliche Reihenfolge der Transaktionen elementar für die Analyse. Ab wann eine Sub-Sequenz als häufig gilt, wird durch den Minimum Support definiert.

2. Grundlagen

SID	Sequence	PID	Pattern	Support
1	({a, b}, {c}, {f, g}, {g}, {e})	1	({a}, {f})	3
2	({a, d}, {c}, {b}, {a, b, e, f})	2	({a}, {c}, {f})	2
3	({a}, {b}, {f}, {e})	3	({b}, {f, g})	2
4	({b}, {f, g})	4	({g}, {e})	2
		5	({c}, {f})	2

Abbildung 2.1.: Sequenz-Datenbank (links) und dazugehörige häufige Muster (rechts)

Abbildung 2.1 zeigt eine Sequenzdatenbank und die dazugehören extrahierten häufigen Muster. Das Muster ($\{a\}$, $\{f\}$) kommt in den Sequenzen 1, 2 und 3 vor. Wie in der Sequenz-Datenbank zu erkennen, müssen Transaktionen nicht zwingend benachbart sein, sodass auch ($\{a\}$, $\{f\}$) eine Sub-Sequenz von ($\{a\}$, $\{b\}$, $\{f\}$, $\{e\}$) darstellt. Auch können einzelne Elemente einer Transaktion gesondert voneinander betrachtet werden, sodass ($\{a\}$, $\{f\}$) auch eine Sub-Sequenz von ($\{a, b\}$, $\{f\}$) darstellen würde. Wie Sub-Sequenzen definiert sind, ist also abhängig von der konkreten Problemstellung. Bei der Analyse von Logfiles werden diskrete Nutzeraktionen untersucht. Da eine Aktion immer nur einzeln auftreten kann, besteht eine Transaktion hier immer nur aus einem Element. Bei einer Analyse von Kaufverhalten hingegen, können innerhalb einer Transaktion (Kauf) mehrere Elemente (Artikel) vorkommen.

Die Implementierung von Algorithmen zur Extraktion häufiger Muster ist mit einigen Herausforderung verbunden. Ein solcher Algorithmus muss unter Einhaltung des Minimum Support die gesamte Menge häufiger Muster aus einer großen Sequenzdatenbank extrahieren können. Er muss effizient und skalierbar sein und ggf. bestimmte Bedingungen (constraints) unterstützen können. Mittlerweile existieren eine Vielzahl von Sequential Pattern Mining Algorithmen, die sich grob in zwei Kategorien einteilen lassen. Zu den Apriori-basierten Ansätzen zählen der GSP- (*Generalized Sequential Patterns*) und der SPAM-Algorithmus (*Sequential PAtern Mining using A Bitmap Representation*). Diese Algorithmen benutzen einen *Bottom-Up*-Ansatz, bei dem häufige Sub-Sequenz-Kandidaten sukzessive um Elemente erweitert (*Candidate Generation*) und Gruppen dieser Kandidaten gegen die Daten getestet werden. Die Aprori-Heuristik geht davon aus, dass wenn eine Sequenz S nicht häufig ist, alle Super-Sequenzen von S ebenfalls nicht häufig sind [ME10]. Die zweite Kategorie stellen die *Pattern-Growth*-basierten Algorithmen dar. Dazu gehören u.a. der *FreeSpan*- und der *PrefixSpan*-Algorithmus. Diese Algorithmen versuchen auf eine aufwendige *Candidate Generation* zu verzichten und durchsuchen die Datenbasis auf Grundlage eines Suchbaums nach häufigen Mustern. *Pattern-Growth*-basierte Algorithmen zählen zu den effizientesten Algorithmen [ME10].

Eine Erweiterung des SPAM-Algorithmus ist der von Fournier et al. konzipierte CM-SPAM Algorithmus, der mithilfe einer sogenannte CMAP (*Co-occurrence MAP*) die Menge möglicher Sub-Sequenz-Kandidaten reduzieren kann [FVGG⁺14]. Dieser Algorithmus scheint für die meisten Analysen der schnellste zu sein und soll in dieser Arbeit für die Analyse von Logfiles benutzt werden. Eine Implementierung dieses Algorithmus in Java ist in der von Fournier bereitgestellten *SPMF*-Bibliothek¹ zu finden. Die Implementierung erwartet als Input eine Sequenz-Datenbank in dem folgenden Format: Elemente

¹<http://www.philippe-fournier-viger.com/spmf/>, Abruf: 05.07.2016

```

1  1 2 -1 3 -1 4 5 -1 5 -1 6 -2
2  1 7 -1 3 -1 2 -1 1 2 6 4 -2
3  1 -1 2 -1 4 -1 6 -2
4  2 -1 4 5 -2

```

Abbildung 2.2.: Input-Format für den CM-SPAM-Algorithmus

werden als eindeutige, numerische Werte repräsentiert. Transaktionen werden mittels dem Wert **-1** voneinander getrennt, Sequenzen mittels dem Wert **-2**. Abbildung 2.2 zeigt die Sequenz-Datenbank aus Abbildung 2.1 in dem beschriebenen Format.

Zur Extraktion von Nutzungsmustern aus Logfiles müssen diskrete Aktionen bzw. Events aus den Daten extrahiert und für den Algorithmus aufbereitet werden. Wie das geschieht, wird in den Abschnitten 2.3.4.2 und 3.3.2 behandelt. Der folgende Abschnitt gibt zunächst eine Übersicht über das Thema Logfiles und dafür wichtige Formate.

2.1.2. Logfiles und Datenformate

Logfiles sind automatisch generierte und strukturierte Informationen, die beim Betrieb und der Nutzung von Software erzeugt werden können. Eine Software im laufenden Betrieb ist gewissermaßen eine Blackbox, deren inneren Vorgänge von außen nur schwer beobachtbar sind. Um dennoch an Informationen über den Zustand des Systems, sowie über die vom System ausgeführten Prozesse zu gelangen, bedient man sich sogenannter Logfiles. In Logfiles können Informationen von verschiedenen Ebenen einer Anwendungen gespeichert werden. Zur Fehlerbehebung und zum Profiling² können systemnahe Informationen, beispielsweise über den Speicherverbrauch oder den Zustand einzelner Prozesse, gesammelt werden. Auf höherer Ebene können nutzerspezifische Daten erhoben werden. Vor allem in großen, komplexen Anwendungen wie Betriebssystemen, Computer-Netzwerken und verteilten Systemen sind Logfiles fester Bestandteil der Software [Val01].

Durch die Popularität des Internets sind Server-Logfiles und die Analyse derer seit langem Bestandteil von Forschung und Entwicklung [PHMAZ00]. Server speichern standardmäßig Daten über alle eingehenden Verbindungen. Typischerweise werden so nutzerspezifische Daten wie IP oder Session³ in Verbindung mit der angefragten Webseite sowie einem Zeitstempel erhoben. Syntax und Format von Logfiles unterscheiden sich dabei erheblich. Für die Speicherung der Daten existieren zahlreiche, teils standardisierte Formate wie etwa das Extended Log File Format⁴ oder das Common Log File Format⁵. Die Implementierung zur Erhebung und Speicherung der Daten erfolgt meist parallel zur Anwendungslogik. Die Daten werden dabei typischerweise lokal in Dateien gespeichert. Durch die weite Verbreitung von Cloud-basierten Diensten, gibt es mittler-

²Analyse des Laufzeitverhalten einer Software

³Sitzung, bestehende Verbindung eines Clients mit einem Server

⁴Spezifikation: <https://www.w3.org/TR/WD-logfile.html>, Abruf: 20.06.2016

⁵Spezifikation: <https://httpd.apache.org/docs/trunk/logs.html>, Abruf: 20.06.2016

2. Grundlagen

```
1  {
2      "id": 4982,
3      "actor": {
4          "id": 221,
5          "display": "John Doe"
6          "objectType": "person"
7      },
8      "verb": "likes",
9      "object": {
10         "id": 234,
11         "display": "Profile picture"
12         "objectType": "picture"
13     }
14     "published": "2013-08-30T14:06:58Z",
15 }
```

Abbildung 2.3.: Beispiel Activity Stream Eintrag

weile aber auch SaaS⁶-Lösungen wie Loggly⁷ oder Papertrail⁸, denen die Daten online übermittelt werden können.

Bei komplexen webbasierten Anwendungen wie E-Learning-Plattformen oder sozialen Netzwerken sind diese Daten jedoch oftmals nicht ausreichend. Daher werden oft Daten auf höherer Ebene gesammelt, die detaillierte Informationen über den Nutzer und dessen Aktionen umfassen. Dabei kommen offene JSON-basierte Formate wie das JSON Activity Stream Format zum Einsatz. Ein Activity Stream ist eine Abfolge einzelner Aktivitäten eines Individuums, die typischerweise auf einer Webplattform stattfinden. Insbesondere in sozialen Netzwerken finden Activity Streams Anwendung. Durch die weite Verbreitung dieses Konzeptes wurde 2011 eine Spezifikation mit dem Titel *JSON Activity Streams* durch die Activity Streams Working Group vorgenommen [SAN12]. JSON Activity Streams bieten ein maschinenlesbares Vokabular, um Aktionen wie *Nutzer X gefällt Foto Y* oder *Nutzer X besucht Seite Y* abzubilden. Im Gegensatz zur herkömmlichen Logfiles, liegt der Fokus bei Activity Streams auf Aktivitäten der Nutzer und nicht des Systems. Abbildung 2.3 zeigt beispielhaft einen einzelnen Activity Stream Eintrag. Im Folgenden werden nun die Merkmale von Activity Streams erläutert (vgl. [NSW12]).

- Ein Activity Stream besteht aus einer Menge an Aktivitäten, die bei der Nutzung einer Web-Plattform aufgezeichnet werden.
- Eine einzelne Aktivität umfasst mindestens Informationen über den Nutzer (*actor*) und den Zeitpunkt der Aktivität (*published*).
- Die Activity Stream Working Group empfiehlt, dass eine Aktivität die Felder *verb*, *object* und *id* enthält.
- Das Feld *verb* beschreibt die Art der Aktion, also z.B. *gefällt*, *besucht* oder *empfiehlt*.

⁶Software as a service

⁷<https://www.loggly.com/>, Abruf 21.06.2016

⁸<https://papertrailapp.com>, Abruf 21.06.2016

- Das Feld *object* beschreibt den Gegenstand der Aktion. Das kann ein bestimmtes Foto oder eine Seite sein.
- Das *id* Feld identifiziert die Aktion eindeutig anhand von Internationalized Resource Identifiers (IRIs) ⁹.
- Darüber hinaus ist es abhängig von der Aktion (*verb*) möglich, ein Zielobjekt (*target*) zu definieren. So kann etwa die Aktivität *Nutzer X schickt Foto Y an Nutzer Z* abgebildet werden, wobei *Nutzer Z* das Zielobjekt ist.
- Die Werte der Felder *actor*, *object* und *target* sind sogenannte Activity Stream Objekte. Diese enthalten wiederum weitere Felder, die das jeweilige Objekt näher beschreiben. Empfohlen wird, dass die Objekte mindestens ein *id* Feld, zur eindeutigen Identifikation enthalten, sowie ein *display* Feld, das dem Objekt einen Namen gibt. Darüber hinaus gibt es weitere Felder innerhalb des Objektes wie etwa *objectType*, das das jeweilige Objekt einer Kategorie wie z.B. *audio*, *video*, *event*, *issue* oder *place* zuordnet.
- Neben den im Standard definierten Feldern ist es möglich, weitere beliebige Felder anzugeben. Im Kontext einer Nutzeranalyse lassen sich beispielsweise Nutzer in verschiedene Gruppen einteilen. Dazu wäre es möglich, ein Feld *group* zu definieren, das ebenfalls ein Activity Stream Objekt ist.

2.2. Workflow-basierte Analyse-Plattformen

Workflow-basierte Analyse-Plattformen zur Durchführung komplexer wissenschaftlicher Analysen haben in den letzten Jahren große Popularität gewonnen. Der Vorteil dieser Plattformen ist die Möglichkeit, wissenschaftliche Analyse-Prozesse und deren interne Abhängigkeiten visuell zu repräsentieren und somit eine nutzerorientierte Umgebung zu schaffen [CGG10]. Die visuelle Repräsentation abstrahiert dabei die zugrundeliegende Anwendungslogik und erleichtert das Entwickeln und Evaluieren wissenschaftlicher Arbeitsschritte. Die meisten Plattformen bieten dazu ein visuelles Frontend, das eine Graphen-Metapher nutzt, die eine interaktive Spezifikation von Analyse-Workflows ermöglicht. Ein Workflow wird dabei als Graph repräsentiert, dessen Knoten den einzelnen Verarbeitungsschritten entsprechen (auch Pipes-and-Filters Muster genannt). Per Drag-and-Drop lassen sich Knoten hinzufügen und in den Workflow integrieren. In vielen Fällen lässt sich die Repräsentation eines Workflows und die Datenflüsse zwischen den einzelnen Verarbeitungsschritten (lokal oder verteilt) direkt auf eine tiefere Programmebene übertragen, wodurch die technischen Details vor dem Nutzer verborgen bleiben. Workflow-basierte Analyse-Plattformen ermöglichen dabei die Integration unterschiedlicher Datenressourcen aus Datenbanken, Servern, Anwendungen und entfernten Diensten für eine vielfältige Anwendung in verschiedenen Feldern der Wissenschaft [TS07]. Die Art und Weise, wie Daten durch den Graphen geleitet und verarbeitet werden variiert allerdings zwischen den Systemen. So unterscheidet man zwischen *Control-Flow*- und *Data-Flow*-basierten Ansätzen. Bei *Control-Flow*-basierten Ansätzen definieren die Kanten eines Workflows strikt die Reihenfolge der Datenverarbeitung, sodass die Datenverarbeitung eines Ausgangs-Knoten erst abgeschlossen sein muss bevor die

⁹Spezifikation: <https://www.ietf.org/rfc/rfc3987.txt> Abruf: 31.05.2016

2. Grundlagen

des Ziel-Knotens beginnen kann. Bei *Data-Flow*-basierten Ansätzen hingegen, definieren die Kanten lediglich, wie die Datenverarbeitung durch die Knoten zusammengesetzt wird (Transformations-Komposition). Die Reihenfolge der Komposition kann dabei zur effizienteren Verarbeitung durch einen Optimizer verändert werden. (vgl. [CGG10]).

2.2.1. Visuelle Sprachen

Die Benutzeroberflächen Workflow-basierter Analyse-Systeme basieren meist auf visuellen Sprachen, die eine flexible Spezifikation von Analyse-Workflows erlauben. Visuelle Sprachen finden in verschiedenen Bereichen Anwendungen und zeichnen sich durch die Verwendung visueller Metaphern und die Möglichkeit der direkten Manipulation aus. Visuelle Sprachen werden vor allem in der Softwareentwicklung genutzt, um die Zusammenhänge zwischen Komponenten oder gar ganze Architekturen zu modellieren. Ein klassisches Beispiel hierfür ist die Unified Modeling Language (UML), die eine umfangreiche visuelle Syntax und Semantik bereitstellt. Mit UML ist es möglich, verschiedene Probleme in der Softwareentwicklung in Form von Diagrammen zu modellieren. Dazu zählen unter anderen Klassendiagramme, Entity-Relationship-Diagramme, Use Cases und Zustandsdiagramme [MRRR02]. Zahlreiche Modellierungsprogramme sind darüber hinaus in der Lage, visuelle Modelle in Quelltext zu überführen.

Visuelle Sprachen werden aber auch in kollaborativen Lehr- und Lernumgebungen (*collaborative learning environments*) zur Unterstützung in Gruppenarbeit oder Gruppendiskussionen genutzt [HGM⁺00]. Ein Beispiel hierfür sind Concept-Maps, mithilfe derer

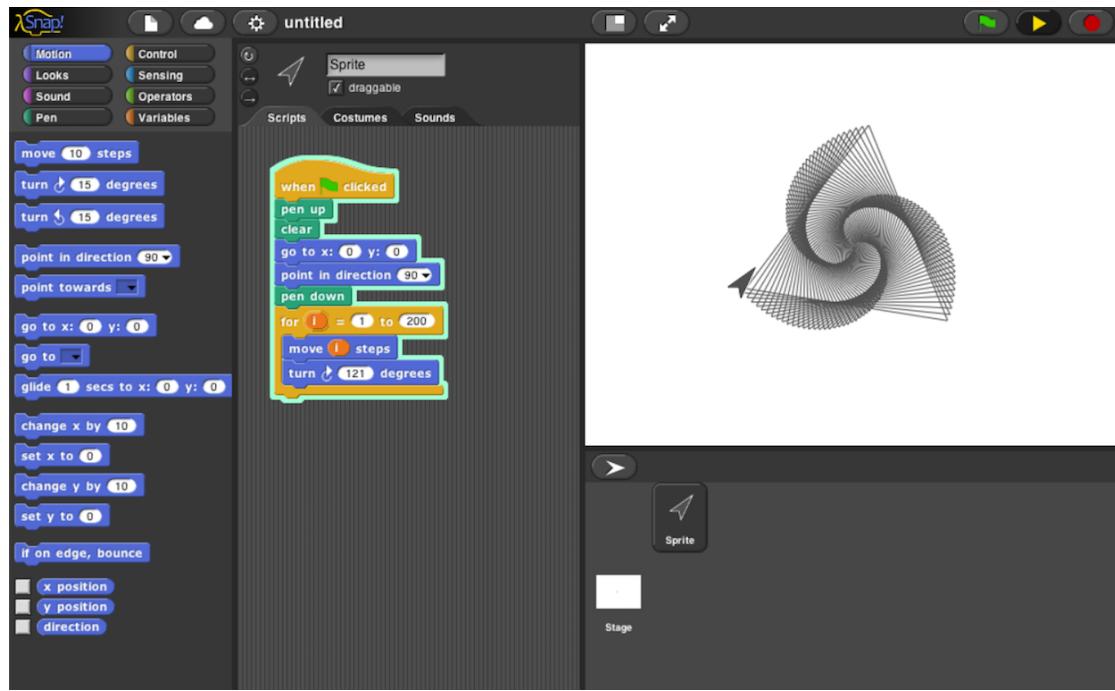


Abbildung 2.4.: Snap! (BYOB), visuelle Programmiersprache

Beziehungen zwischen Begriffen visualisiert und so Konzepte oder Sachverhalte verdeutlicht werden können. Concept-Maps verwenden dabei eine simple visuelle Sprache: Begriffe werden durch Rechtecke repräsentiert; die Beziehung zwischen den Begriffen durch Pfeile; die Art der Beziehung durch die Beschriftung der Pfeile. Die kollaborative Arbeit an Concept-Maps, und die dadurch intensive Beschäftigung mit einer Wissensdomäne, soll Schülern helfen erfolgreicher zu lernen. Mittlerweile gibt es eine Vielzahl an Konzepten, die durch kollaborative Lehr- und Lernumgebungen und mithilfe visueller Sprachen gelehrt bzw. gelernt werden können. Dazu zählen *System Dynamics* für die Simulation dynamischer Systeme, *Turtle-Grafik* zum Lernen von Programmiersprachen oder *Petri-Netze* zur Modellierung von Prozessen [HGM⁺00].

Visuelle Programmiersprachen sind eine weitere Anwendung visueller Sprachen und machen es möglich, Programme mit grafischen Elementen zu erstellen. Eine populäre Entwicklungsumgebung zur visuellen Programmierung im pädagogischen Kontext ist *Snap!* (vorher BYOB, Build your own blocks) [MH11]. Mit *Snap!* können Schüler Animationen, Spiele und Geschichten visuell programmieren. Dazu bietet die Umgebung verschiedene Programmbausteine in Form von Blöcken, deren Funktionalität und Kompatibilität durch Farben und Formen signalisiert werden (siehe Abbildung 2.4). Die visuelle Repräsentation der Programmbausteine soll einen einfachen Einstieg ermöglichen und Schülern helfen, mathematische Konzepte, Algorithmen und Datenstrukturen besser zu verstehen.

Zusammengefasst helfen visuelle Sprachen Komplexität zu reduzieren, da sie grobe Strukturen und Zusammenhänge besser veranschaulichen können als textuelle Sprachen. Workflow-basierte Analyse-Plattformen nutzen visuelle Sprachen, um die komplexe Anwendungslogik hinter einer verständlichen, anwendungsspezifischen Symbolik zu verborgen. Dies macht es einfacher Analysen zu spezifizieren und zu verstehen.

2.2.2. Verwandte Arbeiten

Mittlerweile existiert eine Vielzahl an Workflow-basierten Systemen und Rahmenwerken für verschiedene Anwendungen. In diesem Abschnitt werden zwei der verbreitetsten Systeme, RapidMiner und KNIME vorgestellt, die für die Analyse von Logfiles verwendet werden können. Diesen beiden Systemen steht die Analytics Workbench gegenüber, die im Anschluss in Abschnitt 2.3 vorgestellt wird.

2.2.2.1. RapidMiner

RapidMiner ist eine Software-Umgebung für maschinelles Lernen und Data-Mining. RapidMiner wurde zuvor unter dem Namen YALE (Yet Another Learning Environment) geführt und 2001 am Lehrstuhl für künstliche Intelligenz der Technischen Universität Dortmund entwickelt. RapidMiner wurde entwickelt, um Wissensentdeckungs-Prozesse (*Knowledge discovery processes*) besser durchführen zu können [MKFR03]. Die Software bedient sich einem Workflow ähnlichen Konzept, das verschachtelte Operatorketten (*nested operator chains*) oder Operatorbäume (*operator trees*) genannt wird. Operatoren sind Module, die Daten analysieren oder transformieren und diese an folgende Operatoren weitergeben. So entsteht eine Kette von Datenverarbeitungsschritten, die eine explorative Wissensentdeckung ermöglichen. Die Prozesse werden intern im XML-Format kodiert,

2. Grundlagen

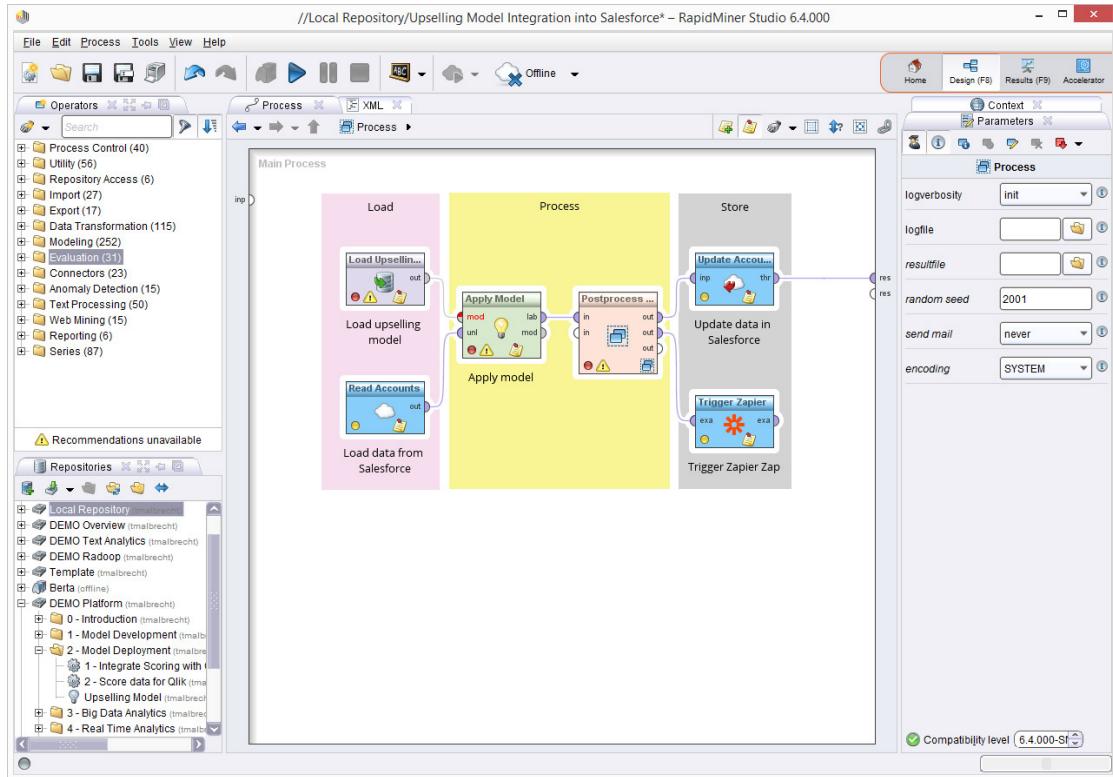


Abbildung 2.5.: RapidMiner Studio

sodass es möglich ist, diese auch ohne GUI¹⁰ zu bearbeiten. Abbildung 2.5 zeigt die Benutzeroberfläche von RapidMiner. Die Benutzeroberfläche erlaubt es, Analyse-Prozesse visuell zu spezifizieren und grafisch aufzubereiten. Operatoren können weitere Operatoren beinhalten. So ist es möglich, Sub-Prozesse innerhalb von Operatoren zu modellieren. Dazu erlaubt die Benutzeroberfläche, den Wechsel der Ansicht auf verschiedenen Ebenen des Prozesses. Aktuell verfügt RapidMiner über 500 Operatoren für verschiedene Aufgaben. So gibt es Operatoren für Ein- und Ausgabe, Datenvorverarbeitung, maschinelles Lernen, Data-Mining, Text Mining, Web Mining oder Zeitreihenanalysen. Darüber hinaus stehen verschiedene Verfahren zur Visualisierung von Daten und Modellen bereit. Die Funktionalität von RapidMiner lässt sich durch zusätzliche Plugins erweitern, die über den RapidMiner Marketplace¹¹ heruntergeladen werden können. RapidMiner Marketplace ist eine Plattform, die es Entwicklern ermöglicht, eigene Operatoren und Algorithmen anderen Nutzern verfügbar zu machen. So gibt es beispielsweise Plugins für die Integration von WEKA¹² oder R-Skripten¹³.

Zur Analyse von Logfiles mittels Sequential Pattern Mining existiert der *Generalized Sequential Patterns*¹⁴ (GPS) Operator. Der Operator scheint für die Analyse von Kun-

¹⁰Graphical User Interface

¹¹<https://marketplace.rapidminer.com>, Abruf: 17.06.2016

¹²Waikato Environment for Knowledge Analysis, <http://www.cs.waikato.ac.nz/ml/weka/>, Abruf: 17.06.2016

¹³R Language, <https://www.r-project.org/>, Abruf: 17.06.2016

¹⁴http://docs.rapidminer.com/studio/operators/modeling/associations/generalized_

2.2. Workflow-basierte Analyse-Plattformen

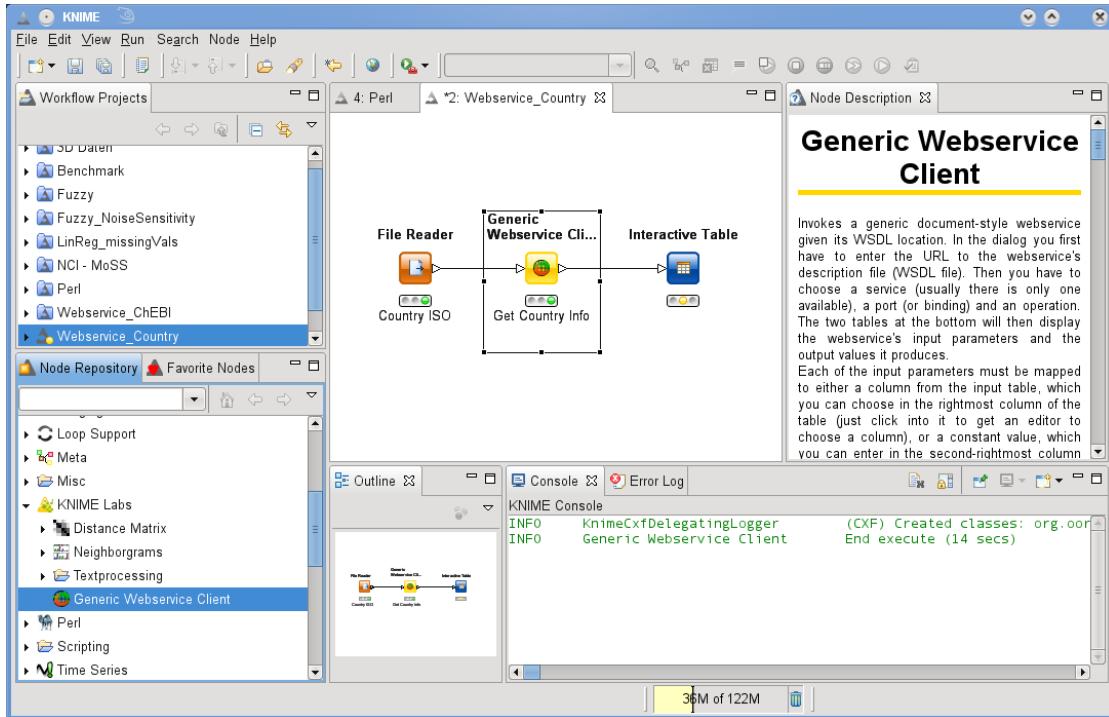


Abbildung 2.6.: Konstanz Information Miner

dendaten bezüglich Einkäufen entwickelt worden zu sein. Eingabe-Daten müssen daher die Attribute Zeitpunkt (*time*), Kunde (*customer*) und Artikel (*item*) aufweisen. Für die Analyse von Activity Streams müssen die Daten also zunächst in das genannte Format übersetzt werden. JSON-Daten lassen sich über den *JSON to data* Operator einlesen und in das interne *Data Table* Format übersetzen. Mithilfe des *Pivot* und *Rename* Operators lassen sich die Daten dann in das für den GPS Operator notwendige Format umwandeln. Die Werte bestimmter Felder der Activity Stream Einträge, z.B. *verb* und *object.objectType*, könnten so zusammengesetzt werden, um das Attribut *item* zu bilden. Das Ergebnis der Analyse sind nach Kunden (*customer*) gruppierte häufige Sequenzen an Artikeln (*item*).

Um die Verarbeitung schwach strukturierter Daten zu ermöglichen, analysiert Rapid-Miner die Eingabe-Daten auf vorhandene Felder und Werte. Diese Meta-Daten werden im gesamten Analyse-Prozess verfügbar gemacht und können zur Validierung der Daten innerhalb und zwischen Operatoren herangezogen werden.

2.2.2.2. Konstanz Information Miner

Der Konstanz Information Miner (KNIME) ist eine kostenlose Plattform zur interaktiven Datenanalyse und wurde 2004 am Lehrstuhl für Bioinformatik und Information Mining der Universität Konstanz entwickelt. Die Software kommt vor allem in der pharmazeutischen Forschung zum Einsatz [TS07]. KNIME folgt einem Pipes-and-Filters-

2. Grundlagen

Ansatz, bei dem Datenverarbeitungs-Module (*nodes*) miteinander verbunden werden und so einen Analyse-Workflow bilden. Aktuell existieren über 1000 Module zur Datentransformation, Data-Mining, Visualisierung oder Integration verschiedener Plattformen wie WEKA, Python, R oder SQL. Die Benutzeroberfläche erlaubt die visuelle Spezifikation der Analyse-Workflows (siehe Abbildung 2.6). Die Architektur von KNIME folgt drei Prinzipien (vgl. [BCD⁺08]):

1. *Visuell und interaktiv*: Datenflüsse können über zahlreiche Datenverarbeitungs-Module visuell per Drag-and-Drop kombiniert werden.
2. *Modular*: Datenverarbeitungs-Module und Daten-Container sind unabhängig voneinander, um eine verteilte Verarbeitung und unabhängige Entwicklung von Algorithmen zu ermöglichen.
3. *Erweiterbar*: KNIME basiert auf der Eclipse-Plattform¹⁵ und ermöglicht die einfache Integration neuer Datenverarbeitungs-Module und Ansichten über ein eigenes Plugin-System.

Um Logfiles mittels Sequential Pattern Mining analysieren zu können, muss auf die WEKA-Integration zurückgegriffen werden. Dazu existiert das *GeneralizedSequentialPatterns*¹⁶ Modul, das den Algorithmus von Srikant und Agrawal [SA96] implementiert. Als Eingabe-Daten wird eine Tabelle mit Sequenzen benötigt. Die Activity Stream Einträge müssen zuvor also in nach Nutzern gruppierte Sequenzen überführt werden. Dazu existieren verschiedene Module zum Einlesen und Transformieren der Daten. Allerdings kann dieser Prozess umständlich sein aufgrund der verschachtelten Struktur der Activity Stream Einträge. Auch muss bekannt sein, welche Felder in den Einträgen existieren, um sie in eine Tabelle überführen zu können.

2.3. Analytics Workbench

Grundlage dieser Arbeit ist die Analytics Workbench, die im Rahmen des SiSOB Projektes¹⁷ von der Collide Forschungsgruppe¹⁸ entwickelt wurde. Die Analytics Workbench ist eine webbasierte Datenanalyse-Plattform, die initial dafür entwickelt wurde, Forschern bei der Analyse von sozialen Netzwerken zu unterstützen [GHHH13]. Das System zeichnet sich dadurch aus, dass Analyse-Workflows visuell spezifiziert und ausgeführt werden können. Die Verarbeitung der Daten basiert auf dem Pipes-and-Filters Architekturmuster [Bus98]. So gibt es eine Vielzahl von Modulen zur Einspeisung, Transformation, Analyse und Visualisierung von Daten, die über sogenannten *Pipes* miteinander verbunden werden können und so einen Analyse-Workflow bilden. Jedes Modul bildet dabei eine Daten-Operation und gibt seine Daten an die jeweils folgenden Module weiter. Die eingespeisten Daten durchlaufen so alle miteinander verbundenen Module des Workflows. Die Ergebnisse eines Analyse-Workflows können abhängig von der Art der Ausgabe-Module, beispielsweise tabellarische Daten oder Visualisierungen sein.

¹⁵<http://www.eclipse.org/>, Abruf: 19.06.2016

¹⁶https://www.knime.org/files/nodedetails/weka_associations_GeneralizedSequentialPatterns.html, Abruf: 19.06.2016

¹⁷<http://sisob.lcc.uma.es/> Abruf: 31.05.2016

¹⁸<http://www.collide.info/> Abruf: 31.05.2016

2.3. Analytics Workbench

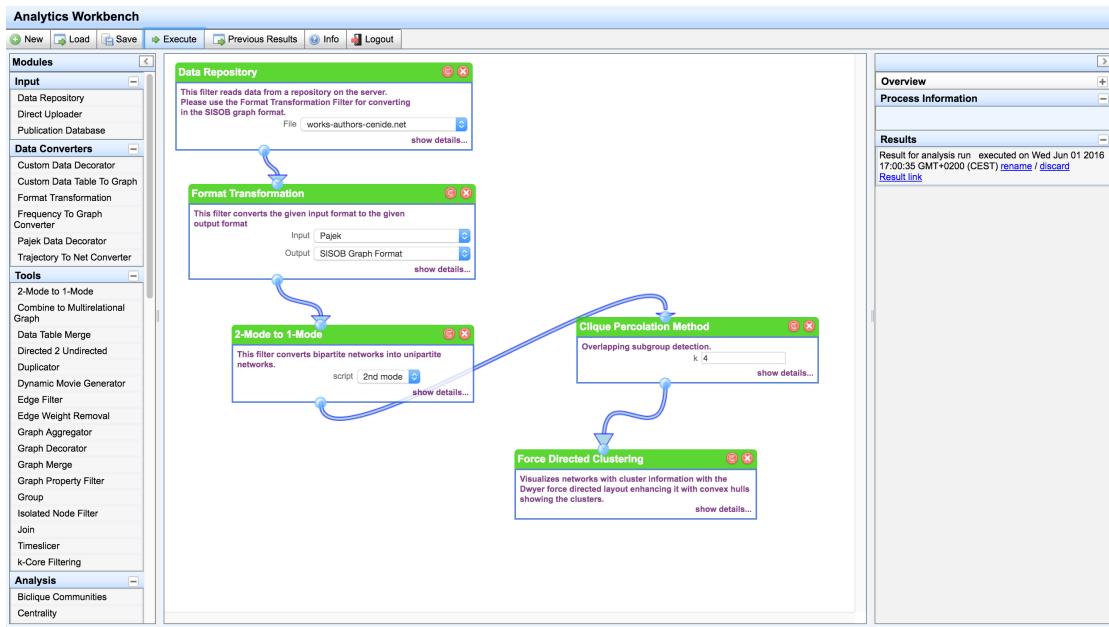


Abbildung 2.7.: Benutzeroberfläche der Analytics Workbench

2.3.1. Benutzeroberfläche

Abbildung 2.7 zeigt die Benutzeroberfläche der Workbench. Grundlage für die Benutzeroberfläche ist eine modifizierte Version der WireIt JavaScript-Bibliothek¹⁹. Auf der linken Seite befindet sich eine Liste verfügbarer Module, die in verschiedene Kategorien wie *Input*, *Analysis* oder *Visualisierung* aufgeteilt sind. Durch die offene Architektur lässt sich die Analytics Workbench leicht um weitere Module erweitern, sodass mittlerweile eine Vielzahl von Analysen-Techniken möglich sind. Die Module können per Drag-and-Drop in den Arbeitsbereich gezogen und miteinander verbunden werden. Dort lassen sich die jeweiligen Module über individuelle Benutzeroberflächen konfigurieren. Bei der Durchführung eines Workflows werden die Module je nach Zustand farblich hervorgehoben. Laufende Module werden gelb gefärbt, abgeschlossene Module grün. Sollten Fehler bei der Durchführung auftreten, wird das betroffene Modul rot gefärbt. Im rechten Bereich werden weitere Informationen zum Status der Analyse, sowie deren Ergebnisse bereitgestellt. Die Ergebnisse der Analysen sind meist einzelne Dateien, wie Excel-Tabellen aber auch HTML-Dokumente, die Visualisierungen enthalten und direkt im Browser geöffnet werden können. In der Menüleiste oben werden Funktionen zum Verwalten der Workflows bereitgestellt. Über diese können Workflows durchgeführt, erstellt, geladen, gespeichert und Ergebnisse bereits durchgeföhrter Analysen aufgerufen werden.

2. Grundlagen

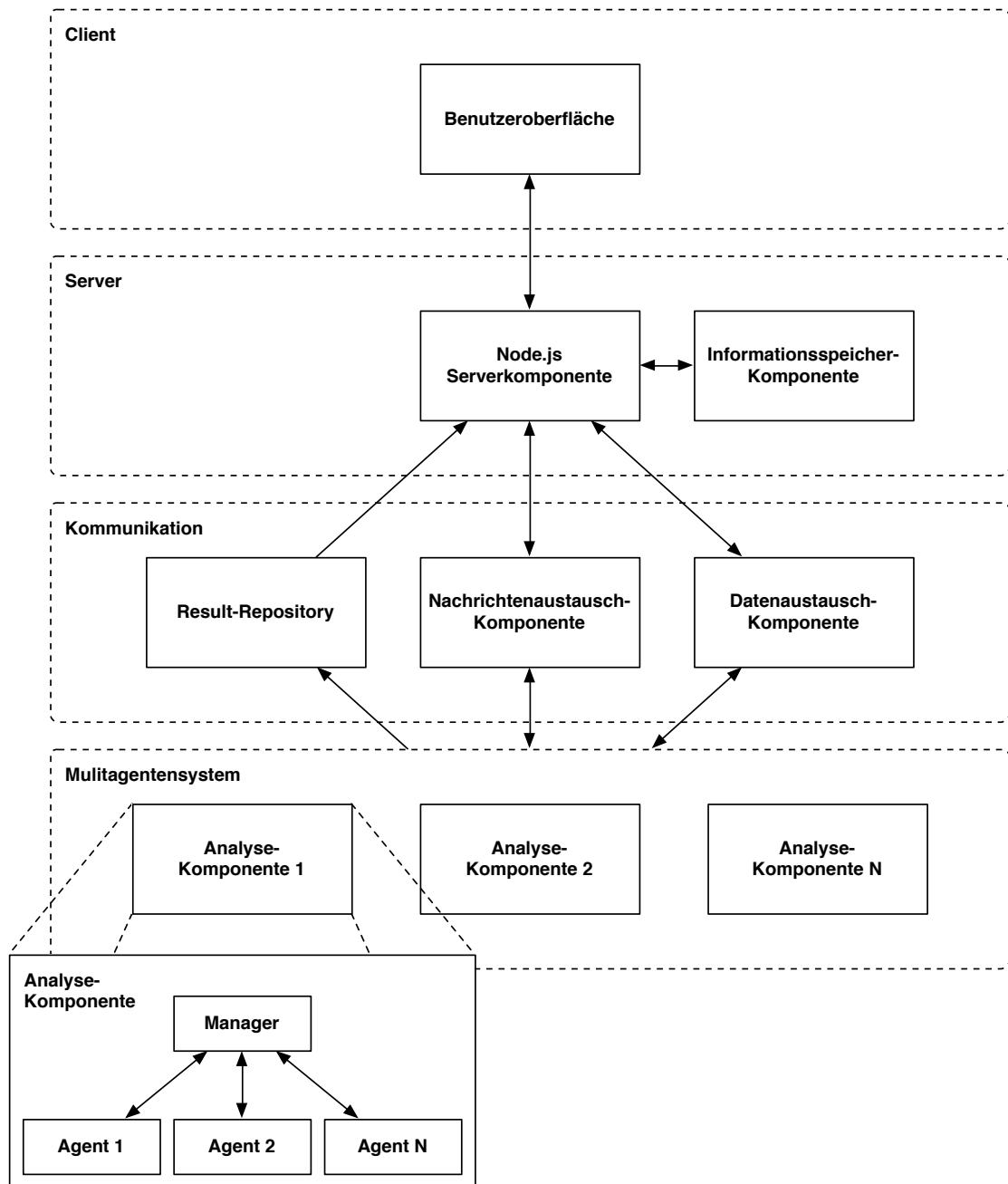


Abbildung 2.8.: Architektur der Analytics Workbench (vgl. [Rem15])

2.3.2. Architektur

Die Analytics Workbench lässt sich in vier Ebenen unterteilen: Client, Server, Kommunikation und Multiagentensystem. Auf den Ebenen Client und Server können Workflows spezifiziert, persistiert und die Durchführung initiiert werden. Die Durchführung selber wird durch mehrere Agenten übernommen, die über die Kommunikationsebene mit dem Server Daten austauschen können. Um eine möglichst einfache Integration weiterer Analyse-Module zu ermöglichen, wurde die Analytics Workbench als sprachheterogenes, verteiltes Multiagentensystem entwickelt. So ist es beispielsweise möglich, Agenten getrennt auf einem eigenen Server laufen zu lassen. Im Detail besteht die Architektur aus sieben Komponenten (siehe Abbildung 2.8):

- **Benutzeroberfläche**

Die in Abschnitt 2.3.1 beschriebene Benutzeroberfläche (im Folgenden auch Frontend genannt) ist die grafische Schnittstelle zwischen System und Nutzer. Sie ist über einen Browser aufrufbar und mit gängigen Web-Technologien wie HTML und JavaScript implementiert. Zur Kommunikation zwischen Benutzeroberfläche und Server wird das WebSocket-Protokoll²⁰ genutzt. Dies ermöglicht eine bidirektionale Kommunikation zwischen den beiden Ebenen.

- **Serverkomponente**

Die Serverkomponente ist ein in Node.js²¹ implementierter Server und ist dafür verantwortlich, die Benutzeroberfläche auszuliefern und die Kommunikation zwischen Benutzeroberfläche und Nachrichtenaustausch- bzw. Datenaustausch-Komponente zu ermöglichen. Zur Speicherung der in der Benutzeroberfläche benötigten Daten wie Workflows und Nutzer ist die Serverkomponente mit der Informationsspeicher-Komponente verbunden.

- **Informationsspeicher-Komponente**

Die Informationsspeicher-Komponente ist für die Langzeit-Speicherung von Workflows und Nutzern zuständig. Grundlage ist eine PostgreSQL-Datenbank²².

- **Datenaustausch- & Nachrichtenaustausch-Komponente**

Der Austausch von Informationen zwischen den einzelnen Analyse-Komponenten und der Serverkomponente ist aufgeteilt in Datenaustausch und Nachrichtenaustausch. Die Nachrichtenaustausch-Komponente ermöglicht den Austausch von Koordinations-Nachrichten. Die Datenaustausch-Komponente ist für den Austausch größerer Datenmengen verantwortlich.

- **Result-Repository**

Das Result-Repository speichert die von den Analyse-Komponenten erzeugten Ergebnisse der Analyse-Workflows und stellt sie der Server-Komponente bereit.

- **Analyse-Komponenten**

Analyse-Komponenten bestehen aus einem Agent-Manager und einem Agenten, von dem zur Laufzeit mehrere Instanzen existieren können. Der Agent-Manager

¹⁹<http://www.lshift.net/downloads/dev.lshift.net/james/wireit/wireit/index.html> Abruf: 14.06.2016

²⁰Spezifikation: <https://tools.ietf.org/html/rfc6455>, Abruf: 15.06.2016

²¹JavaScript-Laufzeit für serverseitige Anwendungen, <https://nodejs.org>, Abruf: 14.06.2016

²²<https://www.postgresql.org/>, Abruf: 15.06.2016

2. Grundlagen

stellt der Weboberfläche Informationen über die eigene Funktionalität bereit und koordiniert die Agenteninstanzen. Jedes Modul, das im Workflow vorhanden ist, wird auf Server-Ebene durch eine eigene, unabhängige Agenteninstanz vertreten. Die Instanzen nehmen die durch den Nutzer festgelegte Konfiguration, sowie die Eingabe-Daten entgegen und führen, entsprechend ihrer Funktionalität, eine Operation auf die Daten durch.

2.3.3. Analyse-Komponenten

```
1  {
2      "name": "Main Path Analysis",
3      "category": "Analysis",
4      "container": {
5          "descriptionText": "The threshold needs...",
6          "legend": "This filter performs a main path analysis.",
7          "inputs": [
8              {
9                  "name": "in_1",
10                 "label": "directed acyclic graph"
11             },
12             "outputs": [
13                 {
14                     "name": "out_1",
15                     "label": "main path as subgraph."
16                 },
17                 {
18                     "name": "out_2",
19                     "label": "decorated input."
20                 }
21             ],
22             "fields": [
23                 {
24                     "selectValues": ["SPC", "SPC ew-pre", "SPC ew-post"],
25                     "name": "value1",
26                     "label": "method",
27                     "required": true,
28                     "type": "select"
29                 }
30             }
31         }
32     }
```

Abbildung 2.9.: Filter Description

Die Analytics Workbench verfügt bereits über zahlreiche Analyse-Komponenten, vor allem zur Analyse sozialer Netzwerke und Graphen. Zur Entwicklung weiterer Analyse-Komponenten stellt die Analytics Workbench ein in Java implementiertes Agenten-Framework bereit. Dennoch ist es möglich, Agenten in anderen Programmiersprachen zu entwickeln. Diese müssen lediglich in die Kommunikationsinfrastruktur integriert werden [Rem15]. Eine Analyse-Komponente besteht, wie in Abschnitt 2.3.2 erwähnt, aus einem Agent-Manager und einem Agenten. Das Agenten-Framework stellt für beide Komponenten entsprechende Klassen bereit, die vom Entwickler genutzt werden können. Essential für die Integration neuer Analyse-Komponenten ist die sogenannte *Filter Description*, die vom Agent-Manager im JSON-Format bereitgestellt werden muss und dem Frontend Informationen über das Modul gibt. Abbildung 2.9 zeigt die *Filter Description* der *Main Path Analysis* Komponente. Neben den deskriptiven Feldern wie *name*, *category*,

descriptionText und *legend* enthält die Beschreibung die Definition der Ein- und Ausgabekanäle (*inputs* und *outputs*) sowie eine Beschreibung der Benutzeroberfläche zur Konfiguration der Analyse-Komponente (*field*). Bei der Durchführung eines Workflows erhält der Agent die Eingabe-Daten über seine definierten Eingabe-Kanäle. Zusätzlich wird die über die Benutzeroberfläche getätigten Einstellungen dem Agenten übermittelt. Der Agent verarbeitet die Eingabe-Daten und stellt seine Ergebnisse über die definierten Ausgabe-Kanäle bereit.

2.3.4. Logile-Analysen

Die Analyse von Logfiles mittels Sequential Pattern Mining ist bereits möglich in der aktuellen Version der Analytics Workbench. Dazu stehen zwei Analyse-Komponenten zur Verfügung: Der Activity Stream Filter und das Sequential Pattern Mining Modul. Mit diesen ist es möglich, Activity Streams (siehe Abschnitt 2.1.2) zu filtern und auf häufige Muster zu untersuchen (siehe Abschnitt 2.1.1).

2.3.4.1. Activity Stream Filter

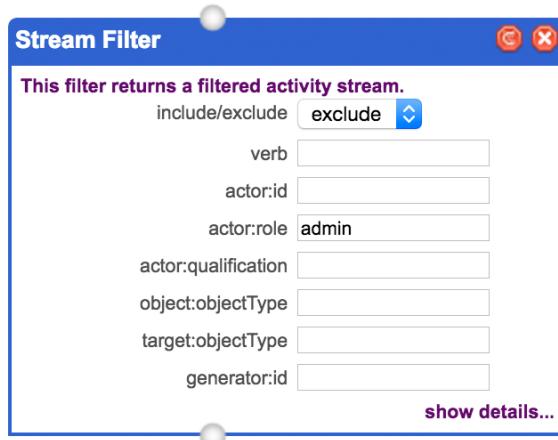


Abbildung 2.10.: Activity Stream Filter Modul

Der Activity Stream Filter kann eingesetzt werden, um Activity Streams anhand bestimmter Feld-Werte-Kombinationen zu filtern. Abbildung 2.10 zeigt die Benutzeroberfläche des Moduls. Im oberen Bereich kann eingestellt werden, ob Einträge, die auf die Feld-Werte-Kombinationen zutreffen, ein- oder ausgeschlossen werden. Die Feld-Werte-Kombinationen lassen sich im unteren Bereich einstellen. Das Modul gibt eine feste Menge an Feldern vor, zu denen Werte eingetragen werden können. Die in der Abbildung zu sehende Konfiguration schließt alle Einträge aus, deren Feld *actor.role* den Wert *admin* hat. Der Filter gibt schließlich eine Teilmenge der eingegangenen Activity Stream Einträgen zurück.

2. Grundlagen

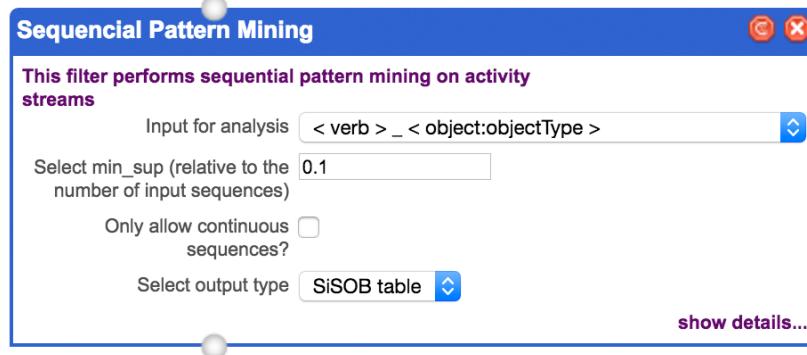


Abbildung 2.11.: Sequential Pattern Mining Modul

2.3.4.2. Sequential Pattern Mining

Das Sequential Pattern Mining Modul ermöglicht die Analyse von Activity Streams auf häufige Muster. Durchgeführt wird die Analyse von dem CM-SPAM-Algorithmus der SPMF-Bibliothek (siehe Abschnitt 2.1.1). Damit Activity Streams von dem Algorithmus analysiert werden können, müssen diese zuvor in ein für den Algorithmus geeignetes Format übersetzt werden. Dies geschieht indem die Werte bestimmter Felder der einzelnen Einträge in eindeutige Bezeichner umgewandelt werden. Aus dem Activity Stream Eintrag aus Abbildung 2.3 könnten beispielsweise die Felder *verb* und *object.objectType* herangezogen werden. Aus den Werten dieser Felder würde der Bezeichner *likes_picture* entstehen. Einträge, die in diesen Feldern dieselben Werte aufweisen, würden denselben Bezeichner erzeugen. Auf diese Weise werden Einträge auf bestimmte Merkmale reduziert und ergeben so einen eindeutigen Bezeichner. Für diese Übersetzung bietet das Sequential Pattern Mining Modul zwei festgelegte Übersetzungsformen: *verb_object.objectType* und *verb_object.objectType_object.name*. Darüber hinaus lassen sich weitere Parameter wie der *Minimum Support*, der angibt, ab wann eine Sequenz als häufig gilt, oder der Ausgabetyp einstellen. Das Modul liefert schließlich tabellarisch eine Menge an häufigen Sequenzen zurück.

2.4. Technologische Grundlagen

Die Analytics Workbench ist ein Webanwendung. Webanwendungen basieren auf einem Client-Server-Modell und bestehen aus einer Vielzahl an Komponenten und Technologien (siehe auch Abschnitt 2.3.2). Grob lassen sich diese in Frontend und Backend einteilen. Das Frontend, also die Benutzerschnittstelle, wird bei webbasierten Anwendungen auf Grundlage von HTML, JavaScript und CSS²³ realisiert. Die Interpretation und Laufzeit findet im Browser des Nutzers statt. Die Auslieferung dieser Ressourcen wird von einem Webserver (dem Backend) übernommen. Serverseitig können verschiedene Technologien und Sprachen wie Java, PHP oder Node.js, das Grundlage für die Analytics Workbench ist, zum Einsatz kommen. Im Folgenden sollen die für die Realisierung der adaptiv konfigurierbaren Web-Umgebung kritischen Technologien vorgestellt werden. Dazu gehören

²³Cascading Style Sheets

Technologien zur Umsetzung von *Single-page Applications* sowie ein Konzept zur Realisierung adaptiver Benutzeroberflächen mit einem *Self-adaptive Software* Ansatz.

2.4.1. Single-page Application

Im Gegensatz zu herkömmlichen Desktop-Anwendungen erfordern Webanwendungen kein bestimmtes Betriebssystem, sondern können plattformunabhängig auf beliebigen Endgeräten über einen Browser aufgerufen und ausgeführt werden. Eine Webanwendung wird gestartet, indem der Nutzer mit einem Browser die URL eines Webservers aufruft. Der Webserver nimmt die Anfrage entgegen und liefert die für die Anwendung benötigten Ressourcen, das HTML-Dokument, CSS-, JavaScript- und Bilddateien aus. Klassische Webanwendungen bestehen aus statischen, untereinander verlinkten HTML-Dokumenten. Beim Wechsel in eine andere Ansicht muss die entsprechende Seite, sowie alle damit verbundenen Ressourcen, erneut vom Server geladen und vom Browser gerendert werden. Mittlerweile existieren aber zahlreiche Technologien, mit denen es möglich ist, Inhalte im Hintergrund nachzuladen und die Benutzeroberfläche dynamisch zu ändern, ohne die gesamte Seite neu zu laden. Webanwendungen, die diesen Ansatz verfolgen, werden *Single-page Applications* (SPA) genannt [Jos15]. SPAs bieten einige Vorteile gegenüber klassischen Webanwendungen. Sobald die Anwendung einmalig geladen wurden, stehen dem Browser alle Ressourcen wie Templates und Bilder bereit. Änderungen in der Benutzeroberfläche können dynamisch und unmittelbar von der Anwendung durchgeführt werden. Das Ergebnis ist eine wesentlich schnellere Reaktionszeit der Anwendung. Auch das Laden von Daten läuft deutlich schneller ab, da die Anwendung mit *rohen* Daten (oft im JSON-Format) arbeiten kann, die im Hintergrund über verschiedene Webservices und Protokolle wie REST²⁴ oder WebSockets²⁵ geladen werden. Das reduziert umfangreiche Anfragen und serverseitiges Rendern von HTML-Dokumenten, wodurch die Serverlast verringert wird. Für die Implementierung von SPAs ist es notwendig, die Anwendungslogik vom Server auf den Client, also dem Browser, zu verlagern. Dazu wird die Sprache JavaScript eingesetzt. JavaScript ist eine dynamisch typisierte, objektorientierte Skriptsprache, die erstmals 1995 im *Netscape Navigator*²⁶ verwendet wurde und mittlerweile fester Bestandteil in der Webentwicklung ist [Kes11]. Die Benutzeroberfläche im Browser wird durch HTML beschrieben, das wiederum in das sogenannte DOM²⁷ überführt wird und Grundlage für das Rendering des Browsers ist. Mittels JavaScript können Templates verarbeitet und Inhalte des DOM verändert werden. Für das Nachladen von Daten können verschiedene vom Browser bereitgestellte Schnittstellen verwendet werden. Zur Kommunikation mit einer REST-Schnittstelle wird der sogenannte AJAX-Ansatz²⁸ genutzt, bei dem über die *XMLHttpRequest*-Schnittstelle des Browsers HTTP-Anfragen an den Server gestellt werden können. Da das HTTP-Protokoll nur eine zustandslose, einseitige, vom Browser initiierte Kommunikation erlaubt, werden bei SPAs oft WebSockets genutzt. WebSockets erlauben eine bidirektionale Kommunikation; so ist es möglich, dass der Server dem Browser (ungefragt) Informationen zukommen lässt. Dieses Protokoll wird auch zum Datenaustausch in der Analytics

²⁴Representational State Transfer

²⁵Auf TCP basierendes Netzwerkprotokoll für bidirektionale Kommunikation

²⁶Webbrowser von Netscape Communications

²⁷Document Object Model, Schnittstelle für den Zugriff auf HTML

²⁸Asynchronous JavaScript and XML

2. Grundlagen

Workbench genutzt. Für die Implementierung komplexer Anwendungen und Benutzeroberflächen existieren mittlerweile zahlreiche JavaScript-Frameworks, wie Backbone²⁹ oder Angular³⁰, die unterschiedliche Architektur- bzw. Entwurfsmuster erlauben. Die meisten Frameworks nutzen eine Variante des MVC-Musters³¹. Das Model (M) wird zur Datenhaltung und Datenaustausch zwischen Server und Browser genutzt; die View (V) ist für die Präsentationslogik verantwortlich und erzeugt seine Benutzeroberfläche auf Grundlage eines Templates und Daten aus einem Model; Model und View kommunizieren meist über ein Publish–Subscribe-Muster; der Controller (C) koordiniert Benutzereingaben und Datenfluss. Benutzeroberflächen, die sich häufig ändern, stellen bei der Entwicklung von SPAs eine Herausforderung dar. Manipulationen des DOMs sind langsam, da bei Änderungen möglicherweise neue Knoten und Teilbäume erstellt und die Rendering-Pipeline bzw. Teile dieser erneut durchlaufen werden müssen. Das Rendering einer View-Komponente geschieht meistens in einer *render*-Funktion, die aus einem textbasiertem HTML-Template tatsächliche DOM-Elemente erzeugt und zurückgibt. Wie View-Komponenten mit Änderungen umgehen ist abhängig von der Implementierung des Frameworks. Oft wird das gesamte Template neu gerendert und die alten DOM-Elemente mit neuen ersetzt. Dies ist bei schmalen Benutzeroberflächen unproblematisch. Umfasst die Benutzeroberfläche aber mehrere hundert DOM-Elemente, so kommt es schnell zu Performance-Problemen. Andere Ansätze implementieren eine programmatiche Vorgehensweise, bei der DOM-Elemente auf Grundlage der Änderungen direkt manipuliert werden. Bei komplexen Anwendungen, kann dies aber schnell unübersichtlich werden. Aktuell gehen JavaScript-Frameworks daher neuen Ansätzen nach. Allen voran ist da der Virtual-DOM-Ansatz zu nennen, der erstmals im React-Framework implementiert wurde. React wird im nächsten Abschnitt vorgestellt.

2.4.2. React-Framework

React³² ist ein JavaScript-Framework, das von Facebook³³ entwickelt und 2013 veröffentlicht wurde. Im Gegensatz zu vielen anderen Frameworks, stellt React lediglich eine View-Engine bereit, mit der Benutzeroberflächen erzeugt werden können. Infrastruktur und Datenmodelle werden von React nicht bereitgestellt. React wurde aus folgender Idee entwickelt:

We built React to solve one problem: building large applications with data that changes over time. Simply express how your app should look at any given point in time, and React will automatically manage all UI updates when your underlying data changes [why13].

Views werden in React Komponenten (*Components*) genannt, die minimal³⁴ nur aus einer Funktion bestehen und beim Ausführen (dem Rendering) einen virtuellen DOM-Baum zurückgeben (siehe Abbildung 2.12). Der virtuelle DOM-Baum ist eine leichtgewichtige aus JavaScript-Objekten bestehende Repräsentation des eigentlich zu erzeugen-

²⁹<http://backbonejs.org/>, Abruf: 30.06.2016

³⁰<https://angularjs.org/>, Abruf: 30.06.2016

³¹Model-View-Controller

³²<https://facebook.github.io/react/>, Abruf: 30.06.2016

³³<https://facebook.com/>, Abruf: 30.06.2016

³⁴React Komponenten können aber auch mit einer Factory-Funktion (`createClass`) erzeugt werden, die die Verwendung weiterer Funktionen wie *Local State* oder *Life-Cycle-Hooks* erlaubt.

```

1  function ListItem(props) {
2    return <li>{props.item.title}</li>;
3  }
4
5  function List(props) {
6    return (
7      <ul>
8        {props.items.map(item => <ListItem item={item}/>)}
9      </ul>
10     );
11   }
12
13 var items = [{ title: 'Item one' }, {title: 'Item two'}];
14 ReactDOM.render(<List items={items}/>, document.body);

```

Abbildung 2.12.: React-Komponenten

den DOMs. Was wie herkömmliches HTML aussieht (Zeile 2), ist sogenanntes *JSX*³⁵, das von einem Transpiler wie *Babel*³⁶ in JavaScript umgewandelt wird. Ein HTML-Tag `` wird so in den Ausdruck `React.createElement('li')` überführt, wodurch ein virtueller DOM-Knoten erzeugt wird. Wird eine Komponente zum ersten Mal gerendert, so erzeugt React aus dem virtuellen DOM die entsprechende (tatsächliche) DOM-Struktur und fügt sie in das HTML-Dokument ein (siehe Zeile 14). Beim erneutem Rendern vergleicht React den bisherigen virtuellen DOM-Baum mit dem neu erzeugten und errechnet die minimal notwendigen Änderungen. Die Änderungen werden schließlich am DOM umgesetzt und erzeugen eine aktualisierte Benutzeroberfläche [Gac15]. Wenn sich beispielsweise nur ein Textfragment geändert hat, muss nicht der gesamte Teil-Baum des DOM, sondern nur ein einzelner Knoten verändert werden. Eine React-Komponente kann auf Grundlage seiner Daten (*props*) zu jeder Zeit beschreiben, wie seine Benutzeroberflächen aussieht. Programmatische Änderungen am DOM sind nicht notwendig. Benutzeroberflächen lassen sich in React funktional beschreiben:

$$UI = Component(State) \quad (2.1)$$

Dies ist auch im Hinblick auf Änderungen der Benutzeroberflächen der Analyse-Modul der Workbench interessant. Analyse-Module beschreiben ihre Oberflächen in einem JSON-Format (siehe Abschnitt 2.3.3). Dazu könnte eine React-Komponente implementiert werden, die aus dieser Beschreibung eine Oberfläche mit entsprechenden Eingabefeldern erzeugen kann. Soll sich eine Oberfläche dynamisch ändern, so müsste der Komponente lediglich eine aktualisierte Oberflächenbeschreibung übergeben werden. Alle damit einhergehenden Änderungen würden dann von React umgesetzt. Ein weiterer Vorteil von React ist die Möglichkeit isolierte wiederverwendbare Komponenten zu erstellen und diese mit *JSX* beliebig zu verschachteln (*Componentization & Composition*). Zur Verschachtlung von Komponenten, können diese ebenfalls wie Tags verwendet werden (siehe Zeile 8). So kann die `List`-Komponente `ListItem`-Komponenten rendern.

³⁵Erweiterung der JavaScript-Syntax

³⁶babeljs.io, Abruf: 30.06.2016

2. Grundlagen

Die Daten für die Komponenten (*props*) werden dabei von oben nach unten durch die Hierarchie durchgeleitet.

Die Stärke von React ist der Fokus auf den View-Layer einer Webanwendung. Infrastruktur und Daten-Modelle werden nicht von React bereitgestellt. Dazu wird in React-Anwendungen häufig die Bibliothek Redux³⁷ genutzt. Redux wird von den Entwicklern als *predictable state container for JavaScript apps*³⁷ beschrieben. Eine Redux-Architektur besteht aus drei Komponenten (siehe Abbildung 2.13): Einem *Store*, der alle in der Anwendung verwendeten Daten (*State*) hält und diese den React-Komponenten bereitstellt (Zeile 26), ein oder mehreren *Reducer*, die beschreiben, wie sich Daten ändern (Zeile 2) und schließlich *Actions*, die Aktionen beschreiben (Zeile 18). Durch den Nutzer getätigte Aktionen (z.B. der Klick auf einen Button einer React-Komponente) oder von außen eingehende Aktionen (z.B. ein Event, das über eine WebSocket-Schnittstelle ausgelöst wurde) können an den *Store* gesendet werden. Dieser Vorgang wird *dispatching* genannt (siehe Zeile 30). Eine *Action* ist eine simple Funktion, die eine Beschreibung erzeugt, um welche Aktion es sich handelt. Der *Store* übergibt diese Beschreibung den *Reducers*, die auf Grundlage der Beschreibung den *State* ändern und zurückliefern. Der neue *State* wird im *Store* gehalten und die React-Komponenten werden informiert, dass sich Daten geändert haben, worauf diese sich erneut rendern. So entsteht ein sogenannter *unidirectional data flow* (Abbildung 2.14 soll dieses Konzept verdeutlichen). Redux verzichtet bewusst auf eine MVC-Architektur. In komplexen Anwendungen, in denen sich Daten ständig ändern, kann es schwierig sein, mit dem MVC-Modell eine übersichtliche Architektur zu implementieren. Anstelle mehrerer *Models*, die von verschiedenen *Views* referenziert werden, setzt Redux ein globales *Store*-Objekt ein, das gesamten Zustand (*State*) der Anwendung hält. Dies vermeidet kaskadierende Änderungen von einander abhängigen Models und soll den Entwickler den Daten-Fluss in der Anwendung besser durchschauen lassen.

2.4.3. Self-adaptive Software

Unter *self-adaptive software* werden Systeme verstanden, die in der Lage sind, sich aufgrund von Änderungen in ihrer Umgebung anzupassen, um dem Benutzer bestmöglich zu unterstützen [OGT⁺99]. Man unterscheidet dabei in verschiedenen Arten der Adaption. Eine dieser Arten nennt sich *dynamische Adaption*, bei der sich ein System aufgrund von Änderungen des Kontextes oder des Ressourcenzustandes zur Laufzeit anpasst [Gei08]. Ein Beispiel dafür sind Laptops, die ihre Bildschirmhelligkeit entsprechend der Lichtverhältnisse anpassen. Die Adoptionsmöglichkeiten eines Systems werden dabei vom Entwickler zur Entwurfszeit festgelegt. Voraussetzung für die Realisierung solcher Anwendungen sind drei Grundfunktionen (vgl. [Gei08]):

1. Kontextmanagement

Das Kontextmanagement ist dafür verantwortlich, Änderungen in der Umgebung zu erkennen und das Adoptionsmanagement zu informieren.

2. Adoptionsmanagement

Das Adoptionsmanagement evaluiert die Änderungen nach bestimmten Kriterien und wählt eine geeignete Adaption aus.

³⁷<http://redux.js.org/>, Abruf 06.07.2016

```

1 // Reducer
2 function reducer(state, action) {
3
4     switch (action.type) {
5         case 'ADD_ITEM': {
6             return {
7                 items: state.items.concat(action.payload)
8             };
9         }
10        default: {
11            return state;
12        }
13    }
14}
15
16
17 // Single action
18 function add(item) {
19     return {
20         type: 'ADD_ITEM',
21         payload: item
22     }
23 }
24
25 // Store
26 var store = createStore(reducer);
27
28 // Dispatching
29 var item = { title: 'Item three' };
30 store.dispatch(add(item));

```

Abbildung 2.13.: Redux-Beispiel

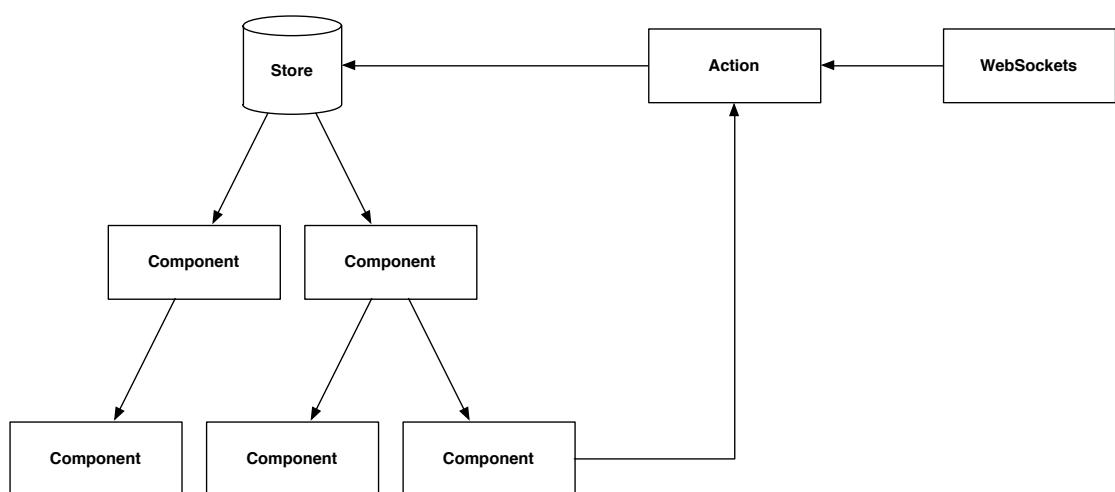


Abbildung 2.14.: Architektur mit React und Redux

2. Grundlagen

3. Konfigurationsmanagement

Das Konfigurationsmanagement setzt die Änderungen schließlich um.

Dieses Konzept lässt sich auch auf Benutzeroberflächen von Webanwendungen übertragen, die sich auf Grundlage geänderter Umgebungsdaten anpassen sollen. Ein Beispiel dafür könnte ein komplexes Formular sein, das beim ersten Aufruf Hilfestellungen bietet. Das System weiß, dass der Nutzer dieses Formular noch nicht kennt und daher eventuell Hilfe braucht beim Ausfüllen. Um den Benutzer zu unterstützen, werden daher im Formular Hinweise und Anleitungen untergebracht. Sobald der Nutzer mit dem Formular vertraut ist, würden die Hinweise nur noch auf Abruf sichtbar werden. Sobald ein Nutzer das Formular mehrmals erfolgreich benutzt hat und nun vertraut damit ist, könnte dies im System vermerkt werden. Das Kontextmanagement wäre nun dafür verantwortlich, diese Änderungen der Umgebungsdaten zu erkennen und die Oberflächenanpassung zu initiieren. Wie sich eine Oberfläche anpasst, wird dabei zur Entwurfszeit in einer entsprechenden View-Komponente festgelegt. Hat die View-Komponente Informationen darüber, dass der Nutzer das Formular kennt, so würden die Hinweise der Übersichtlichkeit halber nicht mehr angezeigt. Im Hinblick auf eine Implementierung mit React (siehe Abschnitt 2.4.2), würde die View-Komponente beim erneuteten Rendering eine geänderte Oberfläche (in Form eines virtuellen DOMs) zurückgeben. Das Konfigurationsmanagement wäre nun dafür verantwortlich, die Änderungen umzusetzen. Diese Aufgabe liegt bei der View-Engine, in diesem Fall also React. Abbildung 2.4.3 soll dieses Konzept verdeutlichen. Im Hinblick auf die Adoptionsfähigkeit der Analytics Workbench ist denkbar, dass sich die Oberflächen zur Konfiguration der Analyse-Module an die Struktur der Eingabe-Daten anpassen. So wäre es für den Nutzer einfacher, valide Konfigurationen vorzunehmen oder einen besseren Einblick in die Daten zu erlangen.

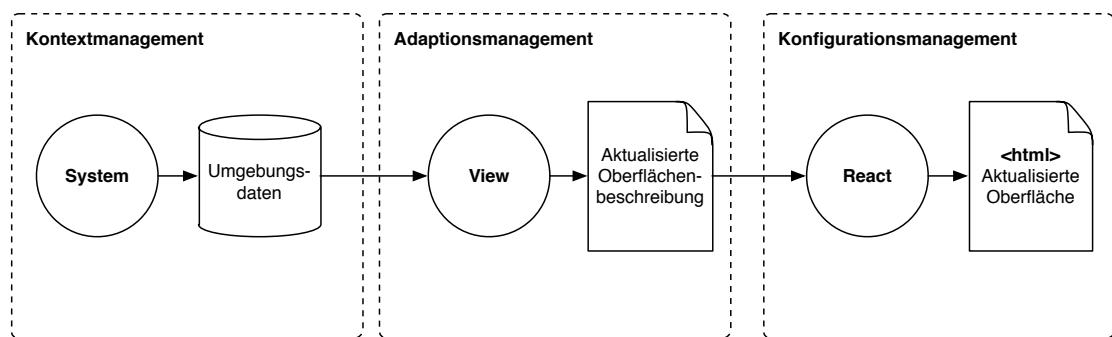


Abbildung 2.15.: Konzept für adaptive Benutzeroberflächen

3. Ansatz

In dem folgenden Kapitel soll zunächst die Analytics Workbench auf ihre Möglichkeiten zur Analyse von Logfiles hin untersucht werden. Dazu werden in Kapitel 3.1.1 die Analyse-Module hinsichtlich ihrer Fähigkeit, den KDD-Prozess (siehe Abschnitt 2.1) abbilden zu können, untersucht. Danach erfolgt eine Auseinandersetzung mit der Problematik schwach strukturierter Daten. Auf Grundlage der festgestellten Probleme werden in Abschnitt 3.1.3 die Anforderungen an die Analytics Workbench festgehalten. In den Abschnitten 3.2, 3.1 und 3.4 wird schließlich ein Konzept für die Umsetzung der Anforderungen erarbeitet.

3.1. Anforderungsanalyse

3.1.1. Analyse Module

Ein KDD-Prozess, wie die Analyse von Logfiles mittels Sequential Pattern Mining, lässt sich, wie in Abschnitt 2.1 beschrieben, in fünf Schritte unterteilen: Selektion, Vorverarbeitung, Transformation, Data Mining und Evaluation. Jeder dieser Schritte muss innerhalb des Analyse-Workflows abbildbar sein. Dies ist in der aktuellen Version der Analytics Workbench, wenn auch eingeschränkt, gegeben. Im Folgenden soll festgehalten werden, wie die Schritte durch die Analytics Workbench abgebildet werden können und welche Probleme dabei auftreten.

- **Selektion**

Die Selektion der Daten passiert über die Input-Module, wie etwa das Data Repository Modul, das Dateien, die im Vorfeld auf dem Server abgelegt wurden, in den Workflow einspeisen kann. Auch ist es möglich, direkt Daten in das System hochzuladen oder aus einer Datenbank zu extrahieren. Das Einspeisen von Daten in die Analytics Workbench ist also bereits recht gut umgesetzt.

- **Preprocessing**

Das Preprocessing, wie beispielsweise das Filtern von Daten, ist möglich mit dem Activity Stream Filter Modul. Das Modul bietet die Möglichkeit, für bestimmte Felder der Activity Stream Einträge, Werte zu definieren. Einträge, die auf diese Feld-Wert-Kombination zutreffen, können dann ein- oder ausgeschlossen werden. Aktuell ist es allerdings nicht möglich, mehrere Filter gleichzeitig auf die Daten anzuwenden, ohne mehrere Filter-Module in Reihe zu schalten. So kann es vorkommen, dass zahlreiche Filter-Module den Workflow unübersichtlich machen.

Nach welchen Feldern gefiltert werden kann ist festgelegt im Modul selber und kann zur Laufzeit nicht geändert werden. Dies ist vor allem problematisch bei schwach strukturierten Daten wie Activity Streams. Abschnitt 3.1.2 geht näher auf diese Problematik ein.

3. Ansatz

Ein weiteres Problem stellt das Filtern nach Zeiträumen dar. Möchte man einen bestimmten Zeitabschnitt innerhalb der Daten untersuchen, so müssen die Daten aktuell schon auf diesen Zeitabschnitt getrimmt sein. Der Nutzer müsste also im Vorfeld die Möglichkeit haben, Einträge eines bestimmten Zeitabschnitts aus den Logfiles zu extrahieren.

- **Transformation**

Um Activity Streams mittels eines Sequential Pattern Mining Algorithmus untersuchen zu können, müssen die Daten zunächst in ein für den Algorithmus geeignetes Format übersetzt werden. So müssen die einzelnen Einträge in eindeutige Bezeichner¹ umgewandelt werden. Die Abfolge dieser Bezeichner kann dann beispielsweise nach Nutzern und Sitzung gruppiert werden, sodass pro Nutzer und pro Sitzung eine Sequenz von Einträgen vorhanden ist. Diese Sequenzen bilden dann die Grundlage für die Analyse. Abschnitt 2.1.1 erklärt die Funktionsweise im Detail. Das Sequential Pattern Mining Modul der Workbench übernimmt die oben genannte Funktionalität. Allerdings gibt es hier einige Einschränkungen. So lässt sich die Übersetzung von Activity Stream Einträgen in eindeutige Bezeichner nicht flexibel gestalten. Zur Auswahl stehen zwei festgelegte Übersetzungs-Formen, die aus einer Kombination der Werte bestimmter Felder einen Bezeichner generieren. Activity Streams, die nicht über diese Felder verfügen, können also nicht untersucht werden. Auch das Feld, nach dem die Sequenzen gruppiert werden, ist nicht veränderbar. So werden alle Einträge nach Nutzern gruppiert. Es kommt allerdings vor, dass man Aktivitäten nach Nutzergruppen gruppieren möchte, um z.B. zu untersuchen, welche Nutzungsmuster bei kollaborativen Arbeiten häufig vorkommen. Diese Funktionalität fehlt aktuell gänzlich.

- **Data-Mining**

Das Data-Mining umfasst die Anwendung von Algorithmen auf die zuvor verarbeiteten Daten. Dies geschieht, wie die Transformation, auch innerhalb des Sequential Pattern Mining Moduls. Die Analyse wird durch den CM-SPAM Algorithmus der SPMF-Bibliothek [FVGG⁺14] durchgeführt. Der Algorithmus setzt ein bestimmtes Format voraus, um häufige Muster in den Daten zu finden (siehe Abschnitt 2.1.1). Sofern sich die Transformation der Daten so gestalten lässt, dass der Algorithmus mit dem Format arbeiten kann, ist dieser Schritt unproblematisch.

- **Evaluation**

Die Ergebnisse der Analyse können in verschiedenen Formaten ausgegeben werden. Als Ausgabetypen bietet das Sequential Pattern Mining Modul SiSOB-Tabellen und Excel-Tabellen an. Diese lassen sich anschließend durch das Result Downloader Modul herunterladen oder durch das Table Viewer Modul visualisieren.

Für die Spezifikation von Analyse-Workflows im Kontext der Logfile-Analyse sind vor allem die Schritte *Preprocessing* und *Transformation* kritisch. Hier bietet die Analytics Workbench aktuell zu wenig Flexibilität. Ursache dafür ist zum einen die Implementierung der Module selber, zum anderen das Modul-System der Workbench an sich. Die Module zur Analyse von Logfiles scheinen eher rudimentär implementiert zu sein, wie im Abschnitt oben deutlich wurde. Auch das Modul-System der Workbench lässt Raum für Verbesserungen, beispielsweise im Hinblick auf die Oberflächenbeschreibung der Module, die aktuell nur statische Oberflächen erzeugen kann (siehe Abschnitt 2.3).

¹Ein Bezeichner kann z.B. eine Zahl sein, die eine bestimmte Art von Einträgen eindeutig identifiziert.

3.1.2. Schwach strukturierte Daten

Analyse-Workflows lassen sich gut automatisieren. Voraussetzung dafür ist allerdings, dass die Eingabe-Daten stark strukturiert sind. Das bedeutet, die jeweiligen Analyse-Module haben Informationen darüber, wie die zu verarbeitenden Daten aufgebaut sind. Grundlage dafür können Industrie-Standards oder feste Daten-Formate sein [Gan07]. Offene Daten-Formate mit hohen Freiheitsgraden, wie das JSON Activity Stream Format, das im Kontext dieser Arbeit zentral ist, sind in der Verarbeitung und Analyse allerdings problematisch [Jon07]. Wie in Abschnitt 2.1.2 bereits erwähnt, können die Felder der einzelnen Activity Stream Einträge beliebig erweitert werden. Das hat den Vorteil, dass die Daten mit weiteren Informationen über Nutzer und Aktion angereichert werden können. Beispielsweise können Nutzer in verschiedene Gruppen aufgeteilt werden, um A/B-Tests durchzuführen [KLSH09]. Für die Konfiguration eines Analyse-Workflows bedeutet das jedoch, dass das System wissen muss, wie die zu analysierenden Daten strukturiert sind. Möchte der Nutzer die Daten anhand der Gruppenzugehörigkeit filtern, so muss das System erkennen können, dass entsprechende Informationen zur Gruppenzugehörigkeit in den Eingabe-Daten vorliegen. Aktuell verfügen die Module der Analytics Workbench über statische Benutzeroberflächen. Das hat zur Folge, dass nur Daten analysiert werden können, die strukturell kompatibel sind mit den jeweiligen Analyse-Modulen. Das Activity Stream Filter Modul bietet daher nur die Möglichkeit, nach Feldern zu filtern, die in der JSON Activity Stream Spezifikation als Standard-Felder festgelegt sind. Das Filtern nach einem Feld, das die Gruppenzugehörigkeit definiert, wäre somit nicht möglich. Auch die Übersetzung der Activity Stream Daten für den Sequential Pattern Mining Algorithmus ist durch diese Problematik eingeschränkt. Die Generierung eindeutiger Bezeichner, das Gruppieren und Schneiden der Sequenzen wird anhand bestimmter Felder oder Feld-Werte-Kombinationen der Activity Stream Einträge vorgenommen (siehe Abschnitt 3.1.1). Für eine flexible Übersetzung der Daten ist es aber notwendig, dass bekannt ist, welche Felder in den Einträgen präsent sind. Diese Problematik bezieht sich auf alle Module der Analytics Workbench, unabhängig vom Kontext der Logfile-Analyse und hat zur Folge, dass das System wenig robust gegenüber schwach strukturierten Daten ist.

3.1.3. Anforderungen

Auf Grundlage der zuvor festgestellten Probleme sollen nun die Anforderungen an die neue Version der Workbench festgehalten werden.

- **Adaptive Konfiguration**

Die Konfigurationsmöglichkeiten der Module sollen sich abhängig von den Eingabe-Daten anpassen können, um den Nutzer bei der Konfiguration zu unterstützen und eine valide Konfiguration der Module zu ermöglichen.

- **Verbesserung der Benutzeroberfläche**

Die Flexibilität der Benutzeroberfläche soll ausgebaut werden, sodass es möglich ist, dynamisch auf Änderungen in der Konfiguration und den Eingabe-Daten reagieren zu können.

3. Ansatz

- **Verbesserung der Analyse-Komponenten**

Die Komponenten zur Analyse von Logfiles sollen verbessert werden, sodass eine flexiblere Konfiguration möglich ist.

- **Activity Stream Filter**

- * Die Felder und Werte, nach denen gefiltert werden kann, sollen sich durch die Struktur der Eingabedaten ergeben.
 - * Es soll möglich sein, mehrere Filter gleichzeitig je Modul definieren zu können.
 - * Für das Trimmen von Zeiträumen soll die Möglichkeit bestehen, einen Zeitraum auswählen zu können. Einträge die außerhalb des definierten Zeitraums liegen, werden ausgeschlossen.

- **Sequential Pattern Mining**

- * Es soll möglich sein, die Activity Stream Einträge anhand der in den Eingabe-Daten vorhandenen Feldern in ein für den CM-SPAM Algorithmus geeignetes Format übersetzen zu können.
 - * Die Gruppierung der Einträge zur Bildung der zu untersuchenden Sequenzen soll konfigurierbar sein, sodass nicht nur einzelne Nutzer sondern auch Nutzergruppen untersucht werden können.

- **Der bisherige Funktionsumfang soll erhalten bleiben.**

3.2. Adaptive Konfiguration

Adaption und Konfiguration sind eigentlich gegenteilige Begriffe. Mit adaptiver Konfiguration ist gemeint, dass sich die Konfigurationsmöglichkeiten der Module eigenständig an ihre Umgebung, also die Eingabe-Daten, anpassen sollen. Grundlage dafür soll das Konzept der *Self-adaptive Software*, das in Abschnitt 2.4.3 vorgestellt wurde, sein. Dazu ist es zunächst notwendig, dass dem System Informationen über die Struktur der Eingabe-Daten, im Folgenden Meta-Daten genannt, vorliegen. Welche Informationen mittels Meta-Daten transportiert werden können und wie diese erzeugt werden wird im folgenden Abschnitt 3.2.1 behandelt. Danach wird in Abschnitt 3.2.5 ein Ansatz entwickelt, wie sich die Benutzeroberflächen auf Grundlage der Meta-Daten anpassen können.

3.2.1. Meta-Daten

Meta-Daten sollen dem System Aufschluss über die Charakteristika der eigentlichen, zu analysierenden Daten geben. Generell lässt sich die Art von Daten, die in einen Workflow eingespeist werden an zwei Attributen identifizieren. Das Dateiformat und der Datentyp. Das Dateiformat ist essentiell für die Interpretation der in den Daten abgelegten Information. Activity Streams sind meist im JSON-Format kodiert, allerdings wären auch Formate wie CSV² denkbar. Der Datentyp beschreibt, um welche Art von Daten es sich

²Comma-Separated Values, Spezifikation <https://tools.ietf.org/html/rfc4180> Abruf 12.06.2016

```

1  {
2      "fileType": "json",
3      "dataType": "ActivityStream",
4      "fields": [
5          "actor.id",
6          "actor.display",
7          "actor.objectType",
8          "actor.group",
9          "verb",
10         "object.id",
11         "object.objectType",
12         "published",
13         "target.id",
14         "target.objectType"
15     ]
16 }

```

Abbildung 3.1.: Beispiel für Meta-Daten für Activity Streams

handelt. Eine JSON- oder CSV-Datei kann selbstverständlich nicht nur Activity Streams enthalten, daher ist der Datentyp notwendig, um die Daten näher zu beschreiben. In den Meta-Daten sollen diese zwei Attribute immer präsent sein. Für die Analyse von Activity Streams ist es notwendig zu wissen, welche Felder und Werte in den einzelnen Einträgen vorhanden sind (Siehe Abschnitt 3.1.2). Neben Dateiformat und Datentyp ist daher bei Activity Streams u.a. ein weiteres Attribut *fields* notwendig, das eine Liste der in den Einträgen präsenten Felder enthält. Abbildung 3.1 zeigt beispielhaft Meta-Daten für einen Activity Stream. Diese Daten könnten nun den Analyse-Modulen bereitgestellt werden, um dem Nutzer über eine adaptierte Oberfläche eine valide Konfiguration der Module zu ermöglichen. Ein ähnliches Konzept wurde bereits in RapidMiner (siehe Abschnitt 2.2.2.1) implementiert. Dort dienen Meta-Daten aber vor allem der Validierung. So können schon vor der Ausführung eines Workflows Inkompatibilitäten festgestellt werden [Rap14].

Nun soll eine Lösung für die Analyse der Eingabe-Daten zur Generierung der Meta-Daten, sowie die Versorgung der Module mit diesen, erarbeitet werden. Die Multiagenten-Architektur der Analytics Workbench (siehe Abschnitt 2.3.2) stellt dabei eine weitere Herausforderung dar. Jedes Modul, das Bestandteil des Analyse-Workflows ist, arbeitet komplett autonom. Bei der Durchführung eines Workflows wird je Modul eine Agenteninstanz erzeugt und nach Beendigung der Datenverarbeitung wieder verworfen. Während der Spezifikation und Konfiguration eines Workflows, aber auch zwischen zwei Iterationen, existieren also keine Agenten, die mit der Benutzeroberfläche in Kontakt stehen, um beispielsweise Eingabe-Daten zu analysieren oder auf die getätigte Konfiguration zu reagieren. Jeder Agent wird erst bei Durchführung erstellt und erhält Eingabe-Daten von den Agenten einer oder mehrerer Vorgänger-Module, sowie die durch die Benutzeroberfläche getätigte Konfiguration. Dabei haben die Agenten im Vorfeld keinerlei Information darüber, wie die eingehenden Daten strukturiert sind. In den Abschnitten 3.2.2, 3.2.3 und 3.2.4 werden nun drei alternative Ansätze zur Erzeugung von Meta-Daten diskutiert.

3. Ansatz

3.2.2. Erzeugung von Meta-Daten durch Module

Ein erster Ansatz wäre, dass jeder Agent die bei ihm eingehenden Daten selber analysiert und die Ergebnisse (Meta-Daten) zurück an das Frontend schickt. Dazu könnte die Nachrichtenaustauschkomponente der Kommunikationsebene genutzt werden (siehe Abschnitt 2.3.2 sowie Abbildung 2.8). Die Agenten nutzen diesen Kanal bereits, um Informationen über ihren Arbeitsstatus an das Frontend zu schicken, so wird bei Beendigung der Datenverarbeitung eines Agenten, eine entsprechende Nachricht verschickt. Hier könnten weitere Informationen, wie eben die Meta-Daten, transportiert werden. Abbildung 3.2 zeigt diesen Ansatz.

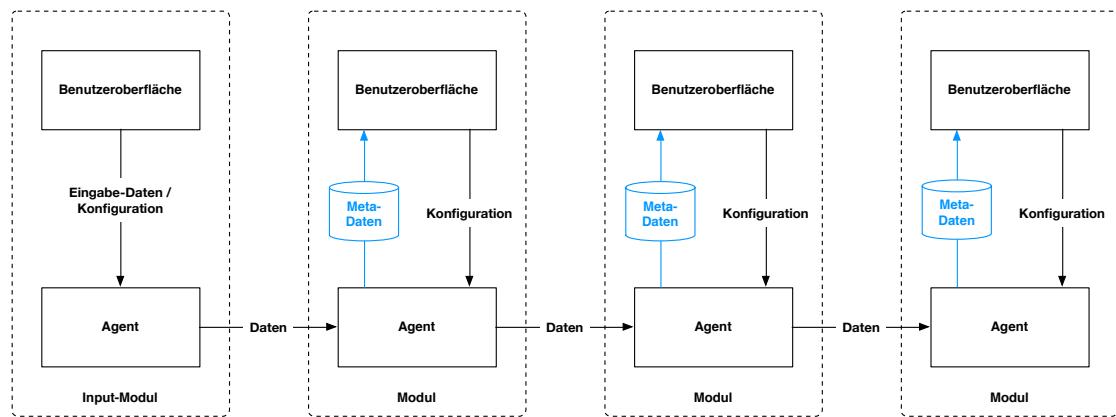


Abbildung 3.2.: Ansatz 1: Erzeugung von Meta-Daten durch Module selber

Problematisch hierbei ist jedoch, dass der Workflow initial durchlaufen werden muss, um die Meta-Daten zu erzeugen. So können sich die Benutzeroberflächen der Module erst dann adaptieren, wenn der Workflow einmalig durchlaufen wurde. Auch bei jeder Änderung der Modul-Konfigurationen ist ein erneuter Durchlauf notwendig, da durch Änderungen, die aktuellen Meta-Daten möglicherweise nicht mehr valide sind. Außerdem nimmt die Durchführung komplexer Workflows mit großen Datenmengen einige Zeit in Anspruch, in der der Nutzer auf die Ergebnisse warten muss. Allerdings bietet dieser Ansatz auch Vorteile. Da ein Modul die bei ihm eingehenden Daten selber analysiert und somit Struktur und Inhalt der Meta-Daten selber bestimmen kann, ist es möglich, nicht nur Meta-Daten sondern auch Statistiken oder gar Visualisierungen zum Frontend zu transportieren, um dem Benutzer ein vollständiges Bild, der eingehenden Daten zu geben. Insgesamt ist dieser Ansatz, aus den oben genannten Gründen, jedoch wenig praktikabel und nicht benutzerfreundlich.

3.2.3. Erzeugung von Meta-Daten durch Workflow

Um bei Änderungen der Eingabe-Daten oder der Modul-Konfigurationen nicht jedes Mal den gesamten Workflow durchführen zu müssen, sollte die Analyse unabhängig vom eigentlichen Workflow ablaufen können. Sobald Eingabe-Daten vorhanden sind, könnte im Hintergrund ein Workflow, der nur für die Erzeugung der Meta-Daten zuständig

ist, angestoßen werden. Dies müsste für jedes Input-Modul, das Daten in den Workflow einspeist geschehen. Dazu bietet sich die bisherige Infrastruktur zur Durchführung der Workflows an. Ein solcher Workflow würde aus dem entsprechenden Input-Modul, sowie einem Analyse-Modul, das die Daten entgegennimmt und Meta-Daten erzeugt, bestehen. Das Input-Modul ist dabei eine Spiegelung des Moduls aus dem eigentlichen Workflow. Die Ergebnisse der Analyse würden, wie bei herkömmlichen Workflows, durch das Result-Repository bereitgestellt (siehe Abschnitt 2.3.2 sowie Abbildung 2.8). Die Benutzeroberflächen der Module könnten somit auf die Meta-Daten zugreifen und ihre Oberflächen anpassen. Abbildung 3.3 zeigt diesen Ansatz.

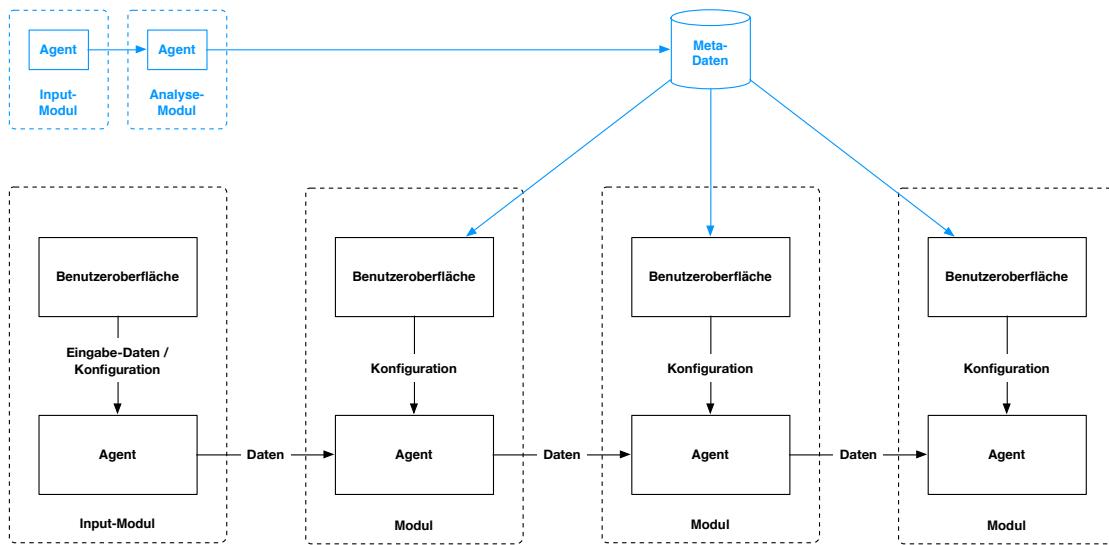


Abbildung 3.3.: Ansatz 2: Erzeugung von Meta-Daten durch unabhängigen Workflow

Problematisch hierbei ist jedoch, dass jedes Modul dieselben Meta-Daten erhalten würde. Allerdings kann es vorkommen, dass durch sich Filterung und Transformation die Struktur der Daten beim Durchlaufen des Workflows ändert. Die eingangs erzeugten Meta-Daten könnten für hintere Module gegebenenfalls ungültig sein. Bei der Analyse von Activity Streams ändert sich die Struktur der Daten eher selten. Allerdings wird die Analytics Workbench für zahlreiche Analyse-Techniken genutzt, bei denen davon ausgegangen werden kann, dass sich Daten oftmals ändern. Daher soll im nächsten Abschnitt eine Möglichkeit gefunden werden, auf mögliche Änderungen in den Daten reagieren zu können.

3.2.4. Transformation von Meta-Daten

Bei der Durchführung einer Analyse werden die Eingabe-Daten entsprechend der Verbindungen zwischen den Modulen durch den Workflow geleitet und transformiert. Jedes Modul erwartet entsprechend seiner Eingänge bestimmte Daten und weiß, wie es diese Daten verändert und über seine Ausgänge weitergibt. Wenn jedes Modul die Möglichkeit hätte, aufgrund seiner eigenen Funktionalität und Konfiguration zu beschreiben, wie es Daten verändert, könnten Meta-Daten eben wie die tatsächlichen Daten, auf einer Art

3. Ansatz

Meta-Ebene durch den Workflow geleitet und durch die Module transformiert werden. Dies könnte ganz unabhängig vom Server oder dem Multiagentensystem im Browser geschehen. Voraussetzung dafür ist, dass der Benutzeroberfläche, also dem Frontend, je Modul eine Funktionen (im Folgenden Meta-Daten-Transformationsfunktion genannt) bereitsteht, die aus den eingehenden Meta-Daten und der Modul-Konfiguration neue Meta-Daten für die folgenden Module erzeugen kann (siehe Abbildung 3.4).

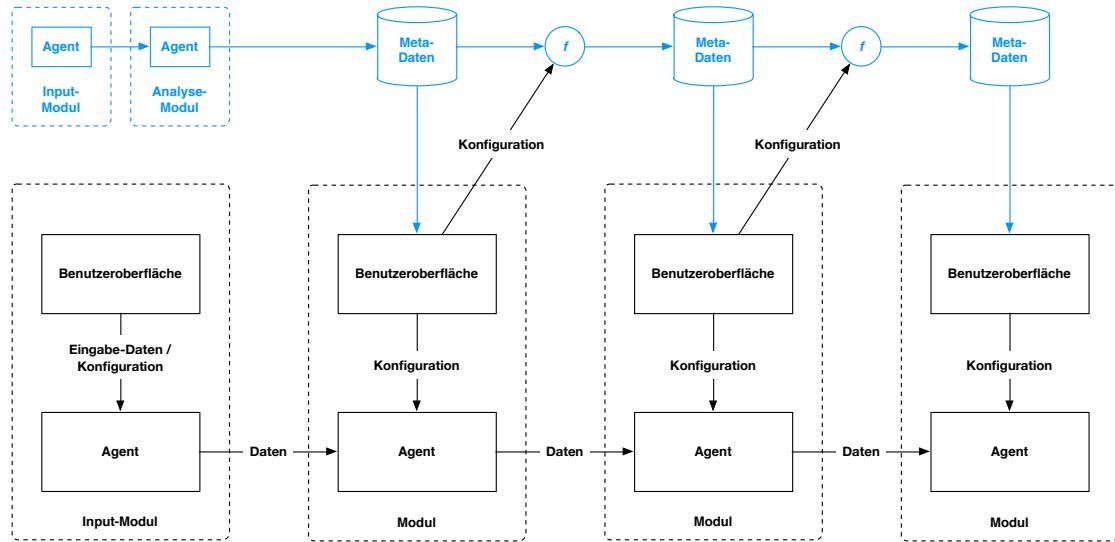


Abbildung 3.4.: Ansatz 3: Erweiterung um Meta-Ebene

Ein Workflow, wie er in der Workbench spezifiziert werden kann, ist ein gerichteter azyklischer Graph³. Jedes Modul entspricht einem Knoten in diesem Graphen, über den die eingehenden Kanten (*incomming_edges*), die Meta-Daten-Transformationsfunktion (*TRANSFORM_META*) und die aktuelle Konfiguration (*config*) bekannt sind. Um die Meta-Daten für ein bestimmtes Modul zu erhalten, kann Algorithmus 1 herangezogen werden. Der Algorithmus erwartet als Input den Knoten, für den Meta-Daten abgeleitet werden sollen. Entspricht der Knoten einem Input-Modul, gibt der Algorithmus die aus dem Analyse-Workflow hervorgehenden Meta-Daten direkt zurück (Zeile 3). Ist der Knoten ein herkömmliches Modul, werden zunächst die Meta-Daten aller vorherigen Module abgeleitet. Dazu durchläuft der Algorithmus alle an dem Knoten eingehenden Kanten bzw. die damit verbundenen Knoten und wendet sich rekursiv an (Zeile 14). Sobald die Meta-Daten für alle Eingänge bekannt sind, wird die Meta-Daten-Transformationsfunktion (*TRANSFORM_META*) mit diesen Meta-Daten und der aktuellen Modul-Konfiguration angewendet (Zeile 20). Das Ergebnis der Funktion wird schließlich zurückgegeben. Auf diesem Wege können für alle Module gültige Meta-Daten abgeleitet werden. Voraussetzung dafür ist, dass zuvor Meta-Daten durch einen Analyse-Workflow für alle Input-Module erzeugt wurden. Dieser Ansatz hat jedoch auch eine Einschränkung: abgeleitete Informationen können nur schwierig transformiert werden. Ein Beispiel dafür wäre die Information, wie viele Einträge in einem Activity Stream vorhanden sind. Die Meta-Daten-Transformationsfunktion des Activity Stream Filters könnte auf Grundlage seiner eingestellten Filter keine Annahmen darüber treffen, wie viele Einträge nach Anwendung

³DAG, directed acyclic graph

Algorithm 1 getMetaData Algorithmus

```

1: function GET_META_DATA(node)
2:
3:   if IS_INPUT_NODE(node) then                                ▷ Input-Modul erreicht
4:     return GET_META_DATA_FROM_ANALYSIS(node)
5:   end if
6:
7:   incomming_edges ← GET_INCOMMING_EDGES(node)
8:   count ← COUNT(incomming_edges)
9:   meta ← ARRAY()
10:
11:  for i ← 0, count do                                     ▷ Durchlaufe alle eingehenden Kanten
12:    edge ← incomming_edges[i]                               ▷ Aktuelle Kante
13:    ancestor ← GET_FROM_NODE(edge)                         ▷ Refenz zum Vorgänger-Knoten
14:    meta[i] ← GET_META_DATA(ancestor)                  ▷ Rekursion
15:  end for
16:
17:  config ← GET_CONFIG(node)                                ▷ Modul-Konfiguration
18:  TRANSFORM_META ← GET_TRANSFORM_META(node)
19:
20:  return TRANSFORM_META(meta, config)                      ▷ Anwendung
21:
22: end function

```

der Filter noch vorhanden sind. Dazu müssten in den Meta-Daten für jede Feld-Werte-Kombination die Anzahl der betreffenden Einträge vorhanden sein. Für die Möglichkeit der adaptiven Konfiguration der Analyse-Komponenten sollte dieser Ansatz jedoch ausreichen.

Wie kann ein Modul dem Frontend nun eine Meta-Daten-Transformationsfunktion bereitstellen? Wie in Abschnitt 2.3.3 erwähnt, stellen Module dem Frontend eine sogenannte *Filter Description* bereit, die Informationen, wie die Beschreibung der Benutzeroberflächen zur Modul-Konfiguration, enthält (siehe auch Abbildung 2.9). Hier soll auch die Meta-Daten-Transformationsfunktion mitgeliefert werden. Problematisch hierbei ist jedoch, dass die *Filter Description* im JSON-Format kodiert ist. Das JSON-Format unterstützt allerdings nur die primitive Datentypen *String*, *Number*, *Boolean*, *Null*, *Object* und *Array* [Bra14]. Funktionen können daher nicht nativ kodiert werden. Allerdings gibt es browserseitig die Möglichkeit, *Strings* als JavaScript-Code zu behandeln und auszuführen. Die Implementierung der Meta-Daten-Transformationsfunktion könnte daher als *String* in der *Filter-Description* mitgeliefert werden (mehr dazu in Kapitel 4). Wie eine Meta-Daten-Transformationsfunktion implementiert werden kann zeigen Abbildung 3.5 und 3.6. Das Activity Stream Filter Modul erwartet als Eingabe-Daten einen Activity Stream und filtert bestimmte Einträge heraus (siehe Abschnitt 2.3.4.1). Der Datentyp der Eingabe-Daten ändert sich also nicht durch die Anwendung des Moduls. In diesem Falle können die am Modul eingehenden Meta-Daten einfach durchgeleitet werden (siehe Abbildung 3.5). Die Funktion erwartet als Parameter die Meta-Daten, die an den jeweiligen Eingängen ankommen (*metaByInput*) sowie die aktuelle Konfiguration des Moduls (*config*). Da das Modul nur einen Eingang (*in_1*) hat, werden die Daten,

3. Ansatz

die dort ankommen, von der Funktion zurückgegeben. Abbildung 3.6 zeigt beispielhaft die Meta-Daten-Transformationsfunktion des Sequential Pattern Mining Moduls. Das Modul bietet die Möglichkeit, die Ergebnisse der Analyse als Excel- (*xls*) oder SISOB-Tabelle (*sdt*) zurückzuliefern (siehe Abschnitt 2.3.4.2). Der *fileType* ist daher variabel und abhängig von der Konfiguration des Moduls. Der *dataType* hingegen ist immer der selbe, nämlich *SequentialPatterns*.

```
1 function transformMeta(metaByInput, config) {
2   return metaByInput.in_1;
3 }
```

Abbildung 3.5.: Meta-Daten-Transformationsfunktion Activity Stream Filter

```
1 function transformMeta(metaByInput, config) {
2   var fileType = config.output === 'Excel file' ? 'xls' : 'sdt';
3   return {
4     fileType: fileType,
5     dataType: 'SequentialPatterns'
6   };
7 }
```

Abbildung 3.6.: Meta-Daten-Transformationsfunktion Sequential Pattern Mining

3.2.5. Adaption der Benutzeroberflächen

Nun, da eine Möglichkeit besteht, Informationen über die Struktur der Eingabe-Daten zu erhalten, muss ein Ansatz zum Adoptionsmanagement (siehe Abschnitt 2.4.3), also der Adaption der Benutzeroberfläche auf Grundlage der Meta-Daten, erarbeitet werden. Module beschreiben ihre Benutzeroberfläche zur Konfiguration innerhalb der *Filter Description* (siehe Abschnitt 2.3.3 und Abbildung 2.9). Dazu verwendet die Analytics Workbench ein eigenes Format. Jedes Eingabe-Feld, das in der Benutzeroberfläche präsent sein soll, kann darin definiert werden. Dazu werden je Feld ein Typ (*type*), z.B. Textfeld oder Auswahlbox, der Name (*name*) der Variable, die später in der Konfiguration den Wert des Feldes hält, sowie weitere Parameter wie die Beschriftung (*label*) oder ein Indikator, ob das Feld ein Pflichtfeld ist (*required*), definiert. Diese Beschreibung ist statisch und kann zur Laufzeit nicht geändert werden. Um die Beschreibung dennoch dynamisch zu gestalten, sollen Module die Möglichkeit erhalten, eine Funktion zu definieren, die die Meta-Daten und die aktuelle Modul-Konfiguration (also die Werte der Eingabe-Felder) erhält und auf dieser Grundlage eine veränderte Oberflächenbeschreibung zurückgibt. Neben der Meta-Daten-Transformationsfunktion soll es also eine Oberflächen-Transformationsfunktion geben. Diese Funktion wird bei jeder Änderung der Meta-Daten und Modul-Konfiguration durchgeführt. So kann mittels der Oberflächenbeschreibung und der dazugehörigen Transformationsfunktion für jeden Umstand die Benutzeroberfläche beschrieben werden. Das macht es nicht nur möglich, die Oberfläche auf Grundlage der Meta-Daten zu ändern, sondern auch auf Grundlage

der aktuellen Konfiguration. So wäre es denkbar, dass bei einer bestimmten Konfiguration (z.B. Auswahl eines Algorithmus zur Analyse) weitere Parameter, die nur der ausgewählte Algorithmus unterstützt, konfiguriert werden können. Für die Filterung der Activity Stream Daten soll das Activity Stream Filter Modul die Möglichkeit bieten, die Daten anhand bestimmter Feld-Werte-Kombination zu filtern (siehe Abschnitt 3.1.2). Mit den Meta-Daten, in denen vermerkt ist, welche Felder in den einzelnen Activity Stream Einträgen präsent sind, ist es nun möglich dem Benutzer eine Auswahl dieser Felder bereitzustellen. Für die Auswahl des Feldes, müsste in der Oberflächenbeschreibung eine Auswahlbox definiert werden. Diese würde standardmäßig die im JSON Activity Stream Standard definierten Felder als Auswahlmöglichkeiten bereitstellen. Sobald Meta-Daten mit den in den Einträgen tatsächlich vorhandenen Feldern bereitstehen, kann die Oberflächen-Transformationsfunktion die Liste Auswahlmöglichkeiten um diese Felder erweitern. Wie dies im Detail für die verschiedenen Module umgesetzt werden soll, wird in Abschnitt 3.3 behandelt. Wie in Abschnitt 2.4.3 bereits aufgegriffen, eignet sich für die Umsetzung dieses Ansatzes die Verwendung von React. Mit React könnte eine Komponente implementiert werden, die die Oberflächenbeschreibung entgegennimmt und daraus entsprechende Eingabe-Felder erzeugt. Bei einer Änderung der Beschreibung würde diese Komponente erneut gerendert und die Änderungen von React umgesetzt.

3.2.6. Zusammenfassung

In diesem Abschnitt wurde ein Ansatz zur Umsetzung einer adaptiver Konfiguration von Analyse-Modulen erarbeitet. Der Ansatz ist angelehnt an das Konzept der *Self-adaptive Software* (siehe Abschnitt 2.4.3). Module können so ihre Oberfläche auf Grundlage der Struktur der Eingabe-Daten adaptieren. Meta-Daten enthalten Informationen über diese Struktur und werden den Modulen bereitgestellt. Für die Erzeugung von Meta-Daten werden die Eingabe-Daten, sobald diese in dem Workflow vorhanden sind, mittels eines im Hintergrund ablaufenden Meta-Analyse-Workflows analysiert. Dazu muss ein Meta-Analyse-Modul implementiert werden, das nur für den Meta-Analyse-Workflow zu Verfügung steht. Die erzeugten Meta-Daten werden dem Frontend bereitgestellt. Über eine Meta-Ebene werden alle im Workflow befindlichen Module mit validen Meta-Daten versorgt. Dazu müssen Module eine Meta-Daten-Transformationsfunktion bereitstellen. Die Benutzeroberflächen zur Konfiguration der Module werden wie gehabt in der Oberflächenbeschreibung des Modul definiert. Um die Oberflächen dynamisch zu machen, müssen Module eine Oberflächen-Transformationsfunktion implementieren, die auf Grundlage der Meta-Daten und der eignen Konfiguration die Oberflächenbeschreibung verändern kann. Oberflächenbeschreibung, Oberflächen-Transformationsfunktion und Meta-Daten-Transformationsfunktion werden dem Frontend über die *Filter Description* bereitgestellt.

3.3. Analyse-Komponenten

In diesem Abschnitt sollen nun die für die Analyse von Activity Streams kritischen Analyse-Module, das Activity Stream Filter Modul und das Sequential Pattern Mining Modul, neu konzipiert werden. Dazu sollen für die Benutzeroberflächen zur Konfiguration der Module, Wireframes erstellt und die zugrundeliegende Funktionalität erarbeitet

3. Ansatz

werden. Zentral sind hierbei die Eingabe-Felder und deren Möglichkeit zur Adaption an die Struktur der Eingabe-Daten.

3.3.1. Activity Stream Filter

In den Anforderungen aus Abschnitt 3.1.3 wurden folgende Funktionen für das Activity Stream Filter Modul festgelegt. Für das Filtern von Activity Stream Einträgen soll das Modul die Möglichkeit bieten, Einträge anhand von Feld-Werte-Kombination zu filtern. Nach welchen Feldern und Werten gefiltert werden kann, ergibt sich durch die Meta-Daten, die Informationen über die tatsächlich, in den Eingabe-Daten vorhandenen Feldern und Werten enthalten. Das Modul soll die Anwendung mehrerer Filter unterstützen, sodass es nicht nötig ist, mehrere in Reihe geschaltete Modul-Instanzen zu erstellen. Außerdem soll es möglich sein, die Einträge auf einen bestimmten Zeitraum zu trimmen.

Die Möglichkeit mehrere Filter gleichzeitig zu konfigurieren, stellt eine Herausforderung für die Benutzeroberfläche dar. In der aktuellen Version der Analytics Workbench, stellt das Modul für bestimmte Felder der Activity Stream Einträge Textfelder bereit, in denen ein Wert eingetragen werden kann (siehe Abschnitt 2.3.4.1 und Abbildung 2.10). Um mehrere Filter gleichzeitig konfigurieren zu können, ist nun eine tabellarische Darstellung der Filter notwendig. Abbildung 3.7 (oben links) zeigt die Tabelle zur Verwaltung der Filter. In dem Beispiel sind bereits zwei Filter in der Tabelle angelegt. Der erste Filter schließt alle Einträge ein, die in dem Feld *actor.group* den Wert *beta* aufweisen. Gleichzeitig schließt der zweite Filter alle Einträge aus, die in dem Feld *actor.role* den Wert *admin* aufweisen. Ein Filter lässt sich aus drei Komponenten zusammenstellen: Der Name des Feldes, der Wert des Feldes und der Modus des Filters (ein- oder ausschließen). Für jede Komponente muss also eine Spalte in der Tabelle vorhanden sein. Um neue Filter hinzufügen zu können, soll ein entsprechendes Formular oberhalb der Tabelle platziert werden. Für die Auswahl von Feld und Wert sollen Eingabe-Felder mit einer Auto vervollständigung-Funktionalität (*Auto Complete*) bereitstehen. Dies macht es möglich, eine Benutzereingabe sinnvoll zu ergänzen und lässt den Nutzer gleichzeitig nach vorhandenen Auswahlmöglichkeiten suchen. Grundlage für die Vervollständigung sind die aus den Meta-Daten hervorgehenden Informationen über die in den Einträgen vorhandenen Felder bzw. Werte. Sollten keine Meta-Daten zu Verfügung stehen, so beschränken sich die Auswahlmöglichkeiten auf die in dem JSON Activity Stream Standard festgelegten Standard-Felder. Die Auswahlmöglichkeiten verkleinern sich entsprechend des Fortschritts der Benutzereingabe. Hat das Eingabe-Feld den Wert *actor*, so werden nur Einträge angezeigt, die diesen Wert enthalten; z.B. *actor.id*, *actor.group*, *actor.role* (siehe Abbildung 3.7, oben rechts). Für die Auswahl des Modus, soll eine einfache Auswahlbox, mit den Auswahlmöglichkeiten *include* (einschließen) und *exclude* (ausschließen), genutzt werden. Um die Eingabe zu bestätigen und den Filter schließlich hinzuzufügen, muss der Button rechts vom Formular gedrückt werden. Um bereits erstellte Filter wieder zu löschen, befindet sich in der jeweiligen Zeile des Filters ein weiterer Button. Die beschriebene Tabellen-Komponente wird von der aktuellen Version der Workbench nicht unterstützt und muss daher implementiert werden. Damit auch zukünftige Erweiterungen die Komponente nutzen können, soll diese möglichst flexibel gestaltet werden. So soll es möglich sein, beliebige Spalten und Eingabe-Felder für die Tabelle konfigurieren zu können. Details zur Implementierung der Tabelle folgen in Kapitel 4.

3.3. Analyse-Komponenten

Activity Stream Filter	
Filter	Filter
<input type="button" value="Select property ▼"/> <input type="button" value="Select value ▼"/> <input type="button" value="Select mode ▼"/> <input type="button" value="+"/> actor.group beta include <input type="button" value="x"/> actor.role admin exclude <input type="button" value="x"/>	<input type="button" value="act ▼"/> <input type="button" value="Select value ▼"/> <input type="button" value="Select mode ▼"/> <input type="button" value="+"/> actor.id actor.group actor.qualification actor.role
Date range	Date range
From: 01.01.2014	From: 01.01.2014
To: 01.01.2015	To: 01.01.2015

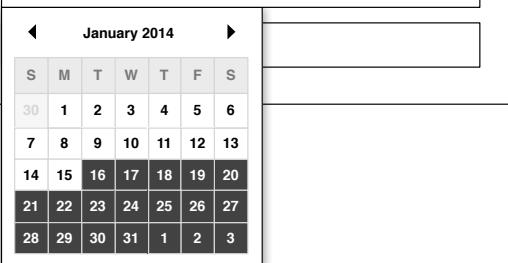
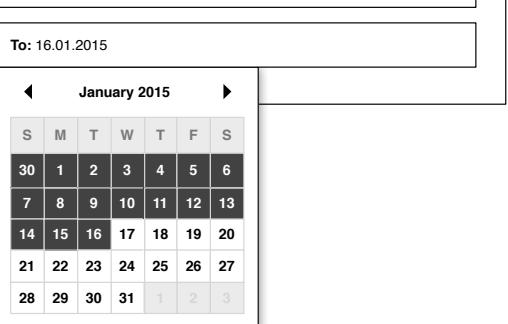
Activity Stream Filter	
Filter	Filter
<input type="button" value="Select property ▼"/> <input type="button" value="Select value ▼"/> <input type="button" value="Select mode ▼"/> <input type="button" value="+"/> actor.group beta include <input type="button" value="x"/> actor.role admin exclude <input type="button" value="x"/>	<input type="button" value="Select property ▼"/> <input type="button" value="Select value ▼"/> <input type="button" value="Select mode ▼"/> <input type="button" value="+"/> actor.group beta include <input type="button" value="x"/> actor.role admin exclude <input type="button" value="x"/>
Date range	Date range
From: 16.01.2014	From: 16.01.2014
	

Abbildung 3.7.: Konzept für die Benutzeroberfläche des Activity Stream Filter Moduls

3. Ansatz

Für die zeitliche Eingrenzung der Activity Stream Einträge soll es möglich sein, ein Start- und End-Datum angeben zu können. Die Auswahl eines Datums soll über einen sogenannten *Date Picker* geschehen (siehe Abbildung 3.7, unten links und unten rechts). Beim Klick auf eines der Eingabe-Felder öffnet sich ein angehefteter Dialog mit Monatsübersicht. Dunkel hinterlegte Tage signalisieren, dass sich diese in der aktuellen Auswahl des Zeitraums befinden. Nach Auswahl des Start-Datums öffnet sich derselbe Dialog zur Auswahl des End-Datums. Auch hier sollen Meta-Daten den Nutzer unterstützen und die Möglichkeit bieten, einen validen Zeitraum auswählen zu können. Dazu muss bei der Analyse der Eingabe-Daten zusätzlich der Zeitraum, aus dem die Activity Stream Einträge stammen, erfasst werden. Die Datumsauswahl beschränkt sich dann auf den erfassten Zeitraum. Tage, die sich außerhalb dieses Zeitraums befinden, sind nicht auswählbar und werden optisch ausgegraut. Auch diese Komponente wird von der aktuellen Version der Workbench nicht unterstützt und muss implementiert werden.

3.3.2. Sequential Pattern Mining

Die Aufgaben des Sequential Pattern Mining Moduls umfassen das Transformieren der Activity Stream Daten in ein von dem CM-SPAM Algorithmus unterstütztes Format, sowie die Anwendung des Algorithmus selber. Der Algorithmus erwartet als Eingabe-Daten eine Sequenzdatenbank. Die Sequenzen bestehen aus einer Abfolge eindeutiger numerischer Bezeichner (eine detaillierte Beschreibung der Funktionsweise ist in 2.1.1 zu finden). Die Transformation der Activity Stream Einträge in das beschriebene Format besteht dabei aus mehreren Schritten. Zunächst müssen aus den einzelnen Activity Stream Einträgen numerische Bezeichner erzeugt werden. Dazu werden die Werte ein oder mehrerer Felder der Einträge herangezogen. Dies könnten beispielsweise die Felder *verb* und *object.objectType* sein. Hat ein Eintrag in den Feldern die Werte *likes* und *picture*, so würde daraus zunächst der Bezeichner *likes_picture* entstehen. Einträge, die in diesen Feldern dieselben Werte aufweisen, erzeugen denselben Bezeichner. Diese Bezeichner werden gesammelt und schließlich in eindeutige numerische Werte übersetzt werden. Wäre *likes_picture* der erste beobachtete Bezeichner, so würde dieser in den Sequenzen als Wert 1 repräsentiert; der nächste als Wert 2 usw. Im nächsten Schritt werden die erzeugten Bezeichner in Sequenzen eingeteilt. Typischerweise wird dazu das Feld *actor.id* herangezogen, das den Nutzer, der die Aktion (also den Eintrag) erzeugt hat, identifiziert. In diesem Fall werden die Sequenzen nach Nutzern gruppiert. Möchte man die Aktionen von Nutzergruppen analysieren, z.B. zur Untersuchung von kollaborativem Arbeiten, so könnte ein anderes Feld, beispielsweise *actor.group* (falls vorhanden), herangezogen werden. Nun liegt je Nutzer eine Sequenz an Aktionen, die bei der Nutzung einer Webanwendung getätigten wurden vor. Dies berücksichtigt allerdings nicht, dass die Aktionen in separaten Sitzungen entstanden sind. Im letzten Schritt müssen die erzeugten Sequenzen daher nach Sitzungen geschnitten werden. Bei jedem Login in eine Webanwendung kann von einer neuen Sitzung ausgegangen werden. Typischerweise wird dazu ein Activity Stream Eintrag erzeugt, der im Feld *verb* den Wert *login* aufweist. Die Sequenzen werden nun an den Stellen geschnitten, an denen ein solcher Eintrag beobachtet wird. Welche Feld-Wert-Kombination eine neue Sitzung einleitet, kann allerdings von System zu System unterschiedlich sein, sodass hier die Möglichkeit bestehen soll, eine beliebige Kombination angeben zu können. Die erzeugte Sequenzdatenbank kann nun

Sequential Pattern Mining	
Identifier	<input type="text" value="x verb x object.objectType"/>
Group by	<input type="text" value="x actor.id"/>
Split at property	<input type="text" value="verb"/>
Split value	<input type="text" value="login"/>
Minimum support	<input type="range" value="0.5"/>
Continuous sequences	<input type="checkbox"/> Only allow continuous sequences
Output type	<input type="text" value="SISOB table"/>

Sequential Pattern Mining	
Identifier	<input type="text" value="x verb x object.objectType obje "/>
object.id	
object.title	
object.name	
object.origin	
verb	<input type="text" value="verb"/>
Split value	<input type="text" value="login"/>
Minimum support	<input type="range" value="0.5"/>
Continuous sequences	<input type="checkbox"/> Only allow continuous sequences
Output type	<input type="text" value="SISOB table"/>

Abbildung 3.8.: Konzept für die Benutzeroberfläche des Sequential Pattern Mining Moduls

vom CM-SPAM Algorithmus auf häufige Muster untersucht werden. Der Algorithmus kann dazu über weitere Parameter konfiguriert werden. Mithilfe des *Minimum Support* Parameters lässt sich einstellen, ab wann eine Sequenz als häufig gilt. Dazu kann ein Wert zwischen 0 und 1 angegeben werden, der den Anteil an der gesamten Menge an Sequenzen repräsentiert. Bei 100 Sequenzen und einem *Minimum Support* von 0.1, gilt eine Sequenz als häufig, wenn sie mindestens zehn mal beobachtet wurde. Außerdem lässt sich einstellen, ob nur ununterbrochene (Sub-)Sequenzen betrachtet werden sollen oder nicht (*Continuous Sequences*).

Die Benutzeroberfläche des Sequential Pattern Mining Moduls muss nun in der Lage sein, die Konfiguration der oben beschriebenen Funktionalität abzubilden. Abbildung 3.8 (links) zeigt das Konzept der Benutzeroberfläche. Für die Erzeugung der Bezeichner soll es möglich sein, mehrere Felder der Activity Stream Einträge auswählen zu können. Dazu wird ein Eingabe-Feld (*Identifier*) mit Auto vervollständigungs-Funktion (wie sie in Abschnitt 3.3.1 bereits beschrieben wurde) und der Möglichkeit zur Mehrfachauswahl bereitgestellt (siehe Abbildung 3.8, rechts). Ausgewählte Felder werden innerhalb des Eingabe-Feldes als optisch getrennte Elemente repräsentiert. Über einen Button (links

3. Ansatz

im Element) können bereits ausgewählte Felder wieder entfernt werden. Die Reihenfolge der ausgewählten Felder ist für die Transformation unerheblich. Für die Gruppierung der Bezeichner sollen ebenfalls mehrere Felder ausgewählt werden können, sodass hier ein Eingabe-Feld (*Group by*) mit derselben Funktionalität zum Einsatz kommt. Für das Schneiden der Sequenzen soll eine beliebige Feld-Wert-Kombination angebbar sein. Die Auswahl des Feldes (*Split at property*) und des damit verbundenen Wertes (*Split value*) sind möglich über herkömmliche Auswahlboxen. Alle bisher beschriebenen Eingabe-Felder (*Identifier*, *Group by*, *Split at property* und *Split value*) sollen ihre Auswahlmöglichkeiten auf Grundlage von Meta-Daten anpassen können. Für die Einstellung des *Minimum Support* Parameters soll ein Schieberegler (*Slider*) verwendet werden. In der aktuellen Version wird dieser Parameter über ein Textfeld eingestellt. Der Schieberegler hat den Vorteil, dass er zu jeder Zeit valide Werte, aus einem im Vorfeld definierten Wertebereich (hier 0.0 - 1.0), zurückgibt und intuitiv bedienbar ist. Diese Komponente wird zur Zeit allerdings nicht unterstützt und muss ebenfalls implementiert werden.

3.4. Arbeitsumgebung

Die Implementierung der adaptiven Benutzeroberflächen, erfordert wie in Abschnitt 3.2 erwähnt, ein Reengineering des Frontends der Analytics Workbench. Da sich Teile des bisherigen Frontends nur schwierig in die neue Umgebung portieren lassen, macht es Sinn, im selben Zuge eine Neugestaltung der Arbeitsumgebung durchzuführen. In diesem Abschnitt soll daher die Arbeitsumgebung zur visuellen Spezifikation der Analyse-Workflows neu konzipiert werden, wobei der aktuelle Funktionsumfang erhalten bleiben soll.

3.4.1. Workflow-Spezifikation

Zentral bei der visuellen Spezifikation von Analyse-Workflows ist die zugrundeliegende Symbolik. Bei Workflow-basierten Analyse-Plattformen werden Workflows oft durch eine Graphen-Metapher dargestellt, bei der die Knoten den Modulen zur Datenverarbeitung entsprechen (siehe Abschnitt 2.2.1). Diese visuelle Sprache ist bereits Teil der Analytics Workbench und soll in der neuen Version ebenso verwendet werden. Die Konfiguration der Analyse-Module findet in der aktuellen Version in der Modul-Repräsentation innerhalb des Workflows statt (siehe Abbildung 2.7). Bei komplexen Workflows kann dies schnell dazu führen, dass der Workflow unübersichtlich wird, gerade durch Analyse-Module, wie dem Sequential Pattern Mining Modul, die eine umfangreiche Konfiguration erlauben und daher viel Platz einnehmen. Angelehnt an die Arbeitsumgebungen von RapidMiner und KNIME (siehe Abschnitt 2.2.2.1 und 2.2.2.2) soll sich die Funktionalität der Workflow-Repräsentation auf die Spezifikation des Workflow-Graphen, also das Verbinden der Ein- und Ausgänge der Module, beschränken. Um mehr Platz für den Workflow zu schaffen, sollen die Modul-Repräsentationen nur noch eine Übersicht ihrer aktuellen Konfiguration darbieten; für die Konfiguration soll ein eigener Bereich geschaffen werden. Abbildung 3.9 zeigt das neue Konzept der Arbeitsumgebung und die Visualisierung des Workflows. Für jeden konfigurierbaren Parameter soll in Modul-Repräsentation eine Zeile mit Bezeichnung und Wert des Parameters vorhanden sein. Sollte die Übersicht eine bestimmte Höhe überschreiten, so wird dieser Bereich über eine

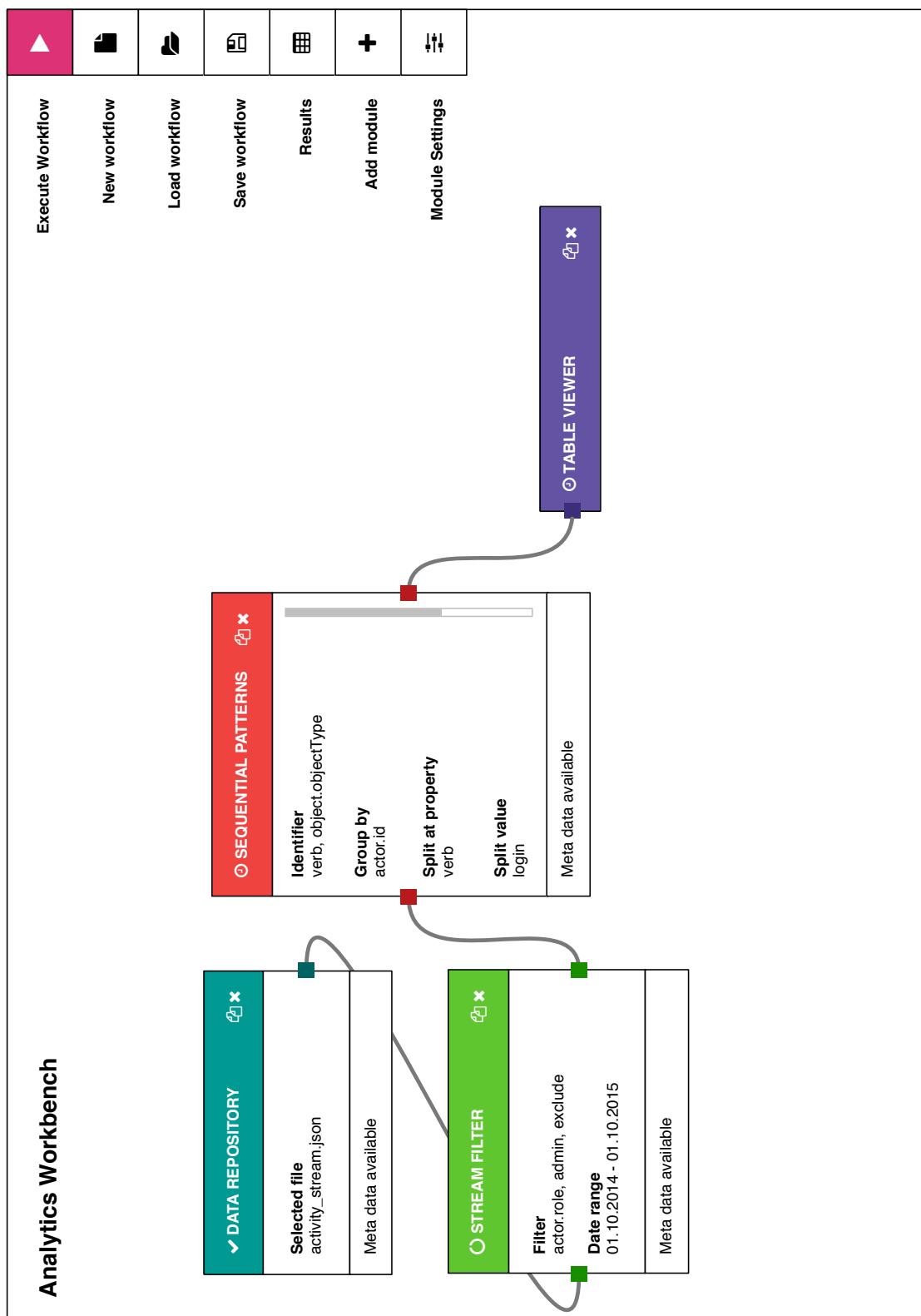


Abbildung 3.9.: Konzept für die Benutzeroberfläche der Workbench

3. Ansatz

Scrollbalken zugänglich gemacht (in der Abbildung im Sequential Pattern Mining Modul zu sehen). Ein weiteres Problem in der aktuellen Version der Repräsentation der Module innerhalb des Graphen ist, dass die Module optisch nur schwierig einzuordnen sind. Lediglich der Titel eines Moduls gibt Aufschluss darüber, um welche Art von Modul es sich handelt. In dem Konzept sind daher die Titelleisten der Module in Abhängigkeit ihrer Zugehörigkeit zu einer Kategorie gefärbt. Input-Module werden blau dargestellt, Tools grün, Analyse-Module rot und Ausgabe-Module violett. Dies kollidiert allerdings mit der Visualisierung des Modul-Status (durch eine entsprechende Färbung der Module) während der Durchführung eines Workflows in der aktuellen Version. Der Status eines Moduls soll in der neuen Version daher mit Icons in der Titelleiste dargestellt werden. In dem Konzept in Abbildung 3.9 sind diese Status in den Titelleisten der Module zu sehen. Die Verarbeitung des Data-Repository-Moduls ist hier bereits erfolgreich durchgeführt, was durch ein Häkchen-Symbol signalisiert wird. Das Activity Stream Filter Modul befindet sich in der Durchführung und zeigt eine Lade-Animation. Module, die auf die Durchführung warten, zeigen ein Uhr-Symbol, wie hier das Sequential Pattern Mining und Table Viewer Modul. Neben dem eigentlich Workflow werden im Hintergrund auch Analyse-Workflows zur Erzeugung der für die adaptive Konfiguration nötigen Meta-Daten durchgeführt. Dies passiert für jedes Input-Modul initial und bei Änderung der Eingabe-Daten. Um den Nutzer über den Status dieses Workflows zu informieren, wird in der Fußleiste von Input-Modulen eine ähnliche Symbolik verwendet. Wird aktuell ein Workflow für die Analyse der Daten eines Input-Moduls durchgeführt so ist in der Fußleiste eine Lade-Animation und der Hinweis *Analyzing input data* zu sehen. Eine erfolgreiche Analyse und das Vorhandensein von Meta-Daten wird durch den Hinweis *Meta data available* deutlich gemacht. Module, die ihre Meta-Daten (über die Meta-Ebene, siehe Abschnitt 3.2.4) von den Input-Modulen erhalten, übernehmen den Hinweis des entsprechenden Input-Moduls. Zentral für die Spezifikation des Workflow-Graphen, sind die Interaktionsmöglichkeiten zum Verbinden der Ein- und Ausgänge der Module. Diese werden links und rechts am Modul durch sogenannte Terminals dargestellt. Um ein bisher nicht verbundenes Terminal mit einem anderen zu verbinden, muss ein Terminal angeklickt und bei gedrückter Maustaste, der Zeiger Richtung des Ziel-Terminals gezogen werden. Dadurch wird eine neue Kante erstellt, die sich bis zum Loslassen über dem Ziel-Terminal an der Position des Mauszeigers anheftet. Um eine Kante zu entfernen, muss das betreffende Terminal angeklickt, und bei gedrückter Maustaste, der Zeiger auf einen leeren Bereich gezogen und losgelassen werden.

3.4.2. Konfiguration, Ergebnisse und Verwaltung

Die aktuelle Version der Workbench ist in vier Bereiche aufgeteilt (siehe Abbildung 2.7). Im Zentrum findet die Spezifikation des Workflows statt; links befindet sich eine Liste verfügbarer Module, die per Drag-and-Drop in den Workflow integriert werden können; rechts werden Informationen zum Status des Analyse-Prozesses und die Ergebnisse bereitgestellt; im oberen Bereich befindet sich eine Funktionsleiste, über die Workflows durchgeführt, geladen und gespeichert werden können. In der neuen Version der Workbench soll der Fokus ganz auf der Spezifikation des Workflows (wie in Abschnitt 3.4 beschrieben) liegen. Alle anderen Funktionen werden über eine Funktionsleiste rechts zugänglich gemacht (siehe Abbildung 3.9). Dazu gehören das Durchführen eines Workflows (*Execute workflow*), das Erstellen eines neuen Workflows (*New workflow*), das

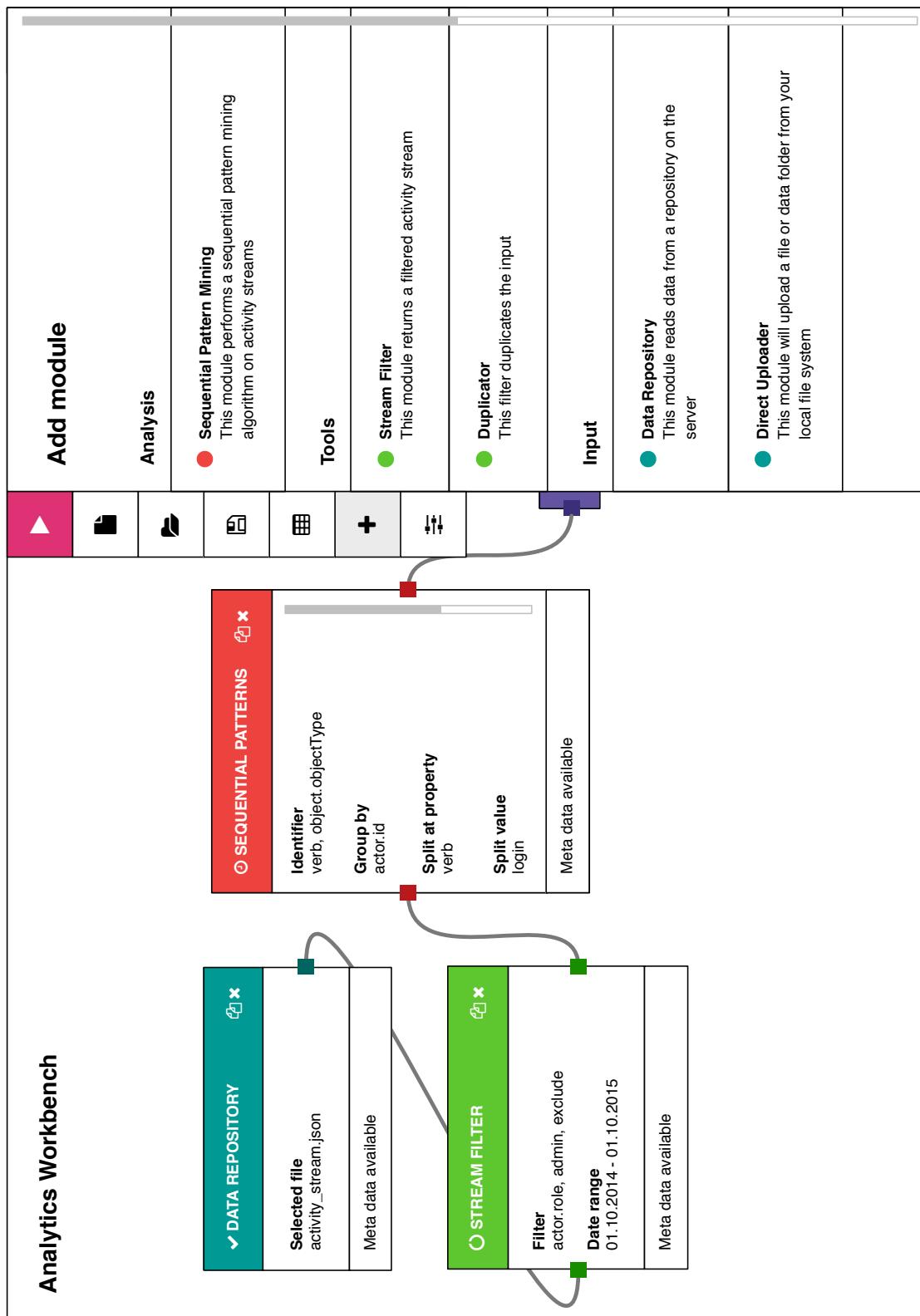


Abbildung 3.10.: Konzept für die Benutzeroberfläche der Workbench, Hinzufügen von Modulen

3. Ansatz

Laden und Speichern von Workflows (*Load/Save workflow*), die Zugang zu den Ergebnissen (*Results*), das Hinzufügen von Analyse-Modulen (*Add module*) und schließlich die Konfiguration der Module (*Module settings*). Die Funktionsleiste besteht aus Buttons, die mit einem Icon auf die jeweilige Funktion hinweisen. Zusätzlich wird beim Berühren eines Buttons mit dem Mauszeiger die Bezeichnung der Funktion sichtbar. Mit Ausnahme von *Execute workflow* und *New workflow*, öffnet sich beim Klick auf die Buttons ein entsprechender Bereich, in Form eines sogenannten *Drawer* (zu Deutsch Schublade), der von rechts eingeschoben wird und an den sich die Funktionsleiste anheftet. Über einen Klick in einen leeren Bereich des Workflow-Graphen kann der *Drawer* wieder geschlossen werden. Abbildung 3.10 zeigt den geöffneten Bereich zum Hinzufügen von Analyse-Modulen in den Workflow. Alle verfügbaren Module werden hier nach Kategorien sortiert aufgelistet und der Kategorie entsprechend farblich markiert. Per Klick auf ein Modul kann dieses zum Workflow hinzugefügt werden. Die übrigen Bereiche sollen nach einem ähnlichen Muster aufgebaut werden.

Ein Spezialfall ist der Bereich zur Modul-Konfiguration (*Module settings*). Die Konfiguration eines Moduls, kann erst angezeigt werden, wenn das Modul ausgewählt wurde. Das passiert über einen Klick auf das entsprechende Modul im Workflow-Graphen. Ist der *Drawer* geschlossen so öffnet sich dieser mit dem Klick auf ein Modul. Ein Modul, das ausgewählt wurde, wird im Workflow-Graphen optisch hervorgehoben; alle übrigen Module werden leicht ausgegraut (siehe Abbildung 3.11). In dem Bereich zur Modul-Konfiguration werden die in Abschnitt 3.3 erarbeiteten Benutzeroberflächen platziert. In Abbildung 3.11 ist das Sequential Pattern Mining Modul ausgewählt. Im Titel des *Drawer* ist die Bezeichnung des Moduls, sowie die farblich hinterlegte Kategorie und eine Kurzbeschreibung des Moduls zu finden. Darunter befindet sich schließlich die eigentliche Benutzeroberfläche zur Konfiguration des Moduls. Um Nutzern und insbesondere Entwicklern von Modulen, eine Einsicht in die am Modul eingehenden Meta-Daten zu geben, kann im Reiter unter der Kurbeschreibung die Ansicht von *Settings* auf *Meta Data* geändert werden. In der Ansicht, werden die Meta-Daten, sortiert nach den Modul-Eingängen in Form einer Baumstruktur dargestellt.

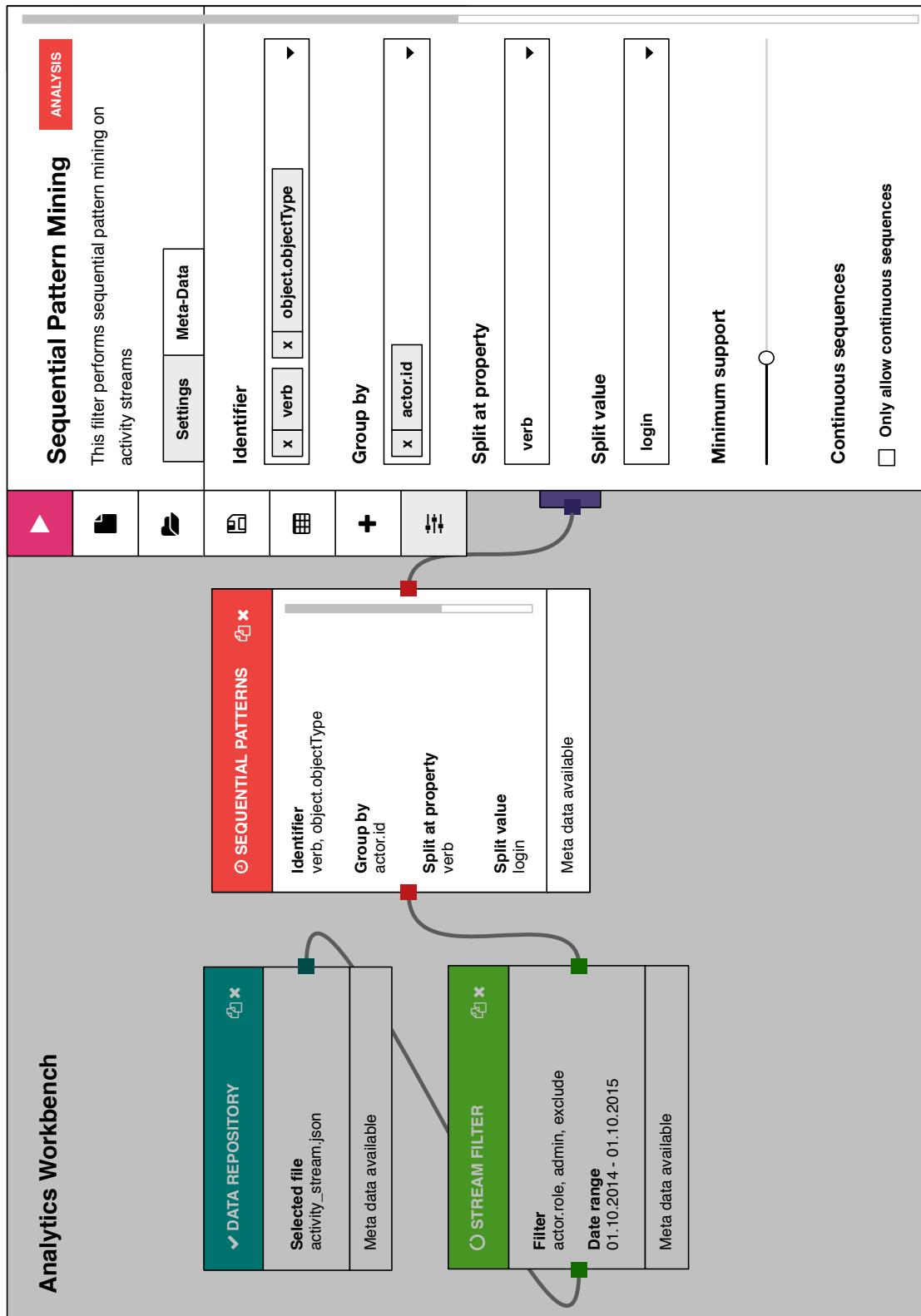


Abbildung 3.11.: Konzept für die Benutzeroberfläche der Workbench, Modul-Konfiguration

4. Implementierung

In diesem Kapitel wird die Implementierung des in Kapitel 3 erarbeiteten Ansatz vorgestellt. Abschnitt 4.1 gibt zunächst einen Überblick über die verwendeten Bibliotheken und Technologien. Darauf folgt in Abschnitt 4.1.1 eine Übersicht der Komponenten, aus denen sich die Anwendung zusammensetzt, sowie in Abschnitt 4.1.2 eine Übersicht der Infrastruktur zur Kommunikation zwischen Frontend und Server. In Abschnitt 4.1.3 wird die für die Durchführung eines Workflows implementierte Funktionalität vorgestellt. Im Anschluss daran wird in Abschnitt 4.1.4 die Umsetzung der adaptiven Konfiguration besprochen. Abschnitt 4.2 behandelt die Implementierung der wichtigsten Analyse-Module. Dazu gehören das Meta-Analyse Modul (Abschnitt 4.2.1), das Activity Stream Filter Modul (Abschnitt 4.2.2) und das Sequential Pattern Mining Modul (Abschnitt 4.2.3).

4.1. Frontend

Das Frontend der Analytics Workbench ist als Single-page Application unter Verwendung von React und Redux implementiert (siehe Abschnitt 2.4.1 und 2.4.2). Als Sprache wurde JavaScript Version 6¹ mit JSX-Erweiterung² verwendet. Da diese Kombination von den meisten Browsern nicht unterstützt wird, muss der Quelltext zuvor unter Verwendung eines Transpilers in die weitverbreitete JavaScript Version 5³ übersetzt werden. Dazu wurde der JavaScript-Module-Bundler Webpack⁴ verwendet. Mit Webpack ist es möglich JavaScript-Komponenten in isolierte Dateien auszulagern und diese bei Bedarf zu importieren. Die einzelnen Dateien werden beim Kompilieren des Bundles unter Verwendung des Babel⁵-Transpilers in gültiges JavaScript Version 5 umgewandelt. Ergebnis ist eine einzelne JavaScript-Datei, die in ein HTML-Dokument eingebunden werden kann. Webpack unterstützt außerdem CSS-Module⁶. CSS-Module erlauben die Verwendung von lokalen CSS. Dabei können die Namen von CSS-Klassen aus CSS-Dateien importiert und in JavaScript-Komponenten verwendet werden.

4.1.1. Komponenten-Hierarchie

Unter Verwendung von React lassen sich die Bestandteile des Frontends in eine Hierarchie von React-Komponenten überführen. Jede Komponente ist dabei eine isolierte, wiederverwendbare Einheit. Abbildung 4.1 zeigt die Architektur des Frontends und die auf dem Konzept aus Abbildung 3.10 aufbauende Komponenten-Hierarchie (blau

¹<http://www.ecma-international.org/ecma-262/6.0/>, Abruf: 06.07.2016

²<https://facebook.github.io/jsx/>, Abruf: 06.07.2016

³<http://www.ecma-international.org/ecma-262/5.1/>, Abruf: 06.07.2016

⁴<https://webpack.github.io/>, Abruf: 06.07.2016

⁵<http://babeljs.io>, Abruf: 06.07.2016

⁶<https://github.com/css-modules/css-modules>, Abruf: 06.07.2016

4. Implementierung

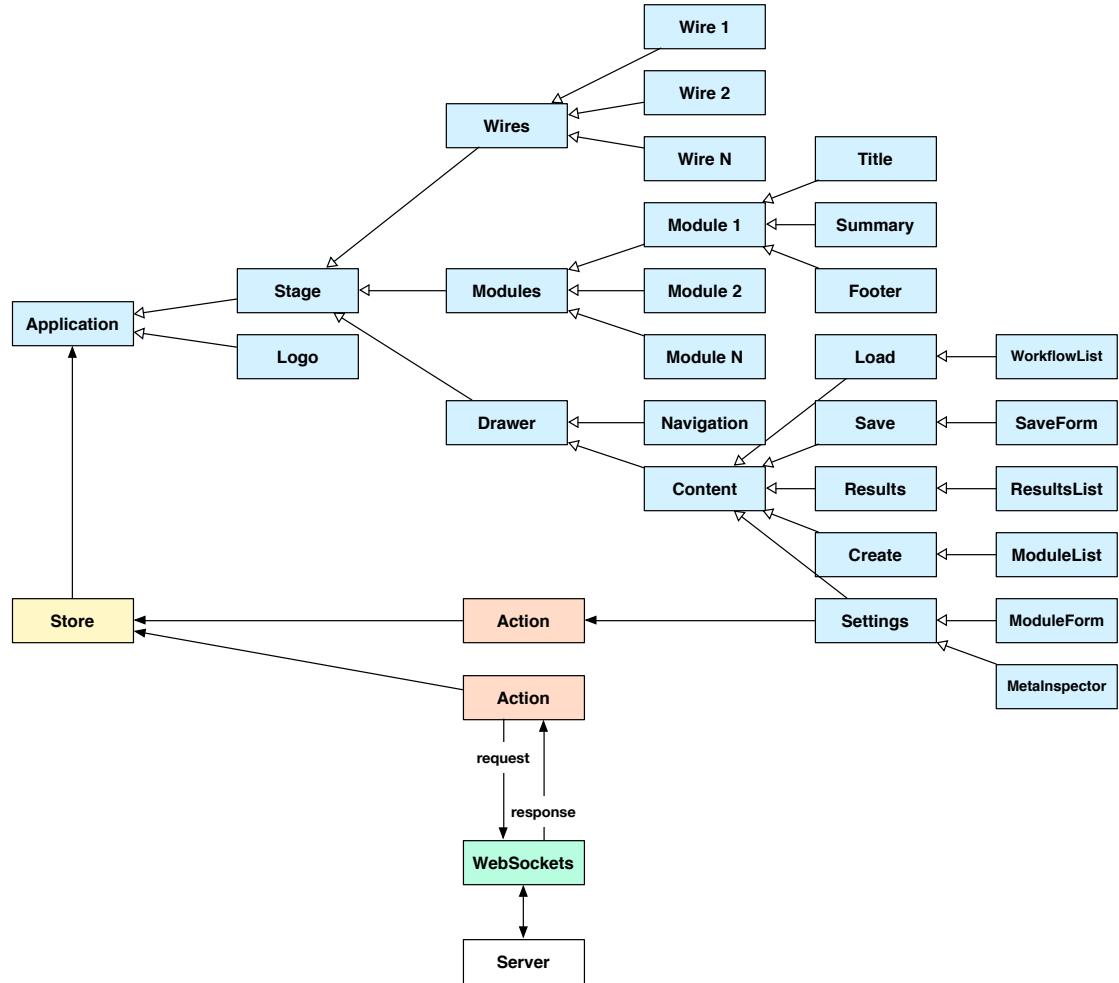


Abbildung 4.1.: Architektur des Frontends

gefärbt). Die **Application**-Komponente fungiert als Wurzel und enthält die **Logo**- und **Stage**-Komponente. Die **Stage**-Komponente ist das Herz der Anwendung. Hier findet die Visualisierung des Workflow-Graphen statt, die sich durch die **Wires**- und die **Modules**-Komponenten zusammenstellt. Die **Wire**-Komponenten rendern die Kanten des Workflow-Graphen, in Form von SVG-Elementen⁷, mit denen es möglich ist, grafische Elemente, wie Pfade, Polygone oder Kreise innerhalb einer Webanwendung darzustellen. Dazu erhalten die Komponenten Informationen über die Position von Start- und Ziel-Terminal der verbundenen Module. Die Modul-Komponenten rendern die Knoten des Workflow-Graphen, also die Modul-Repräsentationen. Diese stellen sich aus einer Titelleiste (**Title**), der Zusammenfassung der Modul-Konfiguration (**Summary**) und der Fußleiste (**Footer**), in der Information zu den Meta-Daten angezeigt werden, zusammen. Die **Stage**-Komponente enthält darüber hinaus auch den **Drawer**. Der **Drawer** ist der ausfahrbare Bereich zur Durchführung und Verwaltung der Workflows, Einsicht in die Ergebnisse und Konfiguration der Module (siehe Abschnitt 3.4.2). Der **Drawer** rendern die Funktionsleiste (**Navigation**), sowie den ausgewählten Bereich (**Content**). Jeder Bereich ist in

⁷<https://www.w3.org/Graphics/SVG/>, Abruf: 06.07.2016

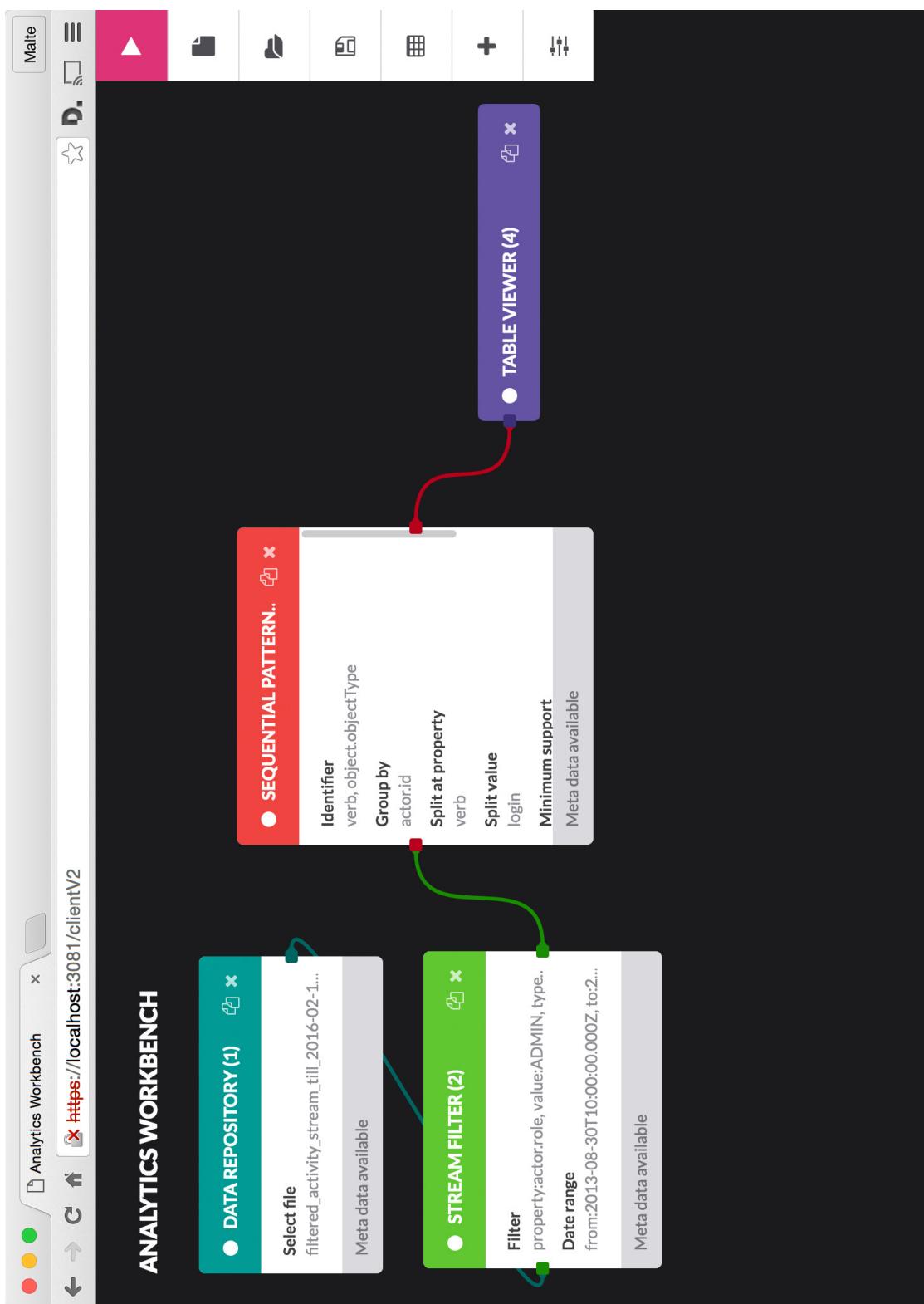


Abbildung 4.2.: Neue Benutzeroberfläche der Workbench

4. Implementierung

einer eigenen Komponente implementiert. Die Ansicht zum Laden von Workflows (**Load**) stellt eine Liste an verfügbaren Workflows (**WorkflowList**) bereit, die in den Arbeitsbereich geladen werden können. Für das Speichern von Workflows (**Save**) wird in dem **Drawer** ein Formular, zur Eingabe von Titel und Beschreibung (**SaveForm**) angezeigt. Für den Zugang zu den Ergebnissen eines Workflows stellt die **Results**-Komponente eine entsprechende Liste bereit (**ResultsList**). Das Hinzufügen von Modulen ermöglicht die **Create**-Komponente, die eine Liste (**ModuleList**) aller verfügbaren Module bereitstellt. Die **Settings**-Komponente stellt die Ansicht für die Konfiguration der Module. Dazu gehören die **ModuleForm**-Komponente, die auf Grundlage der *Filter Description* und Meta-Daten, eine Konfiguration des ausgewählten Moduls erlaubt. Die Inspektion der Meta-Daten erlaubt die **MetaInspector**-Komponente. Die beschriebenen Komponenten-Hierarchie bildet die Oberfläche der neuen Version des Frontends. Diese ist in Abbildung 4.2 zu sehen.

4.1.2. Datenfluss & Kommunikation

Infrastruktur und Daten-Modell wurden unter Verwendung der Redux-Bibliothek implementiert. Wie in Abschnitt 2.4.2 beschrieben, besteht eine solche Architektur aus einer zentralen Datenbasis dem *Store*; Funktionen, die Aktionen beschreiben, den *Actions* und Funktionen, die auf Grundlage der *Actions*, Änderungen in den Daten vornehmen, den sogenannten *Reducers*. Komponenten beziehen ihre Daten aus dem *Store*. Um Änderungen in den Daten zu initiieren, z.B. um ein Modul dem Workflow hinzuzufügen, kann eine Komponente eine entsprechende *Action* am *Store* ausführen. Auf Grundlage der *Action* und der in den *Reducers* implementierten Funktionalität ändern sich die Daten. Der *Store* informiert die Komponenten über die Änderung und diese rendern sich schließlich mit den neuen Daten. So wird ein unidirektonaler Datenfluss sichergestellt. In Abbildung 4.1 geht ein Pfeil aus dem *Store* zur **Application**-Komponente; diese Darstellung ist vereinfacht. Prinzipiell könnte jede Komponente auf den *Store* zugreifen um Daten abzurufen oder *Actions* auszuführen. Die Struktur der in dem *Store* gehaltenen Daten ergibt sich durch die *Reducer*. Diese lassen sich beliebig verschachteln. In der Analytics Workbench wurden drei Reducer implementiert, die für verschiedene Domänen verantwortlich sind. Der Filter-Reducer ist für die Verwaltung der vom Server bereitgestellten *Filter Descriptions* verantwortlich, der User-Reducer ist für die Verwaltung nutzerbezogener Daten und der *Workflow*-Reducer für alle Daten, die mit Workflows in Verbindung stehen.

Die Kommunikation zwischen Server und Frontend geschieht über das WebSocket-Protokoll. Die serverseitige Implementierung musste dazu nicht angepasst werden. Auf Seiten des Frontends wurde dafür, wie in der bisherigen Version, die Socket.io-Bibliothek⁸ genutzt. Der *Store* wurde dazu um eine *Request-Response*- und *Receive*-Funktionalität erweitert. Abbildung 4.3 zeigt exemplarisch die Implementierung von *Action* und *Reducer* zum Abfragen der *Filter Description* vom Server. Innerhalb der *Actions* kann zur Durchführung von Anfragen an den Server über das WebSocket-Protokoll die Funktion *request* benutzt werden. Um Zugriff auf diese Funktion zu erhalten, muss eine *Action* anstelle einer Beschreibung (simples JavaScript-Objekt) eine Funktion zurückgeben. Diese Funktion wird beim Ausführen der *Action* aufgerufen und erhält als Parameter

⁸<http://socket.io/>, Abruf: 06.07.2016

```

1 // Single action
2 function fetch() {
3   return ({ dispatch, request }) => {
4     dispatch({
5       type: 'FILTERS_FETCH',
6       payload: request({ command: 'getFilterDescriptions' })
7     });
8   };
9 }
10
11 // Reducer
12 function reducer(state, action) {
13   switch (action.type) {
14     case 'FILTERS_FETCH': {
15       return {
16         ...state,
17         fetching: true
18       };
19     }
20     case 'FILTERS_FETCH_SUCCESS': {
21       return {
22         ...state,
23         fetching: false,
24         byId: indexFilters(action.payload)
25       };
26     }
27     case 'FILTERS_FETCH_ERROR': {
28       return {
29         ...state,
30         fetching: false,
31         fetchingError: action.payload
32       };
33     }
34   }
35 }

```

Abbildung 4.3.: React-Komponenten

Referenzen zu weiteren Funktionen, wie der `dispatch`-Funktion, mit der weitere *Actions* ausgeführt werden können oder eben der `request`-Funktion (Zeile 3). Die `request`-Funktion liefert ein *Promise*⁹ zurück, das wiederum von der *Action* im `payload` Parameter zurückgegeben wird. *Actions*, die in diesem Parameter ein *Promise* zurückgeben, werden anders behandelt als herkömmliche *Actions*. *Promises* erlauben asynchrone Zustände zu transportieren. Bei einem erfolgreich durchgeföhrten asynchronen Vorgang, wie der Anfrage an einen Server, wird das *Promise* aufgelöst (*resolve*). Sollte ein Fehler auftreten sein, wird das *Promise* zurückgewiesen (*reject*). Bei der Verwendung von *Promises* in *Actions* wird bei einem erfolgreich aufgelösten *Promise* eine weitere *Action* ausgeführt, die im `type`-Parameter den Wert der eigentlichen Action (z.B. `FILTERS_FETCH`) und dem Suffix `SUCCESS` hält, sodass eine *Action* `FILTERS_FETCH_SUCCESS` entsteht. Die Antwort des Server wird im `payload`-Parameter dieser *Action* transportiert. Bei einem

⁹https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Promise, Abruf: 06.07.2016

4. Implementierung

zurückgewiesenen *Promise* wird eine Aktion ausgeführt, die im `type`-Parameter den Suffix `ERROR` verwendet und im `payload`-Parameter eine Beschreibung des Fehlers. In Zeile 6 wird nun die `request`-Funktion mit `getFilterDescriptions` als `command` ausgeführt. Dadurch wird eine Anfrage an den Server gesendet. Gleichzeitig wird die `Action FILTERS_FETCH` ausgeführt. Der *Reducer* kann nun in den Daten vermerken, dass aktuell eine Anfrage an den Server läuft, indem das Feld `fetching` den Wert `true` erhält (Zeile 14). Eine React-Komponente könnte nun eine Lade-Animation anzeigen. Die Anfrage an den Server wird mit einer `requestId` versehen, die der Server beim Antworten wieder mitliefert. So ist es möglich, die Antwort einer Anfrage zuzuordnen. Kommt eine erfolgreiche Antwort vom Server zurück, so wird die `FETCH_FILTERS_SUCCESS Action` mit den entsprechenden Daten ausgeführt. Der *Reducer* setzt das Feld `fetching` auf `false` und indiziert die vom Server erhaltenen *Filter Descriptions*. Sollte bei der Anfrage ein Fehler aufgetreten sein so wird eine `FETCH_FILTERS_ERROR Action` ausgeführt. Der `payload` dieser *Action* ist ein *Error*-Objekt, das Aufschluss über den Fehler gibt. Neben dieser *Request-Response*-Funktionalität, ist es möglich, Benachrichtigungen, die (ohne Anfrage) vom Server erfolgen, z.B. wenn Agenten bei der Durchführung eines Workflows ihren Status ändern, werden mit Hilfe einer *Action-Map* in *Actions* umgewandelt. Für jedes Event, das vom Server abgegeben wird, kann in dieser Map eine entsprechende *Action* angegeben werden.

4.1.3. Workflow-Durchführung

Die Durchführung eines Workflows ist, bedingt durch die Multiagenten-Architektur der Analytics Workbench etwas komplexer. Abbildung 4.4 zeigt den zeitlichen Ablauf und die nötigen Schritte, vom Start der Durchführung hin zum Abruf der Ergebnisse. Sofern der Workflow bisher nicht gespeichert wurde, geschieht dies zunächst innerhalb der `WORKFLOW_AUTOSAVE Action`. Dabei wird der Workflow-Graph, der sich aus den Modul-Konfigurationen (`modules`) und den Kanten (`wires`) zusammensetzt, aus dem *Store* geholt und mit einem `saveWiring`-Befehl an den Server geschickt. Dieser antwortet mit einer `saveId`, die später dazu verwendet werden kann, dem gespeicherten Workflow, Ergebnisse zuzuordnen. Im nächsten Schritt findet mit der `WORKFLOW_EXECUTE Action` die Initiierung der Durchführung statt. Dazu wird der Workflow in ein von dem Server vorgegebenes Format übersetzt und mit dem `executeWiring`-Befehl an den Server geschickt. Dieser antwortet mit einer `runId`, die dazu benötigt wird, die Ergebnisse später aus dem Result-Repository zu erhalten. Wurde das Direct-Uploader-Modul verwendet, so müssen die ausgewählten Dateien in einem weiteren Schritt zum Server hochgeladen werden. Dazu steht der `handleDataUpload`-Befehl zur Verfügung. Serverseitig läuft nun die sukzessive Abarbeitung der einzelnen Module ab. Für jedes Modul wird dazu ein Agent gestartet, der die eingehenden Daten verarbeitet und die Ergebnisse den folgenden Agenten bereitstellt. Über die Kommunikationsebene (siehe Abschnitt 2.3.2) informieren die Agenten den Server über ihren Status. Der Server leitet diese Events (über die WebSocket-Schnittstelle) an das Frontend weiter. Die Events werden über die oben beschriebenen *ActionMap* in entsprechende *Actions* umgewandelt. In diesem Fall werden `WORKFLOW_SET_AGENT_STATUS Actions` erzeugt. Diese führen dazu, dass im *Store* für jedes Modul vermerkt wird, in welchem Zustand sich der dazugehörige Agent befindet. In den Repräsentationen der Module werden nun die Status-abhängigen Icons angezeigt (siehe Abbildung 4.5). Sobald alle Agenten mit der Durchführung fertig sind und die Ergebnisse

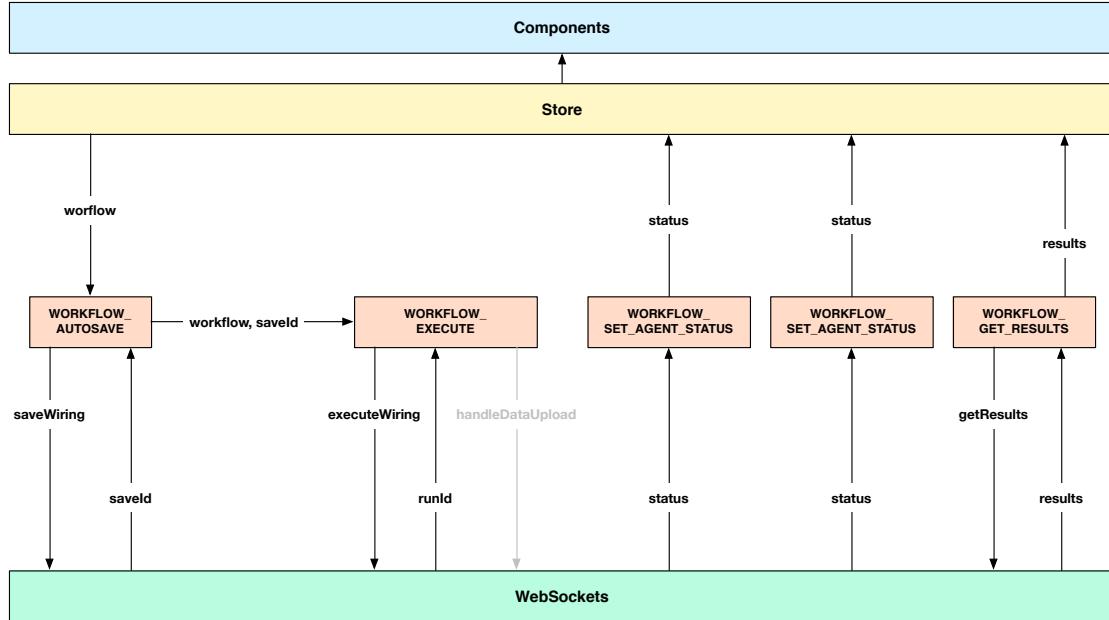


Abbildung 4.4.: Ablauf einer Workflow-Durchfhrung

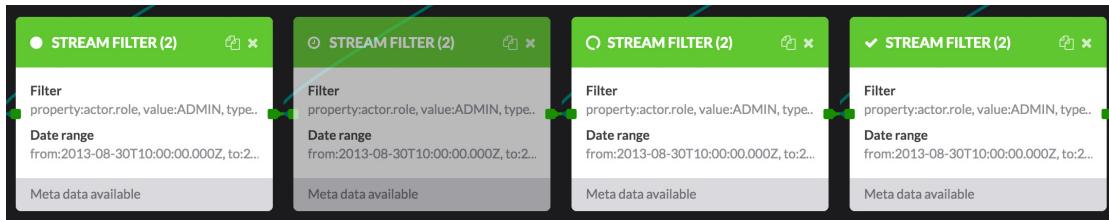


Abbildung 4.5.: Module-Status, von links nach rechts: Standard, Warten, Durchfuhren, Fertig

im Result-Repository zur Verfügung stehen, wird innerhalb der `WORKFLOW_GET_RESULTS` *Action* der `getResults`-Befehl mit der `runId` an den Server geschickt. Der Server liefert eine Liste mit Links zu den Ergebnissen, die schließlich im Frontend angezeigt wird und von dem Nutzer geöffnet oder heruntergeladen werden können.

4.1.4. Adaptive Konfiguration

Für die Implementierung der adaptiven Konfiguration werden verschiedene Komponenten benötigt. Grundlage sind Meta-Daten, die Informationen über die Beschaffenheit der Eingabe-Daten transportieren. Um für jedes Modul gültige Meta-Daten erzeugen zu können, müssen Module beschreiben, wie sie ihre Daten und somit auch Meta-Daten transformieren (siehe Abschnitt 3.2.4). Dies geschieht mit der Meta-Daten-Transformationsfunktion. Um schließlich für die Benutzeroberfläche eines Moduls gültige Meta-Daten zu erhalten, wird Algorithmus 1 beim Rendern einer Benutzeroberfläche herangezogen. Welche Eingabe-Felder in einer Benutzeroberfläche

4. Implementierung

```

1 // Ausschnitt der Oberflächen-Beschreibung
2 {
3     "input_encoding": {
4         "rank": 0,
5         "type": "AutoComplete",
6         "label": "Input for analysis",
7         "multi": true,
8         "default": ["verb", "object.objectType"],
9         "required": true,
10        "options": [
11            "published",
12            "actor.id",
13            "actor.objectType",
14            "verb",
15            "object.id",
16            "object.objectType"
17        ]
18    }
19 }
20
21 // Oberflächen-Transformationsfunktion (vereinfacht)
22 function transform(form, state, metaByInput) {
23     var input = metaByInput.in_1;
24     var fields = input.fields;
25     return merge(form, {
26         input_encoding: {
27             options: fields
28         }
29     });
30 }
```

Abbildung 4.6.: Oberflächenbeschreibung und Oberflächen-Transformationsfunktion des Sequential Pattern Mining Moduls

zur Verfügung stehen, wird vom Modul in der Oberflächenbeschreibung definiert. Diese Oberflächenbeschreibung kann auf Grundlage der Meta-Daten mit der Oberflächen-Transformationsfunktion verändert werden. Die Implementierung der Adaptation wird also durch diese Funktion gestellt. Meta-Daten-Transformationsfunktion, Oberflächenbeschreibung und die Oberflächen-Transformationsfunktion müssen von Modulen in der *Filter Description* bereitgestellt werden.

Für die Erzeugung der Meta-Daten wurde das in Abschnitt 3.2.3 beschriebene Konzept implementiert. Meta-Daten müssen immer dann erzeugt werden, wenn sich die Konfiguration eines Input-Moduls ändert. Dazu werden die Konfigurationen der Input-Modul im *Store* beobachtet. Sobald dort eine Änderung festgestellt wurde, wird für das entsprechende Input-Modul ein Meta-Analyse-Workflow angestoßen. Dieser Workflow läuft wie die herkömmlichen Workflows (siehe Abbildung 4.4) ab; mit Ausnahme der automatischen Speicherung. Wie in Abschnitt 3.2.3 beschrieben, besteht dieser Workflow aus dem Input-Modul und einem Meta-Analyse-Modul, das die Eingabe-Daten entgegennimmt und analysiert. Die Implementierung dieses Moduls wird in Abschnitt 4.2.1 erläutert. Sobald der Workflow fertig ist, werden die daraus erzeugten Meta-Daten aus dem Result-Repository geladen und in dem *Store* gespeichert. Sobald Meta-Daten zur

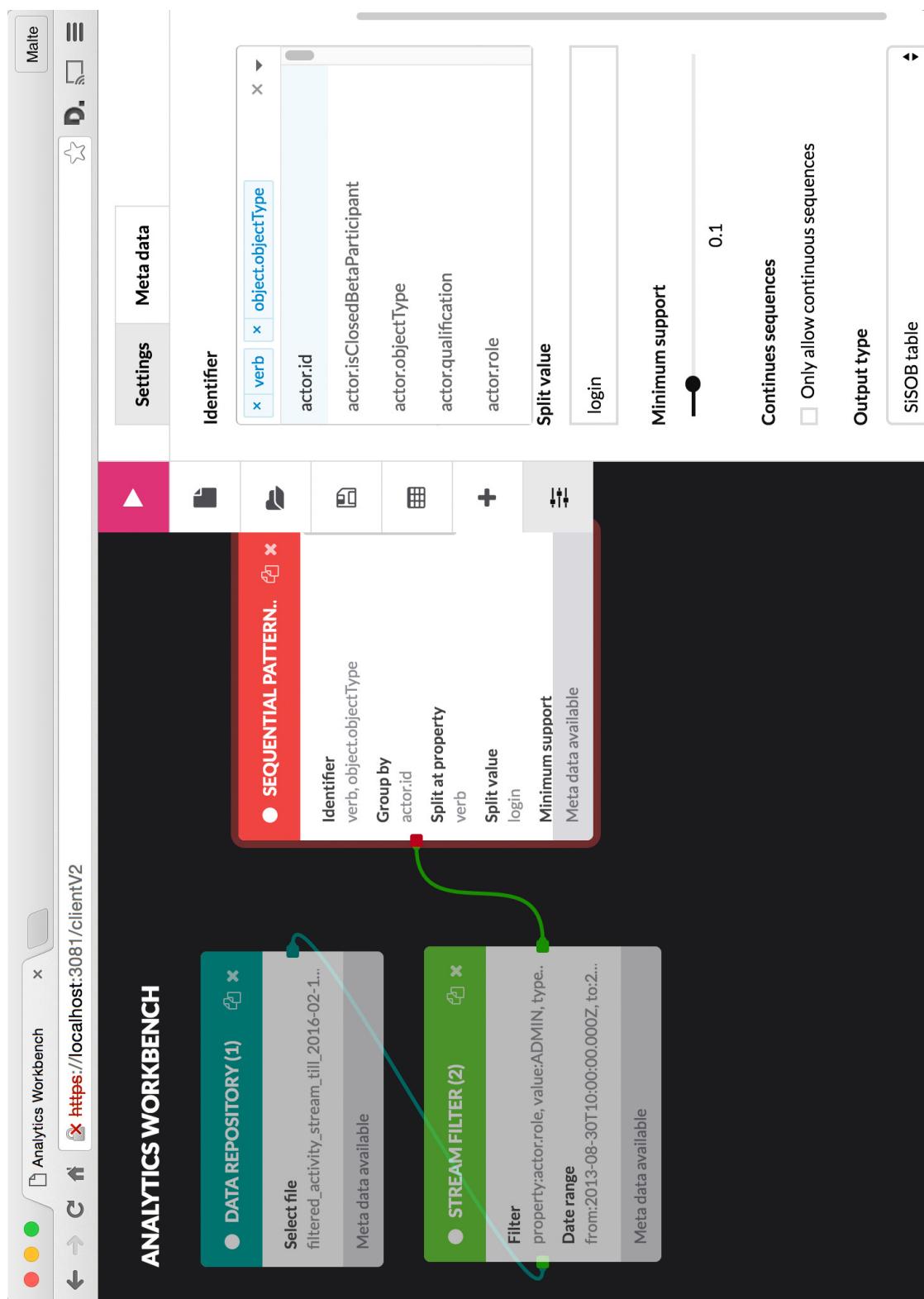


Abbildung 4.7.: Adaptive Konfiguration

4. Implementierung

Verfügung stehen, können sich die Benutzeroberflächen der Module adaptieren. Die Benutzeroberflächen werden von einer React-Komponente gerendert, die in der Lage ist, die Oberflächen-Beschreibung der Module in tatsächliche Eingabe-Felder zu überführen. Abbildung 4.6 zeigt die Oberflächenbeschreibung und Oberflächen-Transformationsfunktion des Sequential Pattern Mining Moduls. Beim Rendern wird nun die Oberflächen-Transformationsfunktion mit der initialen Oberflächenbeschreibung (`form`), den aktuellen Werten der Eingabe-Feldern (`state`) und den mit Algorithmus 1 abgeleiteten Meta-Daten (`metaByInput`) ausgeführt. Im Fall des Sequential Pattern Mining Moduls aus Abbildung 4.6, werden die Auswahlmöglichkeiten der Auswahlbox `input_encoding` mit den aus den Meta-Daten hervorgehenden Felder der Activity Stream-Einträge ersetzt (Zeile 27). Das Ergebnis ist eine adaptierte Oberflächenbeschreibung, die schließlich von der React-Komponente gerendert wird. Abbildung 4.7 zeigt die adaptierte Benutzeroberfläche des Sequential Pattern Mining Moduls, das nun in der Lage ist, auf Grundlage der Meta-Daten dem Nutzer Vorschläge zu machen. Die Syntax der bisherigen Oberflächenbeschreibung musste erweitert werden, um auch Eingabe-Felder mit Auto vervollständigungs-Funktionalität oder der in Abschnitt 3.3.1 beschrieben Tabelle zu unterstützen.

4.2. Analyse-Module

Serverseitig konnte auf die bisherige Infrastruktur der Analytics Workbench zurückgegriffen werden. An der Infrastruktur selber waren keine Änderungen notwendig. In der neuen Version der Workbench müssen Analyse-Module jedoch in der *Filter Description* neben der Oberflächenbeschreibung, eine Oberflächen-Transformationsfunktion und eine Meta-Daten-Transformationsfunktion bereitzustellen. Beide Funktionen müssen in der im Frontend verwendeten Sprache JavaScript implementiert werden. Um die Funktionen in der *Filter Description* bereitzustellen, musste die dafür verantwortliche Klasse des Agenten Frameworks `eu.sisob.components.framework.componentdescription.Container` um die beiden Felder erweitert werden. Die Implementierung beider Funktionen wurde für alle (in dieser Version der Workbench verfügbaren) Analyse-Module vorgenommen.

Für die Analyse der Eingabe-Daten musste ein Meta-Analyse Modul implementiert werden. Die Implementierung des Moduls wird in Abschnitt 4.2.1 behandelt. Die Erweiterung des Funktionsumfangs vom Activity Stream Filter und Sequential Pattern Mining Modul erforderte umfangreiche Änderungen in den jeweiligen Modulen, die in Abschnitt 4.2.2 und 4.2.3 erläutert werden.

4.2.1. Meta-Analyse Modul

Das Meta-Analyse Modul ist dafür verantwortlich, Eingabe-Daten aus Input-Modul zu analysieren und Meta-Daten zu erzeugen. Das Modul ist ein vollwertiges Analyse-Modul, das unter Verwendung des Agenten-Framework der Analytics Workbench implementiert wurde. Da das Modul nicht für eine herkömmliche Analyse geeignet ist, wird es Nutzern der Workbench im Frontend nicht zur Verfügung gestellt. Der für die Analyse verantwortliche Agent des Moduls analysiert sukzessive die am Modul eingehenden Daten. Meta-Daten bestehen minimal aus einem Feld `fileType`, das den Dateityp definiert

```

1  {
2      "fileType": "json",
3      "dataType": "ActivityStream",
4      "values": {
5          "actor.qualification": ["ARZT", "HAUSARZT", /* ... */],
6          "actor.role": ["MENTOR", "ADMIN", "TEAM_MEMBER", /* ... */],
7          "actor.state": ["BE", "BW", "BB", "MV", "BY", "HH", /* ... */],
8          "actor.isClosedBetaParticipant": [true, false],
9          "object.action": ["edit", "create", "delete", "action"],
10         "object.objectType": ["view", "case", "article", /* ... */ ],
11         /* ... */
12     },
13     "fields": [
14         "actor.qualification",
15         "actor.role",
16         "actor.state",
17         "actor.isClosedBetaParticipant",
18         "object.action",
19         "object.objectType",
20         /* ... */
21     ],
22     "dateRange": ["2013-08-30T14:06:58Z", "2016-02-08T23:22:07Z"]
23 }

```

Abbildung 4.8.: Ausschnitt des Ergebnisses einer Meta-Daten-Analyse

und einem Feld `dataType`, das den Datentyp definiert. Zunächst wird in der Methode `analyseInput` der Dateityp festgestellt und in Abhängigkeit davon, die weitere Analyse durchgeführt. In der derzeitigen Implementierung ist das Modul nur in der Lage, JSON Activity Streams zu erkennen. Die Funktionalität kann aber recht einfach durch die Implementierung weiterer Analyse-Methoden erweitert werden. Wurde die Datei als JSON-Datei identifiziert, so wird die Methode `analyseJSON` herangezogen; sollte die Datei in einem anderen Format vorliegen, so endet der Analyse-Prozess. In diesem Fall wird in den Meta-Daten als Dateityp die Endung der Datei und als Datentyp `Unknown` angegeben. Die Methode `analyseJSON` wandelt die rohen Daten nun in eine entsprechende JSON-Repräsentation um, die schließlich auf verschiedene Merkmale hin untersucht wird. Activity Stream Einträge müssen in einem Array im Feld mit dem Namen `items` gehalten werden. Ist dies in den Eingabe-Daten der Fall, so wird nun stichprobenartig ein Eintrag aus dem Array auf weitere Felder hin untersucht. Activity Stream Einträge haben mindestens ein Feld `actor` und ein Feld `published` [SAN12]. Sind beide Felder in der Stichprobe vorhanden, so wird davon ausgegangen, dass es sich um einen Activity Stream handelt. Nun müssen die Charakteristika aus dem Activity Stream extrahiert werden. Das geschieht in der Methode `analyseJSONActivityStream`. Wie in Abschnitt 3.3 beschrieben, sollen alle in den Einträgen vorhandenen Felder und dazugehörige Werte gesammelt werden. Außerdem wird die Zeitspanne zwischen ältesten und neuestem Eintrag festgehalten. Dazu wird die gesamte Menge der Einträge durchlaufen und einzeln analysiert. Activity Stream Einträge liegen in einer baumartigen Struktur vor. Dieser Baum wird nun mit einem *depth-first*-Ansatz durchlaufen. Die Namen der Felder einer tieferen Ebene setzen sich aus den Namen der Eltern-Knoten, z.B. `actor`, und dem Namen des eigentlichen Feldes, z.B. `role`, zusammen, sodass die Bezeichnung `actor.role`

4. Implementierung

entsteht. Alle beobachteten Felder werden in den Meta-Daten im Feld `fields` gehalten. Die dazugehörigen Werte in dem Feld `values`; die Zeitspanne wird in einem Array im Feld `dateRange` gehalten. Abbildung: 4.8 zeigt das Ergebnis einer Analyse.

4.2.2. Activity Stream Filter Modul

Das Activity Stream Filter Modul musste umfangreich reimplementiert werden. Das Modul ist nun in der Lage, eine beliebige Anzahl an Filtern auf die Daten anzuwenden. Außerdem ist es möglich, Einträge zu filtern, die sich außerhalb eines definierten Zeitraums befinden. Für die Konfiguration dieser Funktionen musste für das Frontend die in Abschnitt 3.3.1 konzipierte Filter-Tabelle und der Date Picker implementiert werden. Diese können nun in der Oberflächenbeschreibung der Module genutzt werden. Abbildung 4.10 zeigt die Oberflächenbeschreibung und Oberflächen-Transformationsfunktion des Moduls. Die Adaption der Benutzeroberfläche umfasst die Erweiterung der Auswahlmöglichkeiten der Felder `property` und `value` der Filter-Tabelle (in Zeile 36-39 zu sehen) sowie die Einschränkung des Zeitraums des Date-Pickers (Zeile 45ff).

Serverseitig wurde die Filter-Funktionalität des Agenten reimplementiert. Die Verarbeitung der Daten läuft nun wie folgt ab: Der Agent beginnt mit einem leeren Array, das sukzessive mit Activity Einträgen gefüllt wird. Dazu wird die gesamte Menge an Einträgen durchlaufen und einzeln geprüft, ob der jeweilige Eintrag ein- oder ausgeschlossen wird. Der erste Test überprüft, ob sich ein Eintrag in dem festgelegten Zeitraum befindet. Dies geschieht in der Methode `matchesDateRange`. Danach werden alle in der Modul-Konfiguration angegebene Filter durchlaufen. Mit der Methode `matchesFilter` wird für jeden Filter geprüft, ob der Eintrag den Bedingungen des Filters standhält. Abbildung 4.9 zeigt eine vereinfachte Version der Implementierung dieser Methode. Ein Filter setzt sich aus drei Komponenten zusammen: Das betreffende Feld `property`, der Wert des Feldes `value` und der Modus des Filters `mode` (ein- oder ausschließen). In Zeile 6 wird zunächst der Wert des angegebenen Feldes des Eintrags geholt. Durch die Baumstruktur der Activity Stream Einträge, musste die Funktion `getNestedProperty` implementiert werden. Die Funktion kann unter Angabe eines Pfades, z.B. `actor.role`, den Wert des Feldes aus einem verschachtelten JSONObject extrahieren. Danach wird in Zeile 7 geprüft, ob der Wert des Feldes dem im Filter angegebenen Wert gleich. Beide Werte

```
1 private boolean matchesFilter(JSONObject item, JSONObject filter) {
2     String property = filter.get("property").toString();
3     String value = filter.get("value").toString();
4     String mode = filter.get("mode").toString();
5
6     String itemValue = this.getNestedProperty(item, property);
7     boolean result = itemValue.equalsIgnoreCase(value);
8
9     if (mode.equals("include")) return result;
10    else return !result;
11 }
```

Abbildung 4.9.: Vereinfachte Implementierung der matchesFilter-Funktion

```

1 // Ausschnitt der Oberflaechen-Beschreibung
2 {
3   "filters": {
4     "type": "Table",
5     "columns": {
6       "property": {
7         "type": "AutoComplete",
8         "options": [
9           "actor.id",
10          "actor.objectType",
11          /* ... */
12        ]
13      },
14      "values": {
15        "type": "AutoComplete",
16        /* ... */
17      },
18      "model": {
19        "type": "AutoComplete",
20        "options": ["include", "exclude"],
21        /* ... */
22      }
23    },
24  },
25  "dateRange": {
26    "type": "DateRange",
27  }
28 }
29
30 // Oberflaechen-Transformationsfunktion (vereinfacht)
31 function transform(form, state, metaByInput) {
32   var input = metaByInput.in_1;
33   return merge(form, {
34     filters: {
35       columns: {
36         property: {
37           options: input.fields
38         },
39         values: {
40           options: input.values
41         }
42       }
43     },
44     dateRange: {
45       min: input.dateRange[0],
46       max: input.dateRange[1]
47     }
48   });
49 }

```

Abbildung 4.10.: Oberflächenbeschreibung und Oberflächen-Transformationsfunktion des Activity Stream Filter Moduls

4. Implementierung

```
1  String generateInputForAnalysis(String[] fields, String splitAtProperty,
2                                String splitAtValue) {
3
4      StringBuilder fileBuilder = new StringBuilder();
5      for (String groupId : items.keySet()) {
6          List<EventLog> itemList = items.get(groupId);
7
8          for (EventLog item : itemList) {
9              int identifier = translateItemToInteger(item, fields);
10
11             String value = getNestedProperty(splitAtProperty, item);
12             boolean split = value.equalsIgnoreCase(splitAtValue);
13
14             if (split) {
15                 fileBuilder.append("-2");
16                 fileBuilder.append("\r\n");
17             }
18
19             fileBuilder.append(identifier);
20             fileBuilder.append(" -1 ");
21         }
22     }
23     return fileBuilder.toString();
}
```

Abbildung 4.11.: Implementierung der Übersetzung von Activity Stream Einträgen in eine Sequenz-Datenbank (vereinfacht)

liegen als Strings vor, sodass auch ein Abgleich von booleschen Werten oder Zahlen kein Problem ist. Abhängig vom Modus (ein- oder ausschließen) gibt die Funktion schließlich ein `true` oder `false` zurück (Zeile 9ff). Das Ergebnis der Verarbeitung ist eine gefilterte Menge an Activity Stream Einträgen.

4.2.3. Sequential Pattern Mining Modul

Das Sequential Pattern Mining Modul ist nun in der Lage, eine flexible Konfiguration zur Transformation von Activity Stream Daten in das für den CM-SPAM-Algorithmus erforderliche Format, zu ermöglichen. Dazu wurden die Konfigurations-Möglichkeiten um die Parameter *Identifier* (Übersetzung einzelner Einträge in einen Bezeichner), *Group by* (Gruppierung der Einträge), *Split at property* und *Split value* (Schneiden der Sequenzen) erweitert. Im Frontend wurden für diese Parameter Eingabe-Felder mit Auto vervollständigungs-Funktion implementiert. Die Eingabe-Felder sind in der Lage, dem Nutzer auf Grundlage der Meta-Daten Vorschläge zu machen. Die Implementierung der dazugehörigen Oberflächenbeschreibung und Oberflächen-Transformationsfunktion wurde bereits zur Illustrierung der adaptiven Konfiguration in Abbildung 4.6 in Abschnitt 4.1.4 gezeigt und besprochen.

Zur Implementierung dieser Funktionalität mussten einige Änderungen am Agenten des Moduls vorgenommen werden. Die Übersetzung der Activity Stream Einträge wird

von der Klasse `SequenceDBGenerator` geleistet. Bisher war dort die oben beschriebene Funktionalität statisch im Quelltext kodiert. Der gesamte Prozess der Generierung der Sequenz-Datenbank wurde daher dynamisiert. Abbildung 4.11 zeigt eine vereinfachte Version der für die Übersetzung verantwortlichen Methode `generateInputForAnalysis`. Der Activity Stream wird durch diese Methode in das in Abschnitt 2.1.1 und Abbildung 2.2 beschriebene Format des CM-SPAM-Algorithmus übersetzt. Der Methode liegen die nach dem *Group by* Parameter gruppierten Activity Stream Einträge (`items`), die Felder aus denen der Bezeichner generiert wird (`fields`) sowie Name und Wert des Feldes, an denen die Sequenzen geschnitten werden sollen (`splitAtProperty` und `splitAtValue`), vor. Alle Gruppen und die dazugehörigen Einträge werden durchlaufen (Zeile 4). Dabei werden die einzelnen Einträge in Zeile 8 in eindeutige numerische Bezeichner umgewandelt. Im nächsten Schritt wird geprüft, ob der Eintrag ein Schneiden der Sequenz veranlasst (Zeile 10-16). Standardmäßig passiert dies, bei Einträgen, die einem Login des Nutzers entsprechen. Sollte dies der Fall sein, so wird der Sequenz-Datenbank der Wert `-2`, der dem Algorithmus signalisiert, dass eine neue Sequenz beginnt, angehängt. Im letzten Schritt wird schließlich der Bezeichner des Eintrags, gefolgt von dem Wert `-1`, der die Einträge voneinander trennt, der Sequenz-Datenbank hinzugefügt (Zeile 18-19). So entsteht aus dem Activity Stream eine Sequenz-Datenbank, die nun von dem CM-SPAM-Algorithmus auf häufige Muster untersucht werden kann.

5. Anwendungsbeispiel

In diesem Kapitel soll die neue Version der Analytics Workbench im Hinblick auf ihre Fähigkeit, Logfile-Analysen mittels Sequential Pattern Mining durchführen zu können, untersucht werden. Als Datengrundlage wird eine Logfile aus dem KOLEGEA-Projekt¹ genutzt. KOLEGEA ist eine Plattform für kooperatives Lernen für Ärzte in Weiterbildung zum Facharzt der Allgemeinmedizin. Nutzer mit verschiedenen Qualifikationen und Rollen benutzen die Plattform. In dem folgenden Anwendungsbeispiel soll eine explorative Analyse von Nutzungsmustern von verschiedenen Nutzergruppen durchgeführt werden. Die Logfile umfasst 58.763 Activity Stream Einträge von August 2013 bis Februar 2016 und kommt auf eine Dateigröße von 24MB.

5.1. Selektion der Daten

In einem ersten Schritt sollen die Daten der Logfile in das System geladen werden. Dazu wird dem leeren Workflow ein Direct Uploader Modul hinzugefügt. Über das Modul kann die Logfile ausgewählt und bei der Durchführung des Workflows zum System hochgeladen werden. Durch die Auswahl der Datei wird der Meta-Analyse Workflow gestartet. Der

¹www.kolegea.de, Abruf: 09.07.2016

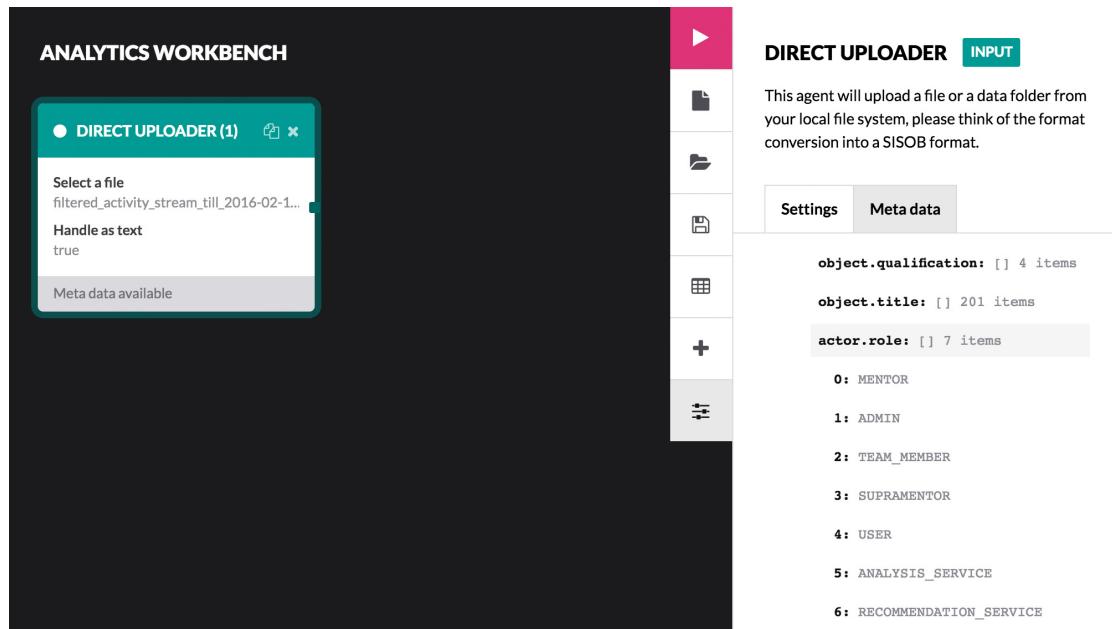


Abbildung 5.1.: Datenselektion durch Direct Uploader und Blick in die Meta-Daten

5. Anwendungsbeispiel

The image contains two side-by-side screenshots of a 'STREAM FILTER' interface. Both screenshots have a header with 'STREAM FILTER' and 'TOOLS'.

Left Screenshot (Initial State):

- Filter:** Shows a dropdown for 'actor.role' with 'Value' set to 'ADMIN'. Below it is a 'Date range' section with 'From' set to 'No date selected' and 'To' set to 'No date selected'.
- Meta data:** On the right, there is a list of user roles: ADMIN, ANALYSIS_SERVIC, MENTOR, RECOMMENDATI, and COMMUNITY_APP.

Right Screenshot (Configured State):

- Filter:** Shows two filter rules:
 - actor.role: ADMIN, Mode: exclude
 - generator.name: COMMUNITY_APP, Mode: include
- Date range:** Shows a date range from '1.1.2014' to '31.12.2014' with a calendar view below it.

Abbildung 5.2.: Adaptive Konfiguration des Activity Stream Filter Moduls

gesamte Prozess dauert bei der vorliegenden Logfile ca. 8 Sekunden. Sobald die Meta-Daten verfügbar sind, wird dies in der Fußleiste des Direct Uploader Moduls durch den Hinweis *Meta data available* signalisiert. Um einen ersten Einblick in die Struktur der Daten zu erhalten, kann in der Modul-Konfiguration des Direct Uploader Moduls der Meta-Daten-Inspektor geöffnet werden (siehe Abbildung 5.1). Dort lassen sich nun die Feld-Werte-Kombinationen einsehen. In der Abbildung sind im Meta-Daten-Inspektor (rechts) die Werte für das Feld `actor.role` zu sehen. Dies gibt dem Nutzer im Vorfeld schon einen Einblick in die in den Daten vorhandenen Nutzergruppen.

5.2. Bereinigung der Daten

Im nächsten Schritt sollen die Daten zunächst bereinigt werden. Einträge, die durch Administratoren erzeugt wurden, sollen bei der Analyse nicht berücksichtigt werden. Die Plattform stellt verschiedene Anwendungen für verschiedene Endgeräte bereit, die sich teilweise in ihrem Aufbau unterscheiden. Einträge aus verschiedenen Anwendungen könnten kollidieren und die Ergebnisse verzerren. Daher sollen nur Daten berücksichtigt werden, die innerhalb der Community-App erzeugt wurden. Außerdem seien für die Analyse nur Daten aus den Jahren 2014 und 2015 interessant. Für die Filterung der Daten wird nun das Activity Stream Filter Modul dem Workflow hinzugefügt. Der Ausgang

des Direct Uploader Moduls wird mit dem Eingang des Activity Stream Filter Modul verbunden. Durch die Verbindung der Module kann das Activity Stream Filter Modul nun auf die Meta-Daten des Input-Moduls zugreifen. Bei der Konfiguration des Moduls können dem Nutzer nun Vorschläge gemacht werden.

Abbildung 5.2 zeigt, wie das System den Nutzer bei der Konfiguration unterstützt. In der Abbildung (links) wird der Filter zum Ausschließen der von Administratoren erzeugten Einträge konfiguriert. Als Feld wurde bereits `actor.role` ausgewählt. Nun muss ein entsprechender Wert angegeben werden, bei dem der Filter greift. Auf Grundlage der Meta-Daten ist bekannt, welche Werte für das Feld `actor.id` in den Daten vorkommen. Beim Klick auf das Eingabe-Feld öffnet sich eine Auswahlbox mit Auto vervollständigungs-Funktionalität. Dort sind alle beobachteten Werte auswählbar. Ein Klick auf die entsprechenden Zeile wählt den gewünschten Wert aus. Als Modus muss nun noch `exclude` ausgewählt werden. Der Filter wird schließlich per Klick auf den Plus-Button in die Tabelle der Filter aufgenommen. Der Filter zur Einschränkung der Daten aus der KOLEGEA Community-App läuft ebenso ab. Hier wird das Feld `generator.name`, das die Umgebung, in der der Eintrag erzeugt wurde, bezeichnet, mit dem Wert `COMMUNITY_APP` und dem Modus `include` (einschließen) ausgewählt. Einträge aus anderen Umgebungen oder Anwendungen (z.B. TABLET) werden somit nicht mit in die Menge aufgenommen.

In Abbildung 5.2 (rechts) ist zu sehen, wie der Zeitraum des Activity Stream eingeschränkt werden kann. In den Meta-Daten ist vermerkt, dass die Daten aus dem Zeitraum August 2013 bis Februar 2016 stammen. Tage außerhalb dieses Zeitraums sind nicht auswählbar. Da für die Analyse nur Daten aus den Jahren 2014 und 2015 genutzt werden sollen, wird als Start-Datum der 1. Januar 2014 und als End-Datum der 1. Januar 2016 ausgewählt. Tage die sich innerhalb der aktuellen Auswahl befinden werden, wie in der Abbildung zu sehen, dunkel hinterlegt.

5.3. Transformation und Analyse

Die Transformation und Analyse der Daten wird vom Sequential Pattern Mining Modul geleistet. Das Modul wird dem Workflow hinzugefügt und mit dem Activity Stream Filter Modul verbunden. Standardmäßig werden bei der Übersetzung der Einträge die Felder `verb` und `object.objectType` herangezogen. Ergebnis wären Einträge wie `open_view` oder `open_case`. Wenn aber konkrete Objekte interessant sind, kann ein weiteres Feld, das das einzelne Objekt identifiziert, mit in die Übersetzung aufgenommen werden; beispielsweise `object.name`. Dazu muss in der Konfiguration des Sequential Pattern Mining Moduls unter *Identifier* das Feld `object.name` aufgenommen werden. Abbildung 5.3 zeigt diesen Prozess. Über die Auto vervollständigungs-Funktion des Eingabe-Feldes kann nach dem entsprechenden Feld gesucht werden. Hier wurde bereits die Eingabe `object` vorgenommen. Die Auswahlmöglichkeiten beschränken sich nun auf alle Felder, die diese Zeichenkette beinhalten. Auch `object.name` gehört dazu und kann über einen Klick mit in die Liste zur Übersetzung aufgenommen werden. Die Bezeichner der Einträge für die Übersetzung setzen sich nun aus den Feldern `verb`, `object.objectType` und `object.name` zusammen. Da in dieser Analyse die Nutzungsmuster einzelner Nutzer interessant sind, kann die voreingestellte Gruppierung nach dem Feld `actor.id` bleiben.

5. Anwendungsbeispiel

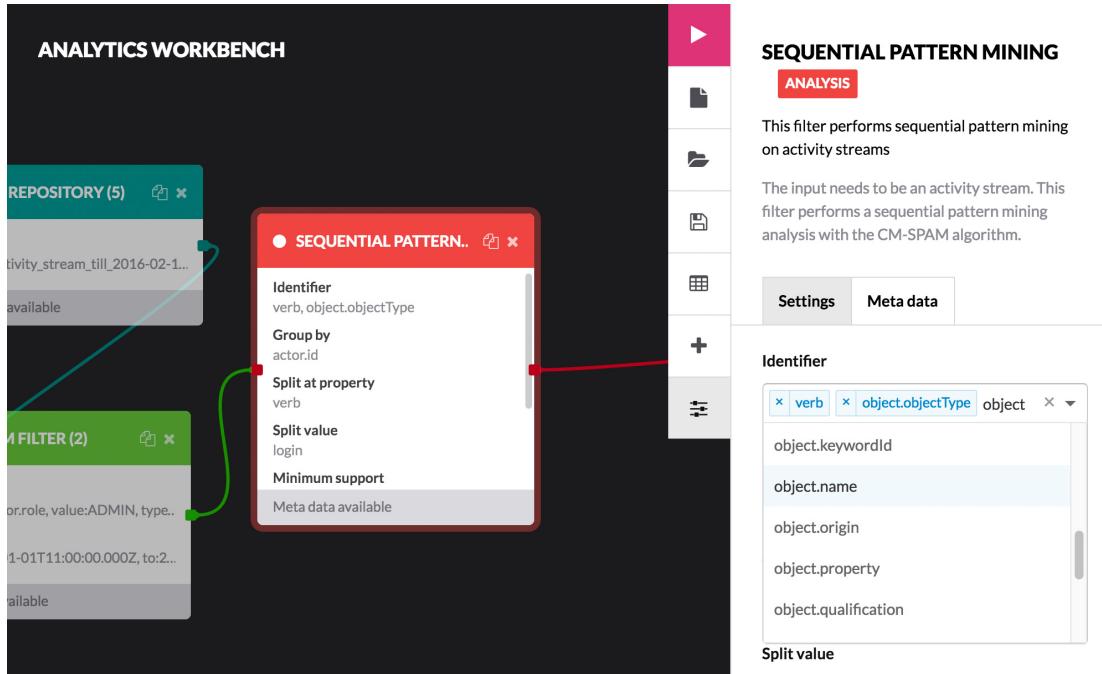


Abbildung 5.3.: Konfiguration des Sequential Pattern Mining Moduls

Das Schneiden der Sequenzen an Einträgen, die in dem Feld `verb` den Wert `login` aufweisen, kann ebenfalls so übernommen werden. Als Ausgabe-Format wird *SiSOB-Tabelle* ausgewählt.

5.4. Evaluation und Exploration

Im letzten Schritt müssen die extrahierten Nutzungsmuster nun in eine Form gebracht werden, die eine leichte Interpretation ermöglicht. Durch das Table Viewer Modul kann die von dem Sequential Pattern Mining Modul erzeugte SiSOB-Tabelle in ein HTML-Dokument umgewandelt und direkt im Browser geöffnet und betrachtet werden. Dazu wird das Modul dem Workflow hinzugefügt und mit dem Sequential Pattern Mining Modul verbunden. Der Workflow kann nun mit einem Klick auf den farblich hinterlegten *Play*-Button in der Funktionsleiste durchgeführt werden. Der Fortschritt des Prozesses wird, wie in Abschnitt 4.1.3 beschrieben, visualisiert. Sobald die Ergebnisse vorliegen, öffnet sich der *Drawer* mit den Ergebnissen für jedes Output-Modul. Ein Klick auf das Ergebnis des Table Viewer Moduls führt zu der in Abbildung 5.4 zu sehenden Tabelle. Hier sind nun alle extrahierten Muster, nach Häufigkeit sortiert, aufgelistet. Ein Muster besteht aus zwei oder mehreren Items. Die Items repräsentieren die vom Nutzer getätigten Aktionen (z.B. `[open, view, Home]`) und setzen sich durch die in dem Transformation-Schritt festgelegten Übersetzungsform (`[verb, object.objectType, object.name]`) zusammen.

Die vorliegende Tabelle gibt nun einen ersten Überblick über die Nutzungsmuster der KOLEGEA Community-App. In weiteren Iterationen können nun verschiedene Nutzer-

5.4. Evaluation und Exploration

length	sequence	support	support (relative)
2	[login, profile] → [open, view, Home]	861	0.7848678213309025
2	[login, profile] → [open, case]	289	0.2634457611668186
2	[login, profile] → [open, article]	180	0.1640838650865998
3	[login, profile] → [open, article] → [open, view, Home]	166	0.15132178669097537
2	[open, article] → [open, view, Home]	166	0.15132178669097537
3	[login, profile] → [open, view, Home] → [open, view, Home]	158	0.1440291704649043
2	[open, view, Home] → [open, view, Home]	158	0.1440291704649043
3	[login, profile] → [open, case] → [open, view, Home]	142	0.12944393801276208
2	[open, case] → [open, view, Home]	142	0.12944393801276208
4	[login, profile] → [open, article] → [open, view, Home] → [open, view, Home]	118	0.10756608933454877
3	[open, article] → [open, view, Home] → [open, view, Home]	118	0.10756608933454877
3	[login, profile] → [open, case] → [open, case]	102	0.09298085688240657
2	[login, profile] → [open, view, My Questions]	102	0.09298085688240657
2	[open, case] → [open, case]	102	0.09298085688240657
3	[login, profile] → [open, view, My Questions] → [open, view, Home]	93	0.08477666362807658
2	[open, view, My Questions] → [open, view, Home]	93	0.08477666362807658

Abbildung 5.4.: Ergebnisse der Analyse

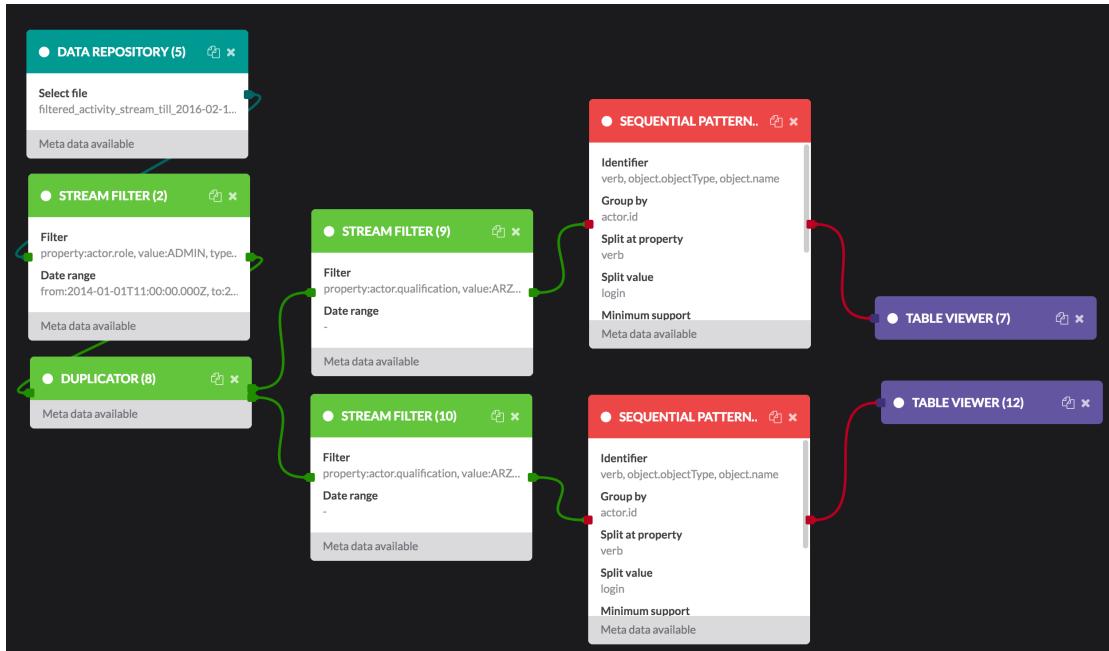


Abbildung 5.5.: Erweiterung des Workflows, zur Gegenüberstellung von Nutzergruppen

gruppen miteinander verglichen werden. In Abbildung 5.5 wurde der bisherige Workflow, für eine Gegenüberstellung von Nutzern mit der Qualifikation ARZT und Nutzern mit der Qualifikation ARZT_IN_WEITERBILDUNG, erweitert. Nach der generellen Filterung von Zeitraum und Nutzern wie Administratoren, wird hier der Activity Stream in die beiden genannten Nutzergruppen aufgeteilt. Dazu wird der Activity Stream mit dem Duplica-

5. Anwendungsbeispiel

tor Modul dupliziert und anschließend erneut gefiltert. Der obere Zweig des Workflows erhält nur Einträge, die in dem Feld `actor.qualification` den Wert ARZT halten, der untere Zweig erhält nur Einträge, die in dem Feld den Wert ARZT_IN_WEITERBILDUNG halten. Beide Gruppen werden anschließend von einem Sequential Pattern Mining Modul analysiert und von dem Table Viewer Modul in eine Tabelle überführt. Die Ergebnisse können schließlich gegenübergestellt werden. Eine explorative Analyse umfasst immer mehrere Iterationen, bei denen auch Zwischenergebnisse wieder in den Prozess mit einfließen. So könnten im weiteren Verlauf der Analyse die Übersetzungsform variiert oder die Daten weiter eingeschränkt werden, wobei die adaptive Konfiguration den Nutzer in diesem Prozess unterstützt.

5.5. Zusammenfassung

Die neue Version der Analytics Workbench macht es möglich, eine explorative Analyse von Logfiles mittels Sequential Pattern Mining durchzuführen. In der bisherigen Version der Analytics Workbench wäre die oben beschriebene Analyse nicht möglich gewesen. Der Nutzer hat in der bisherigen Version keinerlei Einsicht in die Struktur der Eingabe-Daten. Auch die Konfigurationsmöglichkeiten sind durch die statische Implementierung der Analyse-Modul eingeschränkt. Eine Filterung nach Zeitraum oder bestimmten Nutzergruppen wäre daher nicht möglich.

6. Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde auf Grundlage der Analytics Workbench eine adaptiv konfigurierbare Web-Umgebung zur visuellen Spezifikation von Logfile-Analysen erarbeitet und implementiert. Hierzu wurden zunächst Probleme mit der Analytics Workbench im Hinblick auf die Analyse von Logfiles mittels Sequential Pattern Mining festgestellt. Die bisherige Version der Analytics Workbench war durch die statischen Benutzeroberflächen zur Konfiguration der Module nicht für Arbeit mit schwach strukturierter Daten, wie Activity Streams, geeignet. Für die Analyse von Activity Streams existierten aber bereits für die Vorverarbeitung das Activity Stream Filter Modul und für die Transformation und Analyse das Sequential Pattern Mining Modul. Diese Module waren bisher allerdings sehr eingeschränkt in ihren Konfigurationsmöglichkeiten, sodass eine explorative Analyse von Activity Streams nur schwierig durchführbar war. Auf Grundlage der festgestellten Probleme wurden schließlich neue Anforderungen an das System festgehalten. Die Konfigurationsmöglichkeiten der Analyse-Module sollten sich auf Grundlage der Struktur der Eingabe-Daten adaptieren können, um den Nutzer bei der Konfiguration zu unterstützen. Dazu sollte die Flexibilität der Benutzeroberflächen ausgebaut werden, sodass es möglich ist, dynamisch auf Änderungen in der Konfiguration und den Eingabe-Daten reagieren zu können. Auch die Analyse-Module sollten verbessert werden. Mit dem Activity Stream Filter Modul sollte es möglich sein, gleichzeitig mehrere Filter anwenden und eine Filterung anhand eines definierten Zeitraums vornehmen zu können. Das Sequential Pattern Mining Modul sollte eine flexiblere Transformation der Activity Stream Einträge in ein für den CM-SPAM-Algorithmus geeignetes Format erlauben.

Auf Grundlage dieser Anforderungen wurde ein Konzept entwickelt, das nun eine flexible und adaptive Konfiguration von Analyse-Modulen erlaubt. Vorbild für die adaptive Konfiguration war das Konzept der *self-adaptive Software*, das eine auf Umgebungsdaten basierende, dynamische Adaption, die zur Entwurfszeit festgelegt wird, vorsieht. Die Adaption der Konfigurationsmöglichkeiten sollte auf Basis von Meta-Daten, die Informationen über die Struktur der Eingabe-Daten enthalten, geschehen. Zur Erzeugung dieser Meta-Daten wurden drei Ansätze vorgestellt. Der finale Ansatz sieht eine Erzeugung von Meta-Daten durch einen im Hintergrund ablaufenden Analyse-Workflow vor. Die Benutzeroberflächen der Module können schließlich über eine Art Meta-Ebene auf die erzeugten Meta-Daten zugreifen. Um für alle Module valide Meta-Daten ableiten zu können, müssen Module eine Meta-Daten-Transformationsfunktion implementieren, die analog zur Funktionalität des jeweiligen Moduls eingehende Meta-Daten transformiert, auf die nachfolgende Modul zugreifen können. Um nun eine Adaption der Benutzeroberflächen zu ermöglichen, müssen Module eine Oberflächen-Transformationsfunktion implementieren, die die initiale Oberflächenbeschreibung auf Basis der Meta-Daten verändern kann. Auf Grundlage der neuen Möglichkeiten, wurden im Anschluss das Activity Stream Filter und das Sequential Pattern Mining Modul neu konzipiert. Für die Umsetzung dieser Konzepte war eine Reimplementierung des Frontends der Analytics Workbench vorgesehen. Im selben Zuge wurde daher auch die Arbeitsumgebung zur Spezifikation der Analyse-Workflows neu konzipiert.

6. Zusammenfassung und Ausblick

Die Reimplementierung des Frontends wurde unter Verwendung von modernen Web-Technologien vorgenommen. Dabei wurde die aktuellste JavaScript Version 6 sowie das React-Framework als View-Engine und die Redux-Bibliothek für die Infrastruktur genutzt. Für die Kommunikation zwischen Frontend und Server wurde die bisherige WebSocket-Schnittstelle verwendet, die in ihrer Funktionalität nicht geändert werden musste. Serverseitig mussten die Agenten des Activity Stream Filter und des Sequential Pattern Mining Modul an die neue Funktionalität angepasst werden. Auch wurde ein Meta-Analyse-Modul implementiert, das zur Erzeugung von Meta-Daten benötigt wurde. Im Anschluss an die Implementierung wurde in einem Anwendungsbeispiel die neuen Funktionen der Analytics Workbench demonstriert. Mit der neuen Version der Analytics Workbench ist nun eine explorative Analyse von Logfiles mittels Sequential Pattern Mining möglich.

Im Zuge dieser Arbeit wurden nur die für die Analyse von Logfiles benötigten Module an die neue Version angepasst. Um die neue Version der Analytics Workbench im vollen Umfang nutzen zu können, müssten in einem ersten Schritt die restlichen Analyse-Module an die neue Umgebung angepasst werden. Dazu würde es vorerst reichen, für die einzelnen Module die Oberflächenbeschreibung an die neue Syntax anzupassen sowie die Oberflächen- und Meta-Daten-Transformationsfunktion zu implementieren. Die neuen Möglichkeiten der adaptiven Konfiguration bieten auch über die Analyse von Logfiles hinaus interessante Anwendungsmöglichkeiten. Die Analytics Workbench zeichnet sich vor allem durch ihre vielfältigen Möglichkeiten zur Analyse von Netzwerken aus. Aktuell können die bei Netzwerkanalysen verwendeten Graphen mithilfe des Custom Data Decorator Moduls mit weiteren Informationen über die Eigenschaften und Attribute des Graphen angereichert werden. Dies könnte zukünftig automatisiert durch das Meta-Daten-System der neuen Workbench erfolgen. Dazu wäre eine Erweiterung des Meta-Analyse-Moduls notwendig, die die Erkennung entsprechender Datenformate und die Extraktion von Attributen des Graphen umfasst. Interessante Attribute könnten beispielsweise der Typ des Graphen (*directed*, *weighted*, *simple*, *multi*, usw.), die Klasse (*regular*, *complete*, *bipartite*, *cyclic*, usw.) oder weitere Eigenschaften der Knoten und Kanten des Graphen sein. Abhängig von diesen Informationen könnten Analyse-Module erweiterte Konfigurationsmöglichkeiten anbieten oder Vorschläge bei der Konfiguration machen. Im Zuge einer Verbesserung der Arbeitsumgebung wäre es denkbar, die Meta-Daten zur Validierung von Verbindungen zwischen Modulen zu nutzen. Aktuell können unabhängig von der tatsächlichen Kompatibilität alle Modulen miteinander verbunden werden. Aus den Meta-Daten geht hervor, welche Art von Ausgabe-Daten ein Modul erzeugt. Wenn ein Modul nun beschreiben könnte, welche Art von Eingabe-Daten es unterstützt, könnte diese Informationen beim Verbinden der Module abgeglichen werden und mögliche Inkompatibilitäten im Vorfeld festgestellt werden.

Anhang A.

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegen.

Malte Wessel
Düsseldorf, den 12. Juli 2016

Anhang B.

Quellcode der Software

Der Quellcode der Software und eine Anleitung zur Inbetriebnahme befinden sich auf dem angehängten Datenträger.

Literaturverzeichnis

- [BCD⁺08] BERTHOLD, Michael R. ; CEBRON, Nicolas ; DILL, Fabian ; GABRIEL, Thomas R. ; KÖTTER, Tobias ; MEINL, Thorsten ; OHL, Peter ; SIEB, Christoph ; THIEL, Kilian ; WISWEDEL, Bernd: *KNIME: The Konstanz information miner*. Springer, 2008
- [Bra14] BRAY, Tim: *The javascript object notation (json) data interchange format*. <https://tools.ietf.org/html/rfc7159>. Version: 2014, Abruf: 11.07.2016
- [Bus98] BUSCHMANN, Frank: *Pattern-orientierte Software-Architektur: Ein Pattern-System*. Pearson Deutschland GmbH, 1998
- [CEF⁺06] CHAKRABARTI, Soumen ; ESTER, Martin ; FAYYAD, Usama ; GEHRKE, Johannes ; HAN, Jiawei ; MORISHITA, Shinichi ; PIATETSKY-SHAPIRO, Gregory ; WANG, Wei: Data mining curriculum: A proposal (Version 1.0). In: *Intensive Working Group of ACM SIGKDD Curriculum Committee* (2006), S. 140
- [CGG10] CURCIN, Vasa ; GHANEM, Moustafa ; GUO, Yike: The design and implementation of a workflow analysis tool. In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 368 (2010), Nr. 1926, S. 4193–4208
- [FPSS96] FAYYAD, Usama ; PIATETSKY-SHAPIRO, Gregory ; SMYTH, Padhraic: From data mining to knowledge discovery in databases. In: *AI magazine* 17 (1996), Nr. 3, S. 37
- [FVGG⁺14] FOURNIER-VIGER, Philippe ; GOMARIZ, Antonio ; GUENICHE, Ted ; SOLTANI, Azadeh ; WU, Cheng-Wei ; TSENG, Vincent S.: SPMF: a java open-source pattern mining library. In: *The Journal of Machine Learning Research* 15 (2014), Nr. 1, S. 3389–3393
- [Gac15] GACKENHEIMER, Cory: *Introduction to React*. Apress, 2015
- [Gan07] GANNON, Dennis: Component Architectures and Services: From Application Construction to Scientific Workflows. In: *Workflows for e-Science*. Springer, 2007, S. 174–189
- [Gei08] GEIHS, Kurt: Selbst-adaptive software. In: *Informatik-Spektrum* 31 (2008), Nr. 2, S. 133–145
- [GHHH13] GÖHNERT, Tilman ; HARRER, Andreas ; HECKING, Tobias ; HOPPE, H U.: A workbench to construct and re-use network analysis workflows: Concept, implementation, and example case. In: *Proceedings of the 2013 IEEE/ACM international conference on advances in social networks analysis and mining* ACM, 2013, S. 1464–1466

Literaturverzeichnis

- [HGM⁺00] HOPPE, H U. ; GASSNER, Katrin ; MUUML, Martin ; TEWISSEN, Frank u. a.: Distributed visual language environments for cooperation and learning: Applications and intelligent support. In: *Group Decision and Negotiation* 9 (2000), Nr. 3, S. 205–220
- [HPK11] HAN, Jiawei ; PEI, Jian ; KAMBER, Micheline: *Data mining: concepts and techniques*. Elsevier, 2011
- [Jon07] JONES, Andrew C.: Workflow and biodiversity e-Science. In: *Workflows for e-Science*. Springer, 2007, S. 80–90
- [Jos15] JOSEPH, R. J.: Single page application and canvas drawing. In: *International Journal of Web and Semantic Technology* 6 (2015), Nr. 1, S. 29–37
- [Kes11] KESSIN, Zachary: *Programming HTML5 applications: building powerful cross-platform environments in JavaScript*. O'Reilly Media, Inc., 2011
- [KLSH09] KOHAVI, Ron ; LONGBOTHAM, Roger ; SOMMERFIELD, Dan ; HENNE, Randolph M.: Controlled experiments on the web: survey and practical guide. In: *Data mining and knowledge discovery* 18 (2009), Nr. 1, S. 140–181
- [KW00] KNOBLOCH, Bernd ; WEIDNER, Jens: Eine kritische Betrachtung von Data Mining-Prozessen—Ablauf, Effizienz und Unterstützungsotenziale. In: *Data Warehousing 2000*. Springer, 2000, S. 345–365
- [ME10] MABROUKEH, Nizar R. ; EZEIFE, Christie I.: A taxonomy of sequential pattern mining algorithms. In: *ACM Computing Surveys (CSUR)* 43 (2010), Nr. 1, S. 3
- [MH11] MÖNIG, J. ; HARVEY, B.: *Snap! A visual, drag-and-drop programming language*. <http://snap.berkeley.edu/>. Version: 2011, Abruf: 29.06.2016
- [MKFR03] MIERSWA, Ingo ; KLINKENBERG, Ralf ; FISCHER, Simon ; RITTHOFF, Oliver: A flexible platform for knowledge discovery experiments: Yale—yet another learning environment. In: *LLWA 03-Tagungsband der GI-Workshop-Woche Lernen-Lehren-Wissen-Adaptivität*, 2003
- [Mob06] MOBASHER, Bamshad: Web usage mining. In: *Web data mining: Exploring hyperlinks, contents and usage data* 12 (2006)
- [MRRR02] MEDVIDOVIC, Nenad ; ROSENBLUM, David S. ; REDMILES, David F. ; ROBBINS, Jason E.: Modeling software architectures in the Unified Modeling Language. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11 (2002), Nr. 1, S. 2–57
- [NSW12] NIEMANN, Katja ; SCHEFFEL, Maren ; WOLPERS, Martin: An overview of usage data formats for recommendations in TEL. In: *Workshop on Recommender Systems for TEL* Citeseer, 2012, S. 95–100
- [OGT⁺99] OREIZY, Peyman ; GORLICK, Michael M. ; TAYLOR, Richard N. ; HEIMBIGNER, Dennis ; JOHNSON, Gregory ; MEDVIDOVIC, Nenad ; QUILICI, Alex ; ROSENBLUM, David S. ; WOLF, Alexander L.: An architecture-based approach to self-adaptive software. In: *IEEE Intelligent systems* (1999), Nr. 3, S. 54–62

- [PHMAZ00] PEI, Jian ; HAN, Jiawei ; MORTAZAVI-ASL, Behzad ; ZHU, Hua: Mining access patterns efficiently from web logs. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining* Springer, 2000, S. 396–407
- [Rap14] RAPIDMINER (Hrsg.): *RapidMiner Studio Manual*. RapidMiner, Inc. Headquarters, 10 Milk Street, 11th Floor, Boston, MA 02108: RapidMiner, 2014
- [Rem15] REMBERG, Stefan: *Re-Engineering eines Rahmensystems für Multiagentensysteme zur Datenanalyse*. Lotharstr. 65, 47057 Duisburg, Universität Duisburg-Essen, Diplomarbeit, 2015
- [SA96] SRIKANT, Ramakrishnan ; AGRAWAL, Rakesh: Mining sequential patterns: Generalizations and performance improvements. In: *International Conference on Extending Database Technology* Springer, 1996, S. 1–17
- [SAN12] SNELL, J. ; ATKINS, M. ; NORRIS, W: *JSON Activity Streams 1.0*. <http://activitystrea.ms/specs/json/1.0/>. Version: 2012, Abruf: 26.05.2016
- [TS07] TIWARI, Abhishek ; SEKHAR, Arvind K.: Workflow based framework for life science informatics. In: *Computational biology and chemistry* 31 (2007), Nr. 5, S. 305–319
- [Val01] VALDMAN, Jan: Log file analysis. In: *Department of Computer Science and Engineering (FAV UWB)., Tech. Rep. DCSE/TR-2001-04* (2001)
- [why13] *Why React?* <https://facebook.github.io/react/docs/why-react.html>. Version: 2013, Abruf: 30.06.2016