FREE UNIVERSITY OF BOLZANO
FACULTY OF COMPUTER SCIENCE

A Graphical User Interface

for Sequenced Event Set Pattern Matching

Autor
Florian Michaeler
Freie Universität Bozen-Bolzano, Italy
Fakultät für Informatik
florian.michaeler@stud-inf.unibz.it

Supervisor
Prof. Johann Gamper
Freie Universität Bozen-Bolzano, Italy
Fakultät für Informatik
gamper@inf.unibz.it

Danksagungen

Zuerst gebührt Prof. Dr. Johann Gamper der Freien Universität Bozen großer Dank. Er hat das Projekt vermittelt und beaufsichtigt und war bei dessen Umsetzung eine große Hilfe. Des Weiteren muss Bruno Cadonna Dank für seine große Hilfe ausgesprochen werden. Er hat mit seinem Algorithmus für Sequenced Event Set Pattern Matching die Grundlage für diese Arbeit gelegt und diesen Algorithmus außerdem vorgestellt und erklärt und dabei geholfen, ihn auf ein neues System zu portieren und dort lauffähig zu machen. Auch den Mitarbeitern der Informatikabteilung der Freien Universität Bozen gebührt Dank, da die Umsetzung des Projektes ohne die Bereitstellung einer Oracle Datenbank und ihrer Hilfe bei der Umschiffung von einigen Verbindungsproblemen wohl nicht ohne weiteres möglich gewesen wäre.

Abstract

Sequenced Event Set Pattern Matching [1] (SESPM) ist ein Algorithmus zum Abgleich von Sequenzen von Eingabeevents mit komplexen Mustern. Diese komplexen Muster können den Inhalt, die Sortierung, die Quantifizierung und den zeitlichen Abstand zwischen Events beschreiben. Der Benutzer kann also eine Datenbank voller Input Events mit einem Muster abgleichen, um die gewünschten Events herauszufiltern. Im Gegensatz zu normalen Abfragen mit SQL kann ein Zeitraum gewählt werden, innerhalb welchem die Events stattgefunden haben müssen und festgelegt werden, dass bestimmte Events in beliebiger Reihenfolge stattgefunden haben können.

Dieses Dokument beschreibt die Entwicklung einer grafischen Oberfläche für die Verwendung und Veranschaulichung der Eingabe, der Zwischenschritte und der Ausgabe des SESPM Algorithmus. Dieser wurde in der Programmiersprache C geschrieben, verwendet für die Verwaltung der Eingabeevents eine Oracle Datenbank und nutzt zur Eingabe von Mustern Textdateien. Der Benutzer muss also zuerst eine Oracle Datenbank anlegen und mit Events füllen, dann die Verbindungsdaten in eine vorgefertigte Textdatei eingeben, den Algorithmus auf dem eigenen System kompilieren und im Anschluss die ausführbare Datei mit einer Textdatei füttern, welche das gewünschte Muster enthält. Dies alles erfordert einiges an technischem Wissen, ein geeignetes System mit entsprechender Software und viel Zeit. Um diese Probleme zu umgehen und den SESPM Algorithmus einer breiteren Masse an Benutzern zugänglich machen zu können, wurde also die im folgendem beschriebene grafische Oberfläche dafür entwickelt. Sie wurde implementiert und getestet und vereinfacht die Verwendung des vorhandenen Algorithmus stark. Mit der grafischen Oberfläche können erzielte Ergebnisse gut und schnell veranschaulicht werden.

Abstract

Sequenced Event Set Pattern Matching [1] (SESPM) is an algorithm for matching sequences of input events against complex patterns. These patterns can describe the content, sorting and quantification of the events and the time interval within which they happened. A user can thus match a database containing input events against a pattern to find the ones he is looking for. In contrast to normal SQL queries, there can be set a time interval during which different events happened. In addition, one can also specify the sequence of some events to be arbitrary.

This document describes the development of a graphical user interface for the usage and demonstration of the input, the intermediate steps and the output of SESPM. The algorithm has been created using the programming language C, uses an Oracle database to manage the input events and text files for the patterns. A user first has to create an Oracle database and populate it with input events, deposit the connection data in a text file, compile the algorithm on his system and eventually feed the executable with a text file containing the pattern. Doing all this lasts a lot of time and requires a high level of technical knowledge and a proper system with all the software needed. To be able to avoid all of this problems and thus to make the algorithm available to a much bigger audience a graphical user interface has been introduced. It has been implemented and tested and simplifies the usage of the given algorithm. Through the graphical user interface results can be achieved and demonstrated nicely and fast.

Abstract

Sequenced Event Set Pattern Matching [1] (SESPM) è un algoritmo per il pareggio di sequenze di eventi input con degli schemi complessi. Con l´aiuto di questi schemi possono essere descritti il contenuto, l´ordine, la quantificazione o la distanza temporanea di eventi. L´utente potrà dunque pareggiare una banca-dati piena di eventi input con uno schema per trarrne quelli desiderati. A differenza delle richieste dati usuali eseguiti con SQL può essere scelto un periodo entro il quale gli eventi hanno avuto luogo e può essere stabilito che certi eventi possono aver avuto luogo in ordine facoltativa.

Questo documento descrive lo sviluppo di una superfice grafica per l´uso e per l´illustrazione dell´inserimento, dei passi intermediati e dell´output di SESPM. Quest'algoritmo è stato scritto nel linguaggio di programmazione C, utilizza una banca dati Oracle per l´amministrazione di eventi input e usa dei fili di testo per l´inserimento degli schemi. Dunque l´utente per prima cosa deve installare e configurare una banca dati Oracle e inserirci degli eventi. Poi inserisce i dati di connessione in un file di testo e compilisce l´algoritmo sul proprio sistema. Finalmente il file compilato è eseguito usando un file testo come input. Questo file contiene lo schema scelto. Tutto ciò richiede delle conoscenze tecniche e impiega tanto tempo. Inoltre è necessario l´uso di un sistema idoneo. La superficie grafica descritta in seguito è stata sviluppata per evitare questo tipo di problemi e perciò rende disponibile l´uso dell´algoritmo SESPM a un pubblico più ampio. È stata implementata e sottoposta a dei test e semplifica molto l´uso dell´algoritmo. Con la superficie grafica è possibile di dimostrare e illustrare in maniera rapida e semplice i risultati ottenuti.

INHALT

1. Einleitung

In diesem Dokument wird der Entwurf einer grafischen Oberfläche für einen Algorithmus zum Abgleich von Sequenzen von Eingabeevents mit komplexen Mustern beschrieben. Der Algorithmus wurde bereits implementiert und getestet und funktioniert wie folgt: zuerst benötigt er ein Muster als Input, welches in Form einer Textdatei übergeben wird. Die erlaubte Form des Musters kann man der Dissertation über SESPM [1] entnehmen. Aus diesem Muster erstellt der Algorithmus am Anfang einen entsprechenden endlichen Automaten. Anschließend verbindet sich der Algorithmus mit Hilfe einer Textdatei, welche URL, Benutzername und Passwort enthält, zu einer Oracle Datenbank aus welcher dann verschiedene Events gelesen und durch diesen Automaten geschickt werden. Die Events die durch den Automaten durchkommen, also jene die bis zum „accepting state“ kommen, sind die Ergebnisse für das eingegebene Muster. Das Programm gibt einen langen Text zurück, welcher zuerst eine Repräsentation des Automaten und im Anschluss daran die Liste der Events, welche durch den Automaten durchgekommen sind, enthält.

Die grafische Oberfläche für “Sequenced Event Set Pattern Matching“ wurde als Webapplikation realisiert. Dies hat den großen Vorteil, dass die Oberfläche von überall und für beliebig viele Benutzer bereitgestellt werden kann. Ein weiterer Vorteil dieser Lösung liegt darin, dass diese Webapplikation unabhängig von Plattform und System verwendet werden kann und sie kaum Vorwissen oder spezielle Kenntnisse benötigt. Bei der Implementierung musste auf Grund dessen jedoch ein besonderes Augenmerk auf die parallele Ausführbarkeit aller Elemente gelegt werden. Außerdem musste auch auf eine möglichst gute Performance Wert gelegt werden, damit die Serverlast auch bei vielen parallelen Aufrufen nicht zu groß und damit die Wartezeit zu lang wird. Die Webapplikation steuert das Ausführen des Algorithmus, übernimmt die Übergabe des Musters und erstellt mit Hilfe der zurückgegebenen Daten einen grafischen Automaten, welche nach Wahl eines Ergebnisses auch dessen Weg durch ebendiesen Automaten grafisch anzeigen kann. Die Datenbank ist zwar fest vorgegeben, doch für das Muster werden dem Benutzer verschiedene Möglichkeiten geboten. Er kann entweder ein vorgefertigtes Beispielmuster verwenden um die Funktionalität von SESPM genauer zu betrachten, oder selbst ein Muster erstellen um eigene Ergebnisse erzielen zu können.

Dieses Dokument ist wie folgt aufgebaut: In dieser Einleitung gab es eine kleine Übersicht über die Funktionsweise des verwendeten Algorithmus und dem generellen Aufbau der grafischen Oberfläche dafür. In Punkt 2 werden kurz verschiedene Projekte erläutert, die im Zusammenhang mit dieser Arbeit stehen. In Punkt 3 wird der übliche sequentielle Ablauf der Benutzung der Webapplikation und die technische Seite der grafischen Oberfläche diskutiert. Nach einer kleinen Übersicht wird also im Detail auf die Funktionsweise der einzelnen Abschnitte der Webapplikation eingegangen. Dies beinhaltet jeden möglichen Schritt, welchen ein Benutzer auf der Webapplikation durchführen kann und beschreibt was dabei in technischer Hinsicht im Hintergrund abläuft und welche Logik jedem Schritt innewohnt. Abgeschlossen wird die Arbeit mit Punkt 4, in welchem einem kleinen Fazit einige mögliche Weiterentwicklungen für diese Arbeit folgen.

2. THEMENVERWANDTE ARBEITEN

Die Dissertation “Sequenced Event Set Pattern Matching” [1] von Bruno Cadonna, welche er im Jahre 2011 für die Freie Universität Bozen verfasst hat, beinhaltet die Theorie zum SESPM Algorithmus und auch deren Funktionsweise und Umsetzung welche als Grundgerüst für diese grafische Oberfläche verwendet wurde. Dieses Projekt und diese Dokumentation bauen also auf dem Dokument und der Implementation von SESPM von Bruno Cadonna in der Version 0.3.1 auf und stützen sich auf dessen Inhalt. Auch einige Beispiele, welche in diesem Dokument verwendet werden, wurden aus der Dissertation übernommen. Mit diesem Algorithmus ist es möglich, Datenbanken effizient nach Mustern zu durchsuchen, welche den Inhalt, die Sortierung, die Quantifizierung und den zeitlichen Abstand zwischen Events beschreiben können.

Bruno Cadonna hat im Zusammenhang mit SESPM ein weiteres Dokument veröffentlicht, welches sich „Efficient Event Pattern Matching with Match Windows“ [2] nennt. Es erweitert die vorherige Arbeit um eine Datenvorbehandlung und eine Vorfilterung welche die Geschwindigkeit des Algorithmus erhöht.

Es gibt Vorschlag für eine neue SQL Funktionalität [3], welche es erlauben würde, bestimmte Muster in Sequenzen von Datentupeln zu finden. Die Muster werden nach dem Vorbild von normalen Regulären Ausdrücken definiert und beinhalten Konditionen und Quantifizierungen der Tupel. Die Funktionalität erlaubt das beliebige Vertauschen von Tupeln und ermöglicht damit die Definition von Sets von Tupeln. Damit ist es möglich, beliebige SES Muster ohne Kleene Quantifizierer auszudrücken.

DejaVu [4] ist ein System zum Verarbeiten von Events in Verwendung von Relationalen Datenbank Management Systemen wie MySQL. Es implementiert, unter der Verwendung von endlichen Automaten einen Teil der neuen SQL Funktionalität [3], lässt einen Operator für die beliebige Vertauschung der einzelnen Events aber außen vor. Um SES Muster damit ausdrücken zu können, braucht es jeweils mehrere verschiedene Muster. Die Anzahl der benötigten Muster steigt exponentiell mit der Anzahl der Elemente in den Sets der SES Muster. Dies führt natürlich dazu, dass das System bei größeren Sets schnell unhandlich und langsam wird.

SQL-TS [5] beschreibt den Entwurf und die Optimierung einer Abfragesprache, die es erlaubt, Datenbanken effizient nach komplexen sequentiellen Mustern zu durchsuchen. Diese Erweiterung für SQL erlaubt es also solche Muster zu definieren, im Dokument wird nach der Einführung dieser Erweiterung auch nach Möglichkeiten zur Optimierung von ebendieser geforscht. Gleich wie bei DejaVu [4] wird auch hier auf einen Operator für die beliebige Vertauschung einzelner Events verzichtet. Dies führt natürlich dazu, dass diese Erweiterung für SQL daher auch dieselben Nachteile wie das genannte System mit sich bringt.

Auch ZStream [6] dient dem Verarbeiten von komplexen Mustern. Diese Muster können Sequenzen, logische UND oder ODER Verknüpfungen, Negationen und die kleenesche Hülle beinhalten. Es erstellt verschiedene baumförmige Pläne für die Abfragen und wählt dann den vermutlich kosteneffizientesten daraus aus. ZStream erlaubt es auch, das Vorkommen von Events mit einem zeitlichen Rahmen einzuschränken. Außerdem ist es möglich, die Reihenfolge der zeitlichen Vorkommen verschiedener Events zu ignorieren, dies heißt dass einzelne Events beliebig vertauscht werden können. Kleene Quantifizierer in Verbindung mit UND Verknüpfungen verbieten dieses vertauschen jedoch, alle Vorkommen des quantifizierten Events müssen also sequentiell und lückenlos sein.

Es gibt verschiedene Sprachen [7,8,9,10,11], welche dazu benutzt werden, verschiedene Kombinationen von Operationen auf Datenbanken auszuführen. Leider sind sie aufgrund ihrer Eingeschränktheit nicht in der Lage SES Muster auszudrücken.

3. EINE WEB-BASIERTE GUI FÜR SESPM

3.1 Überblick

Die grafische Oberfläche für “Sequenced Event Set Pattern Matching“ wurde als Webapplikation realisiert. Nach einer kurzen Einführung zum Thema SESPM demonstriert sie in 5 einfachen Schritten die Funktionsweise des Algorithmus. Während dieser Demonstration kann der Benutzer den Verlauf interaktiv beeinflussen, das heißt er kann SESPM verwenden während er der Funktionsweise näher gebracht wird. Ein neuer Benutzer kann sich also während der Ausführung in Ruhe mit dem Thema befassen, während ein fortgeschrittener Benutzer das Programm einfach und unkompliziert für seine Zwecke verwenden kann. Die fünf Schritte laufen wie folgt ab:
Schritt 1: Zuerst wird es dem Benutzer ermöglicht, die Daten in der Datenbank zu sichten.
Schritt 2: Im Anschluss daran kann der Benutzer wählen, ob er ein Beispielmuster verwenden oder selbst ein Muster erstellen möchte. Falls er sich für die zweite Möglichkeit entscheidet, so hilft ihm die Oberfläche durch die Anzeige von Beispielen und durch verschiedene Tests auf korrekte Form ein eigenes Muster zu Erstellen.
Schritt 3: Nachdem ein geeignetes Muster gewählt wurde, führt die Webapplikation den Algorithmus damit aus und bekommt das textuelle Ergebnis, welches den endlichen Automaten und die gefilterten Events enthält, zurück. Um das Ergebnis verwenden zu können wird es zunächst zerlegt und umgewandelt. Anschließend wird mit Hilfe einer externen Software ein grafischer Automat damit erstellt. Dieser Automat wird dem Benutzer präsentiert, während alles andere in diesem Schritt nur im Hintergrund abläuft.
Schritt 4: Hier wird dem Benutzer die Anzahl der Events angezeigt, welche den Automaten passieren konnten. Jedes dieser Events wird aufgelistet und zeigt einen „Details“ Knopf an, über welchen man zu Schritt fünf gelangt.
Schritt 5: In diesem letzten Bereich der Applikation bekommt der Benutzer das gewählte Ergebnis nebst dem verwendeten Muster angezeigt. Die Webapplikation verwendet das im vorherigen Schritt gewählte Event um den Pfad, durch welchen es den Automaten passieren konnte, grafisch einzufärben. Der Automat mit gefärbtem Pfad wird dem Benutzer dann auf der Seite präsentiert. Abbildung 2 zeigt diesen Ablauf. Die Abbildungen 1-6 in Anhang 1 zeigen in absteigender Reihenfolge ein Bildschirmfoto eines jeden Schrittes.

Aus der Sicht des Benutzers ist die Webapplikation eine normale Webseite, diese verwendet im Hintergrund jedoch den SESPM Algorithmus und Graphviz um dem Benutzer die gewünschten Ergebnisse präsentieren zu können. SESPM seinerseits verbindet sich mit einer Oracle Datenbank, um Input Events einzulesen. Die Webapplikation wird den Benutzern mit Hilfe eines Apache Servers (XAMPP) online zugänglich gemacht, verwendet also übliche Webstandards zur Kommunikation (HTTP über TCP/IP). Der Benutzer kann folglich mit Hilfe eines handelsüblichen Webbrowsers darauf zugreifen. Auf dem Apache Server liegen alle benötigten PHP bzw. HTML und CSS Dateien, welche zur Darstellung der Applikation nötig sind. Des Weiteren liegen noch Dateien mit Beispielmustern und solche mit vorgecachten Ergebnissen auf dem Server. Dieser läuft auf einem Rechner welcher nebenbei auch noch den ausführbaren C Algorithmus enthält und auf welchem eine Software namens Graphviz installiert ist. Diese Software wird zum zeichnen der Graphen verwendet. Die Aufrufe dieser beiden Komponenten geschehen direkt aus dem PHP Code heraus. Der C Algorithmus verwendet eine auf dem Rechner hinterlegte Datei mit Verbindungsdaten um sich mit einer Oracle Datenbank zu verbinden, welche auf einem Server der Freien Universität Bozen liegt und alle benötigten Input Events enthält. Abbildung 1 zeigt die Architektur der Webapplikation.

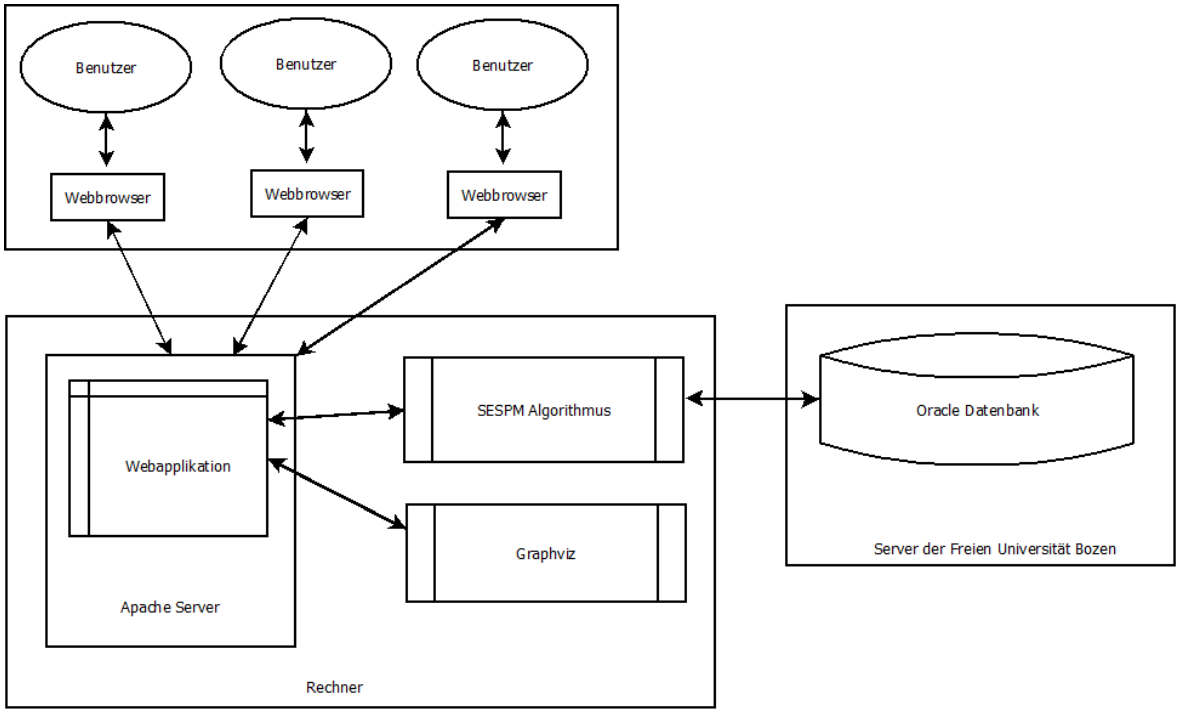

Abbildung 1: Architektur der grafischen Oberfläche

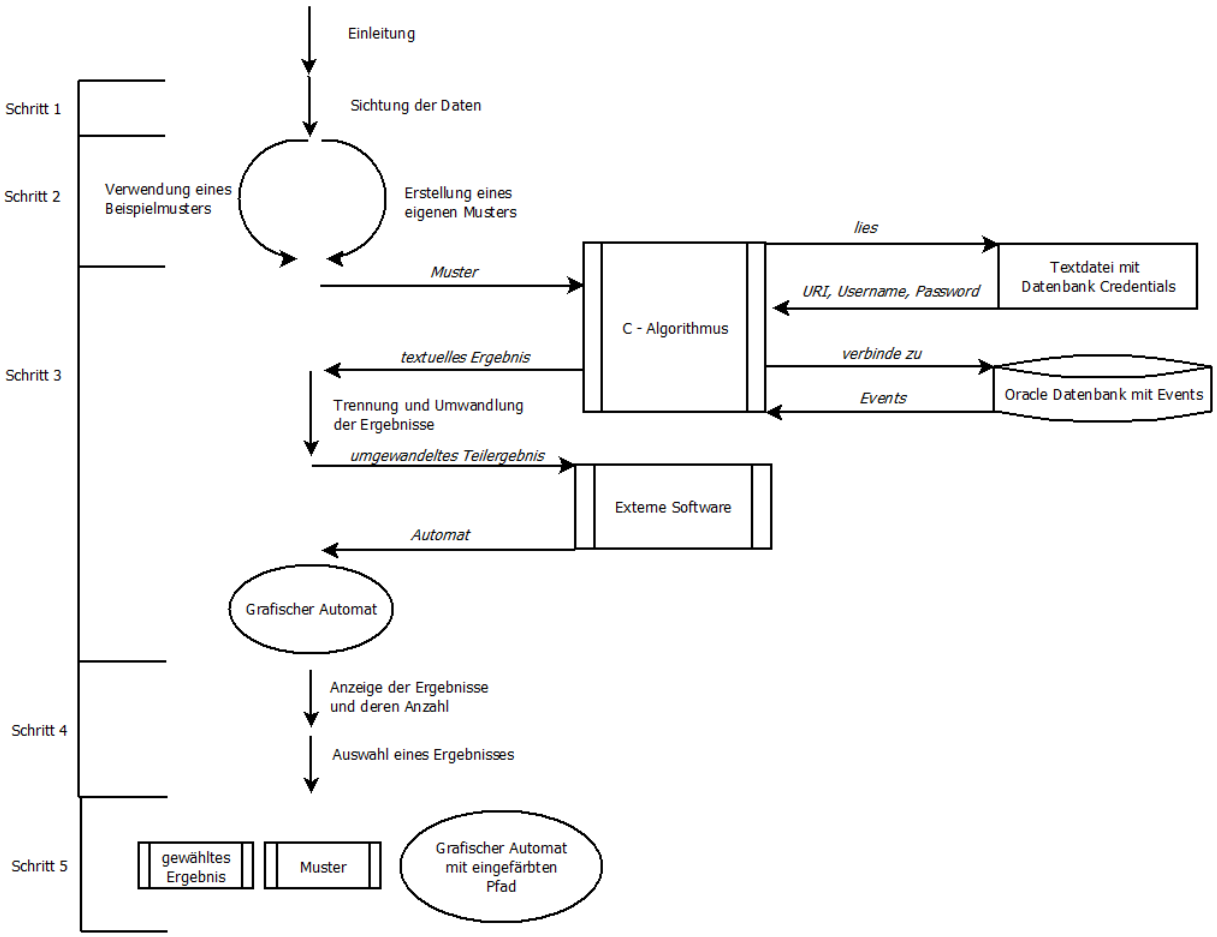

Abbildung 2: Ablauf bei der Benutzung der grafischen Oberfläche

3.2 Änderungen am originalen Code

Um die grafische Oberfläche erstellen zu können, mussten zunächst einige Änderungen am Code vorgenommen werden. Um den Original Code trotz der Änderungen beibehalten zu können (aus Gründen der Testbarkeit), wurden C-Präprozessor Befehle verwendet, welche es erlauben schnell zwischen dem Original und der Version mit den abgeänderten Funktionen zu wechseln. Der Code kann also trotz der Anbindung an die Webapplikation auch noch im Original ausgeführt werden, was vor allem bei zukünftigen Arbeiten daran ein großer Vorteil sein dürfte. Für Weiterentwicklungen stehen jetzt also in der main.c folgende Optionen zur Verfügung (bei Änderungen muss der Code natürlich neu kompiliert werden). Für die grafische Oberfläche werden standartmäßig die mit * markierten Werte verwendet.

#define FILEINPUT	0*	das Muster wird direkt als String übergeben
	1	das Muster ist in einer Datei, der Dateiname wird übergeben
#define ORACLE	0	die Anbindung an die Datenbank wird übersprungen
	1*	SESPM verwendet die Datenbank
#define LOGGING	0*	SESPM legt keine Log Datei an
	1	SESPM legt eine Log Datei an

SESPM war darauf ausgelegt, an einem einzelnen Rechner in der Konsole verwendet zu werden. Um eine Oberfläche zu schaffen, auf welcher man den Code schnell und parallel verwenden kann, mussten also zuerst einige Änderungen vorgenommen werden: Dem Algorithmus musste man im Original eine Datei mit einem Muster als Input übergeben, was jedoch nur für einzelne Aufrufe des Programmes unterstützbar ist. Um das Programm schnell und mehrmals parallel aufrufen zu können, wurde der Code so umgeschrieben, dass das Muster direkt als String übergeben werden kann. Jedes neu generierte Muster hätte ohne diese Änderung zuerst in eine Datei geschrieben werden müssen um es verwenden zu können. Diese vielen Dateiaufrufe hätten die Gesamtperformance der Webapplikation stark verschlechtert und die Serverlast erhöht. Außerdem hätte die parallele Erstellung von Dateien zu Problemen führen können, welche auf diese Weise einfach umgangen werden. Für die grafische Oberfläche ist die Logdatei des Algorithmus nicht von Nöten weshalb dessen Erstellung für die Verwendung mit der grafischen Oberfläche ausgesetzt wurde. Dies spart ein wenig Rechenlast und einen zusätzlichen Dateiaufruf für jeden einzelnen Aufruf von SESPM. Um auch ohne die Anbindung an eine Datenbank Ergebnisse erzielen zu können, wurde die Datenbankanbindung optional gemacht. Auf diese Weise kann man auch ohne eine Datenbank ein Teilergebnis (den Automaten) berechnen und anzeigen lassen, auch wenn das Endergebnis so natürlich ausbleibt. Dies ist für zukünftige Arbeiten am Code sicher hilfreich und gewährleistet darüber hinaus auch die teilweise Funktionalität der Webapplikation bei Problemen mit der Datenbank (z.B. Verbindungsprobleme). Nachdem der Code abgeändert war, konnte die Webapplikation mit einer Einführung und ihren weiteren 5 Einzelschritten erstellt werden. Im folgenden Abschnitt wird im Detail auf die einzelnen Schritte der Webapplikation eingegangen.

3.3 *Step1: Database*

Da die Erstellung und Population einer Datenbank viel Zeit in Anspruch nimmt, dabei sehr viele Fehler gemacht werden können und die Verwaltung der Namen, Daten und Zugriffsrechte für zufällige Benutzer ohne Anmeldung eine große Herausforderung und vermutlich ein Sicherheitsrisiko darstellen würde, stellt die Webapplikation eine vorgefertigte und mit vielen Daten gefüllte Datenbank zum Testen von SESPM zur Verfügung. Die Datenbank enthält über 10.000 Events mit Daten über die Behandlung von Chemotherapie Patienten. Diese Daten wurden vom Krankenhaus Meran zur Verfügung gestellt und wurden bereits für die Dissertation über SESPM [1] verwendet. Dies hat den Vorteil, dass die Ergebnisse, welche in der Dissertation beschrieben werden, gut mit den Ergebnissen dieser Oberfläche abgeglichen werden können um die korrekte Funktionsweise der Webapplikation sicherstellen zu können.

Die Oberfläche erklärt dem Benutzer kurz was er über die Datenbank wissen muss und gibt ihm die Möglichkeit, die Daten einzusehen. Da ein Benutzer die Daten möglicherweise gern auch im späteren Verlauf der Demonstration zur Verfügung hätte, wird die Datenansicht in einem neuen Fenster geöffnet und kann somit im Hintergrund offen bleiben während der Benutzer fortfährt. Abbildung 3 zeigt einen kleinen Ausschnitt dieser Datenansicht. Gleich darunter ist eine kurze Erklärung zum Inhalt ebendieser zu sehen. Bei Schritt 1 kann der Benutzer folglich bis auf die Sichtung der Daten momentan nicht viel machen, hier besteht jedoch viel Potential für ein zukünftiges Projekt. (siehe „Zukünftige Arbeit“ weiter unten in diesem Dokument) Der Benutzer hat von diesem Punkt aus zwei verschiedene Möglichkeiten um mit dem nächsten Schritt fortzufahren: er kann entweder ein Muster aus einigen vorgefertigten Beispielen wählen oder selber interaktiv ein Muster zusammenstellen.

Database: Chemotherapy
Entries: 10017
Row names: EID, PID, L, V, U, T

2785, 637, Emocromo_Ct_Mch, 55.3, Not_reported, 113443208
2786, 637, Emocromo_Ct_Hb, 23.4, Not_reported, 113443209
2787, 637, Emocromo_Ct_Mcv, 167.2, Not_reported, 113443210
2788, 637, Emocromo_Ct_Wbc, 13.7, Not_reported, 113443211
2789, 637, Emocromo_Ct_Mchc, 66.1, Not_reported, 113443212
2790, 637, Emocromo_Ct_Plt, 535, Not_reported, 113443213
2791, 637, Emocromo_Ct_Hct, 70.6, Not_reported, 113443214
2798, 637, Emocromo_Ct_Rbc, 8.45, Not_reported, 113443215
2799, 637, Elettroforesi_Alfa_2, 22, Not_reported, 113443216
2800, 637, Elettroforesi_Gamma, 74, Not_reported, 113443217
2801, 637, Creatinina, 1.72, Not_reported, 113443218
2802, 637, Elettroforesi_Albumina, 93.7, Not_reported, 113443219
2803, 637, Elettroforesi_Rapporto_A/G, 1.57, Not_reported, 113443220
2804, 637, Gpt, 19, Not_reported, 113443221
2805, 637, Elettroforesi_Beta_Tot, 18, Not_reported, 113443222
2806, 637, Elettroforesi_Beta2, 8.3, Not_reported, 113443223
2807, 637, Elettroforesi_Proteine_Totali, 17.1, Not_reported, 113443224

Abbildung 3: Ausschnitt der Datenansicht

Jeder Eintrag, also jede Zeile, entspricht einem Event. Es besteht aus einer Event ID (EID), einer Patienten ID (PID), einem Event Typen (L), dem Wert (V) mit der zugehörigen Einheit (U) und dem Zeitpunkt des Events (T). Die verwendete Oracle Datenbank liegt momentan auf dem Universitätsserver der Freien Universität Bozen. Die Datentypen der einzelnen Einträge sind die folgenden:

EID	NUMBER(10)
PID	NUMBER(10)
L	VARCHAR2(100)
V	NUMBER(11,3)
U	VARCHAR2(100)
T	NUMBER(15)

3.4 Step2: Pattern

3.4.1 Möglichkeit 1: Beispielmuster verwenden

Um SESPM demonstrieren zu können, wurden einige Beispiele für Muster bereitgestellt. Dies hat einige verschiedene Vorteile. Ein Muster auszuwählen geht zum Beispiel weitaus schneller als eines selbst zu erstellen; für eine kurze Demonstration ist dies also ein großer Vorteil. Vorgefertigte Muster haben bei der Verwendung mit SESPM natürlich stets dasselbe Ergebnis; daher kann dieses Ergebnis zwischengespeichert (gecacht) werden um die Performance zu erhöhen; vorgefertigte Muster müssen also nicht jedes Mal aufs Neue berechnet werden, das zwischengespeicherte Ergebnis kann einfach sofort dargestellt werden. Da die meisten Benutzer vermutlich ein solches vorgefertigtes Muster verwenden werden, sinkt die Rechenlast auf dem Server durch diese Form von Zwischenspeichern also beträchtlich. Ein Muster zu erstellen erfordert Vorkenntnisse über den korrekten Aufbau ebendieses; ein Muster auszuwählen ist logischerweise ohne jegliches vorheriges Wissen durchführbar. Vorgefertigte Muster wurden außerdem vorher oft getestet und senken daher die Fehleranfälligkeit stark. Selbst gefertigte Muster sind im Gegenzug logischerweise viel fehleranfälliger.

Die technische Seite dieser bereitgestellten Beispielsmuster funktioniert wie folgt: Vorgefertigte Muster werden einfach als Textdateien im „examples“ Ordner der Webapplikation gespeichert. Die Webapplikation liest diesen Ordner live aus und zeigt jedes gefundene Muster als wählbares Beispiel an. Vorgespeicherte Ergebnisse für die abgelegten Muster werden als Dateien im Ordner „resultCache“ abgespeichert. Falls ein Muster aus dem „examples“ Ordner gewählt wird, sucht die Webapplikation nach einer Datei desselben Namens im „resultCache“ Ordner und verwendet diese anstatt das Muster durch den SESPM Algorithmus zu schicken. In Abbildung 8 kann man sehen, wie sich der normale Programmablauf (Abbildung 2) dadurch ändert. Falls keine solche Datei gefunden wurde, wird SESPM für das Ergebnis live ausgeführt. Dies wäre also ein Beispielmuster ohne vorgespeichertes Ergebnis. Für selbst erstellte Muster wird der Code klarerweise immer live ausgeführt. Abbildung 4 zeigt einige solche vorgefertigte Muster.

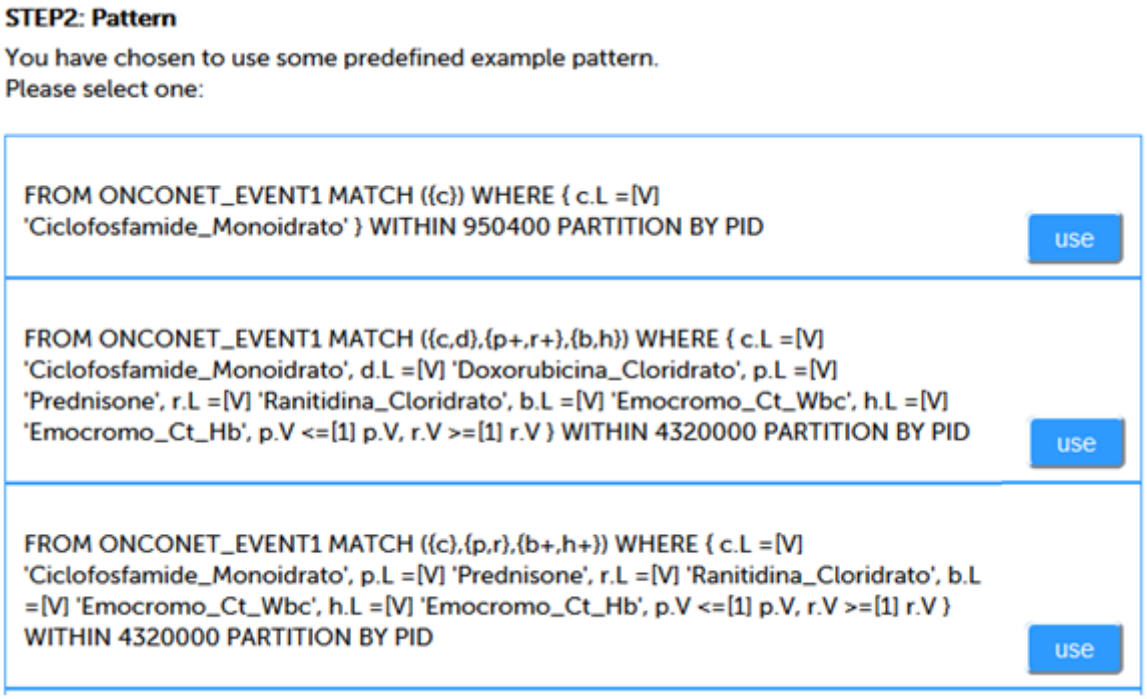

Abbildung 4: vorgefertigte Beispielmuster

3.4.2 Möglichkeit 2: Muster selbst erstellen

Die zweite Möglichkeit, welche eher etwas für fortgeschrittene Benutzer ist, erlaubt es individuelle Muster zu erstellen. Zuerst kann der Benutzer die komplexe Datenbankabfrage zusammen mit einer zeitlichen Beschränkung eingeben (Abbildung 5) um im Anschluss verschiedene Konditionen zur Abfrage hinzuzufügen (Abbildung 6).

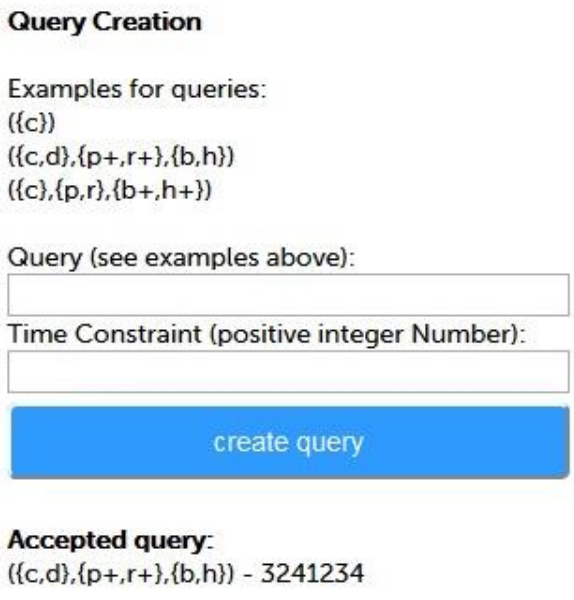

Abbildung 5: Erstellung eines Musters

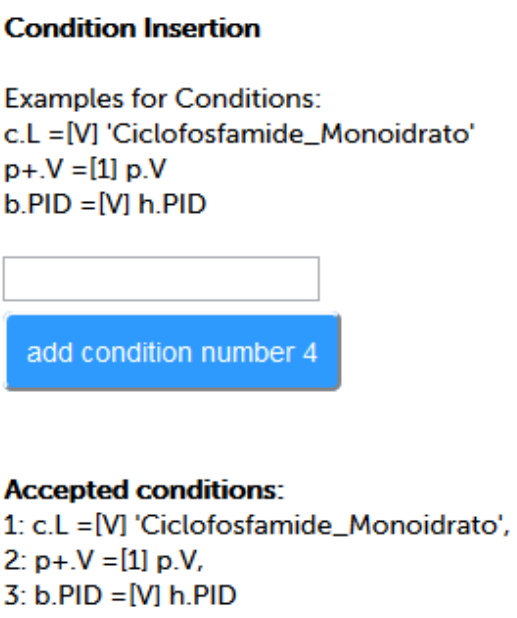

Abbildung 6: Einfügen von Bedienungen

Um den Benutzer vor syntaktischen Fehlern zu bewahren, wird jede Eingabe durch die Verwendung von regulären Ausdrücken kontrolliert.

3.4.3 Muster und zeitliche Einschränkung

Die Eingabe wird zunächst auf ihre Eigenschaften überprüft. Die zeitliche Einschränkung darf nur aus einer Zahl bestehen, welche durch die maximale Größe von „unsigned long long“ in C limitiert wird. Die Zahl muss außerdem größer als 0 sein und das Feld darf natürlich auch nicht einfach leer gelassen werden. Das Muster selbst darf bis zu 20 Variablen haben. Dieses Limit wurde eingeführt, da der Automat ansonsten einfach viel zu groß und komplex würde um damit arbeiten zu können. Abschließend wird das Muster auf eine korrekte Form überprüft. Dies geschieht mit Hilfe vom folgenden regulären Ausdruck welcher alle erlaubten Eingaben für selbst erstellte Muster abdecken sollte:

({([a-zA-Z]+[+|*|?]?)(,([a-zA-Z]+[+|*|?]?))*})

Dieser reguläre Ausdruck beschreibt, dass jede Klammer eines jeden Typs gleich oft geöffnet wie geschlossen werden muss und das Muster nur aus folgenden Elementen besteht. Entweder Buchstaben, welchen möglicherweise ein +, ein ? oder ein * folgt, weitere, optionale, durch Beistrichen getrennte Buchstaben, welche ebenso von einem der 3 Quantifizierungszeichen gefolgt werden können oder geschwungene Klammern, welche einen oder mehrere solcher Buchstaben einschließen können (werden durch Beistriche von anderen solchen Elementen getrennt). Einige Beispiele für erlaubte Muster wären folglich:

({c})
({c,d},{p+,r+},{b,h})
({c},{p,r},{b+,h+})

Um Unerfahrenen Benutzern zu helfen, liegen jeder Eingabe einige Beispiele bei (Abbildung 5 und Abbildung 6)

<u>Konditionen:</u>

Neue Konditionen werden zunächst auf ihre korrekte Form überprüft. Diese kann jedoch sehr unterschiedlich sein. Es folgt eine Liste von verschiedenen korrekten Formen für Konditionen:

- Buchstabe(n) . Buchstabe(n) Symbol Buchstabe(n) . Buchstabe(n)
- dasselbe mit voranstehendem Vergleichsoperator
- Buchstabe(n) . Buchstabe(n) Symbol ` Buchstabe(n) `
  (man beachte die einzelnen Anführungszeichen)
- dasselbe mit voranstehendem Vergleichsoperator
- Buchstabe(n) . Buchstabe(n) Symbol [V] ` Buchstabe(n) `
  (man beachte, dass auch ein _ vorkommen kann)
- Buchstabe(n) . Buchstabe(n) Symbol [P] ` Buchstabe(n) `
  (man beachte, dass auch ein _ vorkommen kann)

Um all diese Möglichkeiten gut leserlich umsetzen zu können, wurde folgende Lösung gefunden: Zuerst werden die verschiedenen Teile abstrahiert.

$letters	[a-zA-Z_] +	(ein beliebiger Buchstabe)
$lettersSurrounded	' [a-zA-Z_] + '	(ein beliebiger Buchstabe innerhalb zweier Hochkommas)
$symbol	<=|>=|<>|=|<|>	(eines der durch | getrennten Zeichen)
$quantifier	+ | * | ?	(ein Quantifizierer)
$preparam	[V] | [1]	(ein Vergleichsoperator)
$dot	.	(ein Punkt)

Anschließend werden der Teil vor dem Symbol und der Teil nach dem Symbol getrennt voneinander abgeglichen:

Abgleich vor dem Symbol:

$letters $quantifier? $dot $letters $symbol

Abgleich nach dem Symbol (die 2 Zeilen stehen für die beiden Möglichkeiten):

$letters $quantifier? $dot $letters
$preparam? $lettersSurrounded

Die gute Leserlichkeit ist deshalb wichtig, da sich diese Formen in neueren Versionen von SESPM bereits geändert haben und eine schnelle und einfache Umstellung und Anpassung an ebendiese ermöglicht werden soll. Nur wenn die Syntax beider Teile stimmt und der Benutzer alle benötigten Werte ausgefüllt hat, erscheint das manuell erstellte Muster auch als Schaltfläche, welche man zum Fortfahren anklicken kann (Abbildung 7). Logische Fehler bei der Erstellung eines Musters können auf diese Weise natürlich nicht kontrolliert werden weshalb das manuelle Erstellen von Mustern auch nur für fortgeschrittene Benutzer zu empfehlen ist, welche bereits wissen wie ein Muster aussehen sollte und selbst auf diese möglichen Probleme achten können. Als kleine Hilfestellung sind neben jeder Eingabemöglichkeit kleine Beispiele angeführt.

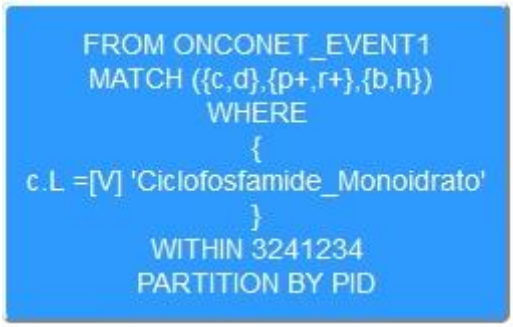

Abbildung 7: Button zum fortfahren mit selbst erstelltem Muster

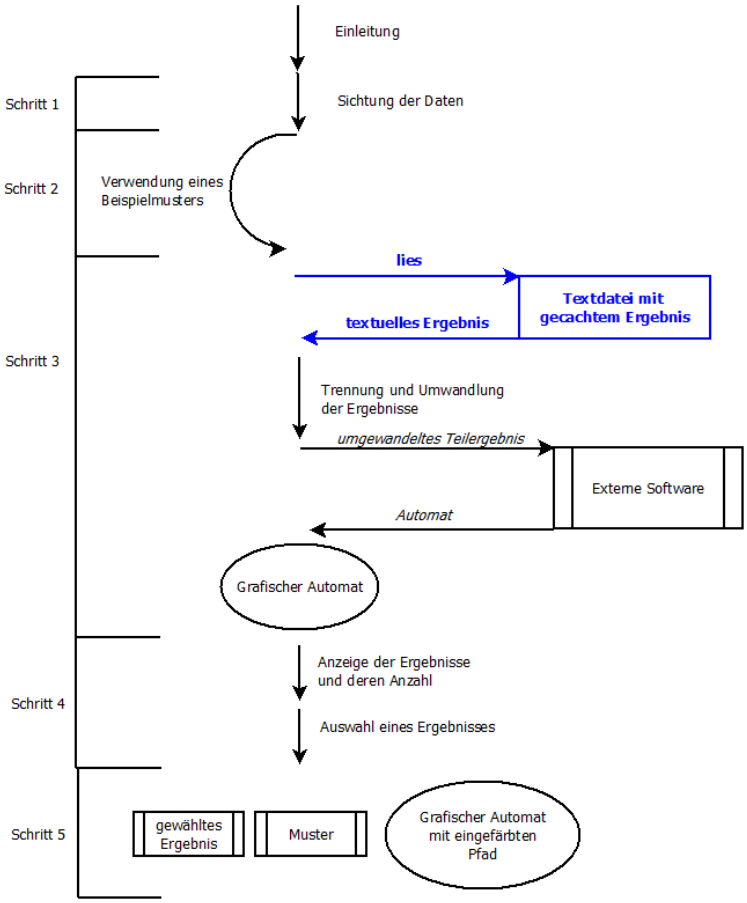

Abbildung 8: Ablauf bei der Benutzung der grafischen Oberfläche mit vorgecachten Beispielen

```
FROM ONCONET_EVENT1 MATCH ({c,d},{p+,r+},{b,h})
WHERE
c.L =[V] 'Ciclofosfamide_Monoidrato'
d.L =[V] 'Doxorubicina_Cloridrato'
p.L =[V] 'Prednisone'
r.L =[V] 'Ranitidina_Cloridrato'
b.L =[V] 'Emocromo_Ct_Wbc'
h.L =[V] 'Emocromo_Ct_Hb'
p.V <=[1] p.V
r.V >=[1] r.V
WITHIN 4320000 PARTITION BY PID
```

Abbildung 9: Beispielmuster, welches in den folgenden Schritten verwendet wurde

3.5 Step3: Automaton

Nachdem ein geeignetes Muster gewählt wurde, braucht es ein Resultat von SESPM um fortfahren zu können. Wie bereits beschrieben, wird dieses bei vorgespeicherten Beispielen aus einer Datei entnommen. Falls keine solche Datei existiert, was zum Beispiel bei manuelle erstellten Mustern immer der Fall ist, so wird SESPM live ausgeführt und das Muster daran übergeben. Das Resultat besteht aus zwei aufeinanderfolgenden Teilen: Der erste Teil des Resultates, welches man bekommt, ist ein Automat welcher dem Muster entspricht. Dieser Automat ist in Textform und muss zuerst vom restlichen Resultat getrennt werden. Der zweite Teil besteht aus den Resultaten, welche den Automaten passieren konnten. Er wird erst in Step4 verwendet. In Step3 wird folglich mit dem ersten Teil gearbeitet: Der textuelle Automat besteht hauptsächlich aus aufeinanderfolgenden Teilen die wie folgt aussehen (die Zeilennummern von 1 bis 6 sind nur zum Referenzieren eingefügt worden, das Beispiel ist ein Teil des textuellen Graphen des Beispielmusters in Abbildung 9):

```
1    STATE START
2    variables:
3    { c (0), d (1), p (2) }
4    transitions:
5    { ( { c (0), d (1), p (2), r (3) }, r (3), { r.L =[V] 'Ranitidina_Cloridrato' }, ) }
6    STATE END
```

Der textuelle Automat muss zuerst in eine Form gebracht, in welcher er als Input für eine Software namens Graphviz verwendet werden kann. Graphviz ist eine Open Source Software für die grafische Darstellung von Diagrammen [12]. Diese Input Form nennt sich DOT Sprache. DOT ist eine einfach gehaltene, aber mächtige Beschreibungssprache für die visuelle Darstellung von Graphen [13]. Das Umwandeln geschieht mit Hilfe von PHP String Befehlen und funktioniert wie folgt: Zeile 1 und 6 werden nur zur Abgrenzung der Teile verwendet. Zeile 2 und 4 werden zur Orientierung verwendet, das heißt um die einzelnen Elemente voneinander abzugrenzen. Zeile 3 entspricht dem Ausgangspunkt der Transition, die Zahlen, Beistriche und Klammern werden entfernt (obiges Beispiel: „cdp“).Zeile 5 kann theoretisch mehrere Elemente enthalten, es läuft eine Schleife darüber die jedes davon als einzelnen Teil abarbeitet. Jeder der Teile wird wie folgt umgewandelt: Der erste Teil innerhalb der geschwungenen Klammern ist das Ziel der Transition, der Teil in der Mitte ist die Variable und der Teil am Ende (wieder in geschwungenen Klammern) sind die Konditionen für die Variable(natürlich werden auch hier wieder die überflüssigen Teile wie Zahlen usw. entfernt, jedoch nicht die wichtigen wie die Vergleichsoperatoren). Obiges Beispiel sieht nach dieser Umwandlung in die DOT Sprache wie folgt aus:

```
cdp->cdpr[label ="r, {r.L =[V] 'Ranitidina_Cloridrato'}"];
```

Nachdem die Webapplikation den Input für Graphviz erstellt hat, gibt er ihn an das Programm weiter. Da Graphviz strikt eine Datei als Input verlangt, muss der Input zuerst in eine solche geschrieben werden. Um die parallele Ausführbarkeit der Webapplikation trotzdem weiterhin gewährleisten zu können, wird beim Erstellen darauf geachtet, einen Dateinamen zu verwenden, welcher noch nicht existiert. Falls nämlich zwei oder mehrere Benutzer denselben Dateinamen verwenden würden, würde die Datei des letzten Benutzers die der vorherigen überschreiben. Die vorherigen Benutzer würden also ein falsches Ergebnis bekommen. Der Pseudo-Code für diesen Vorgang ist unterhalb dieses Absatzes zu finden. Graphviz erstellt dann mit Hilfe der Datei einen Graphen, welcher direkt an die Webapplikation übergeben wird. Auf der Webapplikation wird dieser dann dem Benutzer präsentiert (Abbildung 10 – verwendet Beispielmuster aus Abbildung 9).

Pseudocode für die Erstellung einer nicht vorhandenen Datei:

```
Nummer = 0
Solange Datei ( „DOTINPUT“ + Nummer ) existiert
        Erhöhe Nummer
Dateiname = „DOTINPUT“ + Nummer
Schreibe Datei mit Dateinamen
```

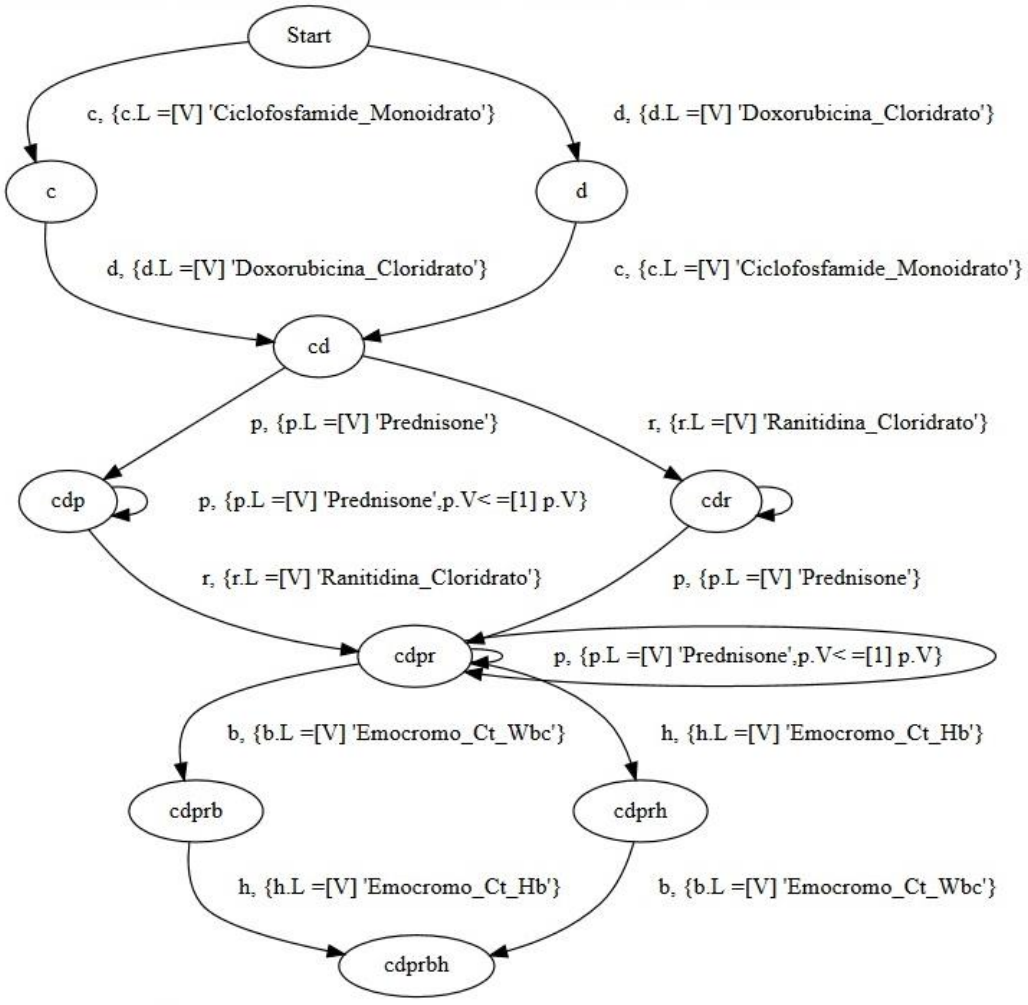

Abbildung 10: mit Hilfe von Graphviz gezeichneter Automat

Folgende Punkte sollten dabei beachtet werden: Alle Informationen welche vom textuellen Graphen beschrieben werden, werden auch in der grafischen Version angezeigt. Auf den Transitionen (dargestellt als Pfeile) sieht man die Variablen, welche mit einem Komma von den Konditionen der Variablen getrennt sind. Falls eine Variable mehrere Konditionen hat, so werden diese durch Kommas getrennt. Looping Transitionen werden als Transitionen von Elementen auf sich selbst dargestellt. Leider kann man Graphviz bei der Erstellung des Graphen nur bedingt beeinflussen, daher ist es leider nicht möglich die Position der verschiedenen Elemente zu beeinflussen um die Lesbarkeit weiter zu erhöhen, diese kann also nur durch Symbole und Abstände im Text geregelt werden. Um die Grafik im Dokument größer und damit leserlicher darstellen zu können wurden 2 Transitionen (ganz rechts) ihrer Variablen und Konditionen beraubt, dies ist also kein Fehler und wird auf der Webapplikation korrekt dargestellt.

3.6 Step4: Results

Der vierte Schritt ist das Präsentieren der Ergebnisse, welche von SESPM geliefert wurden. SESPM schickt alle Daten der Datenbank durch den erstellten Automaten und liefert alle jene zurück, welche hindurch kommen. Sie sind im zweiten Teil des textuellen Resultates welches wir in Step3 erhalten haben zu finden. Je nach gewähltem Muster (in diesem Fall das Beispielmuster aus Abbildung 9) wird also eine Reihe von Daten präsentiert, welche ebendiesem entsprechen (Abbildung 11).

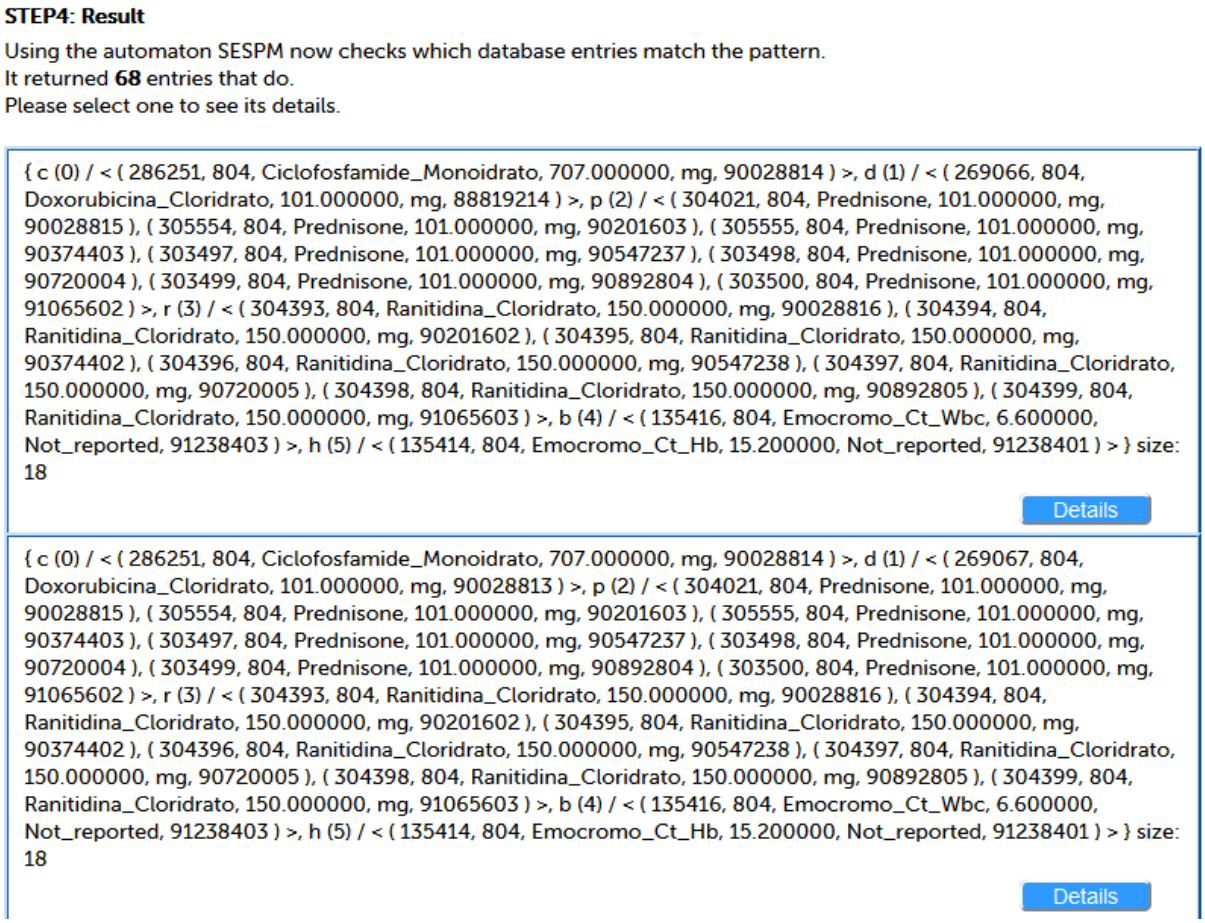

Abbildung 11: Ausschnitt der Ergebnisse des Beispielmusters aus Abbildung 9

Wie man sieht liefert das Beispielmuster 68 Ergebnisse, welche den Automaten passieren konnten. Jedes davon wird als Schaltfläche dargestellt, welche dazu dient, dessen Details einzusehen. Wählt ein Benutzer ein Ergebnis, so kommt er zu Step5.

3.7 Step5: Details

Wenn der Benutzer in Schritt 4 ein Ergebnis wählt, dann kann er im letzten Schritt dessen Details übersichtlich einsehen (Abbildung 12). Alle der folgenden Bilder wurden mit Hilfe des Beispielmusters aus Abbildung 9 erstellt.

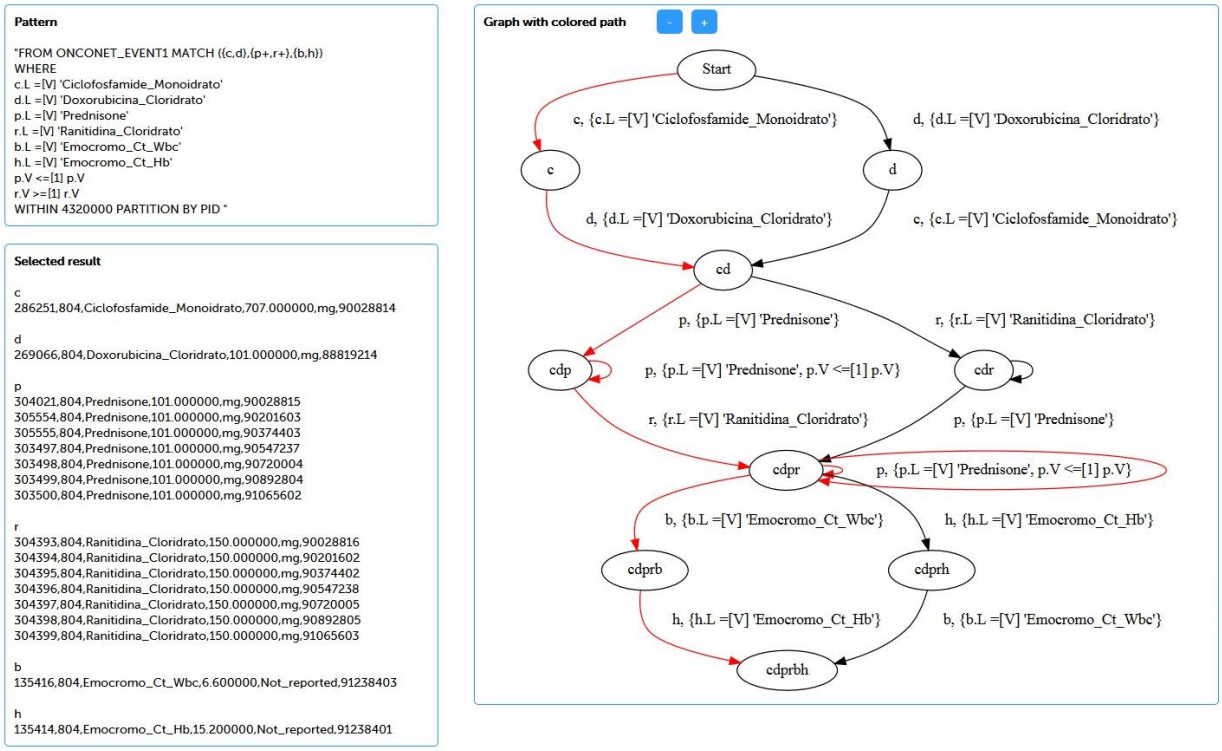

Abbildung 12: Seitenansicht des letzten Schrittes

Um diesen Schritt besser analysieren zu können, wird die Abbildung 12 in drei Teile zerlegt. Der erste Teil (Abbildung 13), welcher das vorher gewählte Muster übersichtlich anzeigt, dient dem Zwecke den Zusammenhang zwischen dem Muster und dem grafischen Automaten direkt einsehen zu können. Sowohl das Muster selbst als auch seine Konditionen werden hier übersichtlich dargestellt. Auf Abbildung 14 kann man sehen, dass das Ergebnis sämtliche Konditionen einhält (z.B. ist jedes c ein „Ciclofosfamide_Monoidrato“). Dieser zweite Teil zeigt das gewählte Ergebnis in übersichtlicher Formatierung. Dies dient dem Zwecke, es am nebenstehenden grafischen Automaten verfolgen zu können. Man sieht das gewählte Ergebnis und die einzelnen Events, welche dazu gehören. Außerdem ist sofort ersichtlich, welcher Event durch welchen Teil des Automaten geflossen ist. Wenn man sich das Muster vor Augen führt sieht man sofort und intuitiv den Zusammenhang (Reihenfolge und Quantifizierung der Ergebnisse entsprechen dem Muster).

Pattern

```
"FROM ONCONET_EVENT1 MATCH ({c,d},{p+,r+},{b,h})
WHERE
c.L =[V] 'Ciclofosfamide_Monoidrato'
d.L =[V] 'Doxorubicina_Cloridrato'
p.L =[V] 'Prednisone'
r.L =[V] 'Ranitidina_Cloridrato'
b.L =[V] 'Emocromo_Ct_Wbc'
h.L =[V] 'Emocromo_Ct_Hb'
p.V <=[1] p.V
r.V >=[1] r.V
WITHIN 4320000 PARTITION BY PID "
```

Abbildung 13: formatierte Ausgabe des gewählten Musters

Selected result

```
c
286251,804,Ciclofosfamide_Monoidrato,707.000000,mg,90028814

d
269066,804,Doxorubicina_Cloridrato,101.000000,mg,88819214

p
304021,804,Prednisone,101.000000,mg,90028815
305554,804,Prednisone,101.000000,mg,90201603
305555,804,Prednisone,101.000000,mg,90374403
303497,804,Prednisone,101.000000,mg,90547237
303498,804,Prednisone,101.000000,mg,90720004
303499,804,Prednisone,101.000000,mg,90892804
303500,804,Prednisone,101.000000,mg,91065602

r
304393,804,Ranitidina_Cloridrato,150.000000,mg,90028816
304394,804,Ranitidina_Cloridrato,150.000000,mg,90201602
304395,804,Ranitidina_Cloridrato,150.000000,mg,90374402
304396,804,Ranitidina_Cloridrato,150.000000,mg,90547238
304397,804,Ranitidina_Cloridrato,150.000000,mg,90720005
304398,804,Ranitidina_Cloridrato,150.000000,mg,90892805
304399,804,Ranitidina_Cloridrato,150.000000,mg,91065603

b
135416,804,Emocromo_Ct_Wbc,6.600000,Not_reported,91238403

h
135414,804,Emocromo_Ct_Hb,15.200000,Not_reported,91238401
```

Abbildung 14: formatierte Ausgabe des Selektiertes Resultates

Der wichtigste Teil des letzten Schrittes ist der grafische Automat, auf dem der Pfad, welchen das gewählte Ergebnis verfolgt hat um den Automaten zu durchlaufen, farblich markiert ist (Abbildung 15). Auch hier kann man sofort einen Zusammenhang zu den vorherigen beiden Abbildungen sehen. Dieser letzte Schritt zeigt also das gewählte Muster mit zugehörigem Automaten und das gewählte Ergebnis welches auf dem Automaten farblich markiert ist. Dies ist eine kleine Übersicht über den Zusammenhang zwischen den einzelnen Teilen.

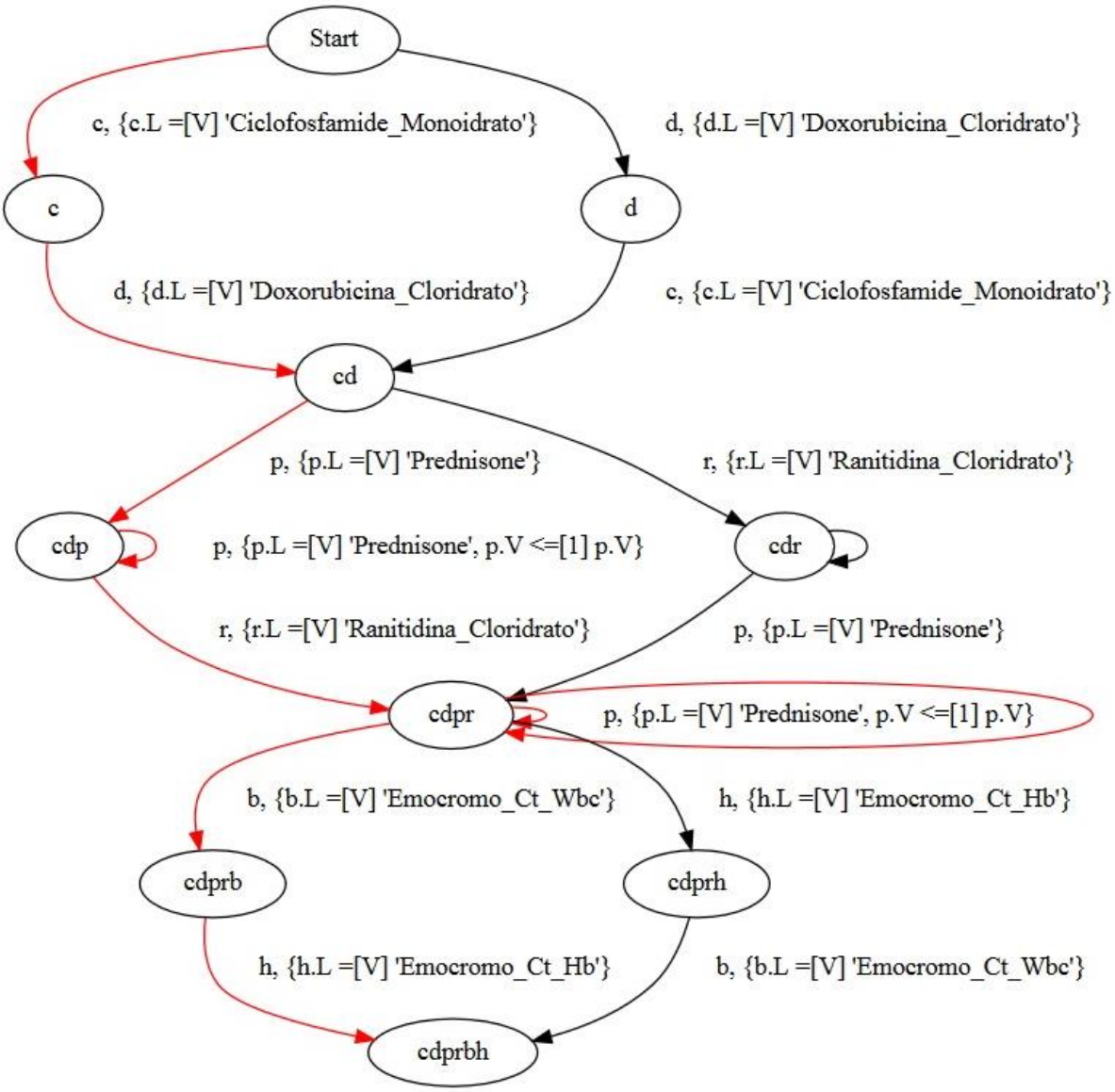

Abbildung 15: Graph des Automaten mit eingefärbtem Weg des selektierten Resultates

Um den Pfad farblich hervorheben zu können, muss der Graph zuerst in DOT Sprache verändert und dann mit Hilfe von Graphviz neu erstellt werden. Um zu vergleichen, durch welche Transition das Resultat geflossen ist, wird einfach jeder Teil des Resultates (c, d, p, r, b, h in diesem Fall) mit den Teilen des Automaten verglichen und bei Übereinstimmung farblich hervorgehoben. Falls ein Teil des Resultates mehrmals vorkommt, dann bedeutet das dass eine looping Transition verwendet wird und diese wird folglich auch farblich markiert. Das farbliche markieren einer Transition wird einfach durch das Ersetzen der öffnenden eckigen Klammer „[„ in der DOT Sprache durch „[color=red “ erreicht.

Obiges Beispiel in DOT Sprache vor der farblichen Markierung:

```
cdp->cdpr[label ="r, {r.L =[V] 'Ranitidina_Cloridrato'}"];
```

Falls das Resultat hindurchgeflossen ist, sieht es dann so aus:

```
cdp->cdpr [color=red label ="r, {r.L =[V] 'Ranitidina_Cloridrato'}"];
```

4. Schlussbemerkungen

Sequenced Event Set Pattern Matching ist ein Algorithmus zum Abgleich von Sequenzen von Eingabeevents mit komplexen Mustern. In diesem Dokument wurde die Erstellung einer grafischen Oberfläche für diesen Algorithmus besprochen. Es enthält einleitend eine kurze Erklärung zur Funktionsweise des Algorithmus und im Anschluss genaue Informationen über jeden einzelnen Schritt der zur Umsetzung der grafischen Oberfläche dafür notwendig war. Dies schließt die Datenbankanbindung der Implementierung, das Verwenden von Beispielmustern oder das Erstellen eigener Muster, die automatische Erstellung einer Grafik welche einen endlichen Automaten repräsentiert, die Präsentation der Ergebnisse und die Erstellung einer weiteren Grafik, welche einen endlichen Automaten mit eingefärbten Pfad für ein Ergebnis enthält, mit ein.

Das Projekt könnte in Zukunft in verschiedenen Richtungen weiterentwickelt werden. Das Erste und Wichtigste wäre sicher, auf die neueste Version von SESPM umzusteigen und alle Muster, Beispiele und Ähnliches daran anzupassen. Die Oberfläche könnte komplett vom Code gelöst werden indem die Änderungen die daran vorgenommen wurden durch Parser ersetzt werden. Dadurch könnte man ohne Umwege auf neue Versionen von SESPM umsteigen was große Vorteile bringen würde da aktuell noch immer aktiv am Code gearbeitet wird. Die Datenbank ist im Moment fest vorgegeben, könnte in einem zukünftigen Projekt jedoch vom Benutzer gewählt und/oder verwaltet werden. Dafür müsste natürlich eine Oberfläche bereitgestellt werden. Auch könnten verschiedene vorgefertigte Datenbanken mit entsprechenden Beispielmustern zur Verfügung gestellt werden. Man könnte dem Benutzer eine Funktionalität zum Abspeichern selbst erstellter Muster zur Verfügung stellen, damit er sie später wiederverwenden oder auch mit anderen Benutzern austauschen könnte. Natürlich müsste dafür auch eine entsprechende Import-Funktion entwickelt werden. Auch könnten die Ergebnisse zum Exportieren in verschiedene Formate bereitgestellt werden um weiteres Arbeiten damit zu ermöglichen bzw. vereinfachen.

5. REFERENZEN

[1] B. Cadonna, J. Gamper and M. H. Böhlen.
Sequenced event set pattern matching.
In EDBT, pages 33-44, 2011.

[2] B. Cadonna, J. Gamper and M. H. Böhlen.
Efficient event pattern matching with match windows.
KDD '12, Pages 471- 479, 2012.

[3] F. Zemke, A. Witkowski, M. Cherniak, and L. Colby.
Pattern matching in sequences of rows.
Technical report, 2007.

[4] N. Dindar, B. G¨u¸c, P. Lau, A. Ozal, M. Soner, and N. Tatbul.
Dejavu: declarative pattern matching over live and archived streams of Events.
In SIGMOD, pages 1023–1026, 2009.

[5] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi.
Expressing and optimizing sequence queries in database systems.
ACM Trans. Database Syst., 29(2):282–318, 2004.

[6] Y. Mei and S. Madden.
Zstream: a cost-based query processor for adaptively detecting composite events.
In SIGMOD, pages 193–206, 2009.

[7] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim.
Composite events for active databases: Semantics, contexts and detection.
In VLDB, pages 606–617, 1994.

[8] S. Gatziu and K. R. Dittrich.
Events in an active object-oriented database system.
In Rules in Database Systems, pages 23–39, 1993.

[9] N. H. Gehani, H. V. Jagadish, and O. Shmueli.
Composite event specification in active databases: Model & implementation.
In VLDB, pages 327–338, 1992.

[10] R. Meo, G. Psaila, and S. Ceri. Composite events in Chimera.
In EDBT, pages 56–76, 1996.

[11] D. Zimmer and R. Unland.
On the semantics of complex events in active database Management systems.
In ICDE, page 392, 1999.

[12] Graphviz
http://www.graphviz.org/

[13] DOT (Graphviz)
http://de.wikipedia.org/wiki/DOT_%28GraphViz%29

Anhang 1:

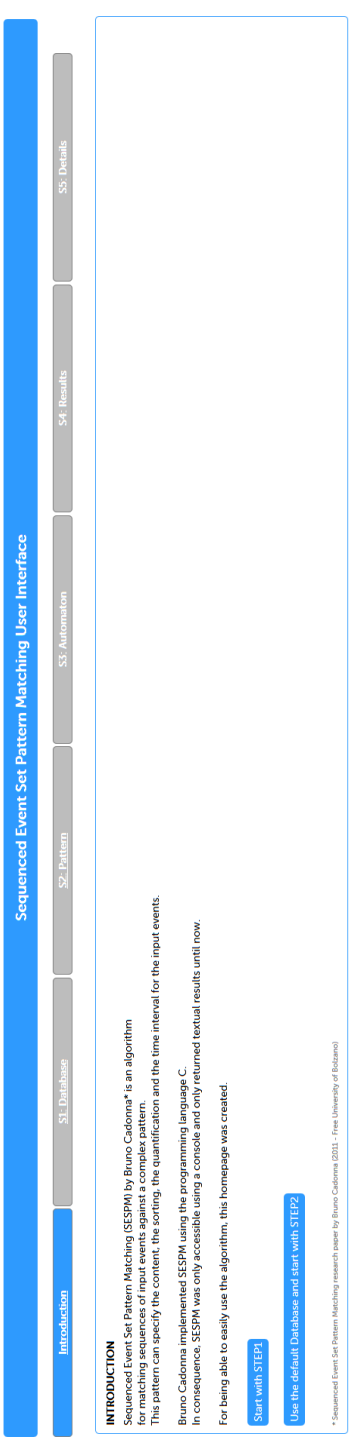

Abbildung 1

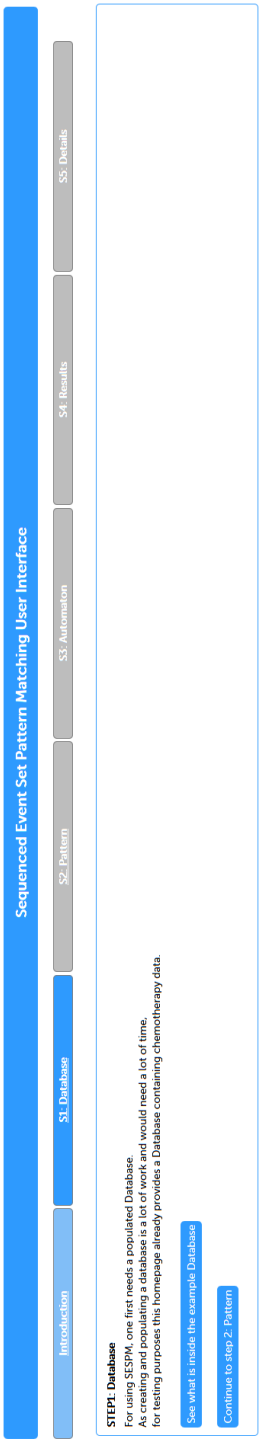

Abbildung 2

Abbildung 3

Abbildung 4

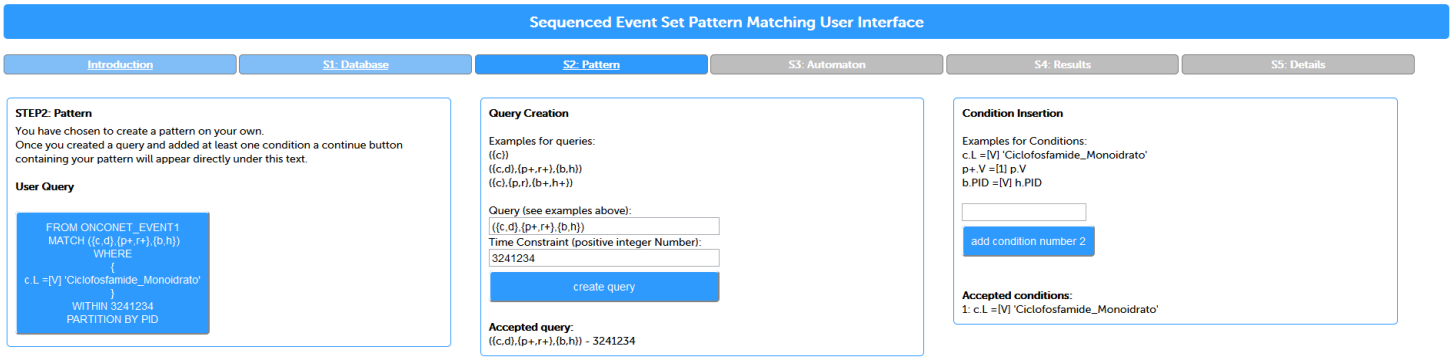

Abbildung 5

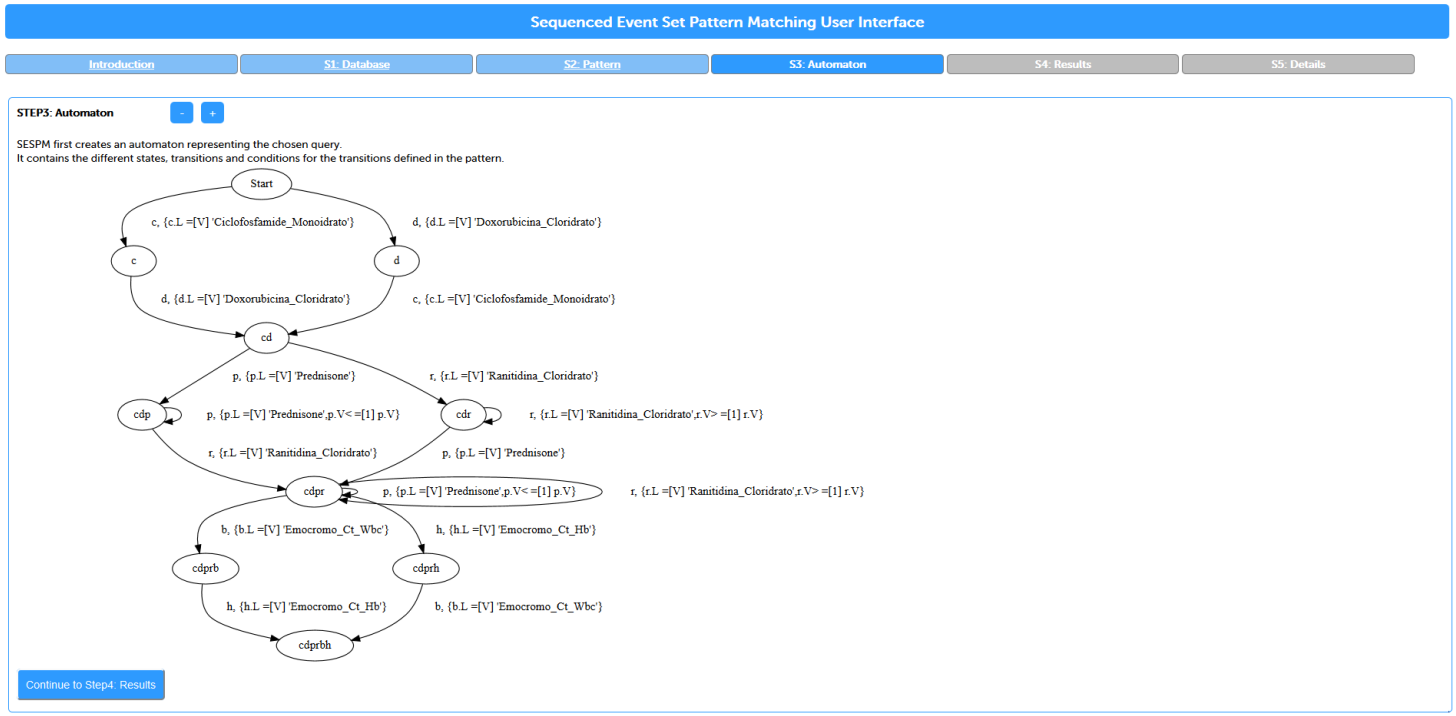

Abbildung 6

Sequenced Event Set Pattern Matching User Interface

Introduction | S1: Database | S2: Pattern | S3: Automaton | S4: Results | S5: Details

STEP4: Result

Using the automaton SESPM now checks which database entries match the pattern.
It returned **68** entries that do.
Please select one to see its details.

{ c (0) / < (286251, 804, Ciclofosfamide_Monoidrato, 707.000000, mg, 90028814) >, d (1) / < (269066, 804, Doxorubicina_Cloridrato, 101.000000, mg, 88819214) >, p (2) / < (304021, 804, Prednisone, 101.000000, mg, 90028815), (305554, 804, Prednisone, 101.000000, mg, 90201603), (305555, 804, Prednisone, 101.000000, mg, 90374403), (303497, 804, Prednisone, 101.000000, mg, 90547237), (303498, 804, Prednisone, 101.000000, mg, 90720004), (303499, 804, Prednisone, 101.000000, mg, 90892804), (303500, 804, Prednisone, 101.000000, mg, 91065602) >, r (3) / < (304393, 804, Ranitidina_Cloridrato, 150.000000, mg, 90028816), (304394, 804, Ranitidina_Cloridrato, 150.000000, mg, 90201602), (304395, 804, Ranitidina_Cloridrato, 150.000000, mg, 90374402), (304396, 804, Ranitidina_Cloridrato, 150.000000, mg, 90547238), (304397, 804, Ranitidina_Cloridrato, 150.000000, mg, 90720005), (304398, 804, Ranitidina_Cloridrato, 150.000000, mg, 90892805), (304399, 804, Ranitidina_Cloridrato, 150.000000, mg, 91065603) >, b (4) / < (135416, 804, Emocromo_Ct_Wbc, 6.600000, Not_reported, 91238403) >, h (5) / < (135414, 804, Emocromo_Ct_Hb, 15.200000, Not_reported, 91238401) > } size: 18

Details

{ c (0) / < (286251, 804, Ciclofosfamide_Monoidrato, 707.000000, mg, 90028814) >, d (1) / < (269067, 804, Doxorubicina_Cloridrato, 101.000000, mg, 90028813) >, p (2) / < (304021, 804, Prednisone, 101.000000, mg, 90028815), (305554, 804, Prednisone, 101.000000, mg, 90201603), (305555, 804, Prednisone, 101.000000, mg, 90374403), (303497, 804, Prednisone, 101.000000, mg, 90547237), (303498, 804, Prednisone, 101.000000, mg, 90720004), (303499, 804, Prednisone, 101.000000, mg, 90892804), (303500, 804, Prednisone, 101.000000, mg, 91065602) >, r (3) / < (304393, 804, Ranitidina_Cloridrato, 150.000000, mg, 90028816), (304394, 804, Ranitidina_Cloridrato, 150.000000, mg, 90201602), (304395, 804, Ranitidina_Cloridrato, 150.000000, mg, 90374402), (304396, 804, Ranitidina_Cloridrato, 150.000000, mg, 90547238), (304397, 804, Ranitidina_Cloridrato, 150.000000, mg, 90720005), (304398, 804, Ranitidina_Cloridrato, 150.000000, mg, 90892805), (304399, 804, Ranitidina_Cloridrato, 150.000000, mg, 91065603) >, b (4) / < (135416, 804, Emocromo_Ct_Wbc, 6.600000, Not_reported, 91238403) >, h (5) / < (135414, 804, Emocromo_Ct_Hb, 15.200000, Not_reported, 91238401) > } size: 18

Details

{ c (0) / < (286251, 804, Ciclofosfamide_Monoidrato, 707.000000, mg, 90028814) >, d (1) / < (269068, 804, Doxorubicina_Cloridrato, 101.000000, mg, 91238414) >, p (2) / < (303501, 804, Prednisone, 101.000000, mg, 91238415), (303502, 804, Prednisone, 101.000000, mg, 91411203), (303503, 804, Prednisone, 101.000000, mg, 91584002), (303504, 804, Prednisone, 101.000000, mg, 91756802), (303505, 804, Prednisone, 101.000000, mg, 91929602), (303506, 804, Prednisone, 101.000000, mg, 92102402), (303507, 804, Prednisone, 101.000000, mg, 92275202) >, r (3) / < (304400, 804, Ranitidina_Cloridrato, 150.000000, mg, 91238416), (303818, 804, Ranitidina_Cloridrato, 150.000000, mg, 91411204), (303819, 804, Ranitidina_Cloridrato, 150.000000, mg, 91584003), (303820, 804, Ranitidina_Cloridrato, 150.000000, mg, 91756803), (303821, 804, Ranitidina_Cloridrato, 150.000000, mg, 91929603), (303822, 804, Ranitidina_Cloridrato, 150.000000, mg, 92102403), (303823, 804, Ranitidina_Cloridrato, 150.000000, mg, 92275203) >, b (4) / < (135427, 804, Emocromo_Ct_Wbc, 6.600000, Not_reported, 92448003) >, h (5) / < (135425, 804, Emocromo_Ct_Hb, 15.200000, Not_reported, 92448001) > } size: 18

Details

{ c (0) / < (336559, 753, Ciclofosfamide_Monoidrato, 1267.500000, mg, 90547239) >, d (1) / < (253916, 753, Doxorubicina_Cloridrato, 78.000000, mgl, 90547232) >, p (2) / < (303353, 753, Prednisone, 84.500000, mg, 90633603), (303354, 753, Prednisone, 84.500000, mg, 90720003), (303355, 753, Prednisone, 84.500000, mg, 90806401), (303356, 753, Prednisone, 84.500000, mg, 90892803) >, r (3) / < (253982, 753, Ranitidina_Cloridrato, 150.000000, mg, 90633600), (253983, 753, Ranitidina_Cloridrato, 150.000000, mg, 90720001), (253984, 753, Ranitidina_Cloridrato, 150.000000, mg, 90806400), (253985, 753, Ranitidina_Cloridrato, 150.000000, mg, 90892801) >, b (4) / < (81621, 753, Emocromo_Ct_Wbc, 12.300000, Not_reported, 91324803) >, h (5) / < (81619, 753, Emocromo_Ct_Hb, 25.900000, Not_reported, 91324801) > } size: 12

Details

{ c (0) / < (336559, 753, Ciclofosfamide_Monoidrato, 1267.500000, mg, 90547239) >, d (1) / < (253917, 753, Doxorubicina_Cloridrato, 78.000000, mgl, 92361608) >, p (2) / < (303357, 753, Prednisone, 84.500000, mg, 92361611), (303358, 753, Prednisone, 84.500000, mg, 92448014), (303359, 753, Prednisone, 84.500000, mg, 92534401), (303360, 753, Prednisone, 84.500000, mg, 92620801), (303361, 753, Prednisone, 84.500000, mg, 92707201) >, r (3) / < (253986, 753, Ranitidina_Cloridrato, 150.000000, mg, 92361610), (253987, 753, Ranitidina_Cloridrato, 150.000000, mg, 92448012), (253988, 753, Ranitidina_Cloridrato, 150.000000, mg, 92534400), (253989, 753, Ranitidina_Cloridrato, 150.000000, mg, 92620800), (254010, 753, Ranitidina_Cloridrato, 150.000000, mg, 92707200) >, b (4) / < (19104, 753, Emocromo_Ct_Wbc, 12.300000, Not_reported, 93139203) >, h (5) / < (19102, 753, Emocromo_Ct_Hb, 25.900000, Not_reported, 93139201) > } size: 14

Details

{ c (0) / < (336560, 753, Ciclofosfamide_Monoidrato, 1267.500000, mg, 92361612) >, d (1) / < (253917, 753, Doxorubicina_Cloridrato, 78.000000, mgl, 92361608) >, p (2) / < (303358, 753, Prednisone, 84.500000, mg, 92448014), (303359, 753, Prednisone, 84.500000, mg, 92534401), (303360, 753, Prednisone, 84.500000, mg, 92620801), (303361, 753, Prednisone, 84.500000, mg, 92707201) >, r (3) / < (253987, 753, Ranitidina_Cloridrato, 150.000000, mg, 92448012), (253988, 753, Ranitidina_Cloridrato, 150.000000, mg, 92534400), (253989, 753, Ranitidina_Cloridrato, 150.000000, mg, 92620800), (254010, 753, Ranitidina_Cloridrato, 150.000000, mg, 92707200) >, b (4) / < (19104, 753, Emocromo_Ct_Wbc, 12.300000, Not_reported, 93139203) >, h (5) / < (19102, 753, Emocromo_Ct_Hb, 25.900000, Not_reported, 93139201) > } size: 12

Details

Abbildung 7

Sequenced Event Set Pattern Matching User Interface

Introduction | S1: Database | S2: Pattern | S3: Automaton | S4: Results | S5: Details

Return to result list to see the details of some other match

Pattern

```
"FROM ONCONET_EVENT1 MATCH ((c,d),(p+,r+),(b,h))
WHERE
c.L =[V] 'Ciclofosfamide_Monoidrato'
d.L =[V] 'Doxorubicina_Cloridrato'
p.L =[V] 'Prednisone'
r.L =[V] 'Ranitidina_Cloridrato'
b.L =[V] 'Emocromo_Ct_Wbc'
h.L =[V] 'Emocromo_Ct_Hb'
p.V <=[1] p.V
r.V >=[1] r.V
WITHIN 4320000 PARTITION BY PID "
```

Selected result

```
c
286251,804,Ciclofosfamide_Monoidrato,707.000000,mg,90028814

d
269067,804,Doxorubicina_Cloridrato,101.000000,mg,90028813

p
304021,804,Prednisone,101.000000,mg,90028815
305554,804,Prednisone,101.000000,mg,90201603
305555,804,Prednisone,101.000000,mg,90374403
303497,804,Prednisone,101.000000,mg,90547237
303498,804,Prednisone,101.000000,mg,90720004
303499,804,Prednisone,101.000000,mg,90892804
303500,804,Prednisone,101.000000,mg,91065602

r
304393,804,Ranitidina_Cloridrato,150.000000,mg,90028816
304394,804,Ranitidina_Cloridrato,150.000000,mg,90201602
304395,804,Ranitidina_Cloridrato,150.000000,mg,90374402
304396,804,Ranitidina_Cloridrato,150.000000,mg,90547238
304397,804,Ranitidina_Cloridrato,150.000000,mg,90720005
304398,804,Ranitidina_Cloridrato,150.000000,mg,90892805
304399,804,Ranitidina_Cloridrato,150.000000,mg,91065603

b
135416,804,Emocromo_Ct_Wbc,6.600000,Not_reported,91238403

h
135414,804,Emocromo_Ct_Hb,15.200000,Not_reported,91238401
```

Graph with colored path - +