ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Design und Implementierung eines Logging-Frameworks für ein objekt-zentriertes Prozessmanagementsystem

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Markus Leitz
markus.leitz@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Kevin Andrews

2015

Fassung 13. Dezember 2015

Satz: PDF-LaTeX 2ε

Kurzfassung

Logging ist in vielen Software-Systemen eine wichtige Komponente. Mit einem Logging System können Ereignisse während des Systemablaufes protokolliert und später analysiert werden. Obwohl schon seit vielen Jahren für viele verschiedene Systeme Logging Komponenten entwickelt werden, existiert bis heute kein anerkannter Standard für das Logging. Daher muss für jedes System, dass eine Logging Komponente besitzen soll, diese Komponente neu entwickelt werden und eigene Standards bezüglich des Aufbaus und des Ablaufs des Loggings definiert werden. PHILharmonicFlows ist ein Prozessmanagementsystem, das zur Zeit entwickelt wird. Als eines der ersten Projekte setzt es den objekt-zentrierten Ansatz für Prozessmanagement um.

Im Rahmen dieser Arbeit werden die Anforderungen an eine Logging Komponente für das PHILharmonicFlows System evaluiert und ein Lösungsansatz entworfen und implementiert. Da bisher keine Logging Systeme für objekt-zentrierte Prozessmanagementsysteme existieren, existieren auch keine Erfahrungswerte darüber, was bei dem Logging in objekt-zentrierten Prozessmanagementsystemen zu beachten ist. Die Verwendung von Logging Systemen anderer, aktivitäts-zentrierter, Prozessmanagementsystemen kann zu unvollständigen und unpassenden Logging führen. Die Logs wären für die spätere Verarbeitung unbrauchbar. Daher müssen die Anforderungen an die Logging Komponente von Grund auf neu erhoben werden. Der entworfene Lösungsansatz muss zudem in das bereits existierende PHILharmonicFlows System eingebunden werden.

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

In den meisten heutigen Software-Systemen ist Logging eine wichtige Komponente. Während der Laufzeit eines Systems dient das Logging dazu, bestimmte Ereignisse zu protokollieren und somit Informationen über den Systemablauf für eine spätere Analyse bereitzustellen. Obwohl die ersten Logging Systeme vor vielen Jahren entwickelt wurden und seitdem die meisten Software-Systeme eine Logging Komponente beinhalten, existiert bis heute kein allgemein anerkannter Standard für das Logging. *Common Event Expression (CEE)* war bis 2014 eine Initiative unter Leitung der MITRE Corporation, welche das Ziel hatte einen Standard für das Logging zu entwickeln. In diesem Standard sollten Vorgaben für den Aufbau und das Format von Logs enthalten sein, ebenso wie Vorgaben zur Speicherung von diesen Logs. 2014 wurde das Projekt vorzeitig beendet, da sich Sponsoren wie die U.S. Regierung zurückzogen[1]. Daher müssen noch heute bei der Entwicklung einer Logging Komponente für eine Anwendung eigene Standards definiert werden. Dies ist wichtig um eine vollständige und konsistente Analyse der Logs gewährleisten zu können.

PHILharmonicFlows[1] ist ein Prozessmanagementsystem (PrMS), das zur Zeit am Institut für Datenbanken und Informationssysteme entwickelt wird. Bisherige PrMSe basieren auf dem Ansatz des aktivitäts-zentrierten Prozessmanagements. PHILharmonicFlows ist ein System, das einen objekt-zentrierten Ansatz umsetzt und bisher noch keine Logging Komponente besaß. Da PHILharmonicFlows einer der ersten Projekte ist, das den objekt-zentrierten Ansatz umsetzt, ist das Gebiet des Loggings in objekt-

[1] Process, Humans, Information Linkage in harmonic Business Flows

zentrierten Prozessmanagementsystemen nicht sehr weit erforscht. Der Einsatz eines bestehenden Logging Frameworks, z.B. aus einem aktivitätengesteuerten PrMS, könnte unter Umständen zu einem unvollständigen und für den objekt-zentrierten Ansatz nicht passenden Logging führen.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, ein Logging-Framework für das objekt-zentrierte Prozessmanagementsystem PHILharmonicFlows zu entwickeln. Es soll neben dem eigentlichen Logging auch eine Möglichkeit geboten werden, die Logs visuell darzustellen.

Dabei geht es nicht darum sämtliche Informationen abzuspeichern, sondern die für die spätere Aufarbeitung wichtigen Informationen herauszufiltern und somit das Logging möglichst effizient und ressourcenschonend umzusetzen. Mit dem zu erstellenden Visualisierungstool sollen die im ersten Schritt entstanden Logs visuell dargestellt werden. Dies sollte möglichst übersichtlich sein, sodass der Benutzer die Zusammenhänge von verschiedenen Logs erkennen und nachvollziehen kann. Zusätzlich soll der Benutzer mit Hilfe des Tools einige statistische Werte bezüglich Anzahl und Art der Logs auslesen können. Das entstehende Visualisierungstool ist in erster Linie für die Entwickler des PHLharmonicFlows Systems gedacht. Die Entwickler sollen mit Hilfe des Visualisierungstools den Ablauf des Programmes besser Nachvollziehen und Verifizieren können.

1.3 Struktur der Arbeit

Zu Beginn werden die Grundlagen zum Thema Logging und objekt-zentriertem Prozessmanagement beschrieben. Diese sind für das bessere Verständnis der Arbeit wichtig. Zudem wird in Kapitel 2 erläutert, auf welcher technologischen Grundlage das erarbeitete Framework zustande gekommen ist.

In Kapitel 3 werden die zu Beginn der Arbeit evaluierten Anforderungen aufgezählt und beschrieben. Diese dienen in Kapitel 4 als Grundlage für die Beschreibung der umge-

setzten Implementierung. Des Weiteren werden neben den implementierten Ansätzen auch Alternativen aufgezeigt und erörtert.

Abschließend wird in Kapitel 5 der gesamt Arbeitszeitraum noch einmal reflektiert und ein Ausblick zum Thema Logging in objekt-zentrierten Prozessmanagementsystemen gegeben. Hierbei wird auch angesprochen, für was die Logs in anderen Tools genutzt werden können.

2

Grundlagen

Dieses Kapitel beschäftigt sich mit den Grundlagen, die für ein besseres Verständnis der späteren Kapitel notwendig sind. Die Grundlagen beinhalten eine Einführung in das Thema Logging, eine Erläuterung von objekt-zentrierten Prozessmanagementsystemen anhand des PHILharmonicFlows Systems und abschließend einen Einblick in das technische Umfeld, in welchem während der Implementierung gearbeitet worden ist.

2.1 Logging

Für die weitere Erklärung sind zwei Definitionen nützlich:

Logging Logging (engl. für Protokollierung) Das Sammeln und Speichern von Ereignissen bzw. deren Beschreibungen in Logeinträgen [2]

Logeintrag Ein Logeintrag, kurz Log, ist ein Protokolleintrag der beim Logging erstellt wird und die Eigenschaften und die Beschreibung eines Ereignisses beinhaltet [2]

Das Logging ist also eine Komponente eines Systems, das zur Laufzeit des Systems auf den Eintritt von bestimmten Ereignissen reagiert. Ein solches Ereignis kann das Auftreten eines Fehlers, die Ausführung einer Funktion, ein Zustandswechsel eines Objekts oder ein sonstiges Ereignis sein. Die Logging Komponente sammelt dann die Informationen dieses Ereignisses, erstellt aus diesen Informationen einen Log und speichert diesen Log ab. Ein Log besteht dann aus einem Identifikator, der den Log eindeutig identifiziert, i.d.R. eine fortlaufende Nummer und der sogenannten Lognachricht, welche die gesammelten Informationen enthält. Je nach Logging System wird die Lognachricht in verschiedene Felder aufgeteilt, in welchen die Eigenschaften der Ereignisses einzeln

gespeichert werden, oder die Informationen werden an einem Stück, z.B. in einem String gespeichert. In den meisten Logging Systemen werden die Logs anhand ihrer Lognachricht klassifiziert. In den meisten Fällen werden sie in die folgende Kategorien eingeteilt:[2]

- Information: Logs dieser Kategorie speichern Ereignisse, welche nicht durch einen Fehler ausgelöst worden sind.
- Debug: Logs dieser Kategorie speichern Ereignisse, die dem Entwickler einer Software bei dessen Arbeit helfen.
- Warnung: Logs dieser Kategorie speichern Ereignisse, welche durch einen Fehler ausgelöst worden sind. Diese Fehler haben aber keine Auswirkung auf den weiteren Verlauf des Systems.
- Fehler: Logs dieser Kategorie speichern Ereignisse, welche durch einen Fehler ausgelöst worden sind. Diese Fehler beeinträchtigen den weiteren Verlauf des Systems und können zum Absturz des Systems führen.
- Alarm: Logs dieser Kategorie speichern Ereignisse, welche vor allem bei sicherheitskritischen Systemen wichtig sind, z.B. das sich ein Benutzer eingeloggt hat, oder das ein Benutzer eine sensible Handlung ausgeführt hat (beispielsweise das Öffnen eines geheimen Dokumentes)

Obwohl eine Logging Komponente mittlerweile Bestandteil der meisten Systemen ist, gibt es noch keinen allgemein anerkannten Standard wie Logs aufgebaut sein sollten, wann und wie ein Ereignis geloggt werden sollte oder wie ein Log gespeichert werden sollte. Viele Entwickler orientieren sich daher an Systemen wie *syslog* oder *Event Log* von Microsoft. Diese geben vor, wie die Logs aufgebaut sein müssen und wo diese gespeichert werden. In Abbildung 2.1 ist ein Log zu sehen, der den Vorgaben des Event Log von Microsoft entspricht und über die Ereignisanzeige eingesehen werden kann. Auf der linken Seite befindet sich die von der Ereignisanzeige erstellte übersichtliche Darstellung, auf der rechten Seite ist der dazugehörige Strukturbaum im XML-Format zu sehen. An diesem Strukturbaum ist auch zu erkennen, welchen Aufbau ein Log für das Event Log System haben muss.

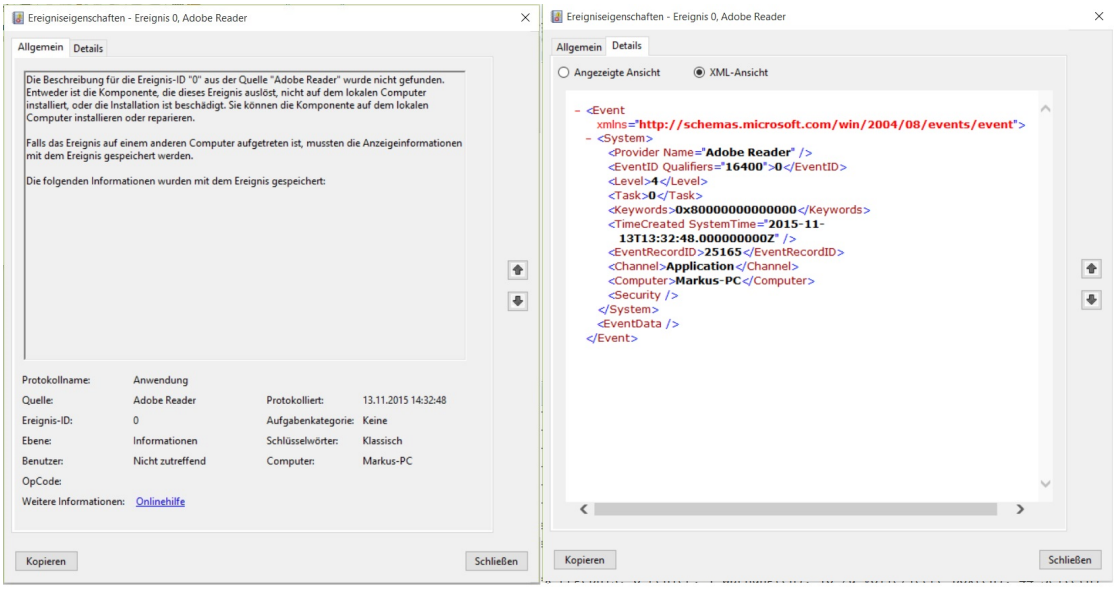

Abbildung 2.1: Darstellung eines Event Log in der Ereignisanzeige

Aufgrund des Fehlens eines Standards sind zu Beginn der Entwicklung eines neuen Logging Systems zwei Punkte zu klären und im weiteren Verlauf stets im Hinterkopf zu behalten:

1. Welchen Zweck soll das Logging später haben?
2. Wie sollen die Logs gespeichert werden?

Aus dem ersten Punkt lässt sich auch ableiten, welche Ereignisse geloggt werden sollen und welche Informationen dabei gespeichert werden müssen. Sollen mit den Logs später beispielsweise nur Fehler aufgedeckt werden, so müssten auch nur solche Ereignisse geloggt werden. In einem Log sollten in diesem Fall Informationen über die Art des Fehlers und Zeitpunkt des Auftretens gespeichert werden. Ebenso nützlich könnten Informationen über die vorausgehenden Abläufe im System sein. Sollte die Rekonstruktion des Systemablaufes das Ziel einer späteren Verwertung der Logs sein, so müssten alle Ereignisse, die einen Zustandswechsel des Systems bewirken, geloggt werden. Von Interesse wären in diesem Fall Informationen über die Auslöser der Ereignisse und über die Änderungen, die das Ereignis bewirkt hat.

Die Entscheidung, welchen Zweck das Logging später haben soll, sollte dann auch in die Klärung des zweiten Punktes einfließen. Wichtig in diesem Zusammenhang ist auch, ob die Lognachricht in Klartext gespeichert werden soll und somit vom Menschen lesbar oder ob die Logs nur von einem anderen Programm verarbeitet werden sollen und somit eine ressourcensparende codierte Speicherung möglich ist. In beiden Fällen ist es für die Verarbeitung hilfreich, wenn alle Logs einen konsistenten Aufbau haben. Beispielsweise, dass die Lognachricht mit einem Zeitstempel beginnt, gefolgt von einer Klassifizierung des Ereignisses und zum Schluss eine Beschreibung des Ereignisses. So kann sowohl ein Klartext automatisiert verarbeitet werden, indem der Text in die vorgegebenen Abschnitte geteilt wird, als auch eine codierte Darstellung, bei welcher sich die Felder direkt ansprechen lassen. In die Klärung des zweiten Punktes fällt aber auch die Entscheidung, wohin die Logs gespeichert werden sollen. Die gängigsten Alternativen sind hierbei: Speicherung in eine oder mehrere Text-Dateien, Speicherung in eine oder mehrere Binärdateien oder Speicherung in eine Datenbank. Alle Varianten haben ihre Vor- und Nachteile. Bei der Entscheidung sollte auf jeden Fall berücksichtigt werden, wie viele Logs gespeichert werden sollen, wie oft diese verändert werden, wie oft diese während der Verarbeitung abgerufen werden und in welcher Form dies geschieht (einzeln, in Gruppen, sortiert oder durcheinander).

Eine spezielle Verwendung von Logs bei Prozessmanagementsystemen ist das Process Mining. Dabei werden Prozesse mithilfe von Logs rekonstruiert und analysiert. So können beispielsweise durch die Analyse der Logs von verschiedenen Ausführungen des gleichen Prozesses, unnötige oder wenig benutze Pfade erkannt und das Prozessmodell optimiert werden.[3]

Zusammenfassend ist Logging das Protokollieren von bestimmten Ereignissen mit dem Zweck die entstehenden Logs später unter gewissen Aspekten zu analysieren. Da zur Zeit keine Standards existieren, müssen diese bei der Entwicklung eines neuen Logging Systems unter Berücksichtigung des Umfelds und des Zwecks des Loggings neu definiert werden.

2.2 Objekt-zentrierte Prozessmanagementsysteme

In der heutigen Unternehmenswelt werden Geschäftsprozesse aufgrund der immer weiter fortschreitenden Spezialisierung immer komplexer. Um solche Prozesse effizient koordinieren zu können, bedarf es spezieller Systeme, sogenannte Prozessmanagementsysteme (PrMS). Bisher waren diese PrMS aktivitäts-zentriert, d.h. eine Aktivität innerhalb eines Prozesses wird erst dann aktiv, und somit ausführbar, wenn die vorausgehenden Aktivitäten abgeschlossen oder zumindest nicht mehr aktivierbar sind. Für datenintensive Prozesse, die immer häufiger auftreten, hat sich gezeigt, dass diese mit einem herkömmlichen aktivitäten-orientierten PrMS nicht effizient ausgeführt werden können [4]. Deshalb wurde der objekt-zentrierte Ansatz entwickelt. Zustandsübergänge innerhalb eines Prozesses werden nicht mehr durch eine Aktivität gesteuert, sondern anhand der vorhandenen Daten errechnet. Der Prozessfortschritt ergibt sich somit nicht mehr aus den durchgeführten Aktivitäten, sondern aus den vorhandenen Daten.

PHILharmonicFlows ist ein System, das diesen objekt-zentrierten Ansatz umsetzt. Basierend auf der Dissertation von Dr. Vera Künzle [5], wird das Projekt zur Zeit federführend von Kevin Andrews und Sebastian Steinau implementiert [6]. Unterstützt wurden und werden sie dabei von verschiedenen Studenten, die im Rahmen von verschiedenen Bachelor-, Master- und Diplomarbeiten ihren Teil dazu beigetragen haben.

Um einen Geschäftsprozess mit Hilfe von PHILharmonicFlows modellieren und ausführen zu können, muss zunächst eine Daten- und eine Prozessstruktur modelliert werden, die bestimmten Kriterien entspricht [5]. Bei der Ausführung einer Prozessinstanz wird durch eine Benutzerinteraktion, also eine Eingabe oder Manipulation von Daten, der sogenannte ProcessRuleManager aktiviert. Der ProcessRuleManager errechnet auf Grundlage von Regeln was für eine Auswirkung die Benutzerinteraktion auf die Instanz des Prozessen, auf welcher gearbeitet wird, hat. Dazu ist es notwendig, dass die Daten- und Prozessstrukturen nicht im Widerspruch zu den Regeln stehen. Da diese Regeln für jeden modellierbaren Prozess gelten sollen, sind sie streng generisch aufgebaut [7]. In Abbildung 2.2 ist zu sehen, wie der ProcessRuleManager funktioniert. Der ProcessRuleMananger überprüft nach seiner Aktivierung welche Regel anwendbar ist. Dazu hat jede Regel eine Menge an Vorbedingungen, die erfüllt sein müssen, sodass diese Regel

zur Anwendung kommen kann. Manche Regeln, die sogenannten *ReactionRules* und *MarkingRules*, haben bei ihrer Ausführung einen Effekt, welcher die Prozessinstanz so verändert, dass wieder andere Regeln anwendbar sind. Andere Regeln, die *ExecutionRules*, haben den Effekt, dass ein sogenanntes *ExecutionEvent* erzeugt wird, welches der Benutzerinteraktion dient, indem z.B. eine neue Benutzereingabe freigeschaltet wird. Neben diesen drei Arten von Rules gibt es noch die *CompletionRules* und die *InvalidationRules*, welche dazu dienen einzelne Prozesspfade als beendet (*Completion*) oder als ungültig bzw. nicht erreichbar (*Invalidation*) zu markieren. So errechnet das PrMS über den ProcessRuleManager mit den angegebenen Daten den Prozessfortschritt und gibt an was der Benutzer als nächstes zu tun hat.

Abbildung 2.2: Funktionsweise des ProcessRuleManagers

2.3 Technologische Grundlagen

Implementiert wurde das Framwork in der objektorientierten Programmiersprache C#, in der auch das PHILharmonicFlows System implementiert ist. Als Entwicklungsumgebung wurde *Visual Studio 2015 Enterprise* verwendet. Für die Gestaltung des Layouts des Visualisierungstools wurde *Blend for Visual Studio 2015* verwendet. Bei der Implementierung wurden für den Zugriff und die Verarbeitung von Daten LINQ[1] Ausdrücke genutzt. LINQ ist eine Sprachergänzung die mit .NET 3.5 eingeführt wurde. Mit LINQ ist es möglich, auf eine einheitliche Art Daten aus verschiedenen Datenquellen, wie Datenbanken, XML-Dokumenten oder Auflistungen wie Listen oder *Collections* zu erhalten. Für LINQ Ausdrücke existieren zwei Syntax-Varianten. Zum einen eine Syntax, welche ähnlich der SQL Syntax ist. Die Abfragequerys haben aber im Gegensatz zur SQL Syntax die Form *from where select*. Die zweite Syntax-Variante beinhaltet sogenannte Erweiterungsmethoden der Enumerable Klasse. Der Compiler übersetzt letztendlich alle LINQ Ausdrücke in Aufrufe der Erweiterungsmethoden [8]. Innerhalb der Erweiterungsmethoden werden meistens anonyme Funktionen in Form von Lambda Ausdrücken benutzt. Die Vorteile von LINQ Ausdrücken sind zum einen der universelle Zugriff auf verschiedene Datenquellen. Ein Entwickler braucht somit statt der verschiedenen Techniken für den Zugriff auf die verschiedenen Datenquellen nur noch eine Technik beherrschen. Ein weiterer Vorteil von LINQ ist die bessere Lesbarkeit des Quellcodes, da mit LINQ eine kompaktere und aussagekräftigere Darstellung im Quellcode möglich ist. Als Beispiel dafür werden in Listing 2.1 drei Möglichkeiten zur Addition der geraden Zahlen einer gegeben Liste von Zahlen aufgezeigt.

[1] Language Integrated Query

```
1  var numbers = new List<int>();
2  //old-style via foreach loop
3  var count = 0;
4  foreach (var number in numbers)
5  {
6      if(number\%2 == 0)
7          count += number;
8  }
9  //LINQ style with SQL syntax
10 count = from number in numbers
11         where number\%2 == 0
12         select number;
13 //LINQ style with extension methods
14 count = numbers.Where(number => number\%2 == 0).Sum();
```

Listing 2.1: Beispiel für die Verwendung von LINQ

Für die Umsetzung des Visualisierungstools wurde eine neue Universal Windows Application (UWA) implementiert. UWAs wurden mit Visual Studio 2013 Update 2 für Entwickler eingeführt. Sie bieten den Vorteil, dass sie auf verschiedenen Gerätetypen wie Tablets, PCs oder Handys abspielbar sind, ohne dass für den jeweiligen Typ eine komplett neue Implementierung notwendig ist. Als einzige Voraussetzung für die Verwendung von UWAs muss das Gerät das Betriebssystem Windows 8.1 bzw. Windows Phone 8.1 oder neuere Versionen installiert haben.

Das PHILharmonicFlows System basiert auf einer Client-Server Architektur. Der Server, die *PHILharmonicFlows Runtime*, wurde als Webservice implementiert. Die Client-Seite, das *Runtime Demonstration Tool* (RDT), ist als UWA umgesetzt. Das Visualisierungstool wurde in diese Struktur eingebunden und enthält die gleiche Schnittstelle zum Server wie das RDT. Für die Datenbank nutzt das PHILharmonicFlows System das ADO.NET Entity Framework (EF) von Microsoft. Das EF hat den Vorteil, dass ein Entwickler auch bei Datenbankzugriffen mit Klassen und Objekten arbeiten kann, das EF übernimmt die Übersetzung in die entsprechende Datenbanksprache. Somit ist ein einfacher Zugriff z.B. über LINQ Ausdrücke möglich.

3

Anforderungen

In diesem Kapitel werden die zu Beginn der Arbeit evaluierten Anforderungen aufgezählt und beschrieben. Aufgrund dieser Anforderungen werden in einem nächsten Schritt Lösungen entworfen, deren Umsetzung in Kapitel 4 beschrieben wird. Die Anforderungen werden in die für Softwareprojekte üblichen funktionalen und nichtfunktionalen Anforderungen unterteilt.

3.1 Funktionale Anforderungen

Funktionale Anforderungen sind Anforderungen an die Funktionalität des zu entwickelnden Frameworks. Sie beschreiben was das Framework leisten soll.

F1 Ereignisbasiertes Logging

Um später den Systemablauf analysieren und nachvollziehen zu können, sollten alle an der Client-Server-Schnittstelle auftretenden Ereignisse in Logs gespeichert werden. Ebenso sollten die wichtigsten Ereignisse innerhalb des Servers, die zum analysieren und nachvollziehen der Funktionsweise des Systems dienen, in Logs gespeichert werden.

F2 Speicherung in einer Datenbank

Da es verschiedene Verwendungsmöglichkeiten für die Logs gibt, sollten die Logs zentral gespeichert werden. Bei einer dezentralen Speicherung, z.B. an den verschiedenen

Standorten des Webservices des PHILharmonicFlows Systems könnten nicht alle Logs zur Analyse hergezogen werden bzw. der Aufwand dafür wäre zu groß. Da die Logs zudem nicht nur gespeichert, sondern auch zur Analyse gelesen werden müssen, ist eine Datenbank dafür am besten geeignet. So wird ein individueller und zugleich effizienter Zugriff auf die Logs ermöglicht. Da das PHILharmonicFlows System schon eine zentrale Datenbank besitzt, sollten alle Logs in dieser gespeichert werden.

F3 Erkennung und Speicherung der Abhängigkeit von Logs

Viele Logs stehen in einer bestimmten Beziehung zueinander. Wenn ein Ereignis an der Client-Server-Schnittstelle eingeht, kann durch die Bearbeitung des Ereignisses der ProcessRuleManager gestartet werden und serverintern Regeln angewendet werden. Dabei wird sowohl das eingehende Ereignis geloggt als auch jeweils die angewendeten Regeln. Der Log des eingehenden Ereignisses steht dann mit dem Log der ersten angewendeten Regel in Beziehung. Ebenso stehen zwei Logs von angwendeten Regeln miteinader in Beziehung, wenn die Regel des einen Logs die Regel des anderen Logs anwendbar gemacht hat. In Abbildung 3.1 sind diese Beziehungen nochmal verdeutlicht. Diese Beziehungen sollten erkannt und für die spätere Aufarbeitung gespeichert werden.

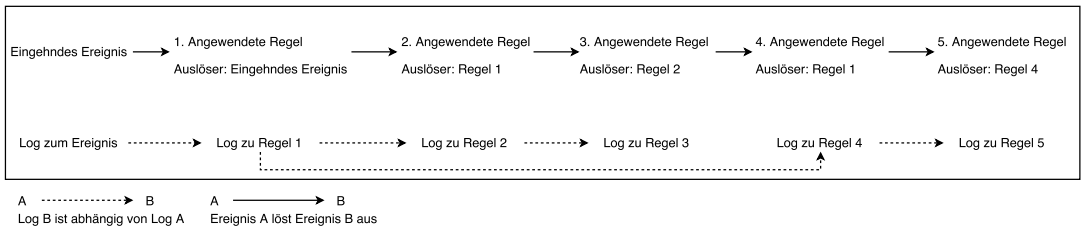

Abbildung 3.1: Beziehung von verschiedenen Logs

F4 Visualisierung von zusammenhängenden Logs

Damit der Benutzer die Beziehungen zwischen verschiedenen Logs, die nach Anforderung F3 erkannt und gespeichert werden, besser verstehen und analysieren kann, sollten die Logs und deren Beziehungen untereinander in einem dafür zu entwickelnden Tool auf eine verständliche und übersichtliche Art dargestellt werden.

F5 Wiedergabe der in den Logs gespeicherten Informationen

Um den Ablauf des Systems anhand der Logs besser nachvollziehen und analysieren zu können, ist es nicht nur wichtig die Beziehungen zu visualisieren, sondern auch andere Informationen zu den Logs darzustellen. So ist es für die Analyse des zeitlichen Ablaufs hilfreich, die Zeitpunkte der Erstellung der Logs zu kennen. Oder um die in Anforderung F4 geforderte Visualisierung der Beziehungen von Logs besser nachvollziehen zu können, ist es hilfreich Informationen zu Vorbedingungen und Effekten von Regeln zu haben. Daher sollten die in einem Log gespeicherten Informationen übersichtlich dargestellt werden.

F6 Filterung der Logs nach bestimmten Kategorien

Aufgrund der zu erwartenden großen Menge an Logs, sollte es eine Funktion geben, die es ermöglicht Logs zu filtern. Ansonsten könnte es zu Ineffizienzen beim Analysieren der Logs kommen. Wenn der Benutzer beispielsweise nur nach Logs von eingehenden Ereignissen innerhalb eines bestimmten Zeitraumes sucht müsste er alle gespeicherten Logs durchsuchen. Durch die Filterung nach bestimmten Kategorien, zum Beispiel nach eingehenden Ereignissen und einen bestimmten Zeitraum, soll die Arbeit des Benutzers erleichtert werden.

F7 Wiedergabe von grundlegenden statistischen Werten

Bei der Analyse des Systemverhaltens anhand der Logs ist es unter Umständen erforderlich bestimmte statistische Kennzahlen zu kennen. Wenn beispielsweise die Anzahl

der Logs von Interesse ist, kann es aufgrund der zu erwartenden großen Menge an Logs sehr umständlich sein diese Zahl durch Zählen von Hand zu erfahren. Daher sollte das Visualisierungstool grundlegende statistische Werte, wie Anzahl oder Art der Logs, automatisch berechnen und übersichtlich darstellen.

3.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen sind Anforderungen an die Qualität und Form des zu entwickelnden Systems. Sie beschreiben also welche Eigenschaften das Framwork besitzen soll.

NF1 Nur geringe Leistungsminderung für das PHILharmonicFlows-System

Aufgrund von Anforderung F1 ist eine große Menge an Logs zu erwarten. Die benötigte Systemleistung zum Erstellen und zum Speichern dieser Logs sollte die Leistung des PHILharmonicFlows Systems jedoch nicht zu sehr mindern. D.h. das Logging sollte so gestaltet werden, dass für einen Benutzer mit den für das System benötigten Fähigkeiten kein Unterschied in der Ausführungszeit wahrnehmbar ist.

NF2 Effizientes loggen

Um das gesamte PHILharmonicFlows System nicht unnötig zu belasten, sollten nur die notwendigsten Informationen protokolliert werden. Es sollten ebenso nur die wichtigsten Ereignisse innerhalb des Servers gespeichert werden. Z.B. ist es für die Analyse nicht wichtig, welche Regeln im ProcessRuleManager überprüft, aber aufgrund von nicht erfüllten Vorbedingungen nicht angewendet wurden. Das Speichern der Logs sollte also hinsichtlich der Verarbeitungszeit und Menge der Informationen effizient sein.

NF3 Benutzerfreundlichkeit des Visualisierungstools

Die Analyse des Systemablaufes ist ein wichtiges Mittel für die Entwickler und später für den technischen Support, um Fehlerursachen herauszufinden oder Optimierungspotenzial zu entdecken. Um mit dem Visualisierungstool den Systemablauf mit Hilfe der Logs zu analysieren, sollte das Tool für einen Benutzer mit grundlegenden Informatikkenntnissen leicht und intuitiv bedienbar sein.

NF4 Richtigkeit der Logabhängigkeiten

Um eine korrekte Analyse des Systemablaufes mit Hilfe der Logs durchführen zu können ist es wichtig, dass die angezeigten Beziehungen zwischen den Logs auch richtig sind. Daher sollte die Berechnung der Zusammengehörigkeiten und Abhängigkeiten der Logs vollständig und korrekt sein.

NF5 Verständliche Darstellung der Logabhängigkeiten

Die Visualisierung der Abhängigkeiten von Logs, wie sie in Anforderung F4 gefordert ist, sollte für einen Benutzer mit grundlegenden Informatikkenntnissen auf den ersten Blick ersichtlich und für unterschiedliche Logs konsistent sein. Eine rein textuelle Darstellung erfüllt dies nicht, da die Beziehungen zwischen mehreren Logs zum Teil sehr komplex sind und so auf den ersten Blick nicht die vollständige Beziehungsstruktur ersichtlich ist.

NF6 Einbindung der Entity Klasse

Um ein vollständiges und korrektes Logging gewährleisten zu können, muss das Framework in das bestehende PHILharmonicFlows System eingebunden werden. Um dies zu realisieren, sollte die Log-Hierarchie Teil der Entity-Struktur sein. Somit ist u.a. auch ein Zugriff auf die bestehende Datenbank für das Speichern und Abrufen der Logs möglich, wie es in Anforderung F2 gefordert ist.

NF7 Datenverträge für Client-Server-Kommunikation

Es sollte sichergestellt sein, dass die Informationen eines jeden Logs sowohl im Server, als auch auf Seite des Visualisierungstools gleich sind. Daher sollte das Framework und das Visualiserungstool in den Datenvertrag, der im PHILharmonicFlows System schon zwischen dem Webserivce und dem RDT besteht, aufgenommen werden.

4

Lösung

In diesem Kapitel wird beschrieben, wie die Vorgaben aus der Aufgabenstellung (vgl. Abschnitt 1.2) und den erarbeiteten Anforderungen (vgl. Kapitel 3) umgesetzt wurden. Es wird dabei mehr die Idee und deren Umsetzung erörtert und weniger auf den entstandenen Quellcode eingegangen. An einigen Stellen werden auch Alternativen zu der gewählten Lösung angesprochen und diskutiert.

Dabei wird zuerst der Aufbau der verschiedenen Logs und die zu beachtenden Eigenschaften erklärt. Danach wird das eigentliche Logging auf der Serverseite betrachtet. Abschließend wird noch gezeigt, wie die Visualisierung der Logs umgesetzt worden ist.

4.1 Aufbau der Datenstruktur

Für das PHILharmonicFlows System wird ein Logging Framework benötigt, welches bestimmte eingetretene Ereignisse für die spätere Prozess- und Systemablaufanalyse protokolliert (vgl. Anforderung F1). Am besten dazu geeignet ist ein Logging, mit welchem verschiedene Ereignisse, wie z.B. die Benutzerinteraktion oder die serverinternen Arbeiten des ProcessRuleManagers, gespeichert werden können. Zur Speicherung der verschiedenen Ereignisse braucht es verschiedene Logs mit unterschiedlichen Feldern zum Speichern der Eigenschaften. Dabei gibt es einige Grundeigenschaften die alle Logs teilen, z.B. den Zeitpunkt, zu welchem sie erstellt worden sind, und andere Eigenschaften welche nur eine Teilmenge der Logs gemeinsamen haben. Daher können die unterschiedlichen Logs in eine baumartige Struktur gepackt werden, welche in dem Klassendiagramm in Abbildung 4.1 zu erkennen ist.

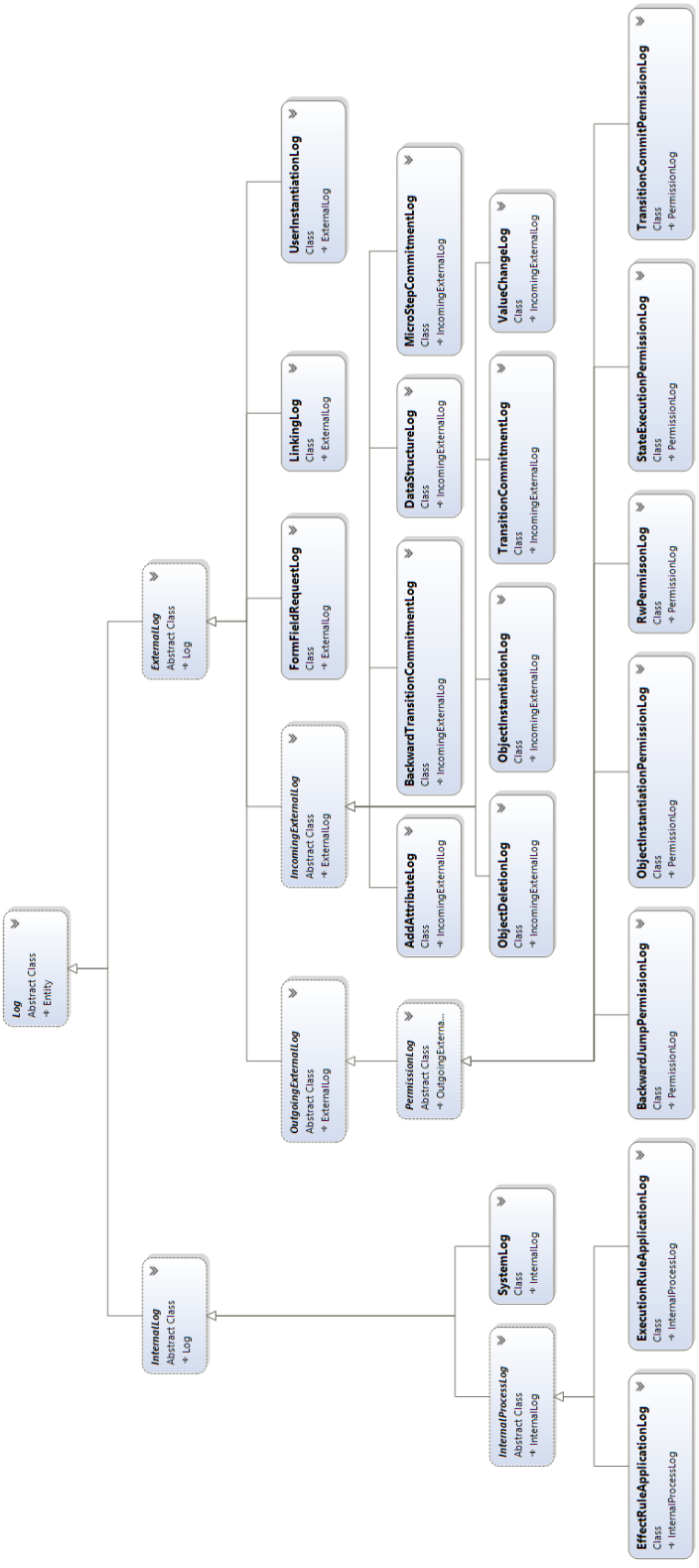

Abbildung 4.1: Klassendiagramm der Datenstruktur

Dadurch ist es möglich die Logs für die spätere Verarbeitung auf verschiedene Arten zu gruppieren. In dem Visualisierungstool ist diese Eigenschaft beispielsweise für das Filtern der Logs nützlich, da sich so die Logs nach bestimmten Gruppen filtern lassen.

An dem Klassendiagramm ist zu erkennen, dass sich die Logs in zwei grundlegende Kategorien unterteilen lassen. Zum einen die sogenannten *internen Logs*, die protokollieren was innerhalb des Servers abläuft. Dabei wird wiederum unterschieden in allgemeine System Logs, die z.B. Fehlerzustände mit entsprechender Fehlermeldung, aber auch sonstige Ereignisse wie das Einloggen eines Benutzers speichern. Der zweite Teilbaum innerhalb der internen Logs beinhaltet Logs, die Zustände betrachten, die entstehen wenn die verschiedenen Regeln im ProcessRuleManager abgearbeitet werden.

Die andere Kategorie von Logs, neben den internen Logs, sind die *externen Logs*. Sie dokumentieren die Aktivitäten an der Schnittstelle zwischen Client und Server. Externe Logs werden in *ausgehende und eingehende Logs* unterteilt. Ausgehende Logs protokollieren Ereignisse, welche vom Server zu einem Client gesendet worden sind, z.B. Rechte Informationen (*PermissionLogs*). Eingehende Logs protokollieren Ereignisse, welche von einem Client an den Server gesendet worden sind, z.B. dass ein neues Objekt instanziiert wurde. Die eingehenden Logs werden noch unterteilt in Logs, welche Ereignisse protokollieren, die den ProcessRuleManager aktivieren (*IncomingLogs*), z.B. *ValueChangeLogs*, und Logs, welche Ereignisse gespeichert haben, die nicht diese Auswirkung haben, z.B. *LinkingLogs*.

In Listing 4.1 wird die Implementierung der *Log* Klasse dargestellt. Sie ist die Oberklasse der Log-Datenstruktur. Im Folgenden werden anhand dieser Log Klasse beispielhaft gezeigt, wie einige Anforderungen aus Kapitel 3 umgesetzt worden sind.

```
1  namespace Server.Logging.AbstractLogs
2  {
3      [DataContract(IsReference = true)]
4      [KnownType(typeof(InternalLog)), KnownType(typeof(ExternalLog))]
5      public abstract class Log : Entity
6      {
7          protected Log()
8          {
9              Timestamp = DateTime.Now;
10         }
11         [DataMember]
12         public DateTime Timestamp { get; set; }
13         [DataMember]
14         public List<string> PreviousLogs { get; set; } = new List<string>();
15         [DataMember]
16         [NotMapped]
17         public virtual string DisplayName {
18             get { return GetType().ToString().Substring(
19                 GetType().ToString().LastIndexOf(".")+1); }
20             set {}
21         }
22         public string DependentLog { get; set; }
23     }
24 }
```

Listing 4.1: Log-Klasse: Oberklasse der Logeinträge

Damit die Logs wie in Anforderung F2 gefordert in die vorhandene Datenbank gespeichert werden können, müssen sie von der Entity-Klasse erben (siehe Zeile 5). Damit ist gleichzeitig die Anforderung NF6 für alle Log-Klassen erfüllt. Darüber hinaus ist mit der Annotation `[DataContract(IsReference = true)]` in Zeile 3 die Anforderung NF7 ebenfalls erfüllt und damit die Kommunikation zwischen Server und Client sichergestellt. Ein *Data Contract* ist ein Mechanismus um Klassen zu serialisieren. Sobald eine Klasse mit der Annotation `[DataContract(IsReference = true)]` ausgezeichnet ist, können Objekte dieses Typs zwischen verschiedenen Partner dieses Datenvertrages übermittelt werden. Dies geschieht über einen SOAP Versand da sich die Klassen mittels dieser Datenverträge in einem XML Format speichern lassen. Im Falle des PHILharmonicFlows Systems können beispielsweise die Logs von dem Webservice zu den Clients und dem Visualisierungstool übermittelt werden, da alle Teilanwendungen den selben

Data Contract referenzieren. Durch diese Annotationen ist auch die Speicherung in einer relationalen Datenbank möglich.

Durch die Annotation `[NotMapped]` in Zeile 16 wird die Eigenschaft DisplayName nicht in der Datenbank gespeichert. Dies wäre auch nicht sinnvoll, da nicht der Wert an sich von Interesse ist, sondern wie dieser berechnet wird (siehe Zeile 18, 19). Die Berechnung der Eigenschaft DisplayName kann sich von Klasse zu Klasse ändern, da das Keyword virtual angehängt wurde. Genutzt wird die Eigenschaft später in dem Visualisierungstool, um die Art des Logs anzuzeigen. Da der Wert von DisplayName nicht in der Datenbank gespeichert wird, wird die Datenbank weniger belastet und somit die Anforderung NF2 zum Teil erfüllt.

Um den Abruf der Logs aus der Datenbank effizient bzw. überhaupt erst möglich zu machen, ist es notwendig, dass möglichst auf Fremdschlüsselbeziehungen verzichtet wird und stattdessen die benötigten Informationen als String gespeichert werden. In der Log-Klasse wird daher die Eigenschaft DependentLog nicht als Log, sondern die dazugehörige ID als String gespeichert (Zeile 22). Würde die Fremdschlüsselbeziehung gespeichert werden, so würde das für die Datenbank verwendete Entity Framework beim Abruf aller Logs eine SQL-Query generieren, die so komplex und verschachtelt ist, dass es ständig zu Laufzeitfehlern kommen würde. Die Speicherung der Fremdschlüsselbeziehungen hat auch den Nachteil, das bei der Löschung der verlinkten Entitäten die Informationen über diese für den Log auch nicht mehr vorhanden wären. Durch die Speicherung der Werte sind diese auch nach Löschung der eigentlichen Entität für die Log Verarbeitung noch vorhanden.

Da bisher als einzige Verarbeitung der Logs nur das im Zuge dieser Arbeit entstandene Visualisierungstool existiert, ist es noch nicht klar welche Eigenschaften für andere Verarbeitungsmöglichkeiten, wie zum Beispiel dem Process Mining, genau von Interesse sind. Daher wurde bei den verschiedenen Logs darauf geachtet, dass möglichst alle Informationen, die mit dem Auftreten eines Ereignisses entstehen, gespeichert werden. So können bei einer späteren Verarbeitung alle Eigenschaften betrachtet und analysiert werden. Im Speziellen wurde mit Blick auf das Process Mining darauf geachtet, dass eine Möglichkeit existiert, den Prozessablauf mit Hilfe der Logs zu rekonstruieren.

4.2 Ablauf des Loggings

Um nun einen bestimmten Zustand in einem Log festzuhalten, gibt es die Klasse Logger, welche entsprechende Methoden bereitstellt. Diese Methoden werden an gewünschter Stelle aufgerufen und die zu protokollierenden Informationen werden übergeben. Die vorhandenen Methoden lassen sich in zwei Kategorien unterteilen. Zum einen für Logs, die nicht in Beziehung zu anderen gesetzt werden z.B. wenn sich ein Benutzer einloggt. Dabei wird nur ein entsprechender Log deklariert und initialisiert und anschließend der Datenbank hinzugefügt. Diese Methode ist in Listing 4.2 dargestellt.

```
1 public static void LogUserLogin(UserInstance user)
2 {
3     var log = new SystemLog
4     {
5         Type = "Information",
6         Message = user.Username+" successfully logged in"
7     };
8     DB.Context.Logs.Add(log);
9 }
```

Listing 4.2: Methode zum Loggen des Logins

Daneben gibt es Methoden für Logs, die eine Beziehung zu anderen Logs besitzen. Wenn beispielsweise ein Ereignis mit einer Wertänderung von einem Client kommt, wird sowohl dieses Ereignis geloggt, als auch der ProcessRuleManager aktiviert, der verschiedene Regeln innerhalb des Servers anwendet. Die Regelanwendungen des ProcessRuleManager werden ebenfalls geloggt, sollten später aber mit dem auslösenden Ereignis, in diesem Beispiel der Wertänderung, in Verbindung gebracht werden können (vgl. Anforderung F3 und F4). Um dies zu realisieren wurde ein Ansatz gewählt, der die Arbeit der Beziehungsherstellung aufteilt. Innerhalb des Servers werden nur Beziehungen von internen Logs zu einem externen Log gesetzt. Dazu werden die internen Logs, die im Zusammenhang mit dem ProcessRuleManager entstanden sind, in einer Liste gespeichert. Sobald die entsprechende Methode zum Loggen des eingehenden Ereignisses aufgerufen wird, wird für jeden Log aus der Liste der internen Logs die Beziehung zu diesem externen Log hergestellt, indem die DependentLog Eigenschaft gesetzt wird. Diese Reihenfolge ist wichtig, da die Log-Methode des externen Logs

immer erst am Ende, also wenn der ProcessRuleManager mit der serverinternen Arbeit fertig ist, aufgerufen wird. Dieser Prozess wird in Abbildung 4.2 bildlich veranschaulicht.

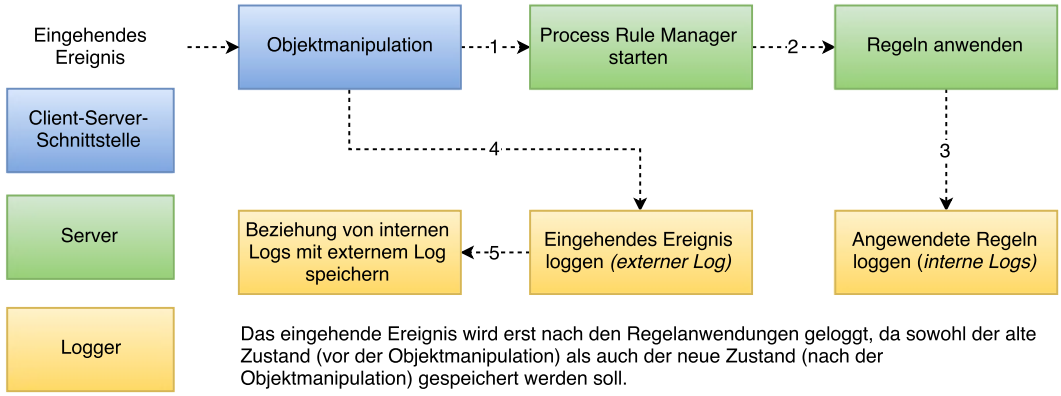

Abbildung 4.2: Ablauf des Loggings von in Beziehung stehender Logs

In Listing 4.3 wird der ganze Prozess auf Codeebene anhand der Methoden für das Loggen von Wertänderungen und von MarkingRules gezeigt.

```
1  public List<InternalProcessLog> InternalLogs { get; set; }
2    = new List<InternalProcessLog>();
3  public  static void LogEffectRuleApplication<T>(...) where T : Entity
4  {
5      var log = new EffectRuleApplicationLog {...};
6      log.DependentLog = TopLog.WCFID;
7      InternalLogs.Add(log);
8      DB.Context.Logs.Add(log);
9  }
10
11 public static void LogAttributeValueChange(...)
12 {
13     var log = new ValueChangeLog{...};
14     foreach (var internalLog in Logger.InternalLogs)
15     {
16         internalLog.DependentLog = log.WCFID;
17     }
18     DB.Context.Logs.Add(log);
19 }
```

Listing 4.3: Methode zum Loggen von Wertänderungen und von MarkingRules

Neben dem für die Implementierung gewählten Ansatz standen während der Entwicklung noch zwei weitere Möglichkeiten zur Diskussion. Ein möglicher Ansatz wäre der, den ganzen Prozess der Beziehungsherstellung, also auch der genauen Abhängigkeitsbeziehungen zwischen den internen Logs auf Serverseite auszuführen. Dies hat aber den Nachteil, dass so während der Ausführung wertvolle Rechenkapazität für das Logging verloren geht. Anforderung NF1 wäre somit nicht erfüllt. Ein weiterer Ansatz wäre der, den gesamten Prozess der Beziehungsherstellung auf der Seite des Visualisierungstools auszuführen. Das hat aber den Nachteil, dass so immer alle Logs aus der Datenbank über die Schnittstelle zwischen Server und Visualisierungstool gesendet werden müssen. Bereits mit nur einem Benutzer und nur wenigen Benutzeraktionen ist die Anzahl der Logs schon ziemlich hoch, hochgerechnet auf einen Mehrbenutzerbetrieb mit vielen ausgeführten Aktionen könnte die Anzahl der zu übertragenden Logs mit hoher Wahrscheinlichkeit zu Laufzeitfehlern wegen einer zu großen Datenmenge führen. Daher wurde der Ansatz gewählt, in welchem die miteinander in Beziehung stehenden Logs auf Serverseite, über den ersten Log einer Beziehungsgruppe, gruppiert werden und die genaue Abhängigkeitsbeziehungsherstellung auf Seiten des Visualisierungstools erfolgt. Dadurch werden nicht alle Logs auf einmal über die Schnittstelle von dem Server zu dem Visualisierungstool übertragen. Beim Öffnen des Tools wird nur der erste Log einer jeden Beziehungsgruppe und die Logs, die keine Beziehung zu anderen Logs haben übertragen. Die restlichen Logs werden dann nach Bedarf übertragen. Der genaue Ablauf auf Seiten des Visualisierungstools wird in Abschnitt 4.3.2 erklärt.

4.3 Das Visualisierungstool

Zusätzlich zu dem Logging sollte noch ein Werkzeug entwickelt werden, mit dem die Logs und deren Beziehungen untereinander visualisiert werden können. Dieses Visualisierungstool sollte die Anforderungen F4, F5, F6, F7, NF3, NF4 und NF5 erfüllen. Dazu wurde eine neue Universal Windows Application implementiert, die über einen Datenvertrag mit der PHILharmonicFlows Runtime Daten austauschen kann. Das Visualisierungstool erhält von der Runtime Gruppen von Logs die in Beziehung stehen. Wie diese Beziehungen innerhalb einer Gruppe aussehen ist zu diesem Zeitpunkt aber noch

nicht bekannt. Daher enthält das Visualisierungstool auch die Logik zur Berechnung der genauen Beziehungen der Logs. Die Umsetzung dieser Logik und des Layouts für das Visualisierungstool wird in den nächsten beiden Abschnitten genauer erläutert.

4.3.1 Layout

Für das Layout wurde zunächst ein Entwurf erstellt indem die Beziehungen in Form eines Graphen und vier Listen dargestellt werden. Dieser erste Entwurf ist in Abbildung 4.3 zu sehen. Die in Anforderung F5 geforderte Wiedergabe der Informationen eines Logs wird in diesem Entwurf mit Pop-Up-Fenstern umgesetzt, die erscheinen wenn der Benutzer einen Log in dem Graphen auswählt. Für die Listen wird davon ausgegangen, dass sich die Logs in Ebenen einteilen lassen. So sind in Ebene 0 die Logs enthalten, welche keine Abhängigkeitsbeziehung zu anderen Logs besitzen. In Ebene 1 sind Logs, welche von Logs aus Ebene 0 abhängig sind, also Logs zu Regeln, deren Vorbedingung von dem Ereignis aus dem Log aus Ebene 0 erfüllt wurde. In Ebene 2 sind Logs, welche von Logs aus Ebene 1 abhängig sind. Diese können aber zusätzlich auch von Logs aus Ebene 0 abhängig sein. Analog sind die weiteren Ebenen aufgebaut. In der linken, inneren Liste (vgl. Abb. 4.3 M1) werden die Logs aufgelistet, die auf der gleichen Ebene sind, wie der aktuell ausgewählte Log. In der Liste rechts davon (vgl. Abb. 4.3 M2) sind die Logs aufgelistet, die von dem aktuell ausgewählten Log direkt abhängig sind. In der rechten äußeren Liste (vgl. Abb. 4.3 M3) sind Logs aufgelistet die indirekt von dem aktuell ausgewählten Logs abhängig sind. Die indirekte Abhängigkeit ergibt sich aus der transitiven Relation, die für die Abhängigkeit von Logs gilt. Wenn also ein Log A von Log B abhängig ist und dieser Log B von einem Log C, dann ist auch Log A von Log C abhängig. In der linken, äußeren Liste (vgl. Abb. 4.3 M4) sind die Logs aufgelistet, die in der übergeordneten Ebene zu dem aktuell ausgewählten Log enthalten sind.

Da die Beziehungen der Logs untereinander aber zum Teil komplex sein können, wie in Abbildung 4.4 skizziert, lassen sich die Logs bezüglich ihrer Abhängigkeiten untereinander nicht immer in eine Baumstruktur einteilen. Eine Baumstruktur ist aber für die Einteilung in Ebenen wichtig. Aus diesem Grund ist die Darstellungsform über verschiedene Listen wie in dem ersten Entwurf nicht praktikabel. Daher wurden diese Listen

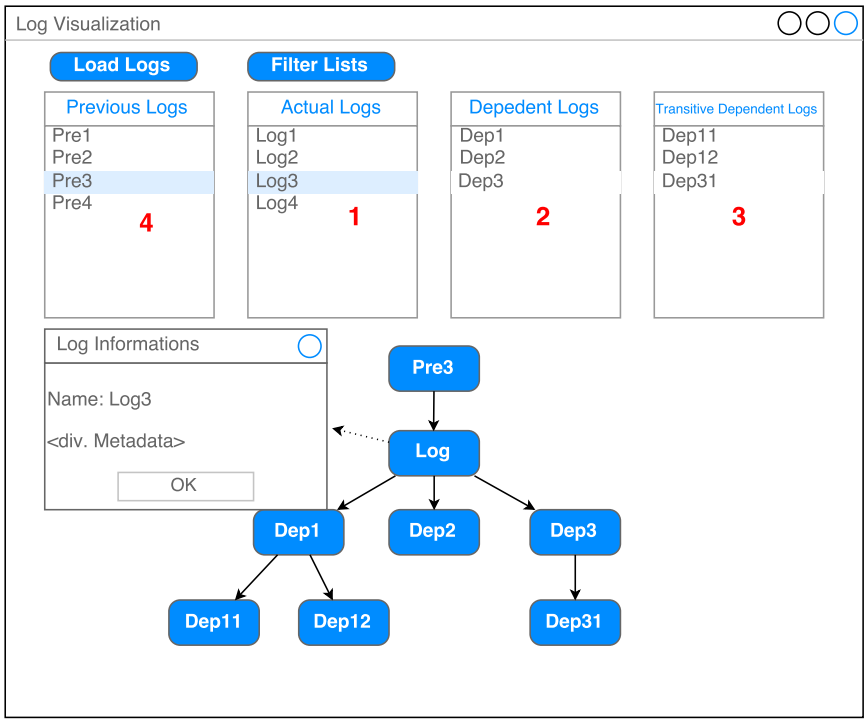

Abbildung 4.3: Entwurf des Visualisierungstools

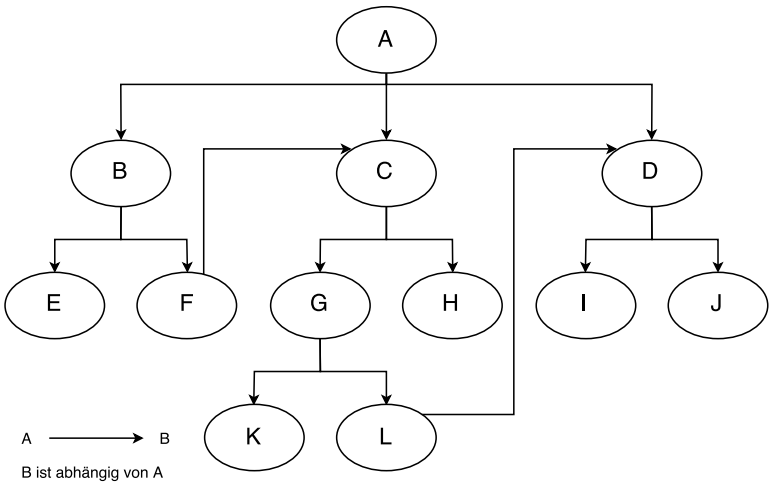

Abbildung 4.4: Beispiel für komplexe Beziehungen von Logs

in einem zweiten Entwurf entfernt und die Fläche für den Graphen vergrößert. Dieser endgültige Entwurf ist in Abbildung 4.5 zu sehen. Der Graph, in welchem die Beziehun-

Abbildung 4.5: Entgültiger Entwurf

gen der Logs dargestellt werden, ist in diesem Entwurf zentral angeordnet (vgl. Abb. 4.5 M1). Ein Pfeil in diesem Graph sagt aus, dass der Log auf den der Pfeil zeigt, von dem Log von dem der Pfeil ausgeht, abhängig ist. In Logs, auf die ein Pfeil zeigt, sind also immer die Informationen einer angewendeten Regel enthalten. Eine Vorbedingung dieser Regel wurde dabei von dem Ereignis erfüllt, das in dem Log gespeichert wurde von dem der Pfeil ausgeht. Über dem Graphen sind Informationen über den Aufbau des Graphen dargestellt. Darin wird beschrieben aus wie vielen Logs der Graph besteht und wie viele Logs davon Informationen über Reaction Rules, Marking Rules und Execution Events beinhalten. Ein Teil von Anforderung F7 ist dadurch umgesetzt. Dabei kann es vorkommen das die Anzahl der Logs und die Summe der Aufzählung der verschiedenen Logs nicht übereinstimmt. Wenn beispielsweise der erste Log einer Beziehungsgruppe ein eingehendes Ereignis gespeichert hat, kann dieser keiner der aufgezählten Kategorien zugeteilt werden. Die Aufsummierung der Logs über die Kategorien würde den ersten Log also nicht beachten, die Gesamtanzahl der Logs aber schon. In der linken oberen

Ecken befinden sich zwei Möglichkeiten die Liste der Logs zu aktualisieren (vgl. Abb. 4.5 M2). Über die linke Schaltfläche (*Refresh*) werden die am aktuellen Tag erstellten Logs geladen. Über die rechte Schaltfläche (*Load older Logs*) hat der Benutzer die Möglichkeit Logs aus einem bestimmten Zeitraum zu laden. Dazu wird der Benutzer nach einem Klick auf die Schaltfläche in einem sich daraufhin öffnenden Dialogfenster dazu aufgefordert, den entsprechenden Zeitraum einzugeben. Unterhalb dieser Schaltflächen befindet sich eine Schaltfläche (*Filter (off)*, vgl. Abb. 4.5 M3) mit welcher der Benutzer die Möglichkeit hat die Filtereinstellungen auszuklappen und die Einstellungen zu ändern. Die ausgeklappten Filtereinstellungen sind in Abbildung 4.6 zu sehen. Die Schaltfläche zeigt auch an, ob zur Zeit eine Filtereinstellung ausgewählt ist (*Filter (on)*) oder ob die Liste der Logs nicht gefiltert ist (*Filter (off)*). Damit wurde die Anforderung F6, die eine Funktionalität zum Filtern der Logs fordert, umgesetzt.

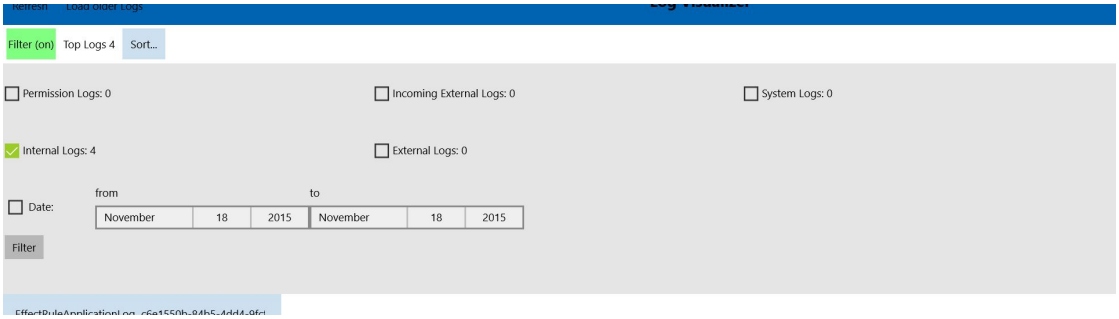

Abbildung 4.6: Filtereinstellungen

Rechts neben der Filterschaltfläche wird die aktuelle Anzahl an Logs in der Liste angezeigt, somit ist auch Anforderung F7 erfüllt. Wiederum rechts von der Anzahl der Logs ist eine Schaltfläche (*Sort...*), die bei einem Klick ein Menü öffnet, dass dem Benutzer die Möglichkeit gibt, die Liste der Logs nach Zeit, Prozess und Name zu sortieren. Die Liste der aktuell geladenen Logs befindet sich unterhalb der Filter Schaltfläche (vgl. Abb. 4.5 M4). Hier sind nur Logs aufgelistet die von keinem anderen Log abhängig sind. Unterhalb dieser Liste werden die Informationen die ein Log enthält dargestellt (vgl. Abb. 4.5 M5). Die Darstellung der Informationen folgt immer dem gleichen Schema, so wird oben immer die Art des Logs angezeigt und am Ende immer der genaue Zeitpunkt, wann der Log erstellt wurde.

4.3.2 Logik

Für die Logik, also die Berechnung der genauen Beziehungen der Logs, ist es erst einmal wichtig, die genaue Schnittstelle zwischen dem Server und dem Visualisierungstool zu kennen. Wenn der Benutzer über die Schaltflächen *Refresh* oder *Load older Logs* Logs von dem Server abruft, lädt der Server nur die Logs aus der Datenbank, die nicht von anderen Logs abhängig sind. Die `DependentLog` Eigenschaft ist bei diesen Logs also `null`. Diese Logs werden dann in der Liste im Visualisierungstool angezeigt. Wenn der Benutzer einen dieser Logs auswählt, werden über die Schnittstelle die mit dem ausgewählten Log in Beziehung stehenden Logs aus der Datenbank nachgeladen. Über diese Logs ist nur bekannt, dass sie Informationen über angewendete Regeln gespeichert haben und dass sie miteinander in Beziehung stehen. Die genauen Abhängigkeitsbeziehungen dieser Logs sind aber noch unbekannt. Die Berechnung der Abhängigkeitsbeziehungen ist in der Logik des Visualisierungstools umgesetzt. Dazu wurde ein Algorithmus entwickelt, der die Vorbedingungen der angewendeten Regeln betrachtet und dann nach passenden Effekten von davor angewendeten Regeln sucht. Wenn ein Effekt gefunden wird, wird die Abhängigkeit gespeichert. Der Algortihmus ist in Listing 4.4 als Pseudocode dargestellt.

```
1 DependencyDetection(List logs)
2   foreach(logA in logs)
3     foreach(logB in logs)
4       if(logA == logB) next;
5       foreach(precondition in logA.Preconditions)
6         foreach(effect in logB.Effects)
7           if(precondition match effect)
8             save matching;
9             next;
```

Listing 4.4: Pseudocodedarstellung des Algorithmus zur Erkennung und Speicherung von Logabhängigkeiten

Um nun den Algorithmus möglichst effizient zu gestalten, sind noch ein paar Informationen zu dem Aufbau und der Funktionsweise der Regeln nötig. In Listing 4.5 ist der Aufbau einer Regelklasse zu sehen, die im Folgenden dabei hilft, die Punkte, an denen der Algorithmus effizienter gemacht wurde, zu erläutern.

```
1  namespace Server.ProcessRules.MarkingRules
2  {
3    public class MarkingRuleMr13F
4            : AbstractEffectRule<MicroTransitionInstance>
5    {
6      public MarkingRuleMr13F()
7      {
8        Name = "Marking Rule Mr13F";
9        ShortDescription = "Marking micro steps as Bypassed";
10       Description = "..."
11       IsIdempotent = true;
12       PreconditionFor(trans => trans.Marking)
13         .IsMarked(MicroTransitionMarkings.Bypassed);
14       PreconditionFor(trans => trans.TargetMicroStep.Marking)
15         .IsMarkedAny(MicroStepMarkings.Ready,
16         MicroStepMarkings.Enabled, MicroStepMarkings.Blocked,
17         MicroStepMarkings.Activated);
18       PreconditionForEach(trans => trans.TargetMicroStep
19         .IncomingTransitions.Select(x => x.Marking))
20           .IsMarked(MicroTransitionMarkings.Bypassed);
21
22       //Mark the target step as Bypassed
23       EffectFor(trans => trans.TargetMicroStep.Marking)
24         .AssignMarking(MicroStepMarkings.Bypassed);
25     }
26   }
27 }
```

Listing 4.5: MarkingRule13F als Beispiel für den Aufbau einer Regel.[7]

Die meisten Regeln (90-95%) besitzen nur eine Vorbedingung, die von einem Effekt einer anderen Regel erfüllt wird und somit die Regel ausführbar macht. Die restlichen Vorbedingungen dienen dazu, die Regeln zu differenzieren. D.h. in den meisten Fällen kann der Algorithmus nach der ersten gefundenen Verbindung von Effekt zu Vorbedingung für diesen Log abbrechen und für den nächsten Log den Log suchen, der den aktivierenden Effekt hat. In dem Beispiel (vgl. Listing 4.5) ist in Zeile 4 die Vorbedingung für die ein passender Effekt gesucht werden muss. Diese Vorbedingung zeichnet sich dadurch aus, dass der Typ der Vorbedingung und der Typ der Regel gleich sind. Im Beispiel ist die Regel vom Typ `MicroTransitionInstance` (Zeile 4) und die Vorbedingung bezieht sich auf ein `MicroTransitionMarking` (Zeile 12), also ein

Marking für ein `MicroTransitionInstance` Objekt. Die Vorbedingung bezieht sich auch immer auf das Marking des entsprechenden Objektes, im Beispiel zu erkennen an `PreconditionFor(trans => trans.Marking).IsMarked` (Zeile 12). Für die Suche nach dem passenden Effekt braucht der Algorithmus nur die Logs zu durchsuchen, die zeitlich vor dem Log mit der zu erfüllenden Vorbedingung erstellt wurden. Dies gilt, da eine angewendete Regel nicht erst durch eine später angewendete Regel anwendbar gemacht worden sein kann. Der gesuchte Effekt ist ein Effekt, der das Marking des betreffenden Objektes auf den gewünschten Wert ändert. In dem Beispiel ist ein Effekt für eine `MicroTransitionInstance` mit der selben `ID` gesucht wie die von dem Objekt, von dem die Vorbedingung erfüllt werden soll, und bei dem das Marking auf `Bypassed` geändert wird.

Mit diesen Informationen konnte ein effizienter Algorithmus erstellt werden, welcher die Abhängigkeitsbeziehungen von Logs korrekt berechnet. Der umgesetzte Algorithmus ist in Listing 4.6 abgebildet.

```
1  private void MatchPreconditionsToEffects(InternalProcessLog actualLog,
2      EffectRuleApplicationLog dependentLog)
3  {
4      foreach (var pre in actualLog.Preconditions)
5      {
6          if (!pre.Contains("IsMarked")) continue;
7          foreach (var effect in dependentLog.Effects)
8          {
9              if (!effect.Contains("AssignMarking")) continue;
10             var preMarkings = pre.Substring(pre.LastIndexOf("(") + 1)
11                 .Trim(')').Split('.');
12             var effectMarking = effect
13                 .Substring(effect.LastIndexOf("(") + 1).Trim(')');
14             foreach (var marking in preMarkings)
15             {
16                 var startIndex = effect
17                     .IndexOf('(', effect.IndexOf('(') + 1)+1;
18                 var endIndex = effect.IndexOf(')');
19                 if (marking.Equals(effectMarking) &&
20                     actualLog.Instance.Equals(effect
21                         .Substring(startIndex, endIndex-startIndex)))
22                 {
23                     actualLog.PreviousLogs.Add(dependentLog.WCFID);
24                 }
25             }
26         }
27     }
28 }
```

Listing 4.6: Algorithmus zur Erkennung und Speicherung von Logabhängigkeiten

Somit entstand im Endeffekt eine Logging Komponente für das PHILharmonicFlows System, dass alle zu Beginn der Arbeit evaluierten Anforderungen erfüllt. Die baumartige Datenstruktur der Logs erlaubt es, neue Log Klassen ohne größeren Aufwand zu integrieren, falls dies im Laufe der weiteren Entwicklung des PHILharmonicFlows System notwendig ist. Das eigentliche Logging wurde in Form einer Logger Klasse realisiert, in welcher alle Methoden zum Loggen der einzelnen Ereignisse enthalten sind. Auch diese ist für die Zukunft erweiterbar, sofern dies notwendig sein sollte. Zusätzlich zu der Logging Komponente wurde auch ein Werkzeug entwickelt, um die Logs und deren Beziehungen untereinander zu visualisieren. Hierbei wurde nicht nur auf eine übersichtliche und gut bedienbare Oberfläche geachtet, sondern auch auf eine möglichst effiziente Arbeitsweise bei der Berechnung der Beziehungen zwischen den Logs.

5

Fazit und Ausblick

5.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Logging Framework für ein objekt-zentriertes PrMS entwickelt. Zu Beginn wurden dazu die Anforderungen evaluiert und anschließend ein erster Entwurf angefertigt. Nach einigen Verbesserungen wurde der finale Entwurf dann unter Berücksichtigung der technologischen Vorgaben umgesetzt. Ergebnis war ein Framework, welches eine baumartige Struktur für verschiedenen Arten von Logs bietet, mit denen es möglich ist, verschiedene Ereignisse zu loggen. Zudem besteht die Möglichkeit mit diesen Logs den Zusammenhang von verschieden Ereignissen zu speichern, zu rekonstruieren und somit auch in einem speziellen Tool darzustellen.

Während der Entwicklung gab es verschiedene Herausforderungen die zu bewältigen waren, einige die sich aus der Problemstellung heraus ergaben und zu erwarten waren, aber auch einige die eher plötzlich und unerwartet aufgetreten sind. Aus der Problemstellung ergab sich z.B. die Herausforderung erst einmal zu wissen, was beim Logging hinsichtlich objekt-zentrierter PrMS überhaupt zu beachten ist, da es in diesem Gebiet noch keine bzw. nur sehr wenig Erfahrungswerte und Berichte gibt. Weiter musste ich mich zuerst in das bestehende PHILharmonicFlows System einarbeiten. Dazu gehörte das Einstudieren der Programmiersprache C# aus technischer Sicht mit den Besonderheiten wie LINQ. Da mit dem PHILharmonicFlows System bereits zu Beginn der Arbeit ein funktionierendes System gegeben war, musst ich mich neben den technischen auch mit den inhaltlichen Aspekten auseinandersetzen, da für das Logging die Funktionsweise und die Abläufe des PHILharmonicFlows System sehr wichtig sind. Am schwierigsten war hierbei der serverinterne Ablauf der Rule Evaluation. Hinzu kamen

unerwartet Probleme wie z.B. die nicht immer klare Arbeitsweise des für die Datenbankschnittstelle verwendeten Entity-Frameworks (EF), wodurch es zu einem späten Zeitpunkt noch nötig war die Datenstruktur der Logs so umzubauen, dass möglichst wenige Fremdschlüsselbeziehungen entstehen. Das EF nutzt diese beim Abruf von Logs aus der Datenbank dazu, sehr komplexe und verschachtelte SQL-Querys zu generieren die zu einem Laufzeitfehler führen.

5.2 Ausblick

Bisher werden mit dem entstandenen Framework nur Logs generiert und gespeichert. Eine Verarbeitung im Rahmen einer Anwendung für den Endbenutzer ist noch nicht realisiert. Das zu dieser Arbeit gehörende Visualisierungstool ist viel mehr für die Entwickler gedacht und höchstens noch für einen technischen Support geeignet. Die Logs können aber auch für den Endbenutzer einen Nutzen haben. Es wäre denkbar das Process Mining für objekt-zentrierte PrMS umzusetzen. Hierbei wäre zu klären wie diese Technologie für diesen Ansatz aussieht. Wie lassen sich Prozesse rekonstruieren? Wie können Prozesse optimiert werden? Welches sind die dafür notwendigen Kennzahlen? Bei dem Process Mining für aktivitäts-zentrierte PrMS ist beispielsweise die Nutzungshäufigkeit von Aktivitäten interessant um dadurch wenig genutzte Aktivitäten bzw. Aktivitätspfade aus dem Prozessmodell löschen zu können und das Prozessmodell somit zu optimieren. Diese Optimierungsmöglichkeiten sind bisher für objekt-zentrierte PrMS nicht erforscht, Analysen der Logs könnten dafür aber hilfreich sein. Eine weitere Aufgabe für die Zukunft könnte es sein, wie mit Hilfe der Logs, die Prozess- und Datenstrukturen rekonstruiert bzw. falls diese bekannt sind, der Ablauf eines Prozesses mit den Strukturen abgeglichen werden kann. Zusammengefasst ist die Entwicklung des Process Mining auf dem Gebiet des objekt-zentrierten Prozessmanagements eine große Baustelle für die Zukunft.

Literaturverzeichnis

[1] MITRE-Corporation: Common event expression - CEE (2014) [Archiv; Stand 28. November 2014].

[2] Chuvakin, A., Schmidt, K., Phillips, C.: Logging and log management: The authoritative guide to understanding the concepts surrounding logging and log management. Newnes (2012)

[3] Van Der Aalst, W., Adriansyah, A., de Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, J.C., van den Brand, P., Brandtjen, R., Buijs, J., et al.: Process mining manifesto. In: Business process management workshops, Springer (2012) 169–194

[4] Künzle, V., Reichert, M.: Herausforderungen auf dem Weg zu datenorientierten Prozess-Management-Systemen. In: EMISA Forum. Volume 29. (2009) 9–24

[5] Künzle, V.: Object-aware Process Management. PhD thesis, Ulm University (2013)

[6] Andrews, K., Steinau, S., Reichert, M.: A Runtime Environment for Object-Aware Processes. In: Proceedings of the BPM Demo Session 2015 (BPMD 2015). (2015)

[7] Steinau, S.: Design and implementation of a runtime environment of an object-aware process management system. Master's thesis, Ulm University (2015)

[8] Baltes-Götz, B.: Einführung in das Programmieren mit C# 4.0. Zentrum für Informations-, Medien- und Kommunikationstechnologie (ZIMK) an der Universität Trier (2011)

A

Quelltexte

In diesem Anhang sind einige wichtige Quelltexte aufgeführt.

A.1 Log Klassen

```
1  namespace Server.Logging.AbstractLogs
2  {
3      [DataContract(IsReference = true)]
4      [KnownType(typeof(InternalLog)), KnownType(typeof(ExternalLog))]
5      public abstract class Log : Entity
6      {
7          protected Log()
8          {
9              Timestamp = DateTime.Now;
10         }
11
12         [DataMember]
13         public DateTime Timestamp { get; set; }
14         //List of Logs-IDs which are dependent on this log
15         [DataMember]
16         public List<string> PreviousLogs { get; set; } = new List<string>();
17         //Type of the Log for the representation in the visualization tool
18         [DataMember]
19         [NotMapped]
20         public virtual string DisplayName {
21             get { return GetType().ToString().Substring(GetType().ToString().LastIndexOf(".")+1); }
22             set {}
23         }
24         //Log-ID of the Log from which this log dependent on
25         public string DependentLog { get; set; }
26     }
27 }
```

```
1  namespace Server.Logging.AbstractLogs
2  {
3      [KnownType(typeof(EffectRuleApplicationLog)), KnownType(typeof(ExecutionRuleApplicationLog))]
4      [DataContract(IsReference = true)]
5      public abstract class InternalProcessLog : InternalLog
6      {
7          //ID of the instance on which the rule has been applied
8          [DataMember]
```

```
9           public string Instance { get; set; }
10          //Name of the rule which has been applied
11          [DataMember]
12          public string RuleName { get; set; }
13          //List of the preconditions of the rule
14          [DataMember]
15          public List<string> Preconditions { get; set; } = new List<string>();
16          [DataMember]
17          [NotMapped]
18          public override string DisplayName
19          {
20              get
21              {
22                  return GetType().ToString().Substring(GetType().ToString().LastIndexOf(".") + 1) + "  " + Instance;
23              }
24              set { }
25          }
26      }
27  }
```

```
1   namespace Server.Logging.InternalProcessLogs
2   {
3       [DataContract(IsReference = true)]
4       public class EffectRuleApplicationLog : InternalProcessLog
5       {
6           [DataMember]
7           public List<string> Effects { get; set; } = new List<string>();
8       }
9   }
```

```
1   namespace Server.Logging.IncomingExternalLogs
2   {
3       [DataContract(IsReference = true)]
4       public class ValueChangeLog : IncomingExternalLog
5       {
6           //ID of the attribute which is changed
7           [DataMember]
8           public string ValueAttributeInstance { get; set; }
9           //Old value of the attribute
10          [DataMember]
11          public string OldValue { get; set; }
12          //New value of the attribute
13          [DataMember]
14          public string NewValue { get; set; }
15      }
16  }
```

```
1   namespace Server.Logging.PermissionLogs
2   {
3       [DataContract(IsReference = true)]
4       [KnownType(typeof(BackwardJumpPermissionLog)), KnownType(typeof(ObjectInstantiationPermissionLog)),
5           KnownType(typeof(RwPermissonLog)), KnownType(typeof(StateExecutionPermissionLog)),
6           KnownType(typeof(TransitionCommitPermissionLog))]
7       public abstract class PermissionLog : OutgoingExternalLog
8       {
9           //Name of the user
10          [DataMember]
11          public string User { get; set; }
12          //Result of the permission evaluation
```

```
13          [DataMember]
14          public bool Result { get; set; }
15      }
16  }
```

```
1   namespace Server.Logging.PermissionLogs
2   {
3       [DataContract(IsReference = true)]
4       public class RwPermissonLog : PermissionLog
5       {
6           [DataMember]
7           public string Permission { get; set; }
8           [DataMember]
9           public string StateInstance { get; set; }
10          [DataMember]
11          public string ValueAttributeInstance { get; set; }
12      }
13  }
```

```
1   namespace Server.Logging
2   {
3       [DataContract(IsReference = true)]
4       public class SystemLog : InternalLog
5       {
6           [DataMember]
7           public string Type { get; set; }
8           [DataMember]
9           public string Message { get; set; }
10      }
11  }
```

A.2 Logger Klasse

```
1   namespace Server.Logging
2   {
3       public static partial class Logger
4       {
5           public static LoggingLevel LoggingLevel { get; set; }
6           public static Log TopLog { get; set; }
7           public static List<InternalProcessLog> InternalLogs { get; set; } = new List<InternalProcessLog>();
8
9           public  static void LogEffectRuleApplication<T>(AbstractEffectRule<T> abstractEffectRule, T instance,
10              List<PreconditionEvaluation> preconditionEvaluationResult, List<EffectApplication> effectApplications)
11              where T : Entity
12          {
13              if (preconditionEvaluationResult.Any(pre => !pre.IsValid)) return;
14              var log = new EffectRuleApplicationLog
15              {
16                  Instance = instance.WCFID,
17                  RuleName = abstractEffectRule.Name,
18                  Preconditions = preconditionEvaluationResult.Select(precondition => "("
19                      + precondition.Precondition.Expression + ")"
20                      + precondition.Precondition.ConditionList.FunctionsToString()).ToList(),
21                  Effects = (from effectApplication in effectApplications
```

```
                        let restResult = effectApplication.RestrictionEvaluations.All(x => x.EvaluationResult)
                        from intermediate in effectApplication.Intermediates
                        select "(" + intermediate + "("+intermediate.WCFID+"))"
                            + effectApplication.EffectItemApplications.Select(x => x.EffectItem)
                                .FunctionsToString()).ToList(),
            };
            if (abstractEffectRule is ReactionRuleRr01 || abstractEffectRule is ReactionRuleRr20)
            {
                TopLog = log;
                InternalLogs.Add(log);
            }
            else
            {
                log.DependentLog = TopLog.WCFID;
                InternalLogs.Add(log);
            }
            DB.Context.Logs.Add(log);
        }

        public static void LogExecutionRuleApplication<T>(AbstractExecutionRule<T> abstractExecutionRule,
            T instance, List<PreconditionEvaluation> preconditionEvaluationResult,
            ExecutionEventRaise executionEventRaise) where T : Entity
        {
            var log = new ExecutionRuleApplicationLog
            {
                Instance = instance.WCFID,
                RuleName = abstractExecutionRule.Name,
                Preconditions = preconditionEvaluationResult.Select(precondition => "("
                    + precondition.Precondition.Expression + ")"
                    + precondition.Precondition.ConditionList.FunctionsToString()).ToList(),
                ExecutionEvent = executionEventRaise.Type.ToString(),
                DependentLog = TopLog.WCFID
            };
            InternalLogs.Add(log);
            DB.Context.Logs.Add(log);
        }

        public static void LogInvalidationRuleApplication<T>(AbstractInvalidationRule<T> abstractInvalidationRule,
            T instance, List<PreconditionEvaluation> preconditionEvaluationResult,
            ExecutionEventRaise invalidationEventRaise) where T : Entity
        {
            var log = new ExecutionRuleApplicationLog
            {
                Instance = instance.WCFID,
                RuleName = abstractInvalidationRule.Name,
                Preconditions = preconditionEvaluationResult.Select(precondition => "("
                    + precondition.Precondition.Expression + ")"
                    + precondition.Precondition.ConditionList.FunctionsToString()).ToList(),
                ExecutionEvent = invalidationEventRaise.Type.ToString(),
                DependentLog = TopLog.WCFID
            };
            InternalLogs.Add(log);
            DB.Context.Logs.Add(log);
        }
    }
}
```

```
namespace Server.Logging
{
    public partial class Logger
```

```
4   {
5       public static void LogNewObjectInstantiation(ObjectType objectType, ObjectInstance objectInstance,
6          DataStructure dataStructure)
7       {
8           //initialize log
9           var log = new ObjectInstantiationLog
10          {
11              Process = objectInstance.MicroProcessInstance,
12              ObjectType = objectType.Name,
13              ObjectInstance = objectInstance.Name,
14              DataStructure = dataStructure.ProjectName
15          };
16          //set the DependentLog property of the logs in the InternalLogs list
17          foreach (var internalLog in Logger.InternalLogs)
18          {
19              internalLog.DependentLog = log.WCFID;
20          }
21          DB.Context.Logs.Add(log);
22      }
23
24      public static void LogNewUserInstantiation(UserInstance userInstance)
25      {
26          var log = new UserInstantiationLog
27          {
28              UserInstance = userInstance.Username
29          };
30          DB.Context.Logs.Add(log);
31      }
32
33      public static void LogObjectInstanceDeletion(ObjectInstance objectInstance)
34      {
35          var log = new ObjectDeletionLog
36          {
37              Process = objectInstance.MicroProcessInstance,
38              ObjectInstance = objectInstance.Name
39          };
40          DB.Context.Logs.Add(log);
41      }
42
43      public static void LogAttributeValueChange(ValueAttributeInstance valueAttributeInstance, string oldValue,
44          string newValue)
45      {
46          var log = new ValueChangeLog
47          {
48              Process = valueAttributeInstance.Parent.MicroProcessInstance,
49              ValueAttributeInstance = valueAttributeInstance.Name,
50              OldValue = oldValue,
51              NewValue = newValue
52          };
53          foreach (var internalLog in Logger.InternalLogs)
54          {
55              internalLog.DependentLog = log.WCFID;
56          }
57          DB.Context.Logs.Add(log);
58
59      }
60      public static void LogBackwardTransitionCommitment(BackwardTransitionInstance backwardTransitionInstance)
61      {
62          var log = new BackwardTransitionCommitmentLog
63          {
```

```
64                  Process = backwardTransitionInstance.MicroProcessInstance,
65                  BackwardTransitionInstance = backwardTransitionInstance.Name
66              };
67              foreach (var internalLog in Logger.InternalLogs)
68              {
69                  internalLog.DependentLog = log.WCFID;
70              }
71              DB.Context.Logs.Add(log);
72          }
73
74          public static void LogTransitionCommitment(MicroTransitionInstance microTransitionInstance)
75          {
76              var log = new TransitionCommitmentLog
77              {
78                  Process = microTransitionInstance.MicroProcessInstance,
79                  MicroTransitionInstance = microTransitionInstance.Name
80              };
81              foreach (var internalLog in Logger.InternalLogs)
82              {
83                  internalLog.DependentLog = log.WCFID;
84              }
85              DB.Context.Logs.Add(log);
86          }
87
88          public static void LogMicroStepCommitment(MicroStepInstance microStepInstance)
89          {
90              var log = new MicroStepCommitmentLog
91              {
92                  Process = microStepInstance.MicroProcessInstance,
93                  MicroStepInstance = microStepInstance.Name
94              };
95              foreach (var internalLog in Logger.InternalLogs)
96              {
97                  internalLog.DependentLog = log.WCFID;
98              }
99              DB.Context.Logs.Add(log);
100         }
101
102         public static void LogFormFieldsRequest(string userInstanceId, ObjectInstance objectInstance,
103             StateInstance currentState)
104         {
105             var log = new FormFieldRequestLog
106             {
107                 UserInstance = userInstanceId,
108                 ObjectInstance = objectInstance.Name,
109                 StateInstance = currentState.Name
110             };
111             DB.Context.Logs.Add(log);
112         }
113
114         public static void LogNewDataStructureInstantiation(DataStructure dataStructure)
115         {
116             var log = new DataStructureLog
117             {
118                 DataStructure = dataStructure.ProjectName,
119                 Type = "Instantiation"
120             };
121             DB.Context.Logs.Add(log);
122         }
123
```

```
124         public static void LogDataStructureSerialization(DataStructure dataStructure)
125         {
126             var log = new DataStructureLog
127             {
128                 DataStructure = dataStructure.ProjectName,
129                 Type = "Serialization"
130             };
131             DB.Context.Logs.Add(log);
132         }
133 
134         public static void LogNewDataStructureDeployment(DataStructure dataStructure)
135         {
136             var log = new DataStructureLog
137             {
138                 DataStructure = dataStructure.ProjectName,
139                 Type = "Deployment"
140             };
141             DB.Context.Logs.Add(log);
142         }
143 
144         public static void LogAttributeAdd(ObjectInstance objectInstance,
145             ValueAttributeInstance valueAttributeInstance)
146         {
147             var log = new AddAttributeLog
148             {
149                 Process = objectInstance.MicroProcessInstance,
150                 ObjectInstance = objectInstance.Name,
151                 ValueAttributeInstance = valueAttributeInstance.Name
152             };
153             DB.Context.Logs.Add(log);
154         }
155 
156         public static void LogObjectInstancesLinked(ObjectInstance objectInstance1,
157             ObjectInstance objectInstance2)
158         {
159             var log = new LinkingLog
160             {
161                 IsLinked = true,
162                 Source = objectInstance1.Name,
163                 Target = objectInstance2.Name
164             };
165             DB.Context.Logs.Add(log);
166         }
167 
168         public static void LogObjectInstancesUnlinked(ObjectInstance objectInstance1,
169             ObjectInstance objectInstance2)
170         {
171             var log = new LinkingLog
172             {
173                 IsLinked = false,
174                 Source = objectInstance1.Name,
175                 Target = objectInstance2.Name
176             };
177             DB.Context.Logs.Add(log);
178         }
179     }
180 }
```

```
1  namespace Server.Logging
2  {
```

```
3       public partial class Logger
4       {
5           public static void LogReadWritePermissionNotFound(UserInstance userInstance,
6              ValueAttributeInstance valueAttributeInstance,
7              StateInstance stateInstance)
8           {
9               var log = new RwPermissonLog
10              {
11                  Result = false,
12                  User = userInstance.Username,
13                  ValueAttributeInstance = valueAttributeInstance.Name,
14                  StateInstance = stateInstance.Name
15              };
16              DB.Context.Logs.Add(log);
17          }
18
19          public static void LogReadWritePermissionFound(UserInstance userInstance,
20              ValueAttributeInstance valueAttributeInstance,
21              StateInstance stateInstance, AttributeReadWritePermission permission)
22          {
23              var log = new RwPermissonLog
24              {
25                  Result = true,
26                  User = userInstance.Username,
27                  ValueAttributeInstance = valueAttributeInstance.Name,
28                  StateInstance = stateInstance.Name,
29                  Permission = permission.Name
30              };
31              DB.Context.Logs.Add(log);
32          }
33
34          public static void LogObjectInstantiationPermissionFound(UserInstance userInstance, ObjectType objectType,
35              ObjectInstantiationPermission permission)
36          {
37              var log = new ObjectInstantiationPermissionLog
38              {
39                  Result = true,
40                  User = userInstance.Username,
41                  ObjectType = objectType.Name,
42                  Permission = permission.Name
43              };
44              DB.Context.Logs.Add(log);
45          }
46
47          public static void LogObjectInstantiationPermissionNotFound(UserInstance userInstance, ObjectType objectType)
48          {
49              var log = new ObjectInstantiationPermissionLog
50              {
51                  Result = false,
52                  User = userInstance.Username,
53                  ObjectType = objectType.Name
54              };
55              DB.Context.Logs.Add(log);
56          }
57
58          public static void LogTransitionCommitPermissionFound(UserInstance userInstance,
59              MicroTransitionInstance transitionInstance, TransitionCommitPermission permission)
60          {
61              var log = new TransitionCommitPermissionLog
62              {
```

```
63                 Result = true,
64                 User = userInstance.Username,
65                 MicroTransitionInstance = transitionInstance.Name,
66                 Permission = permission.Name
67             };
68             DB.Context.Logs.Add(log);
69         }
70 
71         public static void LogTransitionCommitPermissionNotFound(UserInstance userInstance,
72             MicroTransitionInstance transitionInstance)
73         {
74             var log = new TransitionCommitPermissionLog
75             {
76                 Result = false,
77                 User = userInstance.Username,
78                 MicroTransitionInstance = transitionInstance.Name
79             };
80             DB.Context.Logs.Add(log);
81         }
82 
83         public static void LogBackwardJumpPermissionFound(UserInstance userInstance,
84             BackwardTransitionInstance btransitionInstance, BackwardJumpPermission permission)
85         {
86             var log = new BackwardJumpPermissionLog
87             {
88                 Result = true,
89                 User = userInstance.Username,
90                 BackwardTransitionInstance = btransitionInstance.Name,
91                 Permission = permission.Name
92             };
93             DB.Context.Logs.Add(log);
94         }
95 
96         public static void LogBackwardJumpPermissionNotFound(UserInstance userInstance,
97             BackwardTransitionInstance btransitionInstance)
98         {
99             var log = new BackwardJumpPermissionLog
100            {
101                Result = false,
102                User = userInstance.Username,
103                BackwardTransitionInstance = btransitionInstance.Name
104            };
105            DB.Context.Logs.Add(log);
106        }
107 
108        public static void LogStateExecutionPermissionFound(UserInstance userInstance, StateInstance stateInstance,
109            StateExecutionPermission permission)
110        {
111            var log = new StateExecutionPermissionLog
112            {
113                Result = true,
114                User = userInstance.Username,
115                StateInstance = stateInstance.Name,
116                Permission = permission.Name
117            };
118            DB.Context.Logs.Add(log);
119        }
120 
121        public static void LogStateExecutionPermissionNotFound(UserInstance userInstance, StateInstance stateInstance)
122        {
```

```
123             var log = new StateExecutionPermissionLog
124             {
125                 Result = false,
126                 User = userInstance.Username,
127                 StateInstance = stateInstance.Name
128             };
129             DB.Context.Logs.Add(log);
130         }
131     }
132 }
```

```
1  namespace Server.Logging
2  {
3      public partial class Logger
4      {
5          public static void LogUserLogin(UserInstance user)
6          {
7              var log = new SystemLog
8              {
9                  Type = "Information",
10                 Message = user.Username
11                     + " successfully logged in"
12             };
13             DB.Context.Logs.Add(log);
14         }
15
16         public static void LogFailedUserLogin(string message)
17         {
18             var log = new SystemLog
19             {
20                 Type = "Error",
21                 Message = message
22             };
23             DB.Context.Logs.Add(log);
24         }
25     }
26 }
```

A.3 Visualisierungstool

```
1  namespace Logging
2  {
3      public partial class MainPage
4      {
5          private async void DrawGraph()
6          {
7              graph.Children.Clear();
8              graph.ColumnDefinitions.Clear();
9              graph.RowDefinitions.Clear();
10
11             _dependentLogs = await _phiLharmonicFlowsServiceClient.GetDependentLogsAsync(_selectedLog.WCFID);
12             foreach (var actualLog in _dependentLogs)
13             {
14                 if (!(actualLog is InternalProcessLog)) continue;
15                 foreach (var dependentLog in _dependentLogs)
```

```
16                  {
17                      if (actualLog.WCFID.Equals(dependentLog.WCFID)) break;
18                      if (!(dependentLog is EffectRuleApplicationLog)) continue;
19                      MatchPreconditionsToEffects(actualLog as InternalProcessLog,
20                          dependentLog as EffectRuleApplicationLog);
21                  }
22                  if (actualLog.PreviousLogs.Count == 0)
23                      actualLog.PreviousLogs.Add(_selectedLog.WCFID);
24              }
25              var controlList = new List<Log>(_dependentLogs);
26              var numberOfRows = 1;
27              var row = new RowDefinition();
28              graph.RowDefinitions.Add(row);
29
30              var top = new Button
31              {
32                  Background = _blue,
33                  Content = (_selectedLog is EffectRuleApplicationLog) ?
34                      (_selectedLog as EffectRuleApplicationLog).RuleName : _selectedLog.DisplayName,
35                  Tag = _selectedLog,
36                  HorizontalAlignment = HorizontalAlignment.Center,
37                  VerticalAlignment = VerticalAlignment.Center
38              };
39              top.Click += Graph_Btn_Clicked;
40              Grid.SetRow(top, 0);
41              graph.Children.Add(top);
42
43              var levelLogs = new List<Log> { _selectedLog };
44              while (controlList.Count != 0)
45              {
46                  var tempList = new List<Log>();
47                  foreach (var levelLog in levelLogs)
48                  {
49                      tempList.AddRange(GetNextLogs(levelLog));
50                  }
51                  tempList = tempList.Distinct().ToList();
52                  foreach (var log in tempList.Where(log => !controlList.Any(x => x.WCFID.Equals(log.WCFID))))
53                  {
54                      tempList = tempList.Where(x => x.WCFID != log.WCFID).ToList();
55                  }
56                  var panel = new StackPanel
57                  {
58                      Orientation = Orientation.Horizontal,
59  //                    Padding = new Thickness(10),
60                      HorizontalAlignment = HorizontalAlignment.Center,
61                      VerticalAlignment = VerticalAlignment.Center
62                  };
63                  foreach (var log in tempList)
64                  {
65                      var item = new Button
66                      {
67                          Background = _blue,
68                          Content = (log is InternalProcessLog) ?
69                              (log as InternalProcessLog).RuleName : log.DisplayName,
70                          Tag = log,
71                          Margin = new Thickness(10,0,10,0)
72                      };
73                      item.Click += Graph_Btn_Clicked;
74                      panel.Children.Add(item);
75
```

```
76                      controlList = controlList.Where(x => x.WCFID != log.WCFID).ToList();
77                  }
78                  var nextRow = new RowDefinition();
79                  graph.RowDefinitions.Add(nextRow);
80                  Grid.SetRow(panel, numberOfRows);
81                  graph.Children.Add(panel);
82                  numberOfRows++;
83
84                  levelLogs = new List<Log>(tempList);
85              }
86
87              graph.UpdateLayout();
88              foreach (var log in _dependentLogs)
89              {
90                  foreach (var nextLog in GetNextLogs(log))
91                  {
92                      DrawRelation(log, nextLog);
93                  }
94              }
95              foreach (var nextLog in GetNextLogs(_selectedLog))
96              {
97                  DrawRelation(_selectedLog, nextLog);
98              }
99              graphInfo.Text = "Number of Logs: "+(_dependentLogs.Count+1)
100                 +"   Number of Reaction Rule Logs: "
101                 +_dependentLogs.Count(x =>
102                     x is InternalProcessLog && (x as InternalProcessLog).RuleName.StartsWith("Reaction"))
103                 +"   Number of Marking Rule Logs: "
104                 + _dependentLogs.Count(x =>
105                     x is InternalProcessLog && (x as InternalProcessLog).RuleName.StartsWith("Marking"))
106                 +"   Number of Execution Event Logs: "
107                 + _dependentLogs.Count(x => x is ExecutionRuleApplicationLog);
108
109
110         }
111         private void MatchPreconditionsToEffects(InternalProcessLog actualLog,
112             EffectRuleApplicationLog dependentLog)
113         {
114             foreach (var pre in actualLog.Preconditions)
115             {
116                 if (!pre.Contains("IsMarked")) continue;
117                 foreach (var effect in dependentLog.Effects)
118                 {
119                     if (!effect.Contains("AssignMarking")) continue;
120                     var preMarkings = pre.Substring(pre.LastIndexOf("(") + 1).Trim(')').Split('.');
121                     var effectMarking = effect.Substring(effect.LastIndexOf("(") + 1).Trim(')');
122                     foreach (var marking in preMarkings)
123                     {
124                         var startIndex = effect.IndexOf('(', effect.IndexOf('(') + 1)+1;
125                         var endIndex = effect.IndexOf(')');
126                         if (marking.Equals(effectMarking) &&
127                             actualLog.Instance.Equals(effect.Substring(startIndex, endIndex-startIndex)))
128                         {
129                             actualLog.PreviousLogs.Add(dependentLog.WCFID);
130                         }
131                     }
132                 }
133             }
134         }
135         private List<Log> GetNextLogs(Log log)
```

```
136         {
137             return _dependentLogs.Where(dependentLog =>
138                 dependentLog.PreviousLogs.Any(x => x.Equals(log.WCFID))).ToList();
139         }
140         private void DrawRelation(Log sourceLog, Log targetLog)
141         {
142             Button source = null;
143             Button target = null;
144             StackPanel sourcePanel = null;
145             var horizontal = false;
146             foreach (var child in graph.Children)
147             {
148                 if (child is Button && ((Log) (child as Button).Tag).WCFID.Equals(sourceLog.WCFID))
149                 {
150                     source = child as Button;
151                 }
152                 else if (child is StackPanel &&  (source == null || target == null))
153                 {
154                     foreach (var button in (child as StackPanel).Children)
155                     {
156                         if (((Log) (button as Button).Tag).WCFID.Equals(targetLog.WCFID))
157                         {
158                             target = button as Button;
159                             if (sourcePanel != null && sourcePanel.Equals(child))
160                             {
161                                 horizontal = true;
162                             }
163                         }
164                         if (((Log)(button as Button).Tag).WCFID.Equals(sourceLog.WCFID))
165                         {
166                             source = button as Button;
167                             sourcePanel = child as StackPanel;
168                         }
169                         if(source != null && target != null)  break;
170                     }
171                 }
172             }
173             if (source == null || target == null)
174             {
175                 return;
176             }
177             Point sourcePoint, targetPoint, point2, point3;
178             if (horizontal)
179             {
180                 sourcePoint = source.TransformToVisual(graph)
181                     .TransformPoint(new Point((source.ActualWidth * 0.5)-10, -5));
182                 targetPoint = target.TransformToVisual(graph)
183                     .TransformPoint(new Point((target.ActualWidth * 0.5)+10, -5));
184                 point2 = source.TransformToVisual(graph)
185                     .TransformPoint(new Point((source.ActualWidth * 0.5)-10, -20));
186                 point3 = target.TransformToVisual(graph)
187                     .TransformPoint(new Point((target.ActualWidth * 0.5)+10, -20));
188             }
189             else
190             {
191                 sourcePoint = source.TransformToVisual(graph)
192                     .TransformPoint(new Point(source.ActualWidth*0.5, source.ActualHeight-5));
193                 targetPoint = target.TransformToVisual(graph)
194                     .TransformPoint(new Point(target.ActualWidth * 0.5, -5));
195             }
```

```
196
197             var angle = (180 / Math.PI) * Math.Atan2(targetPoint.Y - sourcePoint.Y, targetPoint.X - sourcePoint.X);
198             if(horizontal)
199                 angle = (180 / Math.PI) * Math.Atan2(targetPoint.Y - point3.Y, targetPoint.X - point3.X);
200
201             var arrowGroup = new GeometryGroup();
202             if (horizontal)
203             {
204                 arrowGroup.Children.Add(new LineGeometry
205                 {
206                     StartPoint = sourcePoint,
207                     EndPoint = point2
208                 });
209                 arrowGroup.Children.Add(new LineGeometry
210                 {
211                     StartPoint = point2,
212                     EndPoint = point3
213                 });
214                 arrowGroup.Children.Add(new LineGeometry
215                 {
216                     StartPoint = point3,
217                     EndPoint = targetPoint
218                 });
219             }
220             else {
221                 arrowGroup.Children.Add(new LineGeometry
222                 {
223                     StartPoint = sourcePoint,
224                     EndPoint = targetPoint
225                 });
226             }
227
228             //draw actual transition line
229             var transition = new Path
230             {
231                 Data = arrowGroup,
232                 StrokeDashArray = new DoubleCollection { 3, 2 },
233                 Stroke = _blue,
234                 StrokeThickness = 1,
235                 VerticalAlignment = VerticalAlignment.Top,
236                 HorizontalAlignment = HorizontalAlignment.Left
237             };
238             Grid.SetRowSpan(transition, int.MaxValue);
239             graph.Children.Add(transition);
240
241             var headGroup = new GeometryGroup();
242             headGroup.Children.Add(
243                 new LineGeometry
244                 {
245                     StartPoint = new Point(targetPoint.X - 10, targetPoint.Y),
246                     EndPoint = targetPoint,
247                     Transform = new RotateTransform
248                     {
249                         Angle = (angle + 30),
250                         CenterX = targetPoint.X,
251                         CenterY = targetPoint.Y
252                     }
253                 });
254             headGroup.Children.Add(new LineGeometry
255             {
```

```
256                 StartPoint = new Point(targetPoint.X - 10, targetPoint.Y),
257                 EndPoint = targetPoint,
258                 Transform = new RotateTransform
259                 {
260                     Angle = (angle - 30),
261                     CenterX = targetPoint.X,
262                     CenterY = targetPoint.Y
263                 }
264             });
265 
266             var head = new Path
267             {
268                 Data = headGroup,
269                 Stroke = _blue,
270                 StrokeThickness = 1,
271                 VerticalAlignment = VerticalAlignment.Top,
272                 HorizontalAlignment = HorizontalAlignment.Left
273             };
274             Grid.SetRowSpan(head, int.MaxValue);
275             graph.Children.Add(head);
276         }
277     }
278 }
```

```
1  namespace Logging
2  {
3      public partial class MainPage
4      {
5          private bool _permissionFilter, _systemFilter, _incomingExternalFilter, _internalFilter, _dateFilter,
6              _externalFilter, _filterOn;
7          private int _permissionCount, _systemCount, _incomingExternalCount, _internalCount, _externalCount;
8          private void SetFilterView()
9          {
10             permissionCb.IsChecked = _permissionFilter;
11             permissionCb.Content = "Permission Logs: " + _permissionCount;
12 
13             systemCb.IsChecked = _systemFilter;
14             systemCb.Content = "System Logs: " + _systemCount;
15 
16             incomingExternalCb.IsChecked = _incomingExternalFilter;
17             incomingExternalCb.Content = "Incoming External Logs: " + _incomingExternalCount;
18 
19             internalCb.IsChecked = _internalFilter;
20             internalCb.Content = "Internal Logs: " + _internalCount;
21 
22             externalCb.IsChecked = _externalFilter;
23             externalCb.Content = "External Logs: " + _externalCount;
24 
25             dateCb.IsChecked = _dateFilter;
26         }
27 
28         private void FilterLists()
29         {
30             topBox.Items.Clear();
31             var list = new List<Log>();
32             if (_filterOn)
33             {
34                 var count = 0;
35                 foreach (var log in _topLogs)
36                 {
```

```
37                  if (_dateFilter && log.Timestamp.Date.CompareTo(from.Date.Date) < 0 &&
38                      log.Timestamp.Date.CompareTo(to.Date.Date) > 0)
39                  {
40                      continue;
41                  }
42                  if (log is PermissionLog)
43                  {
44                      if (_permissionFilter)
45                      {
46                          list.Add(log);
47                          count++;
48                      }
49                      continue;
50                  }
51                  if (log is IncomingExternalLog)
52                  {
53                      if (_incomingExternalFilter)
54                      {
55                          list.Add(log);
56                          count++;
57                      }
58                      continue;
59                  }
60                  if (_internalFilter && log is InternalProcessLog)
61                  {
62                      list.Add(log);
63                      count++;
64                      continue;
65                  }
66                  if (_externalFilter && log is ExternalLog)
67                  {
68                      list.Add(log);
69                      count++;
70                      continue;
71                  }
72                  if (_systemFilter && log is SystemLog)
73                  {
74                      list.Add(log);
75                      count++;
76                  }
77              }
78              topText.Text = "Top Logs " +count;
79          }
80          else
81          {
82              list.AddRange(_topLogs);
83              topText.Text = "Top Logs " + _topLogs.Count;
84          }
85          Sort(list);
86      }
87
88      private void Checked(object sender, RoutedEventArgs e)
89      {
90          var cb = sender as CheckBox;
91          if (cb == null || _trigger) return;
92          if (cb.Equals(permissionCb))
93          {
94              _permissionFilter = !_permissionFilter;
95          }
96          else if (cb.Equals(systemCb))
```

```
97              {
98                  _systemFilter = !_systemFilter;
99              }
100             else if (cb.Equals(incomingExternalCb))
101             {
102                 _incomingExternalFilter = !_incomingExternalFilter;
103             }
104             else if (cb.Equals(internalCb))
105             {
106                 _internalFilter = !_internalFilter;
107             }
108             else if (cb.Equals(externalCb))
109             {
110                 _externalFilter = !_externalFilter;
111             }
112             else if (cb.Equals(dateCb))
113             {
114                 _dateFilter = !_dateFilter;
115             }
116             _filterOn = _incomingExternalFilter || _internalFilter || _systemFilter || _permissionFilter
117                 || _dateFilter || _externalFilter;
118         }
119     }
120 }
```

Abbildungsverzeichnis

Name: Markus Leitz Matrikelnummer: 750984

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Markus Leitz