Modellgetriebene Lösung von ausgewählten Optimierungsproblemen auf Geschäftsprozessen

Bachelorarbeit

Philipps-Universität Marburg
Fachbereich 12 Mathematik und Informatik

dargelegt von: Paul Groß

betreuende Prüferin: Prof. Dr. Gabriele Taentzer

Abgabedatum: 11.10.2024

Abbildungsverzeichnis

Abkürzungsverzeichnis

ARC Activity Relation Cohesion

AIC Activity Information Cohesion

PCCR Process Coupling/Cohesion Ratio

EMF Eclipse Modeling Framework

NAC negative application condition

PAC positive application condition

DSL domain specific language

LHS left-hand side

RHS right-hand side

Inhaltsverzeichnis

1 Einleitung

Geschäftsprozesse spielen eine zentrale Rolle bei der Wertschöpfung von Unternehmen, da sie die Grundlage für die Leistungsbewertung bieten, welche Verbesserungspotentiale und Raum für neue Innovationen aufdeckt. Unternehmen sind gefordert ihre Prozesse kontinuierlich effizienter und flexibler zu gestalten, um Kosten zu senken, die Produktivität zu steigern und schließlich im Wettbewerb zu bestehen. [10]

1.1 Problemstellung

Ein Problem bei der Modellierung von Prozessen ist die Aufteilung der Prozesse in ausführbare Einheiten, welche Aktivitäten genannt werden. Insbesondere besteht bei der Definition von Aktivitäten das Problem, dass Arbeitsvorgänge innerhalb einer Aktivität verschiedene Ressourcen benötigen. Im Bereich administrativer Prozesse sind diese benötigten Ressourcen Informationen. Diese sollten innerhalb einer einzelnen Aktivität homogen gestaltet werden, um eine Kohäsion der Arbeitsschritte zu gewährleisten. Des Weiteren existieren Abhängigkeiten zwischen den einzelnen Aktivitäten. Am Ende liegende Aktivitäten können erst durchgeführt werden, wenn durch vorhergende Tätigkeiten die benötigten Vorbedingungen erfüllt wurden. Sind Aktivitäten auf diese Weise verbunden, werden sie auch als „gekoppelt“ bezeichnet. [8]

Diese Einschränkungen bilden eine Herausforderung bei der Findung von optimalen Prozessen. Zusätzlich stehen die gesetzten Ziele im Konflikt, was das Modellieren eines optimalen Prozesses weiter erschwert.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, Prozessmodelle zu finden, deren Aktivitäten einerseits eine hohe Kohäsion, gleichzeitig aber auch eine geringe Kopplung aufweisen. Die verwendete Methode zur Optimierung der Geschäftsprozesse ist der modellgetriebene Ansatz. Hierbei wird darauf abgezielt, Geschäftsprozesse in Form von Modellen als Softwareartefakte abzubilden, was die direkte Analyse und Optimierung der Prozesse ermöglicht. Es können Transformationen implementiert werden, welche die Modelle in einen gewünschten Zielzustand versetzen oder an gestellte Anforderungen anpassen. Allgemein erlaubt die modellbasierte Abbildung der Prozesse durch ihre abstrahierte Sichtweise des Codes eine erleichterte Kommunikation. [4]

1.3 Aufbau der Arbeit

Anschließend an dieses Kapitel sollen in Kapitel 2 zunächst die Grundlagen geschaffen werden, indem ein Überblick über die Problemdomäne und die

spezifischen Ziele der Prozessoptimierung anhand eines konkreten Anwendungsbeispiels gegeben wird. Hierbei werden auch die verwendeten Metriken formal definiert. Anschließend werden die wichtigsten Konzepte der modellgetriebenen Entwicklung und -Optimierung erklärt, sowie ein Überblick über die Funktionsweise von genetischen Algorithmen gegeben. Um Kapitel 2 abzuschließen werden die Tools und Frameworks erläutert, welche im Rahmen dieser Arbeit verwendet werden. In Kapitel 3 wird das Design der erstellten Software vorgestellt. Zunächst wird ein Metamodell entworfen, welches zur Prozessabbildung verwendet wird. Es folgt das Design von Mutationsoperatoren, um die erstellten Prozessmodelle zu transformieren. Schließlich wird erläutert, wie die definierten Metriken als Zielfunktionen im Optimierungsprozess verwendet werden können. Nach der Vorstellung des Designs folgt in Kapitel 4 die konkrete Implementierung der Software. Hierbei werden die in Kapitel 3 erarbeiteten Ergebnisse aufgegriffen und umgesetzt. Abschließend werden in Kapitel 5 die Ergebnisse der Optimierung vorgestellt und interpretiert. Dazu sollen zunächst die Forschungsfragen und der Aufbau der Experimente beschrieben werden. Außerdem werden die Threats to validity erläutert, die zu einer Einschränkung des Ergebnisses führen könnten.

2 Grundlagen

In diesem Kapitel sollen die Grundlagen näher erläutert werden. Dafür werden zunächst Prozesse, sowie Metriken zur Bewertung von Prozessen vorgestellt. Anschließend folgt das Konzept der Modellgetriebenen Optimierung, welches zur Generierung von Prozessmodellen angewendet wird. Abschließend folgt eine Vorstellung der Tools und Frameworks, die im Rahmen dieser Arbeit verwendet werden.

2.1 Prozesse

Im ersten Schritt wird erklärt, was unter einem Prozess und dessen Bestandteilen zu verstehen ist. Anschließend wird ein Modell zur Abbildung von Prozessen besprochen. Dazu werden zwei Metriken vorgestellt, welche zur Bewertung der Prozesse herangezogen werden.

2.1.1 Gestaltung und Struktur von Prozessen

Ein Prozess beschreibt eine Abfolge von Arbeitsschritten, durch die ein vorgegebenes Ziel erreicht werden soll. Arbeitsschritte benötigen physische Materialien oder Informationen, um ein Resultat für den folgenden Schritt zu erzeugen. Das Ergebnis eines Prozesses ist somit ebenfalls ein Output, der durch eine definierte Reihenfolge der Arbeitsschritte erreicht wird. Grundsätzlich wird zwischen Fertigungsprozessen, die physische Materialien in ihren Arbeitsschritten benötigen, und administrativen Prozessen, welche Informationen verarbeiten, unterschieden. [8]

Im Rahmen dieser Arbeit wird ausschließlich auf administrative Prozesse eingegangen, da diese eine erhöhte Flexibilität in der Gestaltung der Arbeitsschritte bieten. Während Fertigungsprozesse eine feste Abfolge von Schritten erfordern, um ein Produkt zu erstellen, gewähren administrative Prozesse die Möglichkeit, Informationen auf unterschiedliche Arten zu verarbeiten. [8]

Da an der Durchführung eines Prozesses mehrere Personen beteiligt sind, bietet es sich an, die Arbeitsschritte in Aktivitäten zu gruppieren, welche an die Prozessbeteiligten zugewiesen werden. Zwar sind administrative Prozesse beim Entwurf der Aktivitäten flexibel, jedoch entsteht gleichzeitig das Problem, eine adäquate Strukturierung der Prozesse zu finden. Die Aufteilung in kleine Aktivitäten erhöht die Anzahl der Informationsübergaben zwischen Personen. Dies steigert das Risiko für Informationsverluste und somit die Anzahl möglicher Fehler. Sind Aktivitäten zu groß, wird der Prozess unflexibel. [8]

Ein zentrales Ziel bei der Suche nach optimalen Aktivitäten ist daher eine effektive Modularisierung. Die Informationen innerhalb einer Aktivität sollten eng miteinander in Verbindung stehen. Gleichzeitig sollten die Aktivitäten weitgehend voneinander entkoppelt sein, sodass eine Aktivität nur

wenige Schnittstellen zu anderen Aktivitäten aufweist. Dadurch sinkt die Anzahl der Informationsübergaben. Im weiteren Verlauf werden die Metriken Kohäsion und Kopplung vorgestellt, mit denen die Struktur der Aktivitäten innerhalb eines Prozesses mit Hinblick auf die Modularisierung bewertet werden kann.

2.1.2 Modellierung der vorliegenden Prozesse

Es existieren verschiedene Möglichkeiten einen Prozess als Modell abzubilden. Prozesse erzeugen aus einem oder mehreren Inputs einen Output. Diese Input-Output Perspektive soll sich auch im hier vorgestellten Modell widerspiegeln, welches einen Prozess als Informationselement-Struktur darstellt.

Ein Informationselement ist die kleinste Einheit an Informationen, die verarbeitet werden kann. Jeder Prozess startet mit einer bestimmten Anzahl an Informationselementen, die der initiale Input für den Prozess sind. Arbeitsschritte nehmen einen oder mehrere Inputs entgegen und erzeugen genau einen Output, was als Operation bezeichnet wird. Der Output einer Operation dient schließlich wieder als Input für eine folgende Operation. Das Resultat eines Prozesses wird ebenfalls durch einen oder mehrere Outputs beschrieben. [8]

Ein Beispiel für die Darstellung eines solchen Prozesses ist der Antrag auf Studienförderung in den Niederlanden (Abbildung 1).

Der Prozess besteht aus 27 Informationselementen. Hierzu zählen beispielsweise das Datum, an dem das Studium beginnt (1), oder der Geburtstag des Antragstellenden (5). Diese Informationen können nicht in weitere Informationen zerlegt werden und stellen somit Informationselemente dar. Operationen werden jeweils durch eine Menge von Pfeilen dargestellt, welche die Informationselemente verbinden. Das Informationselement 9 ist die Sozialversicherungsnummer des Vaters des Antragstellenden. Element 10 ist das Jahr, in dem der Antragstellende eine Förderung erhalten möchte. Aufgrund dieser beiden Elemente kann mit einer Operation das Einkommen des Vaters bestimmt werden, welches Element 13 darstellt. Ebenso kann durch die Sozialversicherungsnummer der Mutter (11) und dem Bezugsjahr (10), das Einkommen der Mutter bestimmt werden (14). Das Einkommen der Eltern (15) kann schließlich durch die beiden Elemente 13 und 14 bestimmt werden. Der Output des gesamten Prozesses ist der monetäre Betrag, den der Antragstellende erhalten soll (27). [8]

Auf die Beschreibung aller Informationselemente wird verzichtet, da für eine Definition von Aktivitäten die Nummerierung und die Beziehungen zwischen den Elementen ausreicht.

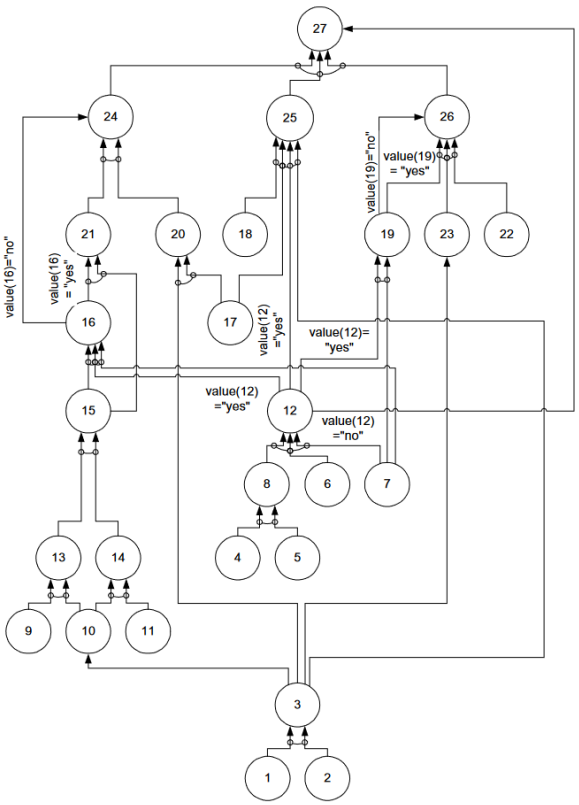

Abbildung 1: Prozess für den Antrag auf Studienförderung

2.1.3 Einführung der Metriken zur Bewertung von Prozessen

Die Aktivitäten eines Prozesses können innerhalb der Informationselement-Struktur als Mengen von Informationselementen und Operationen betrachtet werden. Um kohäsive Aktivitäten zu erstellen, müssen Operationen gefunden werden, welche auf der Grundlage ihrer verwendeten Informationselemente Gemeinsamkeiten aufweisen. Werden Input und Output einer Operation als Menge von Informationselementen dargestellt, können über die Bildung von Schnittmengen mit anderen Operationen Gemeinsamkeiten gefunden werden. Gleichzeitig müssen geeignete Schnittstellen zwischen Aktivitäten gefunden werden, um die Kopplung gering zu halten. Die Kohäsion bezieht sich somit immer auf die Operationen einer einzelnen Aktivität, während die Kopplung beschreibt, wie sehr die gesamten Aktivitäten innerhalb des Prozesses voneinander abhängen.

Die hier zur Bewertung verwendeten Metriken „Kohäsion“ und „Kopplung“ haben ihren Ursprung in der Softwareentwicklung. [8] Hierbei wird innerhalb der Klassen nach hoher Kohäsion und zwischen den Klassen nach niedriger Kopplung gestrebt.

Die Verwendung dieser abgewandelten Metriken in Bezug auf Prozesse ist angemessen, da die Modellierung von Prozessen Ähnlichkeiten zur Modellierung von Software aufweist. Sowohl Software als auch Prozesse befassen sich mit der Verarbeitung von Informationen. Programme verwenden Methoden, welche Informationen als Input entnehmen und einen Output erzeugen. Analog wird dieses Prinzip innerhalb von Prozessen durch Operationen realisiert. Die Struktur von Software und Prozessen ist ebenfalls vergleichbar. Während Programme in Klassen aufgeteilt werden, welche Methoden und Attribute beinhalten, sind Prozesse eine Komposition von Aktivitäten, welche ihrerseits aus Operationen und Informationselementen aufgebaut sind. [8]

Ein Prozessmodell, welches aus stark entkoppelten Aktivitäten besteht, bietet potentiell eine geringere Fehleranfälligkeit bei der Ausführung, da zwischen den einzelnen Akteuren weniger Kommunikation und weniger Austausch von Informationen benötigt wird. Durch eine Verringerung dieser Schnittstellen sinkt auch die Wahrscheinlichkeit, dass Fehler entstehen. [8]

Gleichzeitig können Aktivitäten mit einer hohen Kohäsion die Qualität des Prozesses steigern, da ein Akteur jeweils einen zusammenhängenden Teil des Prozesses ausführt. Aktivitäten mit einer geringen Kohäsion würden im Vergleich einen höheren Informationsaustausch benötigen, welcher wiederum zu einer höheren Anfälligkeit für Fehler führen könnte.[8]

Bei der Modellierung von Prozessen müssen weitere Faktoren betrachtet werden, welche über die Strukturierung der Aktivitäten hinausgehen. Im Rahmen dieser Arbeit wird der Fokus jedoch einzig auf die Ausführung von Prozessen gelegt. Es wird sich nur mit der Suche nach Prozessmodellen befasst, welche mit Hinblick auf die Metriken Kohäsion und Kopplung optimiert sind.

2.1.4 Formale Definition von Prozessen und Metriken

Nach der Vorstellung des vorliegenden Prozessmodells und den Metriken, welche zur Bewertung verwendet werden, sollen diese Konzepte im Folgenden formal definiert werden. Die Definitionen stammen aus [8].

Die Informationselement-Struktur kann als Tupel (D, O) beschrieben werden, wobei D die Menge aller Informationselemente ist und

$$O = \{(p, cs) \in D \times P(D)\}$$

die Menge der Operationen. p stellt den Output dar, während cs die Menge des Inputs ist.

Eine Aktivität innerhalb einer Informationselement-Struktur (D, O) ist eine Teilmenge der Operationen: $t \subseteq O$. Die Menge aller Aktivitäten eines Prozesses wird durch die Menge T dargestellt.

Für ein valides Prozessmodell muss weiterhin gelten, dass alle Operationen in genau einer Aktivität enthalten sind. Außerdem muss eine Sortie-

rung der Aktivitäten möglich sein, sodass eine Reihenfolge existiert, in der die Aktivitäten ausgeführt werden können. Wenn für die Ausführung einer Aktivität ein Informationselement benötigt wird, muss dieses in derselben Aktivität oder in einer vorherigen Aktivität bereits erzeugt worden sein.

Als erste zentrale Metrik soll die Prozesskohäsion definiert werden. Diese wird mithilfe der Activity Relation Cohesion (ARC) und der Activity Information Cohesion (AIC) berechnet.

Für eine Aktivität t innerhalb einer Informationselement-Struktur wird die ARC definiert als:

$$\lambda(t) = \begin{cases} \dfrac{\sum\limits_{(p,cs)\in t} |\{q, ds \in t \mid (p \cup cs) \cap (q \cup ds) \neq \emptyset \wedge (p \neq q)\}|}{|t| \cdot (|t| - 1)}, & \text{for } |t| > 1 \\ 0, & \text{for } |t| \leq 1 \end{cases}$$

Hierbei wird für jede Operation einer Aktivität die Schnittmenge mit jeder anderen Operation gebildet und jede nichtleere Schnittmenge gezählt. In dieser Definition sind die Paare $(p, cs), (q, ds)$ und $(q, ds), (p, cs)$ unterschiedlich, wodurch jede Überschneidung in einer Aktivität doppelt gezählt wird. Für die Normalisierung muss daher die Anzahl der Überschneidungen durch die maximale Anzahl an Überschneidungen geteilt werden. Für jede Operation wären das $|t| - 1$ Überschneidungen, wodurch sich insgesamt $|t| \cdot (|t| - 1)$ ergibt.

Die zweite Komponente der Prozesskohäsion ist die AIC. Für eine Aktivität t innerhalb einer Informationselement-Struktur wird die AIC definiert als:

$$\mu(t) = \begin{cases} \dfrac{|\{d \in D \mid \exists (p, cs), (q, ds) \in t : d \in (p \cup cs) \cap (q \cup ds) \wedge (p \neq q)\}|}{|\{d \in D \mid \exists (p, cs) \in t : d \in (p \cup cs)\}|}, & \text{for } |t| > 0 \\ 0, & \text{for } |t| = 1 \end{cases}$$

Hierbei werden die Informationselemente untersucht, welche entweder als Input oder Output einer Operation verwendet werden und bestimmt, welche Informationselemente in mehr als einer Operation verwendet werden. Diese werden anschließend in Relation zu allen Informationselementen der Aktivität gesetzt.

Für die Berechnung der Kohäsion einer Aktivität werden die ARC und die AIC multipliziert, da eine Aktivität nur als kohäsiv gelten soll, wenn beide Werte hoch sind:

$$c(t) = \lambda(t) \cdot \mu(t)$$

Die Prozesskohäsion ergibt sich schließlich aus der Addition der einzelnen Kohäsionswerte der Aktivitäten und einer anschließenden Division durch die Anzahl der Aktivitäten:

$$c = \frac{\sum_{t \in T} c(t)}{|T|}$$

Die zweite zentrale Metrik, ist die Prozesskopplung. Diese beschreibt, wie sehr die Aktivitäten eines Prozesses miteinander verbunden sind. Eine Verbindung zwischen zwei Aktivitäten liegt genau dann vor, wenn mindestens ein Informationselement existiert, dass in den Operationen der beiden Aktivitäten verwendet wird.

Für die Berechnung werden alle Verbindungen zwischen den Aktivitäten gezählt und normalisiert.

$$k = \begin{cases} \dfrac{\sum_{s,t \in T} \text{connected}(s,t)}{|T| \cdot (|T| - 1)}, & \text{for } |T| > 1 \\ 0, & \text{for } |T| \leq 1 \end{cases}$$

where

$$\mathit{connected}(s,t) = \begin{cases} 1 & \text{if } (s \neq t) \wedge (\exists (p, cs) \in s \wedge (q, ds) \in t : (p \cup cs) \cap (q \cup ds) \neq \emptyset) \\ 0 & \text{otherwise} \end{cases}$$

Um die Prozesskohäsion und die Prozesskopplung in Relation zu setzen und in einen einzigen Wert zu überführen, wird das Process Coupling/Cohesion Ratio (PCCR) eingeführt:

$$\rho = \frac{k}{c}$$

Diese Metrik wird im Verlauf für die Bewertung der Prozessmodelle verwendet. Dabei gilt, je geringer das PCCR ist, desto besser ist das Prozessdesign. [8]

2.2 Der Modellgetriebene Optimierungsansatz

Nach der Vorstellung der Problemdomäne und der Definition der Metriken, soll nun der verwendete Lösungsansatz der modellgetriebenen Optimierung näher erklärt werden. In diesem Kapitel wird dafür zunächst das Konzept der modellgetriebenen Entwicklung erläutert, welches eingesetzt wird, um den vorgestellten Prozess in ein Softwaremodell zu überführen. Für die Generierung geeigneter Lösungsmodelle kommt das Konzept der modellgetriebenen Optimierung zum Einsatz. In diesem Kontext wird außerdem ein Überblick über genetische Algorithmen gegeben.

2.2.1 Modellgetriebene Entwicklung

Modelle beschreiben die Abstraktion eines realen Systems. Sie werden dazu verwendet, einen Ausschnitt der Realität darzustellen, indem unwichtige Teile ausgeblendet und Konzepte vereinfacht werden. Modelle werden im Bereich der Softwareentwicklung in den meisten Fällen zur Dokumentation eingesetzt. Dabei sollen sie ein vorhandenes System beschreiben, oder bei einem neu zu entwickelnden System vorgeben, wie es agieren soll. Über die Abbildung als UML-Klassendiagramm, werden die Zusammenhänge und Funktionalitäten des Systems veranschaulicht. Mittlerweile wurden jedoch neue Ansätze entwickelt, bei denen Modelle nicht nur zu Zwecken der Dokumentation eingesetzt werden, sondern vielmehr als zentrale Artefakte innerhalb des Software-Entwicklungsprozesses angesehen werden. Frameworks unterstützen die Entwickelnden durch Techniken wie etwa Meta-Modellierung, Modelltransformationen, automatische Code-Generierung und der automatischen Ausführung von Softwaresystemen. Diese Techniken können unter dem Begriff der Modellgetriebenen Entwicklung zusammengefasst werden. [4]

Für eine weitere Untersuchung der Techniken ist es notwendig, die Begriffe „Modell“ und „Metamodell“ zu definieren und voneinander abzugrenzen. Während Modelle die Struktur und das Verhalten eines Systems abbilden, stellen Metamodelle eine abstrakte Ebene über den eigentlichen Modellen dar. Ein Metamodell gibt die Struktur vor, welche von den zu erstellenden Modellen eingehalten werden muss. Es definiert die Modellierungssprache, welche dazu verwendet wird, konkrete Modelle zu erstellen.[4] Im Bereich der Prozessmodellierung könnte das Metamodell beispielsweise vorschreiben, dass jedes Prozessmodell eine bestimmte Anzahl an Informationselementen und Aktivitäten enthalten muss. Aus dieser Struktur können dann konkrete Prozessmodelle konstruiert werden. Metamodelle stellen hierbei sicher, dass die erstellten Prozessmodelle den vorgegebenen Regeln entsprechen.

Durch diese Formalisierung können Modelle schließlich als Softwareartefakte eingesetzt werden. So können sie beispielsweise im späteren Verlauf als Input für den Optimierungsprozess eingesetzt werden, und es können Modelltransformationen implementiert werden, welche direkt auf den Klassen des Modells arbeiten. Des Weiteren unterstützt die automatische Generierung des Modell-Codes bei der Implementierung der Metriken.

2.2.2 Optimierung

Im Folgenden Abschnitt wird erläutert, was unter dem Begriff Optimierung im Kontext der Erstellung von Prozessmodellen zu verstehen ist. Zunächst wird vorgestellt, wie ein Optimierungsproblem definiert wird. Ein zentraler Bestandteil des Lösungsansatzes sind genetische Algorithmen, über die im Anschluss ein Überblick gegeben wird. Schließlich werden die Konzep-

te zusammengeführt, um den Ansatz der Modellgetriebenen Optimierung darzulegen.

Das Ziel des Vorgehens ist es, für den gegebenen Prozess Aktivitäten zu definieren, sodass das PCCR des resultierenden Prozessmodells minimal ist. Das Problem liegt darin, aus der sehr großen Menge aller möglichen Prozessmodelle, das Modell mit dem geringsten PCCR zu finden. Zusätzlich ist nicht jedes Modell zulässig, sondern muss die vorgestellten Bedingungen erfüllen. Anhand dieser Kriterien kann das Problem in die Klasse der kombinatorischen Optimierungsprobleme eingeordnet werden. Allgemein kann ein Problem hierbei durch drei Merkmale spezifiziert werden:

- Ein Definitionsbereich des Problems
- Eine Zielfunktion, die jede Lösung auf eine Zahl abbildet
- Instanzen eines Problems, die jeweils eine Menge von Kandidatenlösungen umfassen

[6]

Der Definitionsbereich wird in diesem Fall konkret durch das Metamodell abgebildet. Das Metamodell gibt die Regeln und Bedingungen für die Modellierung der konkreten Prozessmodelle vor und beschreibt somit die möglichen Lösungen. Die Funktion für die Berechnung des PCCR wird für die Bewertung der Lösungsmodelle herangezogen und stellt somit die Zielfunktion dar. Eine Probleminstanz ist in diesem konkreten Fall der vorgestellte Prozess aus dem vorherigen Kapitel. Kandidatenlösungen sind Prozessmodelle aus dem Definitionsbereich für diese Instanz.

Um den Definitionsbereich eines gegebenen Problems nach einer optimalen Lösung abzusuchen, können genetische Algorithmen eingesetzt werden. Genetische Algorithmen sind such-basierte Optimierungsmethoden, welche die natürliche Selektion des Evolutionsprozesses aus der Natur abbilden. [1] Der Algorithmus arbeitet auf einer Menge von Initialen Lösungskandidaten, welche als Population bezeichnet wird. Im Verlauf des Optimierungsprozesses werden iterativ neue Generationen aus der Population mithilfe von Evolution entwickelt. Evolution beschreibt hierbei die Veränderung einzelner Lösungskandidaten durch sogenannte Mutations- und Crossover-Operatoren. Mutationsoperatoren werden isoliert auf einzelne Elemente der Population angewendet und führen zu einer bestimmten Änderung. Crossover-Operatoren verwenden mehrere Lösungskandidaten, hierbei werden beispielsweise Lösungen kombiniert, um eine Neue zu erschaffen. Um die Güte eines Lösungskandidaten zu beschreiben und zu entscheiden, welche Lösung die Beste ist, wird jedem Kandidaten der Population ein quantitativer Wert zugeschrieben. Dieser Wert wird als Fitness bezeichnet. Genetische Algorithmen arbeiten mit Funktionen, welche die Lösungskandidaten auswerten und ihnen, basierend

auf ihren Eigenschaften, eine Fitness zuweisen. Je näher eine Lösung am definierten Ziel liegt, desto höher ist ihre Fitness. Nach Anwendung der Evolutionsoperatoren werden die Lösungen mit der besten Fitness ausgewählt. Die gewählten Lösungen bilden die Population der nächsten Iteration. Der Algorithmus terminiert, wenn eine festgelegte Abbruchbedingung eintritt. Üblicherweise umfassen diese Bedingungen das Erreichen einer bestimmten Fitness für eine Lösung oder eine bestimmte Anzahl an durchgeführten Iterationen. Der Output eines genetischen Algorithmus sind ein oder mehrere Lösungskandidaten mit der höchsten Fitness. [1, 6]

Allgemein sind Optimierungsprobleme nicht nur auf ein Ziel beschränkt, sondern können mehrere Ziele umfassen, die erreicht werden sollen. Der Optimierungsprozess zielt in diesem Fall darauf ab, mehrere Ziele gleichzeitig zu optimieren. Probleme aus der Praxis besitzen häufig Ziele, welche sich gegenseitig ausschließen. Eine gleichzeitige Optimierung aller Ziele ist nicht möglich. Das Ergebnis eines solchen Problems wird daher als eine Menge von Lösungen dargestellt, die sich in ihren Ausprägungen der Ziele unterscheiden. Diese Lösungen können so beschrieben werden, dass kein Ziel einer Lösung weiter verbessert werden kann, ohne dass ein anderes Ziel verschlechtert wird. Die Menge der Lösungen, für die dies zutrifft, wird als paretooptimal bezeichnet. Diese Menge wird durch die Pareto-Front dargestellt, welche eine Hyperebene im Suchraum darstellt. [7] Das Problem der Prozessoptimierung könnte ebenfalls als Pareto-Optimierungsproblem dargestellt werden. Im Gegensatz zum PCCR stellen hierbei die Metriken Kohäsion und Kopplung jeweils eine eigene Zielfunktion dar. Die Kohäsion soll maximiert werden, während gleichzeitig eine geringe Kopplung angestrebt wird.

2.2.3 Modellgetriebene Optimierung

Die Erkenntnis der letzten Kapitel ist, dass das Problem der Erstellung optimaler Prozessmodelle als kombinatorisches Optimierungsproblem definiert werden muss. An diesem Punkt setzt die modellgetriebene Optimierung an. Die Konzepte aus der modellgetriebenen Entwicklung werden verwendet, um das gegebene Suchproblem genau zu spezifizieren. [6] Konkret werden beim Ansatz der modellgetriebenen Optimierung vier Bedingungen an ein Problem gestellt, welche den Bedingungen an ein kombinatorisches Optimierungsproblem ähneln. Wie bei kombinatorischen Optimierungsproblemen, wird bei der modellgetriebenen Optimierung eine genaue Beschreibung des Suchraums benötigt. Der Suchraum stellt den Definitionsbereich dar, der aus allen möglichen Lösungen des Problems besteht. Es wird ein Vorgehen benötigt, welche die einzelnen Lösungen als Modell kodiert, sowie Suchoperatoren, welche zur Erforschung des Suchraums verwendet werden. Analog zu kombinatorischen Optimierungsproblemen, wird eine Fitnessfunktion benötigt, welche die Güte der einzelnen generierten Lösungen mit Hinblick auf die Ziele der Optimierung bewertet.[6]

1. Suchraum
   Um den Suchraum zu definieren, wird zunächst ein Metamodell benötigt. Das Metamodell beschreibt die abstrakte Struktur der Problemdomäne. [6] Im konkreten Fall der Prozessoptimierung könnte das Metamodell eine Menge von Aktivitäten, Informationselementen und Operationen umfassen. Aus einem Metamodell können individuelle Instanzen erstellt werden, welche ein konkretes Problem beschreiben, das gelöst werden soll. Das zu lösende Instanzmodell für die Prozessoptimierung ist hier der vorgestellte Prozess für den Antrag auf Studienförderung.

   Zusätzlich zur Beschreibung des Suchraums ist es außerdem notwendig, die Nebenbedingungen an ein Modell zu definieren, um zu entscheiden, welche Lösungen zulässig sind. [6] Im Fall der Prozessoptimierung muss es beispielsweise möglich sein, die einzelnen Aktivitäten in einer bestimmten Reihenfolge auszuführen, damit die Lösung zulässig ist.

2. Kodierung der Lösungen
   Die Kodierung der Lösungen in Modelle sorgt für einen reduzierten Berechnungsaufwand, da das Modell sowohl für die Transformation durch Suchoperatoren, als auch für die Berechnung der Fitness verwendet werden kann. [6]

3. Suchoperatoren
   Suchoperatoren beschreiben in diesem Fall Mutationsoperatoren, welche bereits im Kontext der genetischen Algorithmen vorgestellt wurden. Mutationen werden an Modellen durchgeführt, indem diese auf eine bestimmte Weise transformiert werden. Es werden je nach Problem Regeln definiert, welche die möglichen Transformationen eines Modells vorgeben. Da in diesem Bereich keine Crossover-Operatoren benutzt werden, wird nur durch Anwendung der Mutationsoperatoren der Suchraum erforscht. [6]

4. Fitnessfunktionen
   Die Qualität der einzelnen Lösungen wird mit Fitnessfunktionen bestimmt. Innerhalb der Funktionen werden Ziele definiert. Lösungsmodelle werden direkt als Input für die Fitnessfunktionen verwendet, und ihre jeweilige Struktur wird ausgewertet. Zusätzlich können Nebenbedingungen definiert werden, die beispielsweise verwendet werden, um zu überprüfen, ob ein generierter Lösungskandidat ein zulässiges Modell darstellt. [6]

2.3 Vorstellung der verwendeten Tools und Frameworks

Nachdem die Konzepte der Modellgetriebenen Entwicklung und -Optimierung erläutert wurden, sollen im folgenden Teil die Frameworks und Tools vor-

gestellt werden, die verwendet wurden, um das Optimierungsproblem zu spezifizieren.

2.3.1 Eclipse Modeling Framework

Für die Spezifikation des Suchraums wird zunächst ein Metamodell benötigt. Auf Grundlage des Metamodells können konkrete Instanzmodelle erstellt werden, um so auch das gegebene Prozessmodell abzubilden. Für die Erstellung dieser Modelle wird das Eclipse Modeling Framework (EMF) verwendet. EMF ist ein Framework welches die Funktionen von Java, XML und UML vereint, um die Konstruktion von Modellen zu ermöglichen, ohne selbst explizit Java-Code zu schreiben. [9]

Das Zentrum der Softwareentwicklung mit EMF ist das sogenannte Ecore-Modell. Dabei handelt es sich um das zu erstellende Metamodell, welches den Ausschnitt aus der Realität abbilden soll, der das Problem beschreibt.

Für die Definition des Metamodells muss der Problembereich untersucht und die einzelnen Bestandteile in Komponenten des Ecore Modells überführt werden. Angelehnt an Klassen, Attribute, Methoden und Referenzen, welche aus UML-Klassendiagrammen bekannt sind, bietet EMF ebenfalls vier Komponenten, welche zur Modellierung des Metamodells verwendet werden:

1. EClass: Wird verwendet, um eine modellierte Klasse darzustellen. Eine Klasse hat einen Namen, eine beliebige Anzahl an Attributen, und eine beliebige Anzahl an Referenzen auf andere Klassen. [9]

2. EAttribute: Wird verwendet, um Attribute der Klassen zu modellieren. Attribute haben einen Namen und einen Typ. [9]

3. EReference: Wird verwendet, um eine Assoziation zwischen Klassen zu modellieren. Eine Referenz hat einen Namen, eine boolean flag um anzudeuten, ob die Referenz ein containment darstellt, und sie bietet die Möglichkeit, die Kardinalität der Referenz zu bestimmen. Außerdem hat jede EReference einen Referenztyp, welche den Datentyp der referenzierenden Klasse angibt. [9]

4. EDatatype: Wird verwendet, um den Typ eines Attributs darzustellen. Ein EDatatype kann beispielsweise ein primitiver Datentyp wie int oder float sein. Komplexe Datentypen wie zum Beispiel selbst erstellte EClass Objekte sind ebenfalls möglich, um als EDatatype verwendet zu werden. [9]

Das Top-Level Element der vier Komponenten bildet das EPackage, welches als Schnittstelle für andere Anwendungen dient, um die einzelnen Bestandteile des Metamodells zu identifizieren und auf diese zuzugreifen. Dies ist notwendig, da auf Grundlage des Metamodells in den nächsten Schritten Suchoperatoren und Zielfunktionen definiert werden sollen.

EMF bietet zwei verschiedene Möglichkeiten Metamodelle zu erstellen. Die erste Möglichkeit ist das „Direct Ecore editing“. Hierbei wird ein Tree-based Editor verwendet, um die einzelnen Klassen des Modells zu erstellen und deren Eigenschaften festzulegen. Eine weitere Möglichkeit ist der Import eines vorher angefertigten UML Modells, hierbei existieren Funktionalitäten diese Modelle zu importieren und in Ecore-Modelle zu überführen. [9]

Nach der Erstellung eines Metamodells können mithilfe eines bereitgestellten Editors Instanzmodelle modelliert werden. Mithilfe des baumbasierten Editors können nun auf Grundlage des Metamodells einzelne Entitäten erstellt und Referenzen zwischen diesen festgelegt werden, um so ein Instanzmodell zu konstruieren. Des Weiteren können die Instanzen manipuliert und automatisch validiert werden, um beispielsweise zu überprüfen, ob eine Änderung des Modells die Konformität zum Metamodell bewahrt. Instanzmodelle können weiterhin als XMI-Dateien serialisiert werden, um die Persistenz erstellter Modelle zu gewährleisten.[9]

Ein weiteres essenzielles Feature des EMF ist die automatische Generierung von Modell-Code. Sobald das Metamodell erstellt wurde, können die einzelnen Entitäten und deren Methoden in Java-Klassen überführt werden. Die erstellten Klassen und Methoden können in den nächsten Schritten der Implementierung verwendet werden.

2.3.2 Henshin

Die nächste benötigte Komponente für die Spezifikation des Problems sind Suchoperatoren. Diese werden dazu verwendet, um die Instanzmodelle während des Optimierungsprozesses zu mutieren und in einen Zielzustand zu transformieren. Für die Erstellung dieser Operatoren wird Henshin verwendet. Henshin ist eine Sprache für Modelltransformationen, welche als Plugin für das Eclipse Modeling Framework geladen werden kann. Es existiert ein Editor, in dem Transformationen visuell implementiert werden können. Außerdem wird eine Schnittstelle geliefert, um die erstellten Transformationen in weiteren Anwendungen nutzen zu können.

Henshin basiert auf der algebraischen Graphtransformation, das bedeutet, dass jede Transformation über einen Graph dargestellt wird. Ein Metamodell, welches zuvor mit EMF erstellt wurde, kann über das EPackage in Henshin geladen werden. Die einzelnen Komponenten des Metamodells werden innerhalb von Henshin als Elemente eines Graphs dargestellt. Dabei korrespondieren die Klassen des Metamodells mit den Knoten eines Graphs. Assoziationen zwischen den einzelnen Klassen des Modells entsprechen den Kanten, mit denen die Beziehungen zwischen den Knoten dargestellt werden. [1]

Für die Erstellung einer konkreten Modelltransformation werden in Henshin sogenannte Regeln verwendet, wobei jede Regel immer genau eine Modelltransformation darstellt. Eine Regel wird durch ein Muster eines Gra-

phen dargestellt, welches im Instanzmodell gefunden und durch einen neuen Teil ersetzt oder abgeändert werden soll. Das Graphenmuster einer Regel wird durch zwei Teile beschrieben. Der erste Teil einer Regel ist die left-hand side (LHS). Diese spezifiziert ein bestimmtes Graph-Muster, welches im gegebenen Input-Modell gefunden werden muss. Der zweite Teil einer Regel ist die right-hand side (RHS). Falls die LHS im Modell gefunden werden kann, dann spezifiziert die RHS eine Änderung, welche im Modell umgesetzt wird. Zur weiteren Spezifikation einer Regel können die Knoten und Kanten der LHS über positive application conditions (PACs) und negative application conditions (NACs) erweitert werden. Das LHS Graphmuster muss dann im Falle einer PAC zusätzliche Eigenschaften erfüllen, welche gematched werden müssen. Im Gegensatz dazu setzt die Erweiterung über eine NAC vor, dass die LHS bestimmte Eigenschaften nicht erfüllen darf.[1]

Die visuelle Darstellung einer gesamten Regel im Henshin-Editor umfasst somit einen Graph mit Knoten und Kanten, welche den Klassen und Assoziationen des Metamodells entsprechen. Die Knoten und Kanten des Graphen können mit bestimmten Tags, darunter auch die PACs und NACs, genauer spezifiziert werden:

1. ≪create≫ Elemente mit diesem Tag sind Teil der RHS einer Regel. Wenn Knoten diese Kennzeichnung besitzen, wird im Modell ein neues Objekt einer Klasse erstellt. Ebenso können Kanten diese Annotation besitzen. In diesem Fall wird im Modell eine neue Assoziation zwischen den zwei verbundenen Objekten erstellt. [1]

2. ≪delete≫ Dieses Tag bildet das Gegenstück zum ≪create≫-Tag. Elemente mit diesem Tag sind Teil der LHS. Bei der Ausführung der Regel wird ein Teil des Modells gelöscht. Entweder betrifft dies ein Objekt einer Klasse oder eine Assoziation zwischen Objekten.[1]

3. ≪preserve≫ Elemente mit dem ≪preserve≫-Tag können sowohl Teil der LHS als auch Teil der RHS sein. Objekte und Assoziationen werden unverändert beibehalten.[1]

4. ≪forbid≫ Ein ≪forbid≫-Tag spezifiziert eine NAC. Wenn die LHS eine ≪forbid≫-Struktur im Modell matchen kann, wird die RHS der Regel nicht ausgeführt.[1]

5. ≪require≫ Ein ≪require≫-Tag ist das Gegenstück zum ≪forbid≫-Tag und spezifiziert eine PAC. Die Regel kann nur ausgeführt werden, wenn die LHS eine ≪require≫-Struktur im Modell matchen kann, ansonsten nicht.[1]

2.3.3 Ziele und Nebenbedingungen

Für die Implementierung der Ziele und Nebenbedingungen der Optimierung wird Java verwendet. Das EMF ermöglicht es, Code aus dem spezifizierten Metamodell zu generieren. Daraufhin können Klassen implementiert werden, welche Instanzmodelle als Input erhalten und anhand deren Struktur eine Fitness berechnen. Aus den Instanzmodellen können die einzelnen modellierten Entitäten entnommen und anhand des definierten Ziels ausgewertet werden. Je näher die Struktur des Instanzmodells dem Ziel der Optimierung entspricht, desto besser ist die Fitness des Modells. Mit dem gleichen Vorgehen können ebenso Nebenbedingungen für das Optimierungsproblem implementiert werden. Die Erfüllung einer Nebenbedingung wird ebenfalls wie ein Ziel behandelt. Wenn der Wert einer Nebenbedingung auf 0 reduziert wurde, gilt sie als erfüllt. Nebenbedingungen müssen daher entsprechend implementiert werden, um diesem Vorgehen gerecht zu werden.

2.3.4 MDEOptimiser

Sobald für das Problem ein Metamodell, Suchoperatoren, Ziele, und Nebenbedingungen entworfen wurden, kann der Optimierungsprozess ausgeführt werden. Im Rahmen dieser Arbeit wird für die Optimierung der Modelle das Tool „MDEOptimiser“ angewendet, welches im Folgenden genauer erklärt werden soll.

MDEOptimiser ist ein Tool, welches die Spezifikation von Modellgetriebenen Optimierungsproblemen vereinfachen soll. Dafür wird eine domain specific language (DSL) bereitgestellt, welche dem Nutzer erlaubt, durch Angabe der benötigten Komponenten, ein Problem zu definieren. Anschließend werden genetische Algorithmen verwendet, um Lösungen für das spezifizierte Problem zu generieren. Für die Spezifiktation werden die Vorteile der Modellgetriebenen Softwareentwicklung genutzt, indem der Optimierungsprozess direkt auf den eingegebenen Modellen stattfindet. Für die Spezifikation des Optimierungsproblems benötigt MDEOptimiser die folgenden Eingaben:

1. Metamodell
   MDEOptimiser benötigt einen Verweis auf das Metamodell. Mithilfe dieses Modells wird der Suchraum abgebildet.

2. Instanzmodell
   Das Instanzmodell ist ein konkrete Modellierung einer Probleminstanz, welche nach den Vorgaben des Metamodells entstanden ist. MDEOptimiser verwendet dieses Modell als Lösungskandidaten der Initialpopulation.

3. Evolver
   Die dritte Komponente besteht aus den Suchoperatoren, welche mit

Henshin für das Metamodell definiert wurden. Wie besprochen, werden in diesem Fall nur Mutationsoperatoren verwendet, um die Lösungen zu optimieren. Suchoperatoren werden im Kontext des MDEOptimiser Evolver genannt.

4. Objectives
   Die Zielfunktionen des Optimierungsproblems werden objectives genannt. Die Lösungskandidaten werden direkt als Input verwendet. Auf Grundlage der Struktur der einzelnen Modelle werden direkte Berechnungen vorgenommen, um die Fitness zu bestimmen.

5. Constraints
   Constraints stellen die Nebenbedingungen eines Optimierungsproblems dar und werden ebenso wie die Objectives mit Java implementiert. Nebenbedingungen können gewünschte Eigenschaften an eine Lösung darstellen, oder vorgeben, welche Bedingungen erfüllt werden müssen, damit eine Lösung zulässig ist. [3]

Für die Eingabe dieser Komponenten wird die DSL des MDEOptimiser verwendet. Die Konfiguration eines Optimierungsproblems wird in einer eigenen .mopt-Datei gespeichert.

Die DSL umfasst bei der Konfiguration die vier Bereiche problem, goal, search und solver. Innerhalb der Bereiche werden die genannten Komponenten über keywords zugewiesen. Nach einem keyword folgt immer der Dateipfad auf das zu spezifizierende Objekt.

Im Bereich problem müssen drei Angaben gemacht werden, um das Metamodell und das Instanzmodell anzugeben:

- basepath: Der basepath bestimmt den relativen Pfad innerhalb des Projekts zu allen Komponenten, die im weiteren Verlauf angegeben werden müssen.
- metamodel: Spezifiziert das Metamodell.
- model: Spezifiziert das Instanzmodell, welches optimiert werden soll.

Im Bereich goal werden die Ziele der Optimierung und die Nebenbedingungen angegeben:

- objective legt das Ziel der Optimierung fest, dafür wird zunächst der Name der Zielgröße angegeben. Das objective wird über eine Java-Klasse implementiert, deren Pfad angegeben werden muss. Hierbei können mehrere objectives festgelegt werden.
- Über das constraint keyword können die Nebenbedingungen spezifiziert werden, dabei können beliebig viele Nebenbedingungen angegeben werden. Nebenbedingungen werden ebenfalls über Java Klassen implementiert und ihr Pfad muss angegeben werden.

Im Bereich „search“ müssen die Suchoperatoren angegeben werden, welche mit Henshin implementiert wurden. Dies geschieht mit der Angabe „mutate using“ und einer darauf folgenden Auflistung aller Suchoperatoren, die im Optimierungsprozess verwendet werden sollen.

Im Bereich „solver“ können weitere Konfigurationen zum Optimierungsprozess durchgeführt werden. Dazu gehören die Auswahl eines genetischen Algorithmus, standardmäßig wird hier der Algorithmus NSGAII verwendet. Die Größe der initialen Population kann verändert werden. Außerdem kann bestimmt werden, wie viele Mutationen pro Iteration auf jedes Inidividuum angewendet werden sollen. Schließlich kann angegeben werden, nach welcher Anzahl an Iterationen der Algorithmus terminieren soll.

Nach der Konfiguration des Problems mithilfe der DSL kann der MDE-Optimiser ausgeführt werden, um den Optimierungsprozess zu starten. Der Algorithmus beginnt, indem in jedem Schritt für jede Kandidatenlösung in der Population eine feste Anzahl an Mutationen angewendet wird. Nach der Mutationsphase wird für jedes Modell anhand der festgelegten objectives eine Fitness berechnet. [3] Der Algorithmus terminiert, wenn eine spezifizierte Anzahl an Mutationen ausgeführt wurde. Das Ergebnis der Optimierung umfasst das Instanzmodell mit der besten Fitness. Zusätzlich wird ein Textdokument erstellt, in dem für das generierte Instanzmodell die Fitness und die Erfüllung der Constraints aufgeführt wird. Des Weiteren wird eine Sequenz der durchgeführten Mutationsoperatoren aufgelistet, durch welche das Instanzmodell im Verlauf des Algorithmus in die abschließende Form transformiert wurde.

3 Design der Softwarelösung

Nachdem in den vorhergehenden Kapiteln die Grundlagen besprochen und die verwendeten Tools und Frameworks vorgestellt wurden, sollen in diesem Kapitel die konzeptionellen Überlegungen und das Design der entwickelten Softwarelösung dargelegt werden. Für die Lösung des Optimierungsproblems wird der bereits vorgestellte modellbasierte Ansatz verwendet. Hierbei muss beachtet werden, dass neben der Konstruktion des Metamodells und der Instanzmodelle, die Suchoperatoren, Ziele, und Nebenbedingungen ebenfalls mit dem Konzept der modellgetriebenen Entwicklung implementiert werden. Sowohl für das Design und die Implementierung der Suchoperatoren, als auch für die Entwicklung der Zielfunktionen und Nebenbedingungen dient das Modell als Grundlage. Das Metamodell bietet Schnittstellen, um die weiteren benötigten Komponenten zu entwickeln. Daher steht das Metamodell im Zentrum des Entwicklungsprozesses und bildet die Grundlage für die weitere Implementierung der Lösung.

Insbesondere sollen im Folgenden die Architektur und die Überlegungen bei der Umsetzung des Metamodells und der hierauf aufbauenden Komponenten vorgestellt werden.

3.1 Design des Metamodells

Der erste Schritt besteht in der Konstruktion eines geeigneten Metamodells, welches dazu verwendet wird, den gegebenen Prozess adäquat zu modellieren. Dafür wird als Orientierung die Informationselement-Struktur verwendet, die zur Modellierung des Beispielprozesses genutzt wurde.

Aus der Informationselement-Struktur können als Entitäten Informationselemente, Operationen und Aktivitäten entnommen werden. Informationselemente und Operationen sind die zentralen Objekte, da sie die Struktur des Prozesses vorgeben.

Informationselemente bilden den erzeugten Output von Operationen ab und dienen wiederum als Input, um mithilfe der Operationen weitere Informationselemente zu erzeugen. Operationen stellen somit die Abhängigkeiten zwischen den Informationselementen dar. Aktivitäten sind in der Informationselement-Struktur bislang nicht enthalten. Ziel ist es, die Informationselemente und Operationen innerhalb der Aktivitäten zu gruppieren.

Die Architektur des Metamodells sollte möglichst einfach gehalten werden, um die Modellierung der Prozesse zu erleichtern und eine Übersichtlichkeit des Modells zu gewährleisten. Gleichzeitig muss dabei bedacht werden, dass das Metamodell für die Implementierung der Suchoperatoren verwendet wird. Hierbei ist der Einsatz einer geringen Anzahl verschiedener Entitäten ebenfalls unterstützend, da durch eine Generalisierung der Objekte auch die Implementierung der Suchoperatoren erleichtert wird.

Für die Architektur des Metamodells sollen daher Informationselemen-

te und Operationen in einer abstrakten Entität „InformationObject“ vereint werden. InformationObjects beinhalten zwei Referenzen „source“ und „sink“, die jeweils auf andere InformationObjects verweisen. Die Struktur des Prozesses kann somit als Graph dargestellt werden, wobei ein InformationObject ein Knoten ist, welcher mit den „source“ und „sink“ Referenzen auf die vorherigen und nachfolgenden Knoten verweisen kann.

Eine konkrete Ausprägung eines InformationObjects ist die Entität „Task“. Ein Task bildet zunächst ein Informationselement ab. Außerdem beinhaltet ein Task die Operation, die das Informationselement als Output erzeugt. Jedes Task Objekt hält dafür eine Referenz auf alle Informationselemente bzw. Tasks, welche als Input benötigt werden. Auf diese Weise können alle Informationselemente und Operationen des Prozesses modelliert werden.

Der Prozess besitzt weiterhin Operationen, welche sich gegenseitig ausschließen. So kann ein Informationselement beispielsweise über zwei verschiedene Operationen erreicht werden. Um diese Situation zu modellieren, kann zunächst jedem Task eine Liste hinzugefügt werden. Jedes Element der Liste stellt eine Menge von Tasks dar, die jeweils einen unterschiedlichen Input der Operation modelliert.

Eine Aktivität beinhaltet nun entgegen der Definition nicht mehr eine Teilmenge der Operationen, sondern eine Menge an InformationObjects. Um eine Referenz auf alle vorhandenen InformationObjects und Aktivitäten zu halten, wird ebenfalls eine Entität erstellt, welche einen Container für die Entitäten des Prozesses darstellt. Diese Klasse soll den Zugriff auf InformationObjects und Aktivitäten ermöglichen.

Die Architektur des Metamodells wird in Abbildung 2 dargestellt.

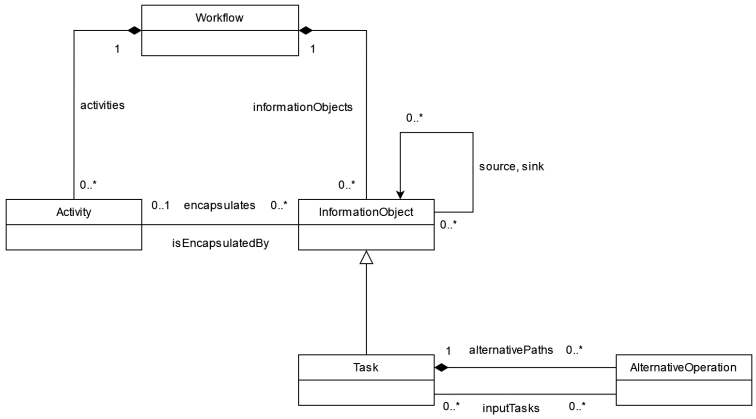

Abbildung 2: Klassendiagramm des entworfenen Metamodells

3.2 Design der Mutationsoperatoren

Die Mutationsoperatoren werden im Verlauf des Optimierungsprozesses eingesetzt, um die vorhandenen InformationObjects in Aktivitäten zu gruppieren. Eine zentrale Anforderung an die Operatoren ist, dass eine effektive Erforschung des Suchraums ermöglicht wird. Außerdem muss sichergestellt werden, dass nach jeder Transformation eines Modells eine weitere Transformation ausgeführt werden kann. Es soll keine Abfolge von Transformationen existieren, die das Modell in einen Zustand überführt, in dem keine Mutation mehr angewendet werden kann. Das dritte Ziel bei der Implementierung der Suchoperatoren ist die Beibehaltung der Validität der Modelle. Modelltransformationen sollen so implementiert werden, dass eine Ausführung zu einem Modell führt, welches weiterhin zulässig ist.

Die Gruppierung der InformationObjects in Aktivitäten wird durch die Suchoperatoren sichergestellt. Hierbei werden Operatoren benötigt, welche neue Aktivitäten erstellen und dem Prozess hinzufügen und Operatoren, welche leere, nicht länger benötigte Aktivitäten löschen.

Um die Zulässigkeit der Prozessstruktur beizubehalten, muss festgelegt werden, auf welche Weise InformationObjects den Aktiviäten zugewiesen werden dürfen. Der Basisfall beschreibt ein InformationObject, welches noch keiner Aktivität zugewiesen wurde. Wird ein solches InformationObject einer Aktivität hinzugefügt, ändert sich die Zulässigkeit des Modells nicht. Um ein InformationObject in eine andere Aktivität zu verschieben, muss sichergestellt werden, dass der Prozess nach der erneuten Zuweisung des Objekts weiterhin zulässig ist. Dies kann realisiert werden, indem nur bestimmte InformationObjects erneut zugewiesen werden dürfen. Hierbei handelt es sich um die Objekte, welche als erstes oder als letztes in einer Aktivität stehen. Diese „Randelemente“ dürfen in die vorherige oder nachfolgende Aktivität verschoben werden. In diesem Fall wird die Zulässigkeit beibehalten.

Durch dieses Vorgehen ist es allerdings möglich, dass alle InformationObjects einer einzigen Aktivität zugeordnet werden. In diesem Fall können keine weiteren Transformationen ausgeführt werden, weil eine Zuweisung der Randelemente zu einer übrigen leeren Aktivität nicht möglich ist. Somit muss ein weiterer Suchoperator implementiert werden, welcher es ermöglicht ein Randelement aus einer Aktivität in eine neue Aktivität auszulagern.

3.3 Design der Ziele und Nebenbedingungen

Als letzter Schritt der Spezifikation des Optimierungsproblems werden Ziele und Nebenbedingungen benötigt. Die Ziele für die Optimierung sind die bereits genannten Metriken zur Kohäsion und Kopplung der Aktivitäten des Prozesses. Insbesondere soll die PCCR eines gesamten Prozesses betrachtet werden. Die Implementierung der Ziele und Nebenbedingungen erfolgt über Java.

Für jedes Ziel und jede Nebenbedingung muss eine eigene Klasse entworfen werden. Die Erfüllung einer Nebenbedingung wird hierbei ähnlich wie eine Fitness, durch einen numerischen Wert angegeben. Die Nebenbedingungen des Optimierungsproblems legen zunächst fest, dass jedes InformationObject einer Aktivität zugewiesen wird. Leere Aktivitäten sollen nicht existieren, da sie keinen Mehrwert für die Lösung bieten.

Ein Nachteil der vorgestellten Modellarchitektur ist, dass InformationObjects existieren, welche keine Operation darstellen. Dies sind die Informationselemente, welche jeweils den initialen Input für einen Prozess liefern. Da mit den Suchoperatoren nur die Gruppierung von InformationObjects im Allgemeinen festgelegt wird, können Aktivitäten entstehen, welche beispielsweise nur initiale Informationselemente beinhalten und somit nicht zulässig sind, weil sie keine Operation besitzen. Daher muss das Auslesen von Operationen innerhalb einer Aktivität zur Laufzeit implementiert werden, was durch die Referenzen der InformationObjects ermöglicht wird. Eine weitere Nebenbedingung zur Sicherstellung der Zulässigkeit ist daher, dass eine Aktivität mindestens eine Operation beinhalten muss.

Für die Zulässigkeit eines Prozesses ist außerdem gefordert, dass eine Reihenfolge von Aktivitäten existiert, um den Prozess auszuführen. Da der Prozess einen Graph darstellt, wird dies durch das Finden einer topologischen Sortierung sichergestellt. Falls eine topologische Sortierung der Aktivitäten nicht möglich ist, ist der Prozess nicht zulässig. Diese Nebenbedingung wird durch eine weitere Klasse implementiert.

Es wird abschließend eine weitere Nebenbedingung implementiert, welche zwar nicht zwingend notwendig ist, aber das Ergebnis einer Lösung beeinflussen kann. Konkret wird eine Nebenbedingung entwickelt, die festlegt, dass Aktivitäten nicht weniger als drei InformationObjects beinhalten sollen. Damit wird sichergestellt, dass Aktivitäten nicht zu klein sind.

4 Implementierung der Softwarelösung

Nach der Vorstellung des Designs und der Architektur der Softwarelösung, soll in diesem Kapitel die konkrete Umsetzung und Implementierung diskutiert werden. In einem ersten Schritt wird dafür das Metamodell mit dem EMF erstellt, welches als Grundlage für die Modellierung der Instanzmodelle dienen wird. Die für den Optimierungsprozess benötigten Suchoperatoren werden mit Henshin implementiert. Durch die Generierung des Model-Code mit EMF können darauffolgend die Zielfunktionen und Nebenbedingungen implementiert werden. Schließlich werden anhand dieser Eingaben Prozessmodelle mit dem MDEOptimiser generiert.

4.1 Implementierung des Metamodells

In diesem Abschnitt soll die konkrete Implementierung des herausgearbeiteten Designs für das Metamodell besprochen werden. Das Projekt kann auf [5] eingesehen werden.

Das Design sieht vor, Informationselemente und Operationen in der abstrakten Klasse „InformationObject“ zu vereinen. Daher wird zunächst eine abstrakte EClass „InformationObject“ erstellt, welche mittels der EReferences „source“ und „sink“ jeweils auf eine weitere Menge von InformationObjects verweist. Bei der konkreten Erstellung der Instanzmodelle können alle eingehenden InformationObjects über „source“ und alle ausgehenden InformationObjects über „sink“ referenziert werden.

Die konkrete Modellierung dieser Beziehungen erfolgt über die EClass „Task“, welche von InformationObject erbt und in diesem Kontext nun Informationselemente und Operationen vereint. Operationen können zur Laufzeit aus den Tasks entnommen werden, indem das Objekt selbst und dessen über „source“ referenzierten Elemente in einer Menge gruppiert werden.

Die EClass „Activity“ stellt die Aktivitäten dar, welche eine zugewiesene Menge an InformationObjects über die EReference „encapsulates“ referenziert. Ein InformationObject besitzt ebenfalls eine Referenz „encapsulatedBy“, welche auf die Aktivität verweist, in welcher dieses Objekt liegt. So kann während des Optimierungsprozesses sichergestellt werden, ob ein InformationObject einer Aktivität zugewiesen wurde, und es werden doppelte Zuweisungen durch Festlegung der Kardinalität verhindert.

Für die Modellierung der alternativen Pfade der Operationen wird die EClass „Alternative Operation“ erstellt. Jedes Task Objekt referenziert eine Liste von Alternative Operations, wobei jede Alternative Operation eine Liste von Tasks enthält. Diese stellen die möglichen verschiedenen Inputmengen dar.

Außerdem wird die EClass „Workflow“ modelliert. Diese dient als Container-Klasse, welche alle InformationObjects und Activities referenziert. Diese Klasse wird relevant bei der Erstellung von Instanzmodellen und der Be-

rechnung der Fitness einzelner Lösungen, da über diese Klasse die Struktur des Instanzmodells ausgewertet werden kann.

Aus dem erstellten Metamodell können nun Instanzmodelle erstellt werden, mit denen Prozesse modelliert werden können. EMF erzeugt hierbei eine neue XMI-Datei, welche zunächst die leere Container-Klasse „Workflow“ beinhaltet. Über einen Tree-based Editor können die spezifische Klassen, welche bereits im Metamodell deklariert wurden, hinzugefügt werden. Zunächst wird hier ein Instanzmodell konstruiert welches den Prozess für einen Antrag auf Studienförderung darstellt, welcher in Kapitel 2 beschrieben wurde. Die Workflow Klasse beinhaltet nach der Modellierung des Prozesses 27 Task-Objekte, welche über ihre „source“ und „sink“ Referenzen auf die anderen Tasks verweisen.

4.2 Implementierung der Mutationsoperatoren

Für die Mutation der Prozessmodelle werden fünf Mutationsoperatoren vorgestellt, welche mit dem visuellen Editor von Henshin entworfen wurden.

Die erste Regel ist „createNewActivity“. Diese Regel wird benötigt, um neue Aktivitäten in den Prozess einzufügen. Zu Beginn der Optimierung enthält ein Prozessmodell noch keine Aktivitäten, weshalb die Erstellung neuer Aktivitäten essenziell ist. Die LHS der Regel besteht hierbei nur aus einem Workflow-Objekt. Sobald dieses im Modell gefunden wurde, wird das Modell transformiert. Dabei wird ein leeres Activity-Objekt in das Modell eingefügt und eine Referenz der Workflow-Klasse auf die Aktivität erzeugt.

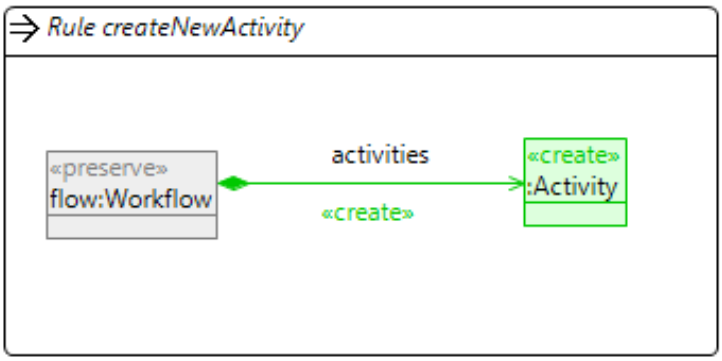

Abbildung 3: Mutationsoperator für die Erstellung einer neuen Aktivität

Die Regel „assignInformationObject“ wird benötigt, um ein InformationObject, welches noch keiner Aktivität zugeordnet ist, einer leeren Aktivität hinzuzufügen. Die Regel zeigt, dass es nicht erlaubt ist, ein Informationselement zu einer Aktivität hinzuzufügen, wenn diese bereits InformationObjects beinhaltet. Sichergestellt wird dies durch die beiden nummerierten

forbid-NAC. Die erste forbid-NAC verbietet die doppelte Zuweisung eines InformationObjects an die gleiche Aktivität. Die zweite NAC stellt sicher, dass nur InformationObjects zugewiesen werden dürfen, die nicht bereits Teil einer anderen Aktivität sind.

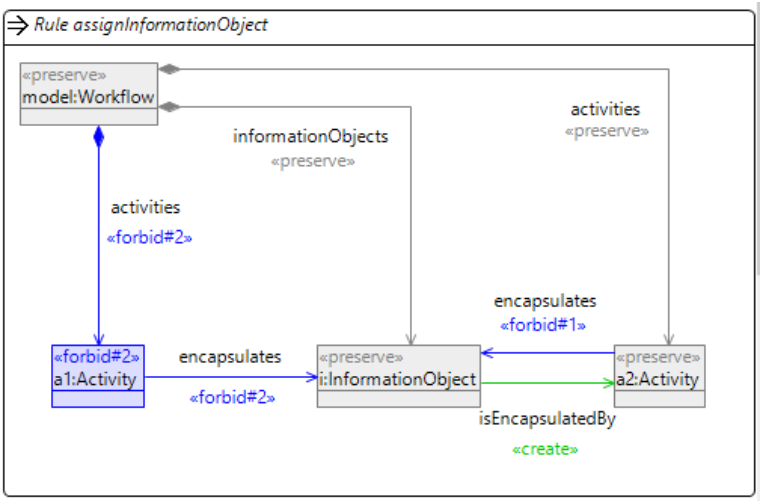

Abbildung 4: Mutationsoperator für die Zuweisung eines InformationObjects zu einer Aktivität

Die Regel „deleteEmptyActivity“ wird verwendet, um leere Aktivitäten aus dem Prozess zu entfernen. Falls durch Transformationen leere Aktivitäten im Prozess entstehen sollten, können diese gelöscht werden. Hierbei wird durch die forbid-NAC sichergestellt, dass eine Aktivität nur gelöscht werden kann, wenn sie kein InformationObject referenziert.

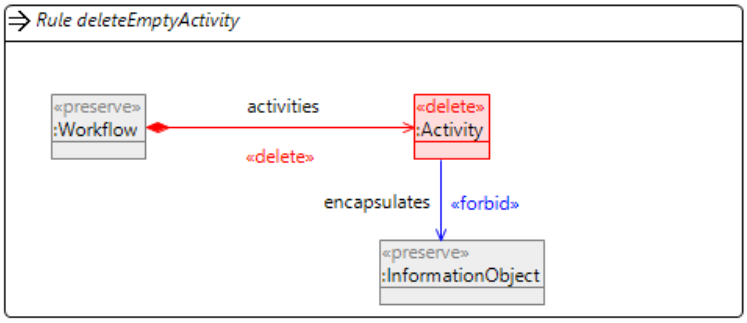

Abbildung 5: Mutationsoperator für das Löschen einer leeren Aktivität

Die drei bisher vorgestellten Regeln stellen sicher, dass einem Modell Aktivitäten zugewiesen werden, und jede erstellte Aktivität höchstens ein InformationObject enthalten darf. Mit der „deleteEmptyActivity“ Regel, wird dafür gesorgt, dass der Prozess möglichst wenige leere Aktivitäten enthält.

Die zwei Regeln für das Verschieben der InformationObjects zwischen den Aktivitäten sind „MoveObjectToNextActivity“ und „MoveObjectToPreviousActivity“. Momentan besitzen die Aktivitäten des Modells höchstens ein InformationObject. Die Regeln ermöglichen es nun, dass InformationObjects zwischen den Aktivitäten verschoben werden können. Hierbei wird sichergestellt, dass nur die InformationObjects am Rand einer Aktivität verschoben werden. Angenommen ein Prozess hat zwei Aktivitäten A1 und A2 mit jeweils einem InformationObject I1 und I2. Wenn I1 im Prozess vor I2 ausgeführt werden muss, dann kann mit „MoveObjectToNextActivity“ I1 in die Aktivität A2 verschoben werden. Ebenso würde „MoveObjectToPreviousActivity“ das Element I2 in die Aktivität A1 verschieben. Voraussetzung ist die Referenzierung über die „source“ und „sink“ EReferences der InformationObjects.

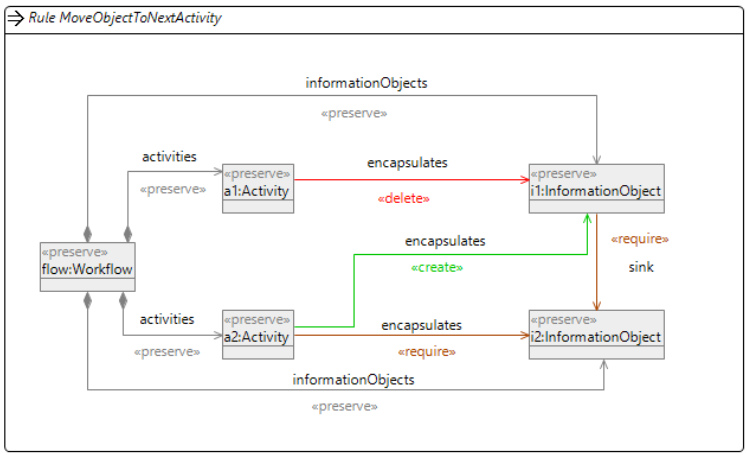

Abbildung 6: Mutationsoperator für das Verschieben in die nächste Aktivität

Das Modell kann nun so transformiert werden, dass Aktivitäten durch Verschieben der Randelemente vergrößert oder verkleinert werden. Für den Optimierungsprozess wären die vorhandenen Regeln ausreichend, jedoch wurde in der Design-Phase die Notwendigkeit einer weiteren Regel für das Verhindern von Situationen, in denen keine weiteren Transformationen mehr ausgeführt werden können, erläutert. Diese Transformation wird durch „extractInformationObject“ realisiert. Diese Regel ermöglicht, dass ein InformationObject, welches am Rand einer Aktivität steht, in eine neue Aktvität ausgelagert werden kann. Diese Regel könnte zusätzlich zu einer höheren Dynamik in der Modelltransformation führen, da potentiell mehr Aktivitäten erzeugt werden, welche wiederum durch die beiden „Move“ Regeln transformiert werden können.

Die Optimierungsstrategie, welche mit den Regeln verfolgt wird, kann in zwei Phasen eingeteilt werden. In einem ersten Schritt sollen leere Ak-

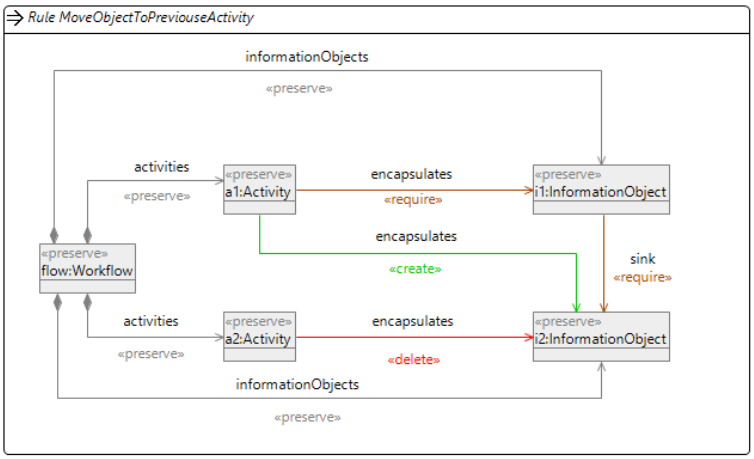

Abbildung 7: Mutationsoperator für das Verschieben in die vorherige Aktivität

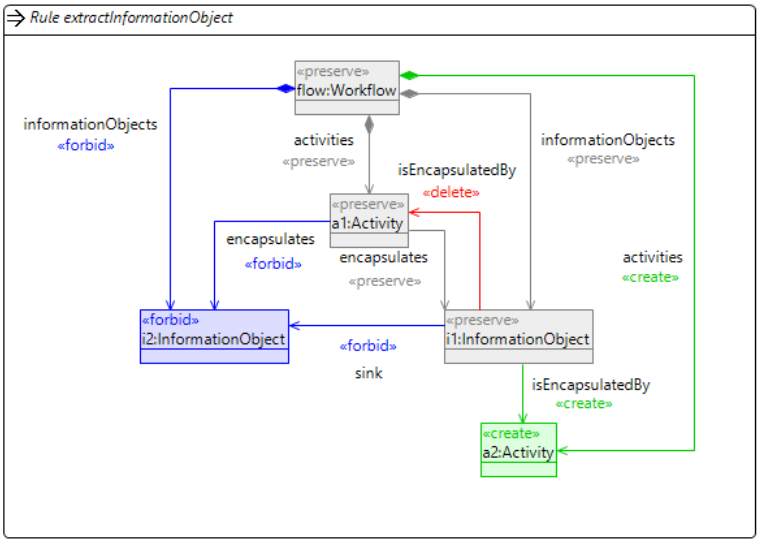

Abbildung 8: Mutationsoperator für das Extrahieren eines InformationObjects in eine neue Aktivität

tivitäten erzeugt werden, sodass für jedes InformationObject im Prozess eine Aktivität vorhanden ist. Mittels der Regel „assignInformationObject“ können dann die InformationObjects jeweils einer Aktivität zugeordnet werden. Im zweiten Teil werden die Randelemente mithilfe der „Move“ Regeln zwischen den Aktivitäten verschoben, oder durch „extractInformationObject“ wird ein InformationObject aus einer Aktivität entfernt und einer neuen Aktivität zugewiesen. Dabei muss angemerkt werden, dass die

zwei Phasen nicht sequentiell stattfinden, sondern nur die Überlegungen des Konzepts verdeutlichen sollen. Wenn beispielsweise zwei Aktivitäten erzeugt wurden und diese InformationObjects beinhalten, welche einander referenzieren, kann bereits eine Verschiebung von InformationObjects stattfinden.

4.3 Generierung des Modelcodes zur Implementierung der Zielfunktionen und Nebenbedingungen

Abschließend werden Zielfunktionen und Nebenbedingungen benötigt, um den Optimierungsprozess zu spezifizieren. Diese werden als konkrete Java-Klassen implementiert, um im MDEOptimiser verwendet werden zu können. Um auf die Objekte des Instanzmodells zugreifen zu können, muss mithilfe des EMF der Modelcode des Metamodells generiert werden. Dabei werden die modellierten Klassen des Metamodells in konkrete Java-Klassen überführt und in einem Package im Projekt abgelegt. Die generierten Klassen besitzen Methoden, um auf die referenzierten Objekte zuzugreifen. So besitzt beispielsweise die Klasse Workflow eine Methode activities(), welche den Zugriff auf die enthaltenen Aktivitäten des Prozesses ermöglicht.

Die Zielfunktionen werden anhand der vorgestellten Metriken implementiert. So werden die Klassen „ProcessCohesion“ und „ProcessCoupling“ erstellt, welche die Kohäsion und die Kopplung des Prozesses berechnen. Für die Berechnung der ProcessCohesion wird die Berechnung der ARC und AIC in eigene Klassen ausgelagert.

MDEOptimiser erlaubt nur die Minimierung von Zielfunktionen. Um die Maximierung der ProcessCohesion im Rahmen einer Pareto-Optimierung zu ermöglichen, muss daher ein alternativer Ansatz verfolgt werden. Da der Wert der ProcessCohesion normiert ist, kann das Minimierungsproblem durch eine Berechnung von $1 - ProcessCohesion$ in ein Maximierungsproblem transformiert werden. Eine Minimierung dieses Terms führt zu einem maximalen Zielwert. Diese Berechnung wird in einer separaten Klasse durchgeführt, welche später bei der Spezifikation des Problems angegeben werden kann. Die Klasse „FitnessUtil“ enthält eine Reihe von Hilfsmethoden, die bei der Berechnung der Fitness relevant sind. Hierunter fällt zum Beispiel das Extrahieren aller Operationen einer Aktivität. Operationen werden ebenfalls durch eine eigene Klasse dargestellt. Sie besitzen ein Task-Objekt als Output und eine Menge von Tasks als Input. Dies wird durch Felder in der Klasse dargestellt.

Für die Implementierung der Ziele und Nebenbedingungen stellt der Code des MDEOptimiser eine Schnittstelle bereit. Jede Klasse, die als Ziel oder Nebenbedingung behandelt werden soll, muss das bereitgestellte Interface „IGuidanceFunction“ implementieren. Nur solche Klassen sind bei der Spezifikation des Problems über die DSL zulässig. Das Interface besitzt als einzige Methode „computeFitness“. Diese Methode wird verwendet, um während des Optimierungsprozesses die Fitness einer Lösung auszuwerten,

oder die Erfüllung einer Nebenbedingung zu bestimmen. Die „computeFitness“ Methode erhält als Parameter ein Objekt des Typs „Solution“. Dabei handelt es sich um eine Klasse, die ebenfalls über den Code des MDEOptimiser bereitgestellt wird. Objekte dieses Typs stellen einen Lösungskandidaten dar und bieten einen Getter, über den auf die Container-Klasse des Instanzmodells zugegriffen werden kann. Konkret wird in diesem Fall ein Workflow-Objekt ausgegeben, welches wiederum einen Abruf der enthaltenen Aktivitäten ermöglicht, um so das Modell auszuwerten. Somit ist lediglich die Implementierung des Interface notwendig, um Ziele und Nebenbedingungen für den MDEOptimiser kompatibel zu machen.

Anhand der gegebenen Aktivitäten in dem Instanzmodell kann die Kohäsion und die Kopplung berechnet werden, indem über die Aktivitäten iteriert und die Erfüllung der Metriken berechnet wird. Die Methode „computeFitness“ gibt den errechneten Fitnesswert zurück und MDEOptimiser nimmt anhand dieser Werte ein Ranking der Lösungen vor.

Die in der Design-Phase vorgestellten Nebenbedingungen werden hier auch wieder aufgegriffen und implementiert. Für die Sicherstellung, dass jedes Task Objekt einer Aktivität zugewiesen wird, wird die Nebenbedingung „NoTaskWithoutActivity“ erstellt. Es sollte keine leeren Aktivitäten geben, da diese trivial für die Ausführung des Prozesses sind. Die Nebenbedingung dazu ist „NoEmptyActivities“. Die Nebenbedinung „NoSmallActivities“ stellt sicher, dass Aktivitäten mindestens drei Tasks enthalten. Diese Nebenbedingung muss für einen zulässigen Prozess nicht erfüllt sein, dennoch sorgt sie für eine angemessene Größe der Aktivitäten. Die Implementierung dieser Nebenbedingung erfolgt über ein einfaches Filtern und Zählen der referenzierten InformationObjects jeder Aktvität.

Eine weitere Nebenbedingung „ValidActivityOrder“ legt fest, ob ein generiertes Modell zulässig ist. Hierbei wird überprüft, ob die erstellten Aktivitäten in einer Reihenfolge angeordnet werden können, um den Prozess auszuführen. Algorithmisch lässt sich dies überprüfen, indem eine topologische Sortierung der Aktivitäten stattfindet. Ein zulässiges Prozessmodell kann als gerichteter azyklischer Graph angesehen werden. Die einzelnen InformationObjects korrespondieren mit den Knoten, während die „source“ und „sink“ Referenzen die Kanten darstellen. Diese Zusammenhänge können auf die Aktivitäten des Prozesses erweitert werden. Wenn ein InformationObject aus einer Aktivität von dem InformationObject einer anderen Aktivität abhängt, dann kann diese Abhängigkeit als gerichtete Kante angesehen werden. Auf diese Weise wird eine Adjazenzliste für alle Aktivitäten erstellt und schließlich kann eine topologische Sortierung berechnet werden. Hierfür wird Kahns Algorithmus verwendet. Falls der Graph einen Zyklus enthält, ist das Modell nicht zulässig, da in diesem Fall zwei Aktivitäten jeweils auf die Ausführung der anderen Aktivität warten müssen. In diesem Fall ist eine topologische Sortierung nicht möglich.

4.4 Tests

Für das Testen der Software wird das JUnit-Framework und Unit-Tests verwendet. Ziel der Tests ist es, die Korrektheit der Zielfunktionen und Nebenbedingungen zu überprüfen und Fehler bei der Implementierung aufzudecken. Zunächst sollen die beiden implementierten Metriken getestet werden. Für die Berechnung der Prozesskohäsion werden die AIC und die ARC benötigt, daher wird die Korrektheit dieser beiden Bestandteile zuerst geprüft. Dafür werden drei Klassen von Aktivitäten getestet. Die erste Klasse beschreibt eine Aktiviät, die eine Anzahl von Task-Objekten enthält, welche Beziehungen zueinander aufweisen. Als zweite Klasse werden Task Objekte mit alternativen Operationen in die Aktivität eingefügt. Die letzte Klasse bildet einen Randfall ab, in der eine Aktivität leer ist. Für die Berechung der AIC und ARC werden diese drei Klassen von Aktivitäten jeweils überprüft. Schließlich werden für die Prüfung der Prozesskohäsion zwei Prozessmodelle erstellt, um hier die Berechnung der gesamten Zielfunktion zu testen.

Für die Berechnung der Kopplung wird ebenfalls das Verhalten von Aktivitäten mit einfachen Task-Objekten und leeren Aktivitäten geprüft. Auch die Berechnung von Aktivitäten, welche Task-Objekte mit alternativen Operationen beinhalten, wird getestet.

Die beiden Metriken werden gemeinsam bei der Berechnung des PCCR verwendet, daher wird auch für diese Berechnung ein Test auf zwei Prozessmodellen durchgeführt.

Da während der Berechnung der Metriken auf die FitnessUtil Klasse zugegriffen wird, müssen die Methoden dieser Klasse ebenfalls getestet werden. Hierbei werden die Methoden jeweils auf einer Prozessinstanz geprüft, da die Methoden keine komplexen Operationen darstellen, sondern lediglich Objekte aus dem Modell ausgelesen werden.

Die angewendete Methode der Äquivalenzklassentests scheint hier angemessen. Ein ähnliches Verhalten zwischen zwei Aktivitäten kann erwartet werden, wenn diese beispielsweise nur Task-Objekte ohne alternative Operationen beinhalten. Auf diese Weise können mit einer geringen Anzahl an Tests die wichtigsten Funktionalitäten geprüft werden. Zusätzlich handelt es sich bei dem vorliegenden Prozess um ein kleines Modell, welches im Optimierungsprozess 5-7 Aktivitäten beinhalten wird. Bei einer weiteren Fortführung des Projekts mit zunehmender Skalierung und steigender Komplexität der Prozessstruktur würde auch die Anforderung an eine ausgereifte Teststrategie steigen. Hierbei könnte beispielsweise eine komplette branch coverage angestrebt werden, um möglicherweise bisher unentdeckte Fehler in der Implementierung zu finden.

Neben der JUnit-Testsuite zur Überprüfung der implementierten Klassen und Methoden, können auch das erstellte Prozessmodell und die Mutationsoperatoren getestet werden. Das EMF bietet die Funktion einer Modellvalidierung, um erstellte Prozessmodelle auf eine Konformität gegenüber

dem erstellten Metamodell zu überprüfen. Hierbei kann festgestellt werden, ob ein erstelltes Instanzmodell den Vorgaben des Metamodells entspricht. Gleichzeitig kann getestet werden, ob durch die Anwendung der Mutationsoperatoren das Modell in einen unzulässigen Zustand transformiert wird.

4.5 Installationsanleitung

Für die Ausführung des vorgestellten Projekts müssen die folgenden Voraussetzungen erfüllt sein:

- Ein Linux-Betriebssystem, im Rahmen dieser Arbeit wird die Standardversion des Windows-Subsystem for Linux (WSL) verwendet.
- Java 11
- Eclipse 2019-03: `https://www.eclipse.org/downloads/packages/release/photon/r/eclipse-ide-java-and-dsl-developers`
- Henshin: `http://download.eclipse.org/modeling/emft/henshin/updates/nightly`
- MDEOptimiser: `http://mde-optimiser.github.io/mdeo_repo/src/composite/release/`
- Der „bpmnmodelling“ Projektordner aus dem Repository
- Die im Repository beigelegte .jar-Datei, um das Projekt als plugin zu laden.

Es wird empfohlen das Projekt in einer Linux Umgebung auszuführen, da unter Windows Fehler bei der Ausführung des MDEOptimiser auftreten können. Des Weiteren wird auf der offiziellen Seite des MDEOptimiser empfohlen Java 11 und Eclipse Photon in der Version 2019-03 zu verwenden, da der letzte Release der Software in dieser Umgebung getestet wurde [2]. Nach der Installation der Eclipse IDE können Henshin und der MDEOptimiser installiert werden:

Installation des MDEOptimiser

- Öffnen Sie Eclipse.
- Öffnen Sie den Wizard zum installieren neuer Software unter dem Reiter Help > Install New Software.
- Kopieren Sie die angegebene URL für den MDEOptimiser.
- Fügen Sie die kopierte URL in das „Work with“ Textfeld ein.
- Nach der Eingabe sollte eine Liste verschiedener Softwares angezeigt werden.
- Setzen Sie einen Haken in das Feld des „MDEOptimiser“ Bundle.
- Wählen Sie „Next“ und folgen Sie den weiteren Installationsschritten.
- Nach der Installation muss Eclipse neu gestartet werden. [2]

Installation von Henshin

- Die Installation erfolgt wie die Installation für den MDEOptimiser.

- Öffnen Sie erneut den Wizard über Help > Install New Software.
- Kopieren Sie die angegebene URL für Henshin.
- Fügen Sie die URL in das „Work with“ Textfeld ein.
- Setzen Sie einen Haken bei dem angezeigten „Modeling“ Bundle.
- Klicken Sie auf „Next“ und folgen Sie den weiteren Installationsschritten.
- Nach der Installation muss Eclipse neu gestartet werden. [2]

Das Projekt als Plugin laden

- Um den MDEOptimiser ausführen zu können, muss das Projekt als Plugin-Projekt geladen werden.
- Zunächst muss die bereitgestellte .jar-Datei in den „dropin“ Ordner der Eclipse Installation abgelegt werden. Der dropin Ordner befindet sich im Installationsverzeichnis.
- Nach diesem Schritt muss Eclipse neu gestartet werden.
- Mit einem Rechtsklick auf den Projektordner öffnet sich ein Reiter, aus dem „Load project as plugin“ ausgewählt werden kann.
- Falls die .henshin Datei einen Fehler wirft muss unter dem Reiter Project > Clean ausgewählt werden.
- Abschließend sollten keine Fehler mehr auftreten.
- Sollten die henshin oder henshin-diagram Dateien Fehler werfen, sollten diese einmal geschlossen und nach einem erneuten Ausführen von Project > Clean erneut geöffnet werden.

Erstellen einer Run Configuration

- Um den MDEOptimiser auszuführen, muss eine Run Configuration erstellt werden.
- Wählen Sie unter dem Reiter Run > Run Configuration aus.
- Wählen Sie in der linken Spalte „MDEOptimiser Search“ aus.
- Mit einem Rechtsklick kann unter „New Configuration“ eine neue Konfiguration erstellt werden.
- Im obersten Feld kann eine Name angegeben werden.
- Im „source“ Textfeld muss die gewünschte .mopt Konfigurationsdatei ausgewählt werden.
- Im Reiter „Classpath“ muss auf „Bootstrap Entries“ und schließlich „Apply“ geklickt werden, um sicherzustellen, dass alle Dependencies im Classpath vorhanden sind.
- Der MDEOptimiser kann schließlich mit einem Klick auf „Run“ gestartet werden.
- Nach der Ausführung des Optimierungsprozesses muss das Projekt mit einem Rechtsklick auf den Projektordner > Refresh aktualisiert werden.
- Dann wird ein Ordner „mdeo-results“ angezeigt, welcher die Ergebnisse beinhaltet.

5 Interne Struktur

Nach der Implementierung der benötigten Komponenten für den Optimierungsprozess, sollen in diesem Kapitel die Ergebnisse der Optimierung besprochen werden. Zunächst sollen die Forschungsfragen vorgestellt werden. Für die Generierung der Prozessmodelle wird anschließend der Aufbau der Experimente dargestellt. Nach der Ausführung der Experimente werden die Ergebnisse erläutert und interpretiert, wobei die Forschungsfragen erneut aufgegriffen werden. Abschließend folgt eine Diskussion über die Einschränkungen des Experimente, welche die Validität der Ergebnisse betreffen.

5.1 Forschungsfragen

Um eine Bewertung der generierten Prozessmodelle durchzuführen, sollen die folgenden Forschungsfragen beantwortet werden:

RQ1: Unterscheiden sich die Lösungsmodelle bei der Optimierung unter einem Ziel von den Ergebnissen einer Pareto-Optimierung? Hierbei soll überprüft werden, ob die Definition über ein Ziel oder zwei Ziele einen Unterschied auf das Ergebnismodell hat.

RQ2: Wie unterscheidet sich ein manuell erstelltes Prozessmodell von generierten Prozessmodellen? Die zweite Frage bezieht sich auf einen Vergleich zwischen einem manuell erstellten Prozessmodell, welches in [8] vorgestellt wird und den generierten Prozessmodellen, die als Ergebnis des Optimierungsprozesses erhalten wurden. Hierbei sollen auch mögliche Unterschiede in der Struktur herausgearbeitet werden.

5.2 Design der Experimente

Für die Erstellung der Prozessmodelle wurden zwei Experimente durchgeführt. Im Rahmen des ersten Experiments wurde als Ziel festgelegt, dass das PCCR als einziges Ziel optimiert werden soll. In einem zweiten Experiment wurde eine Pareto-Optimierung durchgeführt, bei der die Bestandteile des PCCR, die Kopplung und die Kohäsion, jeweils als einzelnes Ziel optimiert werden sollten.

Die Konfiguration für das erste Experiment spezifizierte als Zielobjekt die Minimierung des PCCR. Zu den Constraints des Problems gehörte die Zulässigkeit der Prozessmodelle, diese gibt vor, dass eine Reihenfolge existieren muss, in der die Aktivitäten ausgeführt werden können. Außerdem mussten alle Tasks einer Aktivität zugeordnet sein und es durfte keine leeren Aktivitäten geben. Um einen möglichst realitätsnahen Prozess zu erhalten wurde zusätzlich gefordert, dass Aktivitäten nicht weniger als drei Tasks besitzen sollten. Es wurden 40 Kandidaten in jeweils 500 Iterationen mutiert, dabei wurden alle vorgestellten Suchoperatoren verwendet. Dieses Experi-

ment wurde 20 mal wiederholt und das Modell mit dem geringsten PCCR zur Auswertung herangezogen.

In einem zweiten Experiment sollte eine Pareto-Optimierung durchgeführt werden, dafür wurde festgelegt, dass das Problem mit Hinblick auf zwei Ziele optimiert werden sollte. Die Prozesskohäsion sollte maximiert werden, während die Kopplung der Aktivitäten minimiert werden sollte. Rechnerisch wäre dies gleichzusetzen mit einem geringen PCCR, der Unterschied hierbei ist jedoch, dass durch die Definition von zwei Zielen eine Pareto-Front entsteht. Alle Lösungen die sich auf der Pareto-Front befinden werden ausgegeben. In dieser Durchführung handelt es sich um zwei Modelle, wobei ein Modell den Fokus auf eine geringere Kopplung und das andere Modell einen Fokus auf höhere Kohäsion legt.

5.3 Vorstellung der Ergebnisse

Ergebnis Experiment 1: Das Modell mit dem geringsten PCCR aus dem ersten Experiment wird in Abbildung 9 dargestellt. Die Informationselemente und Operationen sind in fünf Aktivitäten gruppiert. Das PCCR dieses Prozesses beträgt ungefähr 1,8. Die Kohäsion beträgt etwa 0,2723 und die Kopplung 0,5. Die Berechnung für die Kohäsion der einzelnen Aktivitäten kann aus der folgenden Tabelle 1 entnommen werden:

Activity	ARC	AIC	AC	Coupling
A	1	$\frac{1}{4}$	$\frac{1}{4}$	2
B	1	$\frac{1}{2}$	$\frac{1}{2}$	2
C	0	0	0	1
D	1	$\frac{2}{5}$	$\frac{2}{5}$	2
E	$\frac{31}{110}$	$\frac{3}{4}$	$\frac{93}{440}$	3

Tabelle 1: Berechnung der Activity Relation Cohesion, Activity Information Cohesion, Activity Cohesion und Angabe des Coupling jeder Aktivität

Ergebnis Experiment 2: In Experiment 2 wurden zwei Modelle generiert, welche auf der Pareto-Front liegen. Das erste Prozessmodell liefert keine neuen Erkenntnisse, da es identisch zum generierten Prozessmodell aus Experiment 1 ist. Es besitzt eine Kopplung von 0,5 und eine Kohäsion von 0,2723.

Das zweite Prozessmodell besitzt eine höhere Kopplung von 0,66 und ebenfalls eine höhere Kohäsion von 0,3121. Der Prozess wurde in vier Aktivitäten aufgeteilt und kann in Abbildung 10 eingesehen werden.

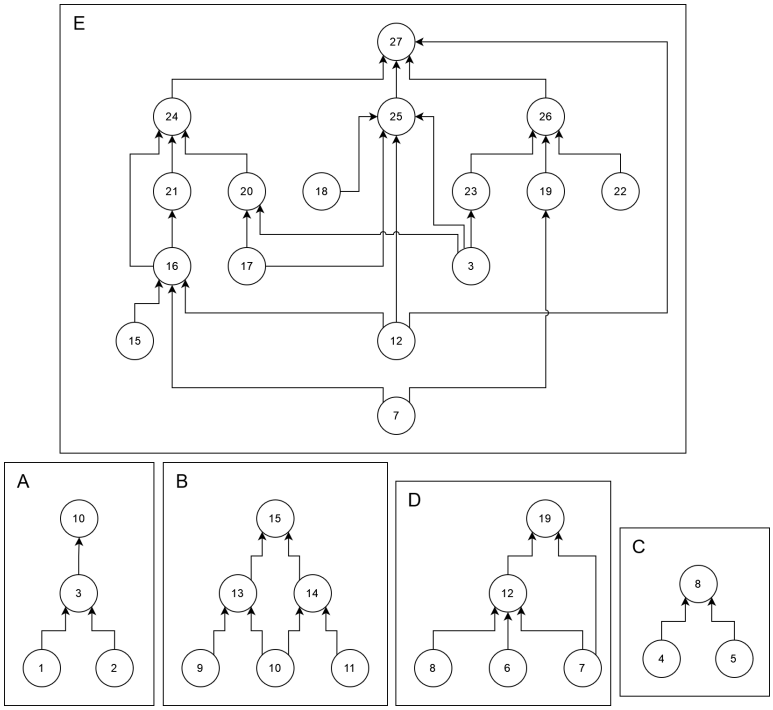

Abbildung 9: Prozessmodell aus Experiment 1

5.4 Interpretation der Ergebnisse und Beantwortung der Forschungsfragen

Zunächst soll ein Vergleich des Modells aus Experiment 1 und den erstellten Modellen aus Experiment 2 durchgeführt werden. Das Modell aus Experiment 1 ist identisch zum ersten generierten Modell aus Experiment 2. Hierbei besteht kein Unterschied im Aufbau der Aktivitäten.

Dennoch bietet die Formulierung des Problems über zwei Zielfunktionen einen Mehrwert, da alle Lösungen entlang der Pareto-Front ausgegeben werden. Die beiden Lösungen stellen den Trade-Off zwischen den gestellten Zielen dar. Das zweite generierte Modell besitzt eine höhere Kohäsion, allerdings gleichzeitig auch eine höhere Kopplung im Vergleich zum ersten Modell. Obwohl das PCCR dieses Modells insgesamt höher ist, könnte für bestimmte Anwendungsfälle und Präferenzen die Formulierung über zwei Ziele sinnvoll sein, beispielsweise wenn eine hohe Kohäsion angestrebt wird. Als Kompromiss wird dabei eine höhere Kopplung eingegangen.

Nach einem Vergleich zwischen den generierten Prozessen, soll nun ein Vergleich dieser mit einem manuell erstellten Prozess stattfinden. Dafür wird ein Prozess aus [8] herangezogen. Dieser Prozess wurde von Hand erstellt und besteht aus sieben Aktivitäten, welche in Abbildung 11 gezeigt werden.

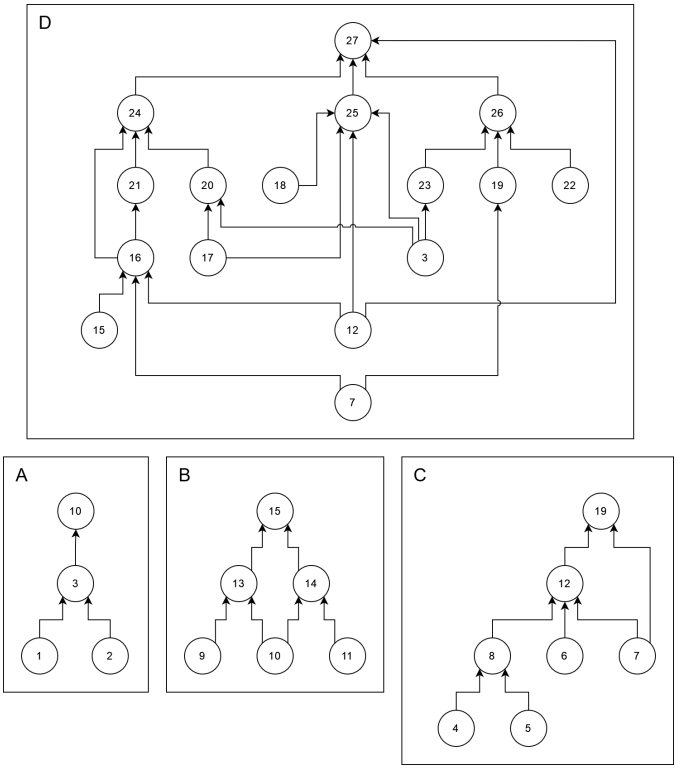

Abbildung 10: Prozessmodell aus Experiment 2

Der Prozess besitzt insgesamt eine Kohäsion von 0,183 und eine Kopplung von 0,714. Somit beträgt das PCCR 3,9. Die Überlegung beim Design dieses Modells war, dass Aktivitäten weder zu groß, noch zu klein sein sollten. [8]

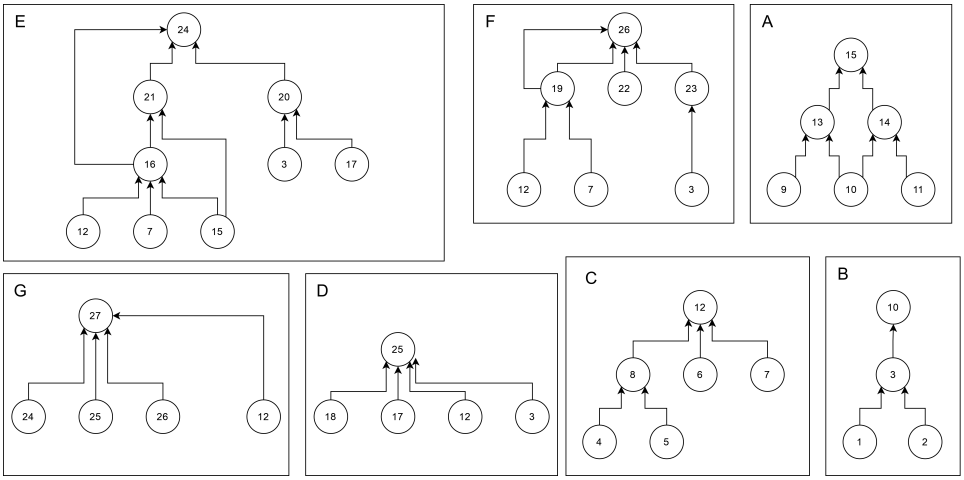

Abbildung 11: Manuell erstelltes Prozessmodell

Der generierte Prozess des ersten Experiments hat eine Kohäsion von 0,2723 und eine Kopplung von 0,5. Das PCCR beträgt 1,83. Der generierte Prozess erzielt somit eine höhere Kohäsion und eine geringere Kopplung als der manuell erstellte Prozess, wodurch das PCCR ebenfalls geringer ist.

Um die Unterschiede zwischen den beiden Prozessen herauszuarbeiten, werden als Erstes die Aktivitäten betrachtet. Der generierte Prozess besitzt vier Aktivitäten, die jeweils 1-3 Operationen umfassen, die fünfte Aktivität umfasst elf Operationen und damit mehr als die Hälfte aller Operationen im gesamten Prozess. Der originale, manuell erstellte Prozess umfasst sieben Aktivitäten, die größte Aktivität besitzt hier fünf Operationen. Daraus geht hervor, dass die Operationen im originalen Prozess gleichmäßiger verteilt wurden. In [8] wird die Hypothese aufgestellt, dass Aktivitäten nicht zu groß und nicht zu klein sein sollen, was sich in der Wahl der Aktivitäten widerspiegelt.

Die Gemeinsamkeit der Prozesse ist, dass zwei der Aktivitäten genau übereinstimmen. Aktivität A und B des originalen Prozesses lassen sich ebenso im generierten Prozess wiederfinden. Die Aktivitäten D,E,F,G des originalen Prozesses werden im generierten Prozess zu einer Aktivität zusammengefasst, mit Ausnahme der Operation für Informationselement 19. Durch die Zusammenfassung kann eine geringere Kopplung erzielt werden. Außerdem werden im generierten Prozess zwei Aktivitäten mit einer Kohäsion von 0 vermieden, im manuell erstellten Prozess waren dies die Aktivitäten G und D. Dadurch steigt die Prozesskohäsion im generierten Prozess insgesamt.

5.5 Threats to validity

Nach der Durchführung der Experimente und der Auswertung der Ergebnisse soll nun auf die Threats to validity eingegangen werden. Insbesondere sollen die Faktoren herausgearbeitet werden, welche die Experimente einschränken und die Validität der Ergebnisse beeinträchtigen könnten.

Zunächst wird die interne Validität der Experimente diskutiert. Bei der Implementierung der Softwarelösung wurde in einem ersten Schritt das Modell der Informationselement-Struktur in ein Metamodell abgebildet. Hierbei wurde die Struktur nicht genau übertragen, sondern es wurden Entscheidungen getroffen, um die einzelnen Bestandteile des Modells zu generalisieren. In der Folge wurden Tests aufgestellt, welche die Korrektheit der Ergebnisse und der Prozessmodelle überprüfen. Jedoch besteht hier die Möglichkeit, dass die Implementierung Fehler enthält, die durch die vorhandenen Tests noch nicht gefunden wurden. Für die Größe des gegebenen Beispielprozesses ist es möglich, eine manuelle Überprüfung der Ergebnisse durchzuführen, bei einer höheren Skalierung kann jedoch nicht mehr sichergestellt werden, ob die Berechnung der Metriken korrekt ist. Ein weiterer Faktor ist die Implementierung der Modelltransformationen. Hierbei müsste im Idealfall sichergestellt werden, ob durch den Einsatz der genannten Transformatio-

nen eine komplette Erforschung des Suchraums möglich ist und jede Abfolge der Transformationen die Zulässigkeit des Modells erhält. Zusätzlich ist fragwürdig, ob sich ein optimaler Prozess in der Realität durch ein geringes PCCR auszeichnet. Möglicherweise existieren hier weitere Metriken, die hinzugezogen werden müssten, um den Prozess zu bewerten.

In Bezug auf die externe Validität der Ergebnisse kann die Generalisierbarkeit der Ergebnisse diskutiert werden. Die implementierte Software wurde nur auf einer sehr kleinen Stichprobe an Prozessen ausgeführt. Fragwürdig ist es, ob das erstellte Metamodell dazu geeignet ist, komplexe Geschäftsprozesse abzubilden und die Ergebnisse auf diese Prozesse zu übertragen. Hierbei kann auch die Konstruktvalidität als ganzes hinterfragt werden. Das implementierte Metamodell bietet neben dem linearen Ablauf der Operationen nur die Möglichkeit eine Verzweigung von Operationen vorzunehmen. Kritisch ist hierbei auch die genaue Definition eines Informationselements, da diese sich in ihrer Komplexität unterscheiden. Operationen werden darüber hinaus identisch behandelt. Es existieren Operationen, welche eine einfache Berechnung darstellen, während andere Operationen einen komplexen Vorgang darstellen, bei dem beispielsweise Informationen aus anderen Organisationen eingeholt werden müssen. Diese vorhandene Komplexität der verschiedenen Operationen wird jedoch nicht im Modell abgebildet, obwohl dies in einem realistischen Szenario auch die Auswahl der Aktivitäten eines Prozesses beeinflussen würde.

6 Fazit

Abschließend lässt sich festhalten, dass sich der Ansatz der modellgetriebenen Optimierung dazu eignet, ausgewählte Optimierungsprobleme von Geschäftsprozessen zu formulieren und zu lösen. Es hat sich herausgestellt, dass eine Verbesserung bei der Modellierung der Prozesse im Hinblick auf Kohäsion und Kopplung der Aktivitäten möglich ist. Durch die Verwendung von Modellen als Softwareartefakte, können Prozesse einfach abgebildet werden. Zusammen mit der Implementierung von Modelltransformationen und der Verwendung genetischer Algorithmen, wie sie im MDEOptimiser verwendet werden, lassen sich Potenziale bei der Suche nach optimalen Prozessmodellen erkennen.

Dennoch sind gewisse Einschränkungen vorhanden, welche berücksichtigt werden müssen. An erster Stellte steht hierbei die Frage, wie ein geeignetes Metamodell für die Abbildung von komplexeren Prozessen aussehen soll. Je nach Kontext des Prozesses und mit steigender Komplexität der Prozesse kann sich die Erstellung eines Metamodells als schwierig erweisen. Dies wirkt sich auch auf das weitere Vorgehen aus, beispielsweise bei der Implementierung der Modelltransformationen und Zielfunktionen. Schließlich sollten die gestellten Optimierungsziele hinterfragt werden, da die Maximierung der Kohäsion und die Minimierung der Kopplung von Aktivitäten in der Realität nicht zwangsweise das beste Prozessmodell darstellt.

Es könnte daher interessant sein, weitere Metriken für die Bewertung von optimalen Prozessen zu entwickeln. Neben der Kohäsion und Kopplung könnten beispielsweise der Ressourcenverbrauch von Operationen und die Ausführungszeit berücksichtigt werden, um den bisherigen Fokus auf die innere Struktur der Prozesse, zu erweitern.

Literatur

[1] K. Born, S. Schulz, D. Strüber, and S. John. Solving the class responsibility assignment case with henshin and a genetic algorithm. In *TTC@ STAF*, pages 45–54, 2016.

[2] A. Burdusel and S. Zschaler. Mdeoptimiser github page. `https://mde-optimiser.github.io`, 2024.

[3] A. Burdusel, S. Zschaler, and D. Strüber. Mdeoptimiser: A search based model engineering tool. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 12–16, 2018.

[4] A. R. Da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.

[5] P. Groß. Link to the repository of the project. `https://github.com/Paul758/workflow-optimisation`, 2024.

[6] S. John, A. Burdusel, R. Bill, D. Strüber, G. Taentzer, S. Zschaler, and M. Wimmer. Searching for optimal models: Comparing two encoding approaches. *J. Object Technol.*, 18(3):6–1, 2019.

[7] J. Lampinen. Multiobjective nonlinear pareto-optimization. *Preinvestigation Report, Lappeenranta University of Technology*, 114:125, 2000.

[8] H. A. Reijers and I. T. Vanderfeesten. Cohesion and coupling metrics for workflow process design. In *International conference on business process management*, pages 290–305. Springer, 2004.

[9] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[10] J. vom Brocke, S. Zelt, and T. Schmiedel. On the role of context in business process management. *International Journal of Information Management*, 36(3):486–495, 2016.

Eidesstattliche Erklärung

Hiermit versichere ich, dass die Arbeit selbstständig verfasst wurde und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden.

Unterschrift: