Technische Hochschule Ingolstadt

Technische Hochschule Ingolstadt

Fakultät Informatik

Studiengang Wirtschaftsinformatik

Bachelorarbeit

Konzeption und Implementierung eines Ansatzes zur Erstellung von Graphdatenschemata für Prozessinstanz- und Entitätsdaten

vorgelegt von

Daria Öhlund
Matrikel-Nr: 00105334

Abgegeben am: 15. Januar 2024

Erstprüfer: Prof. Dr. Jochen Rasch
Zweitprüfer: Prof. Dr. Melanie Kaiser

Inhaltsverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface

BPMN Business Process Model and Notation

CMMN Case Management Model and Notation

CSV Comma-Separated Values

DMN Decision Model and Notation

JSON JavaScript Object Notation

OMG Object Management Group

REST Representational State Transfer

StMWi Bayerisches Staatsministerium für Wirtschaft, Landesentwicklung und Energie

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung in die Arbeit

Im Zentrum der vorliegenden Arbeit steht die intelligente Verarbeitung von Prozessdaten mittels Graphdatenbanken, die ein Schlüsselbaustein in der fortschreitenden Digitalisierung von Unternehmensprozessen darstellen. Die einleitenden Kapitel skizzieren die Motivation, die Forschungslücke und den aktuellen Forschungsstand und legen damit das Fundament für die Entwicklung eines innovativen Konzepts. Das Augenmerk liegt auf der effektiven Modellierung von Prozessinstanz- und Entitätsdaten, um durch deren Integration in Graphdatenbanken einen substantiellen Mehrwert für die Prozessoptimierung zu generieren.

1.1 Motivation

Die vorliegende Bachelorarbeit konzentriert sich auf die Entwicklung und Implementierung eines innovativen Konzepts unter Nutzung von Graphdatenbanken und ihrer Analysemöglichkeiten in der Prozessautomatisierung. Die Motivation hierfür erwächst aus der zunehmenden Bedeutung von Datenanalyse und -verarbeitung in der heutigen Geschäftswelt, gepaart mit dem Bedarf an effizienten, flexiblen und skalierbaren Lösungen im Datenmanagement und der Prozessoptimierung.

In einer Zeit, in der der Wert von Daten stetig steigt, wird die effiziente Sammlung, Speicherung und Analyse großer Datenmengen immer entscheidender für den Geschäftserfolg. Herkömmliche Datenbankmodelle kommen hierbei oft an ihre Grenzen, besonders bei der Handhabung komplexer Beziehungen und dynamischer Datenstrukturen. Graphdatenbanken stellen in diesem Kontext eine fortschrittliche Alternative dar, die eine effiziente Darstellung und Verarbeitung von Datenbeziehungen ermöglichen.

Es mangelt jedoch an umfassenden Konzepten, die eine Integration von Graphdatenbanken in die Prozessautomatisierung verschiedener Unternehmensgrößen erlauben. Diese Arbeit strebt danach, diese Forschungslücke zu schließen, indem sie ein praxisnahes Konzept für die Implementierung von Prozessinstanzdaten in eine Graphdatenbank entwickelt. Dies leistet nicht nur einen Beitrag zur wissenschaftlichen Forschung, sondern hat auch das Potenzial, Effizienz und Effektivität von Geschäftsprozessen wesentlich zu verbessern.

Das Hauptziel dieser Arbeit ist die Entwicklung eines Konzepts, das die Stärken von Graphdatenbanken nutzt, um Prozessinstanzdaten und verarbeitete Entitätsdaten in Unternehmen optimal zu analysieren. Dies umfasst die Erstellung eines Prototyps, der die Möglichkeiten von Graphdatenbanken zur Verbesserung von Entscheidungsprozessen durch präzisere Datenanalysen und -interpretationen demonstriert.

Mein persönliches Interesse an der Schnittstelle zwischen Datenwissenschaft und Unternehmensmanagement motiviert diese Arbeit zusätzlich. Die Chance, theoretisch fundierte und zugleich praktisch anwendbare Forschung zu betreiben, steht im Zentrum meines wissenschaftlichen Engagements.

Zusammenfassend leistet diese Bachelorarbeit einen wichtigen Beitrag zur aktuellen Forschung in der Datenverarbeitung und -analyse, indem sie eine Forschungslücke adressiert und praktikable Lösungen für reale Geschäftsprobleme aufzeigt, die durch die Anwendung von Graphdatenbanken entstehen.

1.2 Einordnung und Abgrenzung im Forschungsprojekt

Diese Arbeit bildet einen spezifischen Teil des Forschungsprojekts *PDA-RobE*, das vom Bayerisches Staatsministerium für Wirtschaft, Landesentwicklung und Energie (StMWi) gefördert wird. Beteiligt sind die exentra GmbH als Konsortialführer, die Conti Temic microelectronic GmbH und der Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik der Friedrich-Alexander-Universität Erlangen-Nürnberg. Die Erarbeitung dieser Arbeit erfolgte bei exentra.

„Im Engineering und dem zur Koordination benötigten Projektmanagement existieren viele nicht automatisierte Prozessschritte. Dieser Zustand ist sehr kosten- und zeitaufwändig sowie fehleranfällig. Ziel des Vorhabens ist es daher, die Transformation von einer reinen Management- und Engineering-Prozessdokumentation und -überwachung hin zur automatisierten Prozesskoordination und -ausführung im Sinne des durchgängigen Engineerings bzw. von Industrie 4.0 zu beschleunigen. [. . .]
Die [. . .] Managementworkflows zur Planung der roboterbasierten Automatisierungslösungen werden auf Basis von [Business Process Model and Notation (BPMN)] modelliert und mittels einer Process Engine in Kombination mit Engineering-Services zur Ausführung gebracht. Hierdurch können die anfallenden Kosten und die Zeitdauer für das Engineering nachhaltig gesenkt sowie die Qualität der Prozesse durch Transparenz gesteigert werden, indem die anfallenden Daten mit Methoden der Künstlichen Intelligenz und Data Science ausgewertet und für die Automatisierung verwendet werden“ [Forschungsteam PDA-RobE 2022, S. 2].

Die Arbeit konzentriert sich auf die Entwicklung eines Konzepts zur Schaffung einer Datenbasis, die als Fundament für zukünftige Anwendungen künstlicher Intelligenz dient. Es wird untersucht, wie Prozessdaten effektiv aufbereitet und in eine Graphdatenbank importiert werden können.

Die spezifischen Forschungsfragen dieser Arbeit lauten:

1. Welche Daten generiert eine Process Engine während des Durchlaufs einer Prozessinstanz?
2. Ist es möglich, ein generisches Graphdatenschema zu erstellen, das diese Daten effektiv integriert?
3. Wie könnte eine Referenzinfrastruktur für die Implementierung dieses Schemas aussehen?
4. Ist es möglich, das Schema effektiv in eine Graphdatenbank zu überführen?
5. Welche Möglichkeiten für zukünftige KI-basierte Auswertungen ergeben sich durch diese Graphdatenbank?

Das folgende Kapitel vertieft den aktuellen Forschungsstand bezüglich des Imports von Prozessinstanzdaten in Graphdatenbanken, um einen umfassenden Überblick über die bestehenden Erkenntnisse und Methoden in diesem Bereich zu bieten.

1.3 Aktueller Stand der Forschung

In diesem Kapitel wird der aktuelle Stand der Forschung in Bezug auf die Entwicklung von Graphschemata für Prozessinstanz- und Entitätsdaten sowie deren Import in Graphdatenbanken untersucht. Der Fokus liegt auf der Bewertung bestehender Methoden und Technologien, die für die Modellierung und Integration von komplexen Datenstrukturen in Graphdatenbanken verwendet werden. Es werden innovative Ansätze und Fallstudien aus der aktuellen Literatur vorgestellt und analysiert, um die Herausforderungen und Lösungsstrategien zu beleuchten, die bei der Konzeption effektiver Graphschemata auftreten. Ziel ist es, ein tiefes Verständnis der technologischen Möglichkeiten und Grenzen zu erlangen, um eine solide Grundlage für die Entwicklung des eigenen Konzepts zu schaffen.

Der Artikel „Konzept der Kontextualisierung in Graphdatenbanken" von von Trotha et al. zeigt ein Konzept auf, das die Nutzung von Graphdatenbanken zur Kontextualisierung von Prozessdaten belegt. Der Fokus liegt darauf, vielfältige Datenquellen in einer Graphstruktur zu verbinden, um so einen ganzheitlichen Kontext zu schaffen. Diese Methode hebt die Komplexität der Datenbeziehungen hervor und verbessert dadurch deutlich die Datenanalyse und -interpretation. Die Studie beweist, dass Graphdatenbanken besonders geeignet sind, um komplexe Datenbeziehungen darzustellen und zu analysieren. Sie zeigt die Effizienz von Graphdatenbanken in der praktischen Anwendung und unterstreicht deren Vorteile in der Dateninterpretation und Erkenntnisgewinnung [vgl. Trotha, Kleinert und Epple 2020, S. 68–76].

Ein aktueller und relevanter Beitrag im Bereich der Verknüpfung von Prozess- und Entitätsdaten stammt von Sheng Ye und Kollegen, die sich in ihrer Arbeit „Recovering Latent Data Flow from Business Process Model Automatically" mit der Herausforderung beschäftigen, Datenflüsse in Geschäftsprozessmodellen nachvollziehbar zu machen [vgl. Ye u. a. 2022, S. 1–11]. Sie adressieren das Problem, dass in vielen BPMN-Modellen Variablen und andere datenbezogene Elemente oft nicht explizit definiert sind, sondern in Formulardeklarationen, Variablendefinitionen oder Programmcode versteckt liegen [vgl. Ye u. a. 2022, S. 1–11]. Die von Ye et al. vorgeschlagene Methodik, die sowohl statische Codeanalyse als auch dynamische Log-Analyse umfasst [vgl. Ye u. a. 2022, S. 1–11], übersteigt den Umfang der hier dargelegten Arbeit, könnte aber in praktischen Umsetzung des Graphschemas einen großen Mehrwert liefern, besonders im Hinblick auf die Identifizierung und Nachverfolgung von datenbezogenen Aktivitäten innerhalb komplexer Prozessstrukturen.

Der Artikel „Verknüpfung von Geschäftsdaten und standardisierten Geschäftsprozessen in einem Data Warehouse" beschreibt die Herausforderungen und Lösungen bei der Integration von Geschäftsprozessdaten in Data Warehouse-Systeme. Er konzentriert sich auf die Verwendung von multidimensionalen Modellen zur Analyse von Geschäftsprozessen und betont die Bedeutung von

Data Warehousing-Technologien. Der Artikel stellt ein generisches Modell vor, das es ermöglicht, Geschäftsdaten automatisch in Geschäftsprozessmodelle zu integrieren, ohne die zugrundeliegende Geschäftslogik zu verstehen. Diese Herangehensweise nutzt Workflow Mining, um ein Meta-Geschäftsprozessmodell im Data Warehouse zu bilden, das Geschäftsobjekte und deren Attribute berücksichtigt [vgl. Kassem und Turowski 2018, S. 284–289].

In dieser Bachelorarbeit liegt der Fokus auf der Nutzung von Graphdatenbanken zur Verbesserung der Prozessautomatisierung, mit einem speziellen Augenmerk auf die Modellierung und Integration von Prozessinstanz- sowie Entitätsdaten in Graphdatenbanken. Im Gegensatz dazu konzentriert sich der diskutierte Artikel hauptsächlich auf den Einsatz von Data Warehouse-Technologien und multidimensionalen Modellen zur Analyse von Geschäftsprozessen [vgl. Kassem und Turowski 2018, S. 284–289]. Während die Bachelorarbeit die Vorteile von Graphdatenbanken für eine flexible und vernetzte Darstellung von Prozessdaten hervorhebt, befasst sich der Artikel von Kassem und Turowski mit der strukturierten und hierarchischen Speicherung von Geschäftsprozessdaten in einem Data Warehouse [vgl. Kassem und Turowski 2018, S. 284–289], was eine grundlegend andere Herangehensweise an das Datenmanagement darstellt.

Abschließend lässt sich sagen, dass die diskutierten Forschungsarbeiten von Trotha et al., Ye et al., und Kassem und Turowski wertvolle Perspektiven im Bereich der Datenanalyse und -verarbeitung aufzeigen. Sie beleuchten verschiedene Methoden und Technologien zur Analyse von Prozessdaten. Diese Bachelorarbeit hingegen hebt sich ab, indem sie spezifisch die Nutzung von Graphdatenbanken für die Kombination von Prozessinstanz- und Entitätsdaten untersucht. Dieser Ansatz, der sich auf die Vorteile von Graphdatenbanken für die Darstellung komplexer Beziehungen zwischen Daten stützt, bietet einen neuen Blickwinkel auf die Datenverarbeitung und hebt sich von den konventionellen Methoden, wie sie in den anderen Studien präsentiert werden, ab. Dadurch wird nicht nur die Vielfalt der Forschung in diesem Bereich betont, sondern auch das Potenzial für innovative Entwicklungen in der Zukunft aufgezeigt.

2 Einführung in die Grundlagen

Kapitel 2 bildet das Fundament dieser Arbeit, indem es die grundlegenden Technologien und Konzepte vorstellt, die für das Verständnis und die Ausführung der nachfolgenden Forschung unerlässlich sind. Beginnend mit einer tiefgehenden Untersuchung von Flowable, einem Schlüsselwerkzeug für die Automatisierung und Modellierung von Geschäftsprozessen, etabliert dieses Kapitel die Basis für die spätere Integration und Auswertung von Prozessdaten. Die folgenden Abschnitte widmen sich Elasticsearch und Kibana sowie Docker und Kubernetes – Technologien, die für die Analyse, Visualisierung und Verwaltung der in automatisierten Geschäftsprozessen generierten Daten entscheidend sind. Abschließend führt dieses Kapitel in die Konzepte und Anwendungen von Graphdatenbanken ein, mit einem besonderen Augenmerk auf Neo4j, um die Handhabung und Analyse von komplexen Datenstrukturen zu erörtern.

2.1 Flowable

In diesem Kapitel wird die Rolle von Flowable im Kontext der Automatisierung und Modellierung von Geschäftsprozessen untersucht. Es werden die Funktionen und Anwendungen von Flowable detailliert beleuchtet, insbesondere im Hinblick auf die Integration und Ausführung von BPMN 2.0 Prozessdefinitionen.

Flowable ist eine Java-basierte, leichtgewichtige Business-Process-Engine, die die Implementierung und Ausführung von BPMN 2.0 Prozessdefinitionen ermöglicht. Sie bietet vielfältige Funktionen, wie das Starten von Prozessinstanzen, das Ausführen von Abfragen und den Zugriff auf aktive oder historische Prozessinstanzen. Flowable kann flexibel in unterschiedlichen Umgebungen eingesetzt werden, sei es eingebettet in eine Anwendung oder über die Representational State Transfer (REST) Application Programming Interface (API). Zusätzlich bietet Flowable verschiedene Anwendungen wie Flowable Modeler oder Flowable Admin, die vordefinierte Benutzeroberflächen zur Prozess- und Aufgabenverwaltung bereitstellen. Der Kern von Flowable besteht aus einer Sammlung von Services, die APIs zur Verwaltung und Ausführung von Geschäftsprozessen bereitstellen [vgl. Flowable 2023d].

In dieser Arbeit kommen spezifisch die Flowable-Anwendungen Design und Work zum Einsatz, welche als zentrale Instrumente für die Prozessmodellierung und -ausführung dienen.

Flowable Design ist eine Modellierungsumgebung, die für die Erstellung verschiedener Modelltypen wie BPMN, Case Management Model and Notation (CMMN), Decision Model and Notation (DMN) und Formulare genutzt wird. Diese Modelle können in den Flowable-Anwendungen Orchestrate, Work und Engage implementiert und ausgeführt werden [vgl. Flowable 2023c]. Flow-Apps sind eine spezielle Art von Anwendungen in Flowable Design, die dazu dienen, verschiedene Definitionen wie Fälle, Prozesse, Entscheidungen und Formulare in einem einzigen Paket zu gruppieren [vgl. Flowable 2023a]. Flowable Design bietet also eine Plattform zur Entwicklung und Verwaltung von Modellen, die in den genannten Flowable-Anwendungen zum Einsatz kommen.

Im Anschluss an die detaillierte Betrachtung von Flowable Design, einem Schlüsselwerkzeug für die Prozessmodellierung, richtet sich der Fokus nun auf Flowable Work.

Flowable Work ist eine Plattform für Prozess- und Fallmanagement, die Prozess- und Case-Management Engines integriert, die vom Entwicklerteam vorab konfiguriert und über eine benutzerfreundliche Oberfläche zugänglich gemacht wurden. Flowable Work wurde so konzipiert, dass es eine umfangreiche Plug-In-Fähigkeit bietet und eine Vielzahl von Konfigurationen für alle seine Funktionen ermöglicht [vgl. Flowable 2023c]. Flowable Work ermöglicht also modellierte Prozesse auszuführen zu können.

BPMN ist ein von der Object Management Group (OMG) entwickelter Industriestandard zur grafischen Darstellung und Modellierung von Geschäftsprozessen. Er ermöglicht die Darstellung von Aktivitäten („Tasks“), Entscheidungspunkten („Gateways“) und Kontrollverbindungen („Sequence Flow“), wobei auch parallele Abläufe und Zuständigkeitsbereiche („Swimlanes“) abgebildet werden können. BPMN zielt darauf ab, unterschiedliche Darstellungsformen zu vereinheitlichen und so Portabilität und Interoperabilität zu ermöglichen. Die Entwicklung des Standards wird kontinuierlich vorangetrieben, wobei in neueren Versionen wie BPMN 2.0 eine stärkere Berücksichtigung der Automatisierung von Geschäftsprozessen erfolgt [vgl. Leymann und Schumm 2018a].

Die Integration von BPMN in Flowable hebt die strategische Bedeutung der Process Engine hervor. Diese Komponente ist nicht nur ein technischer Baustein, sondern auch ein Katalysator für effiziente Geschäftsprozesse in modernen Unternehmensstrukturen.

Eine Process Engine ist ein Kernbestandteil der technischen Infrastruktur für die Automatisierung von Geschäftsprozessen. Sie koordiniert die Abläufe, die durch Prozessmodelle definiert sind, und interagiert mit verschiedenen Anwendungen oder Diensten, die die eigentlichen Arbeitsschritte ausführen. Die Process Engine ermöglicht die flexible Anpassung von Unternehmen an neue Anforderungen und ist zentral für die Integration interner Prozesse mit Systemen anderer Unternehmen. Sie spielt eine entscheidende Rolle in der heutigen und zukünftigen IT-Landschaft von Unternehmen [vgl. Leymann und Schumm 2018b].

Nachdem die Funktionen und Einsatzmöglichkeiten von Flowable umfassend dargestellt wurden, wendet sich das nächste Kapitel dem Thema Elasticsearch und Kibana zu. Diese Werkzeuge spielen eine entscheidende Rolle bei der Analyse und Visualisierung von Daten, die in automatisierten Geschäftsprozessen generiert werden. Die Betrachtung von Elasticsearch und Kibana wird aufzeigen, wie diese Technologien in Verbindung mit Flowable genutzt werden können, um tiefergehende Einblicke in Prozessabläufe und generierten Daten zu gewinnen.

2.2 Elasticsearch und Kibana

Nach der ausführlichen Darstellung der Funktionen und Anwendungsmöglichkeiten von Flowable, einem zentralen Werkzeug zur Automatisierung und Modellierung von Geschäftsprozessen, widmet sich dieses Kapitel nun Elasticsearch und Kibana. Diese beiden Technologien spielen eine wesentliche Rolle in der Analyse und Visualisierung von Daten, die in automatisierten Ge-

schäftsprozessen generiert werden. Der Fokus liegt auf der Aufbereitung und Darstellung dieser Daten mittels Elasticsearch und deren Visualisierung durch Kibana, um tiefere Einblicke in die Prozessabläufe und generierten Daten zu ermöglichen. Die Betrachtung dieser Technologien wird aufzeigen, wie sie in Kombination mit Flowable genutzt werden können, um umfassende Analysen und Einblicke in die Abläufe und Strukturen von Geschäftsprozessen zu gewinnen.

Elasticsearch ist ein verteilter Such- und Analysemotor, der als zentrales Element des Elastic Stack fungiert. Es unterstützt das Sammeln, Aggregieren und Anreichern von Daten mittels Logstash und Beats, gefolgt von Speicherung und Indexierung in Elasticsearch. Kibana erlaubt es Nutzern, interaktiv Daten zu erkunden, zu visualisieren und Einblicke zu teilen. Elasticsearch bietet nahezu Echtzeitsuche und -analyse für verschiedene Datentypen, von strukturierten Texten bis hin zu Geodaten. Die verteilte Natur von Elasticsearch ermöglicht eine effiziente Skalierung mit wachsenden Daten- und Anfragevolumina [vgl. Elasticsearch B.V. 2024d].

Elasticsearch speichert Informationen nicht in herkömmlichen tabellarischen Strukturen, sondern als JavaScript Object Notation (JSON)-Dokumente. Diese Dokumente enthalten komplexe Datenstrukturen, die serialisiert sind, was die Speicherung und Indizierung in Elasticsearch effizient unterstützt. Durch die JSON-Formatierung können vielfältige Datentypen strukturiert und für schnelle Suchvorgänge aufbereitet werden. Elasticsearchs Fähigkeit, mit JSON-Dokumenten zu arbeiten, ermöglicht es, eine breite Palette von Daten flexibel zu verwalten und zu analysieren[vgl. Elasticsearch B.V. 2024a].

JSON ist ein einfaches, leichtgewichtiges Format zur Darstellung von Daten, das häufig in Webdiensten, Client-Server-Anwendungen wie NoSQL-Datenbanken und für den Datenaustausch zwischen verschiedenen Anwendungen verwendet wird. Es basiert auf der JavaScript-Sprachsyntax, was es insbesondere für JavaScript-Programmierer vertraut macht. JSON kann direkt im JavaScript-Code verwendet werden, etwa auf Webseiten. Seine Einfachheit und universelle Anwendbarkeit machen es zu einem wichtigen Werkzeug für die Arbeit mit Daten, besonders im Webkontext [vgl. Nolan und Lang 2014, S. 227].

Kibana ermöglicht es Nutzern, Daten aus Elasticsearch auf vielfältige Weise zu visualisieren. Die Plattform bietet zahlreiche Optionen zur Darstellung von Daten, darunter Diagramme, Messgeräte, Karten und Graphen, die in Dashboards integriert werden können. Diese Visualisierungen helfen dabei, verborgene Einsichten in den Daten zu entdecken und komplexe Datenbeziehungen zu verstehen. Kibana unterstützt die Visualisierung verschiedenster Datentypen, einschließlich strukturierter und unstrukturierter Texte, numerischer Daten, Zeitreihendaten und Geodaten. Dadurch wird Kibana zu einem leistungsstarken Werkzeug für die Datenanalyse und das Reporting [vgl. Elasticsearch B.V. 2024c].

In der vorliegenden Arbeit spielen Elasticsearch und Kibana eine zentrale Rolle bei der Aufbereitung und Analyse von Prozessinstanzdaten. Die effiziente Speicherung und Indizierung von Daten in Elasticsearch sowie die fortschrittlichen Visualisierungsmöglichkeiten in Kibana ermöglichen es, tiefgreifende Einsichten in Prozessabläufe zu gewinnen. Es wurde im Rahmen dieser Arbeit genutzt, um die Prozessinstanzdaten analysieren zu können.

Nach der detaillierten Untersuchung von Elasticsearch und Kibana, die entscheidend sind für die Datenaufbereitung und -analyse, verlagert sich der Schwerpunkt auf die Containerisierung mit Docker. Dieses Thema bildet einen weiteren wichtigen Baustein in der Entwicklung einer flexiblen und effizienten IT-Infrastruktur, die für die Implementierung und das Management von Geschäftsprozessen notwendig ist. Docker erleichtert die Bereitstellung und Skalierung von Anwendungen, was für die Realisierung von komplexen Geschäftsprozessen in dynamischen Umgebungen unerlässlich ist.

2.3 Containerisierung mit Docker und Verwaltung mit Kubernetes

Die Referenzinfrastruktur dieser Arbeit nutzt Docker und Kubernetes für die Bereitstellung von Diensten wie Elasticsearch, Kibana und Neo4j. Docker, eine offene Plattform zur Entwicklung und Ausführung von Anwendungen, ermöglicht eine effiziente Trennung von Anwendungen und Infrastruktur. Kubernetes, eine Open-Source-Plattform zur Verwaltung containerisierter Workloads, unterstützt die Automatisierung und Skalierung von Anwendungen und basiert auf Googles langjähriger Erfahrung. Beide Technologien tragen zur Agilität und Effizienz der Servicebereitstellung bei und bilden eine grundlegende Komponente der IT-Infrastruktur dieser Arbeit.

Docker ist eine offene Plattform zur Entwicklung, Auslieferung und Ausführung von Anwendungen. Es ermöglicht die Trennung von Anwendungen von der Infrastruktur, wodurch Software schnell bereitgestellt werden kann. Docker nutzt Container, die Anwendungen in einer isolierten Umgebung ausführen, wodurch viele Container gleichzeitig auf einem Host laufen können. Docker erleichtert die Lebenszyklusverwaltung von Containern, von der Entwicklung bis zum Einsatz in der Produktion. Es unterstützt schnelle, konsistente Anwendungsbereitstellung und effiziente Skalierung, wobei es auf einer Client-Server-Architektur basiert und durch die Docker-Client- und -Daemon-Kommunikation funktioniert. Docker eignet sich besonders für hohe Arbeitsauslastung und effiziente Ressourcennutzung [vgl. Docker Inc. 2024].

Kubernetes ist eine offene, portable und erweiterbare Plattform zur Verwaltung containerisierter Arbeitslasten und Services, die sowohl deklarative Konfiguration als auch Automatisierung unterstützt. Es wurde 2014 von Google als Open-Source-Projekt eingeführt und basiert auf Googles umfangreicher Erfahrung mit dem Betrieb großer Produktionssysteme. Kubernetes eignet sich als Containerplattform, Microservices-Plattform und portable Cloud-Plattform. Es fördert die Agilität in der Anwendungserstellung und -bereitstellung und unterstützt dabei eine breite Palette von Workloads. Kubernetes vereinfacht die Automatisierung, Skalierung und Verwaltung von Anwendungen, wobei es auf einem Ökosystem von Komponenten und Tools basiert [vgl. The Kubernetes Authors 2024].

Nach der Betrachtung von Docker und Kubernetes, die für die Servicebereitstellung entscheidend sind, richtet sich der Fokus nun auf Quarkus-Service. Quarkus, eine moderne Java-Plattform, ist für die Erstellung effizienter Microservices- und Serverless-Anwendungen konzipiert und ergänzt die containerbasierte Architektur von Docker und Kubernetes ideal. Im nächsten Abschnitt wird erörtert, wie Quarkus zur Gestaltung reaktiver und ressourceneffizienter Services innerhalb der

Referenzinfrastruktur beiträgt.

2.4 Graphdatenbanken und Neo4j

Graphdatenbanken, wie Neo4j, repräsentieren einen innovativen Ansatz im Datenmanagement, indem sie Informationen in Graphstrukturen speichern und verarbeiten. Diese Datenbanken bieten effiziente Lösungen für die Verarbeitung komplexer, vernetzter Daten, was sie besonders relevant für die in dieser Arbeit behandelten Themen macht.

Graphdatenbanken sind eine Form der Datenbankmodelle, die Information in Graphstrukturen speichern und bearbeiten. Im Kern eines Graphdatenbankmodells stehen Datenstrukturen, die als Graphen oder Graphenerweiterungen wie Hypergraphen konzipiert sind. Die Datenmanipulation in Graphdatenbanken erfolgt durch graphorientierte Operationen und Abfragesprachen, die auf Graphmerkmale wie Pfade, Nachbarschaften, Subgraphen und Konnektivität ausgerichtet sind. Graphdatenbanken sind besonders geeignet, um unstrukturierte Daten zu verarbeiten und zeichnen sich durch eine flexible Handhabung der Trennung zwischen Schema und Daten aus. Sie unterstützen zudem die Integrität und Konsistenz von Daten durch verschiedene Arten von Integritätsbedingungen [vgl. Angles und Gutierrez 2017, S. 1–2].

Neo4j ist eine native Graphdatenbank, die Daten in einem natürlichen, vernetzten Zustand speichert und verwaltet. Sie basiert auf einem Eigenschaftsgraph-Ansatz, der sowohl für die Laufzeitperformance als auch für Operationen vorteilhaft ist. Neo4j hat sich von einer Graphdatenbank zu einem umfangreichen Ökosystem mit zahlreichen Werkzeugen, Anwendungen und Bibliotheken entwickelt, die eine nahtlose Integration von Graphentechnologien in bestehende Arbeitsumgebungen ermöglichen. Die Plattform unterstützt verschiedene Arten der Installation und Bereitstellung, darunter On-Premise, Cloud und die vollständig verwaltete Cloud-Service-Option Neo4j Aura. Sie bietet auch fortschrittliche Tools für die Datenvisualisierung und Analyse, sowie Integrationen und Treiber für vielfältige Anwendungsumgebungen [vgl. Neo4j Inc. 2023b].
Ein Graph besteht aus Knoten (Nodes) und Beziehungen (Relationships). Knoten repräsentieren Entitäten oder Objekte, und können mit Labels und Eigenschaften (Properties) versehen sein, die weitere Informationen über die Entität bereitstellen. Beziehungen verbinden Knoten und stellen die Beziehungen zwischen diesen dar. Sie besitzen eine Richtung, einen Typ und können ebenfalls Eigenschaften haben, die die Natur der Verbindung beschreiben. Diese Struktur ermöglicht es, komplexe Datenbeziehungen effizient zu modellieren und abzufragen [vgl. Anthapu 2022, S. 4–6].
Neo4j bietet mit Cypher eine Abfragesprache, die speziell für die Arbeit mit Eigenschaftsgraphen konzipiert wurde. Ihre Syntax erinnert stark an SQL und ermöglicht es Nutzern, komplexe Graphmuster und Pfadabfragen auf intuitive Weise zu formulieren. Dadurch wird der Umgang mit Graphdaten in Neo4j erheblich vereinfacht und effizienter gestaltet [vgl. Angles und Gutierrez 2017, S. 15].

Nachdem nun die Grundkenntnisse, die für diese Arbeit von Bedeutung sind, vermittelt wurden, widmet sich das nächste Kapitel der Konzeption eines Ansatzes, um ein generisches Graphschema für Prozessinstanz- und Entitätsdaten zu erstellen.

3 Konzeption eines Ansatzes für das Graphdatenschema

Dieses Kapitel befasst sich mit der detaillierten Entwicklung und Strukturierung eines Graphdatenschemas, das Prozess- und Entitätsdaten integriert. Ziel ist es, einen umfassenden Rahmen zu schaffen, der die komplexen Beziehungen und Dynamiken innerhalb der Prozessdaten visualisiert und analysierbar macht.

Es beginnt mit einer gründlichen Analyse der Struktur von Prozessdaten, um die Kernkomponenten und ihre Wechselwirkungen zu verstehen. Anschließend wird die Struktur der Entitätsdaten und deren Bedeutung in der Darstellung von Geschäftsprozessen untersucht. Ein wesentlicher Fokus des Kapitels liegt auf den spezifischen Transformationsregeln, die erforderlich sind, um diese vielfältigen Daten effektiv in ein Graphdatenschema zu überführen. Dieses Schema dient als Grundlage für die anschließende Analyse und Interpretation von Prozessabläufen und -interaktionen.

3.1 Analyse der Struktur von Prozessdaten

Um die Datenstruktur von Prozessdaten zu verstehen, ist die Unterscheidung zwischen einem Prozessmodell und Prozessinstanzen entscheidend. Ein Prozessmodell definiert die auszuführenden Arbeitsschritte und ihre Abfolge. Eine Prozessinstanz wiederum ist ein spezifischer Durchlauf eines solchen Modells [vgl. Leymann und Schumm 2018b].
Ein Modell kann mehrere Instanzen haben, die entweder nacheinander oder parallel ablaufen.

Für diese Arbeit werden die Prozessmodelle nicht direkt in die Graphdatenbank importiert; stattdessen konzentriert sich die Analyse auf die einzelnen Prozessinstanzen und deren zugehörigen Daten.
Im Rahmen dieser Arbeit umfassen Prozessinstanzdaten sowohl die Daten der Prozessinstanzen selbst als auch die dazugehörigen Aktivitäts- und Taskdaten. Im Folgenden werden die Taskdaten als Aufgabendaten referenziert. Um jedoch eine klare Unterscheidung zu ermöglichen, wird im Folgenden zwischen *Prozessdaten* – als Sammelbegriff für Prozessinstanz-, Aktivitäts- und Aufgabendaten – und *Prozessinstanzdaten*, die sich ausschließlich auf den Objekttyp *Prozessinstanz* beziehen, differenziert. Diese Unterscheidung dient dazu, Verwechslungen zu vermeiden, wenn im weiteren Verlauf der Arbeit verschiedene Objekttypen betrachtet werden.
Elasticsearch und Kibana wurden zur Datenauswertung verwendet, wobei Kibana die in Elasticsearch gespeicherten Daten in Form von JSON-Dateien oder Tabellen visualisiert. Elasticsearch indiziert jeden Datensatz, beginnend mit dem jeweiligen Objekttyp, was eine klare Zuordnung ermöglicht. Die betrachteten Objekttypen für das prototypische Graphdatenschema umfassen Prozessinstanz-, Aktivitäts- und Aufgabendaten.
Im weiteren Verlauf konzentriert sich die Diskussion auf die detaillierte Analyse relevanter Objekttypen. Vertiefende Informationen finden sich in vier verschiedenen gekürzten JSON-Dateien im Anhang (A1 - A3). Nachfolgend werden die in Flowable generierten Standardfelder erläutert, die für das Graphschema und die Auswertungen dieser Arbeit von Bedeutung sind.
Zusätzliche Felder können je nach den Zielen der späteren Auswertungen als Attribute hinzugefügt

werden.

Prozessinstanzdaten:

Jede Prozessinstanz wird durch eine JSON-Datei repräsentiert, die unterschiedliche Felder enthält. Diese Felder entwickeln sich dynamisch entsprechend dem Fortschritt der jeweiligen Prozessinstanz.

- **deploymentId:**
  Diese Identifikationsnummer, gespeichert als `String`, ist zentral für die Organisation von Deployments in Flowable. Sie bleibt gleich für alle Prozesse innerhalb eines bestimmten Deployments und ändert sich bei jedem neuen Deployment, unabhängig davon, ob es sich um ein einzelnes oder mehrere Prozessmodelle handelt. Änderungen treten auf, wenn Modifikationen an den Prozessmodellen vorgenommen werden. Eine neue *deploymentId* impliziert auch eine Änderung der *processDefinitionId*. Es wird empfohlen, nur die modifizierten Prozesse in ein Deployment zu inkludieren, da es die präzise Nachverfolgung und Analyse von Änderungen in späteren Auswertungen erleichtert. Die *deploymentId* spielt somit eine entscheidende Rolle bei der Zuordnung und dem Tracking von Prozessversionen, indem die Identifikation von Veränderungen und deren Auswirkungen auf jede spezifische Version ermöglicht.

- **processDefinitionKey:**
  Dieses als `String` formatierte Feld definiert den *processDefinitionKey*, der bei der initialen Erstellung eines Prozessmodells festgelegt wird. Typischerweise entspricht dieser Schlüssel dem Namen des Prozesses (angegeben im *processDefinitionName*), konvertiert in Kleinbuchstaben und ohne Leerzeichen. Der *processDefinitionKey* ist entscheidend für die eindeutige Identifikation eines Prozessmodells über verschiedene Deployments hinweg. Um die Konsistenz und die Nachverfolgbarkeit zu gewährleisten, wird empfohlen, diesen Schlüssel während des gesamten Lebenszyklus eines Prozesses unverändert zu lassen. Er spielt eine fundamentale Rolle bei der zuverlässigen Identifikation eines Prozessmodells und wird in Systemen wie Flowable genutzt, um die Konsistenz und Identifikation spezifischer Prozessmodell-Versionen über verschiedene Deployments hinweg zu gewährleisten. Dies ermöglicht eine klare Unterscheidung und Verwaltung verschiedener Versionen des gleichen Prozessmodells.

- **processDefinitionName:**
  Dieses Feld speichert den Namen des Prozessmodells als `String`. Obwohl es technisch möglich ist, diesen Namen bei jedem Deployment zu ändern, wird dies nicht empfohlen. Häufige Änderungen können die Übersichtlichkeit beeinträchtigen, besonders bei späteren Auswertungen. Der *processDefinitionName* ist hauptsächlich für die menschliche Lesbarkeit und Analyse wichtig, da er Aufschluss über den Inhalt und Kontext des Prozessmodells sowie der zugehörigen Prozessinstanzen gibt.

- **processDefinitionId:**
  Diese ID, einzigartig für jede Prozessdefinition oder -version, ändert sich mit jeder Modifikation des Prozessmodells. Sie ermöglicht es, Prozessinstanzen einer bestimmten Modellversion zuzuordnen. Die *processDefinitionId* steht in einer hierarchischen Beziehung eine Stufe un-

ter der *deploymentId*, da jede Prozessdefinition innerhalb eines Deployments ihre eigene, einzigartige ID erhält. Die Kombination aus *deploymentId* und *processDefinitionId* ist von entscheidender Bedeutung, da sie eine detaillierte Zuordnung und Analyse der Prozessinstanzen auf der Grundlage spezifischer Modellversionen und Deployments erlaubt. Sie ermöglichen es, Veränderungen über die Zeit hinweg zu verfolgen und bieten eine präzise Basis für die Bewertung der Auswirkungen von Prozessmodelländerungen.

- **processDefinitionVersion:**
  Dieses Feld, gespeichert als `Integer`, spielt eine wesentliche Rolle bei der genauen Identifizierung der Version eines Prozessmodells. Es ergänzt die *processDefinitionId* und ist entscheidend für die Auswertung und Analyse sowohl vergangener als auch aktueller Versionen von Prozessmodellen. Die *processDefinitionVersion* ermöglicht eine differenzierte Betrachtung der Entwicklungsstufen eines Prozessmodells. Durch die präzise Versionierung ist es möglich, Änderungen im Zeitverlauf nachzuvollziehen und deren Auswirkungen auf die assoziierten Prozessinstanzen und Aktivitäten effektiv zu analysieren.

- **id:**
  Die Identifikationsnummer *id* einer Prozessinstanz ist besonders kritisch, wenn es um die Übertragung von Prozessinstanz-, Aktivitäts- und Aufgabendaten in eine Graphdatenbank geht. Als `String` formatiert, dient sie als unverwechselbares Kennzeichen für jede Prozessinstanz. Diese eindeutige Identifikationsnummer ist essenziell, um jede Prozessinstanz klar zu unterscheiden und die korrekte Zuordnung zu den dazugehörigen Aktivitäten und Aufgaben sicherzustellen. Im Kontext dieser Arbeit wird diese ID im folgenden als *processInstanceId* bezeichnet. Sie ist entscheidend für die Erstellung von Verknüpfungen zwischen Knoten.

- **startTime:**
  Dieses Feld, gespeichert als `String`, markiert den Beginn einer Prozessinstanz und wird zum Zeitpunkt ihrer Erstellung festgelegt. Die *startTime* dokumentiert das genaue Datum und die Uhrzeit bis auf Millisekunden genau und ist somit entscheidend für die zeitliche Nachverfolgung des Prozessstarts.

- **endTime:**
  Analog zur *startTime* gibt die *endTime*, ebenfalls als `String` gespeichert, den Zeitpunkt des Abschlusses einer Prozessinstanz an. Dieses Feld wird erst generiert, wenn die Prozessinstanz beendet ist, sei es durch erfolgreiches Durchlaufen oder durch Abbruch. Vor Beendigung der Prozessinstanz existiert dieses Feld nicht.

- **deleteReason:**
  Das *deleteReason*-Feld, gespeichert als `String`, ist ein weiteres dynamisch erstelltes Feld, das ausschließlich bei einem Abbruch der Prozessinstanz relevant ist. Es liefert detaillierte Informationen über den Grund des Abbruchs, was für die Analyse von Prozessabbrüchen von Bedeutung ist.

- **durationInMillis:**
  Das *durationInMillis*-Feld, dokumentiert als `Integer`, wird parallel zur *endTime* erstellt

und repräsentiert die Gesamtlaufzeit der Prozessinstanz in Millisekunden. Dieses Feld ist besonders wertvoll für die analytische Auswertung, wie beispielsweise die Berechnung der durchschnittlichen Durchlaufzeit pro Prozessversion, und ermöglicht so eine tiefere Einsicht in die Effizienz und Leistung verschiedener Prozessversionen.

- **involvedUsers:**
  Das Feld, implementiert als eine Liste von `Strings`, dokumentiert die Beteiligung von Flowable-Nutzern an einer Prozessinstanz. Obwohl im Rahmen der in dieser Arbeit dargelegten Implementierung das Feld *involvedUsers* eine untergeordnete Rolle spielt – da sämtliche Prozessinstanzen von einem einzigen admin-Account erstellt wurden -, hat es in einem breiteren Anwendungskontext eine erhebliche Bedeutung. In der Praxis ermöglicht dieses Feld die Nachverfolgung der Nutzerbeteiligung an verschiedenen Prozessinstanzen. Dies ist besonders nützlich, um zu analysieren, welche Nutzer regelmäßig in Prozessabläufen involviert sind.

- **startUserId:**
  Dieses Feld, gespeichert als `String`, identifiziert den Nutzer, der eine Prozessinstanz gestartet hat. Obwohl es in der prototypischen Implementierung, die in dieser Arbeit beschrieben wird, keine wesentliche Rolle spielt, ist eine Erwähnung für die Vollständigkeit unerlässlich. In praktischen Anwendungsszenarien kann das *startUserId*-Feld wichtige Einblicke bieten. Es ermöglicht die Analyse, welcher Nutzer bestimmte Prozessinstanzen initiiert hat, was für die Auswertung der Nutzeraktivitäten und -beteiligungen im Prozessmanagement von Bedeutung ist.

Aktivitätsdaten:
Jede Aktivität einer Prozessinstanz repräsentiert Elasticsearch als einzelne JSON-Datei. Diese als Aktivitätsdaten gekennzeichneten Informationen beinhalten sämtliche im BPMN-Diagramm abgebildeten Schritte.
Für die prototypische Implementierung lag der Fokus auf den wesentlichen Elementen, die nachstehend beschrieben werden. Es ist anzumerken, dass einige Aktivitäten im BPMN-Kontext als *Task* bezeichnet werden. Diese Bezeichnung sollte jedoch nicht mit den später analysierten Aufgabendaten verwechselt werden. In dieser Arbeit wird daher der Begriff *Aktivität* als Überbegriff für alle in einem Prozessmodell mit BPMN dargestellten Elemente verwendet.
Folgende Aktivitätstypen wurden berücksichtigt und werden anschließend erläutert, bevor auf die gemeinsamen Datenfelder dieser Aktivitäten eingegangen wird:

- **None Start Event:**
  Ein *None Start Event* signalisiert den Start einer Prozessinstanz ohne spezifischen Auslöser. Subprozesse beginnen standardmäßig mit einem solchen Ereignis [vgl. Flowable 2023b].

- **None End Event:**
  Entsprechend dem *None Start Event* zeigt das *None End Event* das Ende einer Prozessinstanz oder eines aktiven Pfades innerhalb einer Prozessinstanz an [vgl. Flowable 2023b].

- **User Task:**
  Ein *User Task* bezieht sich auf eine Aufgabe, die manuell von einer Person durchgeführt wird [vgl. Flowable 2023b].

- **Manual Task:**
  Dieser Aufgaben-Typ wird außerhalb der BPMN Engine ausgeführt und repräsentiert Aktivitäten, die kein System oder Nutzerinterface erfordern. Er wird von der Engine wie eine Durchlaufaktivität behandelt, sodass die Prozessinstanz ohne Verzögerung fortgesetzt wird [vgl. Flowable 2023b].

- **Sequence Flow:**
  Ein *Sequence Flow* stellt die Verbindung zwischen zwei Elementen in einem Prozessmodell dar [vgl. Flowable 2023b].

- **Subprocess:**
  Ein *Subprocess* ist ein Unterprozess, der entweder parallel zur laufenden Prozessinstanz abläuft oder diese für seine Dauer unterbricht [vgl. Flowable 2023b].

- **Exclusive Gateway:**
  Ein *Exclusive Gateway* dient der Modellierung von Entscheidungen im Prozessablauf [vgl. Flowable 2023b].

- **Parallel Gateway:**
  Ein *Parallel Gateway* ermöglicht es, bestimmte Aktivitäten parallel durchzuführen. Es kann Pfade sowohl aufteilen als auch wieder zusammenführen und wartet am Zusammenführungspunkt, bis alle aktiven Pfade eingetroffen sind [vgl. Flowable 2023b].

- **Inclusive Gateway:**
  Das *Inclusive Gateway* kombiniert Elemente eines *Exclusive Gateway* und eines *Parallel Gateway*. Es ermöglicht das Durchlaufen mehrerer Pfade, die den definierten Bedingungen entsprechen [vgl. Flowable 2023b].

Nach der Erläuterung der relevanten BPMN-Elemente richtet sich der Fokus nun auf die Felder in den JSON-Dateien, die diese Aktivitäten repräsentieren. Da einige dieser Felder bereits im Abschnitt über die Prozessinstanz behandelt wurden, erfolgt bei diesen nur eine ergänzende Betrachtung.
Es ist wesentlich zu betonen, dass eine Prozessinstanz aus einer JSON-Datei besteht und zahlreiche Aktivitäten umfasst, wobei jede Aktivität in einer separaten JSON-Datei dargestellt wird.

- **processDefinitionId:**
  Dieser als `String` gespeicherte Wert stimmt mit dem gleichnamigen Feld einer Prozessinstanz überein und ermöglicht die eindeutige Zuordnung einer Aktivität zu einem bestimmten Prozessmodell.

- **processInstanceId:**
  Wie bereits im Abschnitt zur *id* in der Prozessinstanz erwähnt, wird die *id* einer Prozessinstanz

oft als *processInstanceId* referenziert. Dies trifft auf die Aktivitätsdaten zu und dient der eindeutigen Zuordnung einer Aktivität zu einer bestimmten Prozessinstanz.

- **runtimeActivityInstanceId:**
  Diese einzigartige Kennung, im Folgenden als *activityInstanceId* bezeichnet, differenziert einzelne Aktivitäten voneinander.

- **activityType:**
  Dieses Feld gibt an, zu welchem Typ eine Aktivität gehört, basierend auf den im vorherigen Abschnitt besprochenen Typen.

- **activityId:**
  Die Kennzeichnung wird für jedes Element in einem Prozessmodell verwendet. Sie identifiziert ein Aktivitätselement, ist aber nicht unbedingt einzigartig, wenn das Element mehrfach in einer Prozessinstanz durchlaufen wird. Sie kann automatisch erstellt oder vom Ersteller des Prozessmodells angepasst werden und ermöglicht es, die Häufigkeit des Durchlaufens eines Elements zu erfassen.

- **activityName:**
  Der Name einer Aktivität, der im Feld *activityName* gespeichert wird, hat eine beschreibende Funktion. Er wird meist nur Aktivitäten des Typs *Task* zugewiesen, während andere Aktivitätstypen wie Sequence Flows oder Gateways in der Regel keinen Namen haben.

- **executionId:**
  Die *executionId*, fortan als *pathId* bezeichnet, gibt die ID eines Pfades an. Sie ermöglicht die Unterscheidung verschiedener Pfade, insbesondere bei parallelen oder inklusiven Gateways.

- **startTime, endTime, durationInMillis:**
  Diese Felder haben dieselbe Funktion wie in der Prozessinstanz, sind jedoch auf die jeweilige Aktivität bezogen.

- **assignee:**
  Dieses Feld, das angibt, wem eine Aufgabe zugewiesen wurde, spielt in der prototypischen Implementierung eine untergeordnete Rolle, ist aber erwähnenswert für potenzielle Auswertungen in der Praxis. Es muss über alle Systeme eine eindeutige ID zur Identifizierung des Verantwortlichen geben.

- **taskId:**
  Diese Identifikationsnummer verweist auf die spezifische Aufgabe, der von einer Aktivität erzeugt wurde. Sie wird dynamisch erstellt und ermöglicht die eindeutige Zuordnung eines Aufgaben-Datensatzes zu einer Aktivität.

Aufgabendaten:
Die folgende Erklärung befasst sich mit den Aufgabendaten, die trotz ihres Umfangs nur teilweise für die prototypische Implementierung relevant sind. Jeder Aufgaben-Datensatz ist eindeutig einer bestimmten Aktivität zugeordnet.

- **processDefinitionId, processInstanceId:**
  Diese Felder geben jeweils die zugehörige Prozessmodell-ID und Prozessinstanz-ID an.

- **id:**
  Die *id* ist eine für jeden Aufgaben-Datensatz einzigartige ID. Sie ist identisch zur *taskId* in der zugehörigen Aktivität und ermöglicht eine klare Zuordnung von Aktivität zur Aufgabe. Im Folgenden Verlauf wird sie als *taskId* referenziert.

- **formKey:**
  Der *formKey*, gespeichert als `String`, kennzeichnet das durch die Aktivität oder die Aufgabe ausgelöste Formular und ist entscheidend für die Zuordnung entsprechender Einträge in den *variables*.

- **variables:**
  Diese Liste umfasst alle im Prozessverlauf entstehenden Variablen. Ein Eintrag in den Variablen folgt stets demselben Muster. Folgend die zwei für die Arbeit relevanten Felder:

  - **name:**
    Dieses Feld enthält den Namen des Formulars, aufgebaut als `form_[Formschlüssel]_outcome`.

  - **textValueKeyword:**
    textValueKeyword speichert das Ergebnis des Formulars, standardmäßig `COMPLETED`. Das Ergebnis kann allerdings verändert werden.

- **createTime, endTime:**
  Diese Felder beschreiben das Erstellungsdatum der Aufgabe beziehungsweise dessen Beendigung, analog zur *startTime* und *endTime* in einer Prozessinstanz oder Aktivität.

Die detaillierte Untersuchung der Prozessinstanz-, Aktivitäts- und Aufgabendaten hat ein fundiertes Verständnis der Struktur der Prozessdaten gegeben. Diese Analyse war essenziell, um die Daten in ein Graphdatenschema überführen zu können.

3.2 Erstellen der Struktur von Entitätsdaten

Nachdem zuvor die Struktur der Prozessdaten detailliert untersucht wurde, widmet sich dieses Kapitel nun der Erstellung der Struktur von Entitätsdaten. Diese Daten sind essenziell für die Darstellung und Analyse von Geschäftsprozessen, da sie wichtige Informationen zur Erfassung der Beziehungen und Interaktionen innerhalb eines Prozesses bereitstellen.
Die effektive Charakterisierung und Strukturierung dieser Daten sind entscheidend für deren Integration in ein Graphdatenschema, was wiederum umfassende und effiziente Auswertungen ermöglicht. Entitätsdaten ermöglichen es, die Beteiligung und Entwicklung bestimmter Objekte innerhalb eines Prozesses nachzuvollziehen und unterstützen langfristig die Vorhersage von Mustern und Trends.
Für diese Arbeit wurde der Begriff *Entitätsdaten* speziell definiert. Die Arbeitsdefinition lautet:

„Entitätsdaten sind Daten, die aus verschiedenen Unternehmensanwendungen stammen können. Sie umfassen alle Objekttypen, die von diesen Anwendungen bereitgestellt werden.“
Diese Daten können unter anderem aus Enterprise-Resource-Planning-, Produktionsplanungs- oder Projektmanagement-Systemen stammen und variieren je nach Anwendungskontext. Beispiele für Entitätsdaten sind vielfältig und können unter anderem Maschinen, Rechnungen, Bestellungen, Produkte und Projekte umfassen.
Ein entscheidendes Element für die Integration in das Graphdatenschema ist, dass jede Entität über ein eindeutiges *id*-Attribut verfügt, welches die Verknüpfung zu Prozessdaten ermöglicht. Diese *id* sollte ein klarer und eindeutiger Identifikator des jeweiligen Objekts sein, wie die Seriennummer einer Maschine oder die Rechnungsnummer einer Rechnung. Es ist unerlässlich, alle relevanten Attribute einer Entität im Graphschema zu berücksichtigen, um eine umfassende und aussagekräftige Datenbasis zu gewährleisten.
Zusätzlich ist es wichtig, dass die Datentypen der Attributwerte von der gewählten Graphdatenbank darstellbar sind. In neo4j beispielsweise sind Datentypen wie `Boolean`, `Integer`, `String`, `Map`, `Date` und viele andere darstellbar [vgl. Neo4j Inc. 2023a].
Diese Anforderung an die Struktur der Entitätsdaten ist grundlegend für die spätere effektive Integration in das Graphdatenschema, was wiederum präzise und tiefgehende Auswertungen ermöglicht.
Eine Entität, die eine Maschine beschreibt, könnte zum Beispiel folgendermaßen aussehen:

- id: m1_2
- Hersteller: Machines and More
- Name: Mikrochipproduktionsmaschine
- Kaufdatum: 12-01-2018
- Kaufpreis: 300.000,00 €
- Aktueller Wert: 230.000,00 €
- Standort: Stockholm
- Größe: 8 qm

Je nach Bedarf könnten hier weitere Attribute hinzugefügt oder weggelassen werden. Die Datenquelle ist immer die jeweilige Unternehmensanwendung. Für jeden einzelnen Objekttyp müssen die Attribute individuell gesetzt werden, weshalb als Strukturanforderung lediglich die *id* und weitere Attribute je nach Kontext gegeben werden können.
Nachdem die Struktur der Entitätsdaten präzisiert wurde, folgt nun der nächste entscheidende Schritt: die Überführung dieser Daten in ein Graphdatenschema, welches im nachfolgenden Kapitel ausführlich behandelt wird. Dabei werden im folgenden Kapitel die spezifischen Transformationsregeln für Knoten, Labels und Beziehungen detailliert erläutert.

3.3 Überführung in ein Graphschema

Dieses Kapitel widmet sich der methodischen Transformation der zuvor diskutierten Prozess- und Entitätsdaten in ein systematisch strukturiertes Graphdatenschema. Zunächst werden die grundlegenden Transformationsregeln detailliert erörtert, die die Basis für die strukturierte Überführung der Daten legen. Im Anschluss daran erfolgt im Abschnitt *„Konstruktion und Analyse des Graphdatenschemas“* eine tiefgreifende Untersuchung der entstandenen strukturellen Verknüpfungen zwischen den Knoten und Beziehungen. Das Hauptziel dieses Kapitels ist die Entwicklung eines wissenschaftlich fundierten, kohärenten und transparenten Datenschemas, das sowohl in seiner theoretischen Konzeption als auch in seiner praktischen Anwendbarkeit überzeugt.

3.3.1 Transformationsregeln

Dieser Abschnitt präsentiert detailliert die spezifischen Regeln für die Bildung von Knoten, die Zuweisung von Labels und die Etablierung von Beziehungen innerhalb des Graphdatenschemas, basierend auf den Prozess- und Entitätsdaten.
Die Transformationsregeln werden durch eine tabellarische Übersicht illustriert. In der Spalte *Transformation & Label/Typ* wird die Konversion in Knoten oder Beziehungen beschrieben und definiert die spezifischen Bezeichnungen für diese Knoten oder Beziehungstypen, wobei optional hinzufügbare Labels in Kursivschrift hervorgehoben sind. Diese können je nach den Anforderungen der späteren Implementierung ergänzt werden. Die Spalte *Ursprungsdatensatz* verweist auf die Bezeichnung des jeweiligen Datensatzes, wie beispielsweise Prozessinstanzdaten aus Elasticsearch. *Attribute* listet die wesentlichen Merkmale auf, die den Knoten oder Beziehungen zugeordnet werden sollten. *Ursprungsdatenfelder* benennt konkret jene Felder des ursprünglichen Datensatzes, die für die Transformation von Bedeutung sind. Eine ausführliche Erläuterung der Transformationsregeln folgt direkt im Anschluss an die tabellarische Darstellung Tabelle 1.

Transformation & Label/Typ	**Ursprungsdatensatz**	**Attribute**	**Ursprungsdatenfelder**
Knoten „Application“	Manuell erstellt	id	Manuell erstellt
		applicationName	Manuell erstellt
Knoten „Deployment“	Prozessinstanz	id	deploymentId
		deploymentId	deploymentId
Fortsetzung auf nächster Seite			

Tabelle 1 – Fortsetzung von vorheriger Seite

Transformation & Label/Typ	**Ursprungsdatensatz**	**Attribute**	**Ursprungsdatenfelder**
Knoten „ProcessDefinition“	Prozessinstanz	id	processDefinitionId
		name	processDefinitionName + processDefinitionVersion
		processDefinitionKey	processDefinitionKey
		processDefinitionId	processDefinitionId
		version	processDefinitionVersion
		deploymentId	deploymentId
Knoten „ProcessInstance“ *„Deleted“*	Prozessinstanz	id	deploymentId + processDefinitionId + id (als *processInstanceId* referenziert)
		deploymentId	deploymentId
		processDefinitionId	processDefinitionId
		processInstanceId	id (als *processInstanceId* referenziert)
		deleteReason	deleteReason
		name	processDefinitionName
		processDefinitionKey	processDefinitionKey
		version	processDefinitionVersion
		duration	durationInMillis
		startTime	startTime
		endTime	endTime
		involvedUsers	involvedUsers

Fortsetzung auf nächster Seite

Tabelle 1 – Fortsetzung von vorheriger Seite

Transformation & Label/Typ	Ursprungsdatensatz	Attribute	Ursprungsdatenfelder
Knoten „Activity“ *„Deleted“* *„SubProcess“* *„SubProcessActivity“* *„ParallelGateway“* *„InclusiveGateway“*	Aktivität	id	processDefinitionId
		processInstanceId	processInstanceId
		processDefinitionVersion	processDefinitionVersion
		activityInstanceId	runtimeActivityInstanceId (als *activityInstanceId* referenziert)
		deleteReason	deleteReason
		pathId	executionId (als *pathId* referenziert)
		name	activityName
		type	activityType
		activityId	activityId
		duration	durationInMillis
		startTime	startTime
		endTime	endTime
		taskId	taskId
		involvedUsers	involvedUsers
	Aufgabe	objectType	Spezifischer Variableneintrag
		objectValue	
		formTime	createTime

Tabelle 1 Übersicht der Transformationsregeln

In den nachfolgenden Erläuterungen werden die in der Tabelle dargestellten Inhalte umfassend erweitert und mit zusätzlichen Informationen angereichert. Wie im Grundlagenkapitel dargelegt, bietet Flowable die Möglichkeit, verschiedene *Flow-Apps* innerhalb einer Instanz zu definieren, was eine effiziente Organisation und Verwaltung distinkter Prozesse unterstützt. Der *Application*-Knoten, der an der Spitze der Hierarchie des Graphdatenschemas steht, wird manuell erzeugt – weitere Details hierzu sind im Abschnitt *Anforderungen an die Prozessmodelle für eine erfolgreiche Datenübertragung* zu finden. Er ist in der Tabelle enthalten, um seine Schlüsselfunktion im Schema und seine Verbindung zu den Flowable-Daten zu unterstreichen. Dieser Knoten ist von wesentlicher Bedeutung für die Zuordnung von Prozessdaten zu den entsprechenden Applikationen und trägt dazu bei, eine kohärente Struktur zwischen den *Deployment*-Knoten herzustellen. Die Bedeutung optionaler Labels wird ebenfalls eingehend untersucht. Wenn ein *ProcessInstance*-Knoten das

Label *Deleted* erhält, deutet dies auf das Vorhandensein eines *deleteReason*-Attributs hin, was für die Gewährleistung der Genauigkeit von Effizienzanalysen bestimmter Prozessdefinitionsversionen ausschlaggebend ist, indem es erlaubt, abgebrochene Prozesse bei späteren Auswertungen bewusst auszuschließen. Verschiedene optionale Labels an Aktivitäten bieten tiefe Einblicke in den Prozessverlauf. So kann das *Deleted*-Label an der zuletzt gestarteten Aktivität einer abgebrochenen Prozessinstanz Rückschlüsse darauf zulassen, an welchen Stellen Prozesse tendenziell unterbrochen werden. *SubProcess*- und *SubProcessActivity*-Labels ermöglichen eine gezielte Analyse von Subprozessen, während *ParallelGateway* und *InclusiveGateway*-Labels dazu dienen, zeitaufwändige oder häufig durchlaufene Pfade zu identifizieren. Die Kennzeichnung von Entitätsknoten mit spezifischen Labels, die den Objekttyp definieren, eröffnet die Möglichkeit, maßgeschneiderte Analysen durchzuführen, wie zum Beispiel die Untersuchung der Beteiligung verschiedener Maschinen an Reparaturprozessen. Nachdem die Transformationsregeln etabliert wurden, konzentriert sich der folgende Abschnitt auf die Konstruktion und Analyse des Graphdatenschemas, wobei die Gründe für die etablierten Transformationsregeln detailliert beleuchtet und das Graphdatenschema auf Basis der dargelegten Erklärungen strukturiert aufgebaut wird.

3.3.2 Konstruktion und Analyse des Graphdatenschemas

Die Stärke von Graphdatenbanken manifestiert sich in der Vernetzung von Knoten, die komplexe Beziehungen zwischen einer Vielzahl von Objekten transparent macht und deren eingehende Analyse ermöglicht. In diesem Kapitel erfolgt eine systematische Untersuchung der Verknüpfungen zwischen Knoten, die aus den vorher definierten Transformationsregeln resultieren.
Dabei werden die Beziehungen zwischen *Application*-, *Deployment*- und *ProcessInstance*-Knoten genau analysiert und die Auswirkungen jeder Verbindung auf das Gesamtschema bewertet. Es wird untersucht, wie die `SEQUENCE_FLOW`-Beziehungen die Prozessabläufe abbilden und welche Einsichten in die Prozesseffizienz sie bieten. Des Weiteren werden die Rollen von Entitätsknoten beleuchtet, insbesondere wie diese die realen Objekte und deren Interaktionen im Prozess repräsentieren, ohne dabei die Komplexität der Datenbank unnötig zu erhöhen. Diese Analyse beinhaltet auch eine Betrachtung der funktionalen und strukturellen Merkmale der Knoten, basierend auf den im vorangegangenen Kapitel erläuterten Attributen und Transformationsregeln. Anhand dessen wird ein Graphdatenschema entwickelt, das die gewonnenen Erkenntnisse praktisch anwendet.

Der *Application*-Knoten fungiert als hierarchischer Ankerpunkt, da dieser alle Prozessmodelle umfasst, die in einer *Flow-App* erstellt wurden. Mehrere *Deployment*-Knoten können an diesen Knoten angebunden sein. Die aus den Prozessinstanzdaten abgeleitete *deploymentId* dient als Basis für den *Deployment*-Knoten und ist entscheidend, um die Weiterentwicklung der Prozessmodelle über verschiedene Deployments hinweg verfolgen zu können. Die Prozessdefinitionsknoten, die unter dem *Deployment*-Knoten angeordnet sind, repräsentieren die diversen Versionen eines Prozessmodells. An jeden Prozessdefinitionsknoten schließen sich die Prozessinstanzen-Knoten an, die einzelne Ausführungen dieser Definition symbolisieren. Die Erfassung einer aussagekräftigen Anzahl von Prozessinstanz-Knoten je Prozessdefinition ermöglicht Effizienzanalysen, beispielsweise in Bezug auf die Durchlaufzeit. Jeder Prozessinstanz-Knoten ist mit einem Startknoten verbunden, während die Aktivitätsknoten durch `SEQUENCE_FLOW`-Beziehungen verknüpft sind, welche die

Fließrichtung und die Verbindungen zwischen den Prozesselementen gemäß BPMN-Notation aufzeigen. Jede Aktivität, die eine Entität einbezieht, wird einem entsprechenden Entitätsknoten zugeordnet. Eine direkte Verknüpfung von Entitätsknoten untereinander wird jedoch vermieden, um die Komplexität in der Analyse zu reduzieren.
Beispielsweise könnte eine Maschine des Herstellers *Machines and More* existieren, die durch ein Attribut repräsentiert wird, welches den Hersteller angibt. Obwohl ein Entitätsknoten für *Machines and More* existieren könnte, wird keine Beziehung zwischen den beiden Knoten erstellt, um den Analyseaufwand zu begrenzen und den Fokus der Arbeit zu bewahren. Die Gestaltung und Implementierung von Entitätsknoten sollte daher stets kontextabhängig und zielgerichtet erfolgen.
Das auf diesen Erkenntnissen basierende Graphdatenschema wird nachfolgend präsentiert (Abbildung 1). Um die Klarheit und Verständlichkeit der Darstellung zu wahren, wurden die spezifischen Attribute in der visuellen Ausarbeitung zunächst ausgelassen. Diese können jedoch bei Bedarf aus der zuvor eingeführten Tabelle entnommen werden, um die Diskussion zu ergänzen und das Verständnis des Graphdatenschemas zu vertiefen. Eine größere Darstellung befindet sich im Anhang (A4).

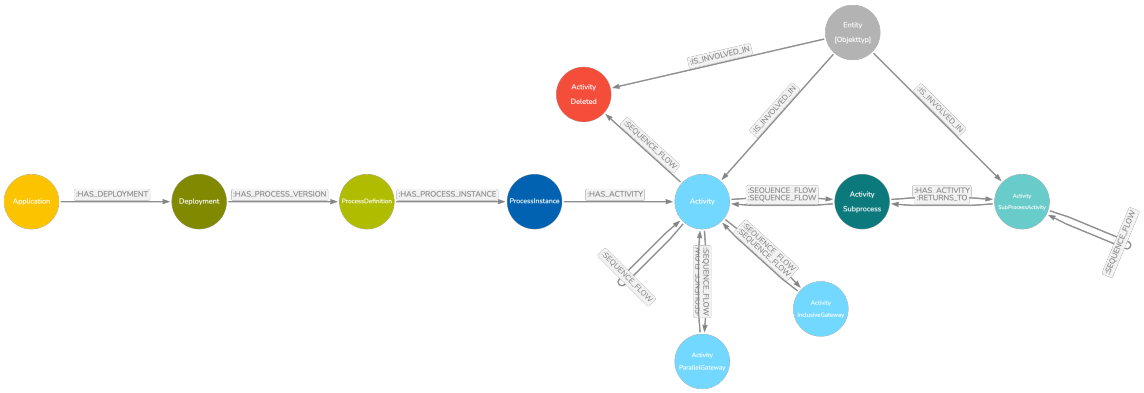


Abbildung 1 Erstelltes Graphschema mit Nutzung der erstellten Transformationsregeln

Das hier präsentierte Graphdatenschema umfasst alle zuvor beschriebenen Knoten und Beziehungen und illustriert den bereits erwähnten hierarchischen Aufbau.
Ein wesentlicher Punkt ist, dass die `SEQUENCE_FLOW`-Beziehungen, die auf Knoten wie *Activity* oder *SubProcessActivity* verweisen, nicht auf denselben physischen Knoten, sondern vielmehr auf Knoten desselben Typs hinweisen. Entsprechend der Prozesslogik werden Aktivitäten in ihrer Bearbeitungsreihenfolge miteinander verknüpft, wobei bei parallelen und inklusiven Gateways die Darstellung der Aktivitäten auch entsprechend parallel laufen kann.
Die Rolle der Labels im Graphschema ist besonders hervorzuheben, da sie spezifische Zustände oder Funktionen der Aktivitätsknoten markieren. Zum Beispiel sind Aktivitätsknoten mit dem Label *Deleted*, *ParallelGateway*, *InclusiveGateway* oder *SubProcess* nicht mit Entitätsknoten verbunden, da sie in BPMN als Kontrollflusselemente und nicht als ausführende Elemente fungieren. Im Gegensatz dazu können *SubProcessActivity*-Knoten normale Entitäten bearbeiten. Ein Aktivitätsknoten mit dem Zusatzlabel *Deleted* verfügt lediglich über eine eingehende Beziehung, da er den Abschluss einer Prozessinstanz kennzeichnet, bei der die Ausführung abgebrochen wurde.
Zusammenfassend lässt sich feststellen, dass durch eine sorgfältige Analyse der Daten ein universell anwendbares Graphdatenschema für Prozessinstanz- und Entitätsdaten entwickelt werden

konnte.
Das nachfolgende Kapitel konzentriert sich auf die praktische Implementierung dieses prototypischen Graphdatenschemas und demonstriert dessen Anwendung in realen Szenarien.

4 Validierung des Ansatzes durch eine prototypische Umsetzung des Graphdatenschemas

Die theoretische Fundierung eines Graphdatenschemas stellt nur die erste Hälfte der Herausforderung dar; seine wirkliche Bewährung erfährt es erst in der praktischen Anwendung. In diesem Kapitel wird die Überführung des konzeptionellen Graphdatenschemas in eine prototypische Implementierung vorgenommen. Es beleuchtet die technische Infrastruktur, die für die Realisierung benötigt wird, und skizziert die Anforderungen an die Prozessmodelle, die eine essenzielle Grundlage für die erfolgreiche Datenmigration bilden.
Im Zentrum des Kapitels steht die zweigeteilte Implementierungsphase: Zunächst wird die Akquisition und Vorbereitung der Daten aus Elasticsearch dargelegt, um sie für den Transfer in die Graphdatenbank aufzubereiten. Daraufhin erfolgt die eigentliche Integration der aufbereiteten Daten in die Graphdatenbank, welche die Grundlage für die nachgelagerte Analyse schafft. Abschließend wird die Effektivität des Ansatzes durch gezielte Auswertungen innerhalb der entstandenen Graphdatenbank evaluiert, um den Erfolg sowohl der Konzeption als auch der praktischen Umsetzung zu verifizieren.

4.1 Aufbau der Referenzinfrastruktur

Die Infrastruktur, die für die prototypische Implementierung im Rahmen des Forschungsprojekts verwendet wurde, ist aufgrund der spezifischen Anforderungen des Projekts und der Bedürfnisse der beteiligten Personen konfiguriert worden. In diesem Abschnitt wird der Fokus auf jene Bestandteile der Infrastruktur gelegt, die für die vorliegende Arbeit von Bedeutung sind. Es wird darauf verzichtet, einen Vergleich mit alternativen Infrastrukturkonfigurationen zu führen, da dies den Rahmen dieser Arbeit überschreiten würde. Zur Veranschaulichung der relevanten Infrastrukturkomponenten folgt eine Skizze (Abbildung 2), die im Anschluss detailliert erläutert wird.

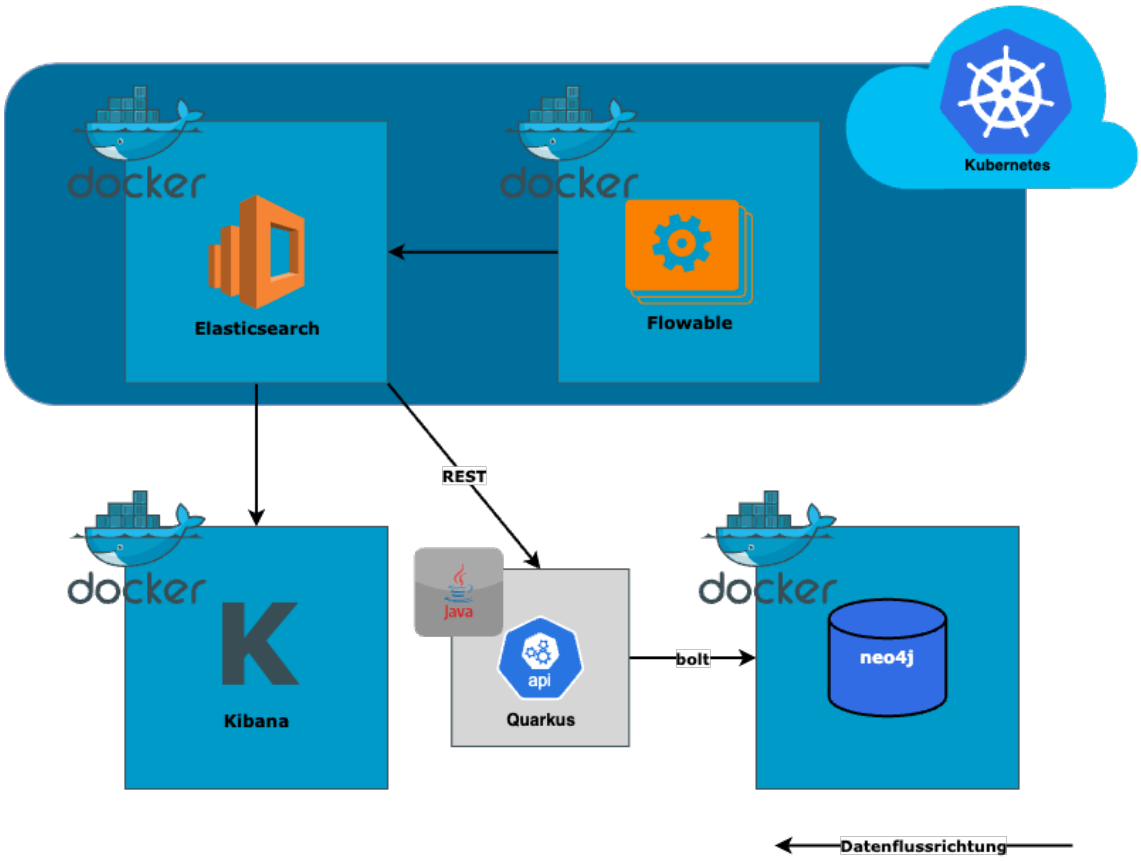


Abbildung 2 Skizze des Aufbaus der Referenzinfrastruktur

Die Kernkomponente der Infrastruktur bildet Kubernetes, das in einer Cloud-Umgebung betrieben wird. Innerhalb von Kubernetes erfolgt die Orchestrierung der Docker-Container für Flowable und Elasticsearch. Diese Container sind für alle Projektteilnehmer zugänglich und bieten die notwendige Flexibilität und Skalierbarkeit für die Projektabläufe. Im Gegensatz zur Cloud-basierten Konfiguration von Kubernetes wird ein Docker-Container für Kibana lokal auf einem spezifizierten Referenzrechner betrieben. Kibana etabliert dabei eine direkte Verbindung zu Elasticsearch, um auf die dort gespeicherten Daten zuzugreifen. Diese Daten werden durch Kibana nicht nur abgerufen, sondern auch verarbeitet und visualisiert, was für die Analyse und Aufbereitung der Forschungsdaten unerlässlich ist. Ergänzend zu Kibana wird ein weiterer Docker-Container lokal betrieben, der für neo4j zuständig ist. Dieser Container ermöglicht die Persistierung von Daten über verschiedene Instanzen hinweg und stellt somit eine wesentliche Komponente für die Datenhaltung und -verarbeitung dar. Eine Schlüsselrolle in der Infrastruktur nimmt die mit dem Java-Framework Quarkus entwickelte REST-API ein. Diese API ist verantwortlich für das Abrufen der Daten aus Elasticsearch. Die Daten werden so aufbereitet, dass sie für den Import in neo4j geeignet sind. Nach der Aufbereitung erfolgt die Übertragung der Daten in die lokale neo4j-Datenbank über den neo4j-Java-Driver. Sämtliche Komponenten – Kibana, die Quarkus REST-API und neo4j – werden auf einem Referenzrechner, dem Linux NUC11TNKi7 mit 32 GiB Arbeitsspeicher und 1 TB Festplattenspeicher, unter localhost betrieben. Diese Konfiguration ermöglicht eine effiziente und lokal kontrollierte Datenverarbeitung und -analyse, die für das Forschungsprojekt essenziell ist. Insgesamt stellt die Infrastruktur eine wohlüberlegte Kombination aus Cloud- und lokal basierten

Komponenten dar, die sowohl Flexibilität als auch Kontrolle in der Datenverarbeitung und -analyse gewährleistet.

4.2 Aufbau und Anforderungen von Prozessmodellen für das Importieren in eine Graphdatenbank

Für diese Arbeit wurden mittels Flowable Design drei Prozessmodelle in BPMN entwickelt, um anhand der aus ihrer Ausführung resultierenden Prozessdaten exemplarische Auswertungen durchführen zu können. Eine Erklärung der einzelnen Prozesselemente erfolgte bereits in Kapitel *3.1 Analyse der Struktur von Prozessdaten*.
Dieses Kapitel widmet sich nun den drei erstellten Prozessmodellen. Der nächste Abschnitt legt den Fokus auf die fachliche Logik dieser Prozessmodelle gefolgt von der Definition der essenziellen Anforderungen, die für die erfolgreiche Integration der Prozessdaten in eine Graphdatenbank notwendig sind.

4.2.1 Beschreibung der Prozessmodelle

In diesem Abschnitt werden die Prozessmodelle detailliert erläutert. Es ist essenziell zu betonen, dass diese Modelle ausschließlich zu Demonstrationszwecken entwickelt wurden und nicht den tatsächlichen Abläufen realer Prozesse entsprechen. Trotz ihres beispielhaften Charakters ist das Verständnis dieser Modelle von zentraler Bedeutung. Dies ermöglicht es, die aus den Prozessen resultierenden Daten korrekt zu interpretieren und darauf basierend fundierte Auswertungen vorzunehmen.
Zu jedem erläuterten Modell (Abbildungen 3, 4, 5) wird eine grafische Darstellung beigefügt. Diese Grafiken dienen lediglich der Orientierung und Veranschaulichung der beschriebenen Prozesse. Für eine detaillierte Betrachtung der Modelle wird auf die vergrößerten Ansichten im Anhang (A5 -A7) verwiesen.

Maschinenwartungsprozess:

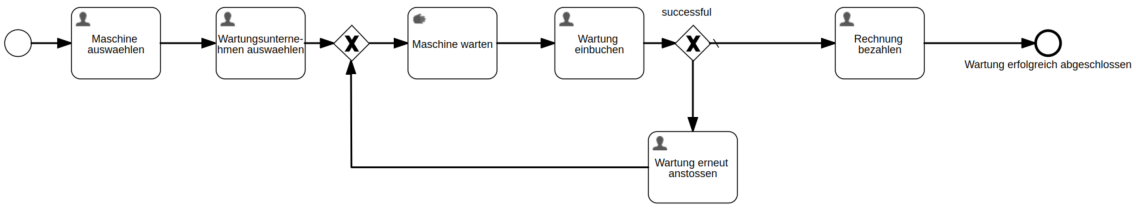


Abbildung 3 Prozessmodell des Maschinenwartungsprozesses

Der Maschinenwartungsprozess, implementiert in Flowable Design, repräsentiert eine sequenzielle Abfolge von Aufgaben, die für die Wartung industrieller Maschinen erforderlich sein könnten. Dieser Prozess dient als exemplarischer Ablauf, um die Interaktion zwischen verschiedenen Aufga-

bentypen und Entscheidungspunkten innerhalb eines Geschäftsprozessmanagementsystems zu illustrieren.
Der Prozessbeginn ist gekennzeichnet durch den User Task *Maschine auswählen*, in dem der Benutzer eine spezifische Maschine für die anstehende Wartung auswählt. Diese Auswahl erfolgt über ein Formular, das eine Liste verfügbarer Maschinen, hinterlegt als Key-Value-Paar, enthält. Darauf folgt der User Task *Wartungsunternehmen auswählen*, der es dem Nutzer ermöglicht, aus vordefinierten Key-Value-Paaren ein Wartungsunternehmen auszuwählen. Nach der Auswahl der Maschine und des Wartungsunternehmens wird der Manual Task *Maschine warten* ausgeführt. Dieser Schritt repräsentiert eine manuelle Tätigkeit, die real von einem Techniker durchgeführt wird, und ist somit nicht automatisiert in der Process Engine abgebildet. Im darauffolgenden User Task *Wartung einbuchen* wird die Durchführung der Wartung dokumentiert. Hierbei wird in einem Formular festgehalten, ob die Wartung erfolgreich war.
Diese Information ist entscheidend für die nachfolgende Entscheidungsfindung am exklusiven Gateway, das auf Basis des Wartungsergebnisses den weiteren Prozessfluss steuert. Das exklusive Gateway *successful* fungiert als Entscheidungspunkt, der je nach Wartungsergebnis unterschiedliche Prozesspfade einleitet. Ein positives Wartungsergebnis leitet über zum User Task *Rechnung bezahlen*, in dem die Rechnungsnummer der Wartung erfasst wird. Ein negatives Ergebnis führt zum User Task *Wartung erneut anstoßen*, der eine Wiederholung des Wartungsprozesses initiiert. Ein zweites exklusives Gateway steuert die Rückführung des Prozesses nach einer erneuten Wartung zurück zum Manual Task *Maschine warten*, was die Möglichkeit einer iterativen Schleife im Falle einer notwendigen Wiederholung der Wartung bietet. Der Prozess mündet in das Endereignis *Wartung erfolgreich abgeschlossen*, das den Abschluss der Wartungsarbeiten und das erfolgreiche Ende des Prozessdurchlaufs symbolisiert.

Standortwechselprozess:

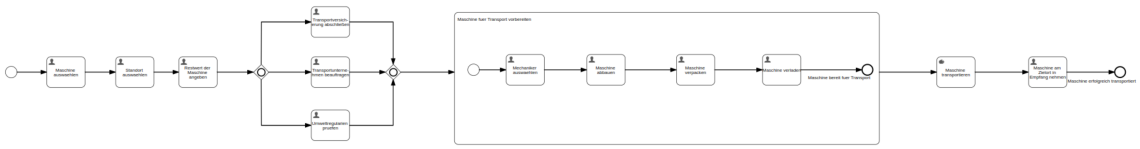


Abbildung 4 Prozessmodell des Standortwechselprozesses

Der Standortwechselprozess, implementiert in Flowable Design, ist ein Beispielprozess, der die logistischen und administrativen Schritte bei der Verlegung einer Maschine von einem Standort zum anderen abbilden soll. Dieser Prozess dient als Muster zur Demonstration der Koordination verschiedener Aktivitäten und Entscheidungspunkte innerhalb eines Geschäftsprozessmanagementsystems.
Der Prozess startet mit dem User Task *Maschine auswählen*, in dem der Anwender eine Maschine für den Standortwechsel aus einer Liste auswählt, die in einem Formular als Key-Value-Paar dargestellt wird. Es folgt der User Task *Standort auswählen*, in dem der neue Standort für die Maschine festgelegt wird, um die logistische Planung zu initiieren. Anschließend wird im User Task

Restwert der Maschine angeben der finanzielle Wert der Maschine ermittelt. Dieser Schritt ist entscheidend, da er die nachfolgenden Entscheidungen im Prozess beeinflusst, wie zum Beispiel die Notwendigkeit einer Transportversicherung.
Das inklusive Gateway leitet den Prozess dann auf unterschiedliche Pfade, abhängig vom Restwert der Maschine. Bei einem höheren Restwert wird im User Task *Transportversicherung abschließen* eine Versicherung für den Transport abgeschlossen. Liegt der Wert darunter, wird direkt im User Task *Transportunternehmen beauftragen* ein Transportunternehmen ausgewählt. Zudem wird im User Task *Umweltregularien prüfen* untersucht, ob spezielle Umweltvorschriften am neuen Standort zu beachten sind, insbesondere wenn dieser in einer umweltsensitiven Region liegt.
Nachdem die initialen Planungen abgeschlossen sind, beginnt der Subprozess *Maschine für Transport vorbereiten*. Dieser beinhaltet mehrere Schritte, beginnend mit der Auswahl des Mechanikers über ein Formular, bei dem mögliche Mechaniker hinterlegt sind, für den Abbau der Maschine, gefolgt vom Abbau, Verpacken und schließlich Verladen der Maschine. Der manuelle Task *Maschine transportieren* repräsentiert den physischen Transport der Maschine zum neuen Standort. Dieser Schritt variiert je nach Entfernung und Komplexität und wird daher außerhalb der digitalen Prozesssteuerung abgebildet. Der finale Schritt ist der User Task *Maschine am Zielort in Empfang nehmen*, der den erfolgreichen Abschluss des Transports und die Bereitschaft der Maschine am neuen Standort dokumentiert. Der Prozess endet mit dem Endereignis *Maschine erfolgreich transportiert*.
Es folgt eine tiefergehende Betrachtung der strukturellen Aspekte des Prozesses. Besonders hervorzuheben ist hierbei die Rolle von Subprozessen und Inklusiven Gateways innerhalb des Modells. Subprozesse bieten in der Praxis die Möglichkeit, standardisierte Abläufe wiederholt in verschiedenen Prozessen einzusetzen, was ihre häufige Verwendung begründet.
Inklusive Gateways sind ebenfalls ein zentraler Bestandteil in der Geschäftsprozessmodellierung, da sie parallele Aktivitäten ermöglichen, die von vorangegangenen Entscheidungen abhängen. Die präzise Modellierung und Darstellung dieser Elemente sind für diese Arbeit von entscheidender Bedeutung, da sie es ermöglicht, diese speziellen Anwendungsfälle bei der Datenimportierung in eine Graphdatenbank angemessen zu berücksichtigen.

Maschinenreparaturprozess:

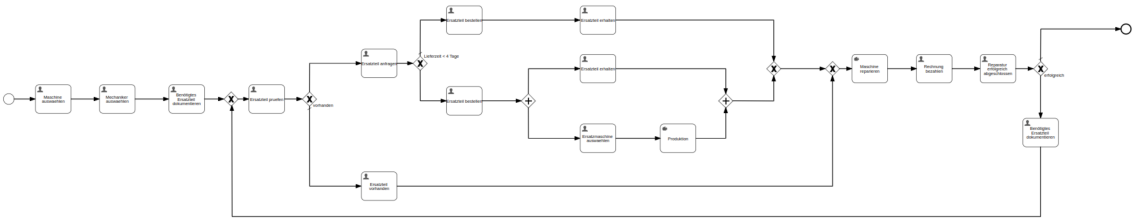


Abbildung 5 Prozessmodell des Maschinenreparaturprozesses

Der Maschinenreparaturprozess, realisiert durch Flowable Design, illustriert die erforderlichen Schritte zur Instandsetzung einer industriellen Maschine. Dieses Modell dient zur Demonstration, wie verschiedene Aufgaben und Entscheidungspunkte effektiv innerhalb eines Geschäftsprozessmanagementsystems koordiniert werden könnten.

Der Prozess beginnt mit dem User Task *Maschine auswählen*, bei dem der Benutzer eine defekte Maschine für die Reparatur auswählt. Dies erfolgt über ein Formular, das eine Liste der verfügbaren Maschinen enthält. Anschließend wird im User Task *Mechaniker auswählen* ein geeigneter Techniker für die Reparatur bestimmt, ebenfalls mittels eines Formulars.
Im nächsten Schritt, dem User Task *Ersatzteil prüfen*, wird die Verfügbarkeit der benötigten Ersatzteile überprüft. Ein exklusives Gateway bestimmt den weiteren Verlauf: Bei Nichtverfügbarkeit des Ersatzteils wird im User Task *Ersatzteil anfragen* die Lieferzeit ermittelt. Ein weiteres exklusives Gateway entscheidet daraufhin auf Basis der Lieferzeit über die nächsten Schritte. Unabhängig vom Ergebnis wird das Ersatzteil im User Task *Ersatzteil bestellen* geordert. Bei einer Lieferzeit von vier Tagen oder mehr wird ein paralleles Gateway aktiviert. Dieses paralleles Gateway ist von zentraler Bedeutung, da es die parallele Ausführung zweier Prozesspfade ermöglicht: einerseits die Auswahl und den Einsatz einer Ersatzmaschine zur Aufrechterhaltung der Produktion, andererseits das Warten auf das bestellte Ersatzteil. Nach Eintreffen des Ersatzteils wird der Prozess mit dem User Task *Ersatzteil erhalten* fortgeführt, um die Eignung des Teils zu bestätigen.
Bei positiver Prüfung erfolgt der Manual Task *Maschine reparieren*. Im User Task *Reparatur erfolgreich abgeschlossen* wird das Ergebnis der Reparatur dokumentiert. Ein exklusives Gateway bewertet, ob die Reparatur erfolgreich war. Bei Misserfolg wird der Prozess erneut angestoßen, beginnend mit dem User Task *Benötigtes Ersatzteil dokumentieren*. Dieser iterative Prozess wiederholt sich, bis die Reparatur erfolgreich abgeschlossen ist.
Die präzise Abbildung dieses Vorgangs ist entscheidend für die Datenimportierung in eine Graphdatenbank, da sie die Wiederholung von Prozessschritten und deren Auswirkungen auf den Gesamtprozess darstellt. Die Modellierung solcher Wiederholungszyklen ist wesentlich für eine realistische Darstellung komplexer Geschäftsprozesse und ermöglicht umfassende Analysen sowie Optimierungen. Das Modell hebt zudem die Bedeutung von parallelen Gateways in der Geschäftsprozessmodellierung hervor, um effiziente Lösungen für simultane Aufgaben in komplexen Prozessen zu ermöglichen. Die adäquate Darstellung und Modellierung dieser parallelen Abläufe sind essenziell für den Datenimport in eine Graphdatenbank, da sie entscheidende Informationen für die Analyse und Optimierung von Geschäftsprozessen bereitstellt.

Die sorgfältige Analyse der Prozessmodelle hat die Komplexität und Vielschichtigkeit in der Gestaltung von Geschäftsprozessen aufgezeigt. Jedes der Modelle bietet Einblicke in spezifische Herausforderungen und Lösungsstrategien, die für das Feld der Geschäftsprozessmodellierung wesentlich sind. Diese Erkenntnisse liefern eine solide Grundlage für die Identifizierung der essenziellen Anforderungen, die für die Integration von Prozessdaten in eine Graphdatenbank entscheidend sind. Im nächsten Abschnitt liegt der Fokus auf diesen unverzichtbaren Anforderungen, die als zentrale Elemente für die effiziente Handhabung und Analyse von Prozessdaten in einer Graphdatenbank fungieren.

4.2.2 Anforderungen an die Prozessmodelle für den Datenimport

Die effektive Integration von Daten in eine Graphdatenbank setzt eine sorgfältige Konfiguration der Prozessmodelle voraus, basierend auf spezifischen Anforderungen, die sowohl für die technische

Implementierung als auch für das Verständnis der Interaktionen zwischen Prozessmanagement und Datenbankarchitektur entscheidend sind. Nachfolgend werden diese Anforderungen detailliert erläutert, um aufzuzeigen, welche Modifikationen in den Prozessmodellen für eine erfolgreiche Datenintegration erforderlich sind.

Anpassen des *processDefinitionKeys*

Um die einzelnen Prozesse eindeutig einer Applikation zuordnen zu können, muss der *processDefinitionKey* entsprechend angepasst werden. Normalerweise steht er lediglich für den Namen des Prozesses, wenn man jedoch pro Applikation ein Präfix hinzufügt, wird es möglich, die Prozessmodelle innerhalb der Flow-Apps korrekt zuordnen zu können. Empfehlenswert wäre folgende Konvention:

`[Name der Applikation].[Name des Prozesses]`

Dabei dürfen weder Leerzeichen, noch Sonderzeichen enthalten sein.

Verknüpfung von Aktivitäten und Sequence Flows

Die Datenübertragung in eine Graphdatenbank erfolgt durch die Verknüpfung der Aktivitäten in ihrer Reihenfolge, wobei die Sequence Flows als verbindende Elemente fungieren. Die Attribute startTime und endTime jeder Aktivität sind hierbei von großer Bedeutung. Insbesondere Manual Tasks und Gateways werden als Durchlaufaktivitäten mit minimaler Dauer behandelt, was zu identischen Zeitstempeln führen kann. Um diese Problematik zu bewältigen, wurden gesonderte Abfragen im Import-Code erstellt. Jedoch wird empfohlen, für Aktivitäten, die außerhalb der Process-Engine ablaufen, ein externes Signal zur Anzeige des Aktivitätsendes einzusetzen.

Anpassung der activityId bei Gateways

Für die eindeutige Zuordnung von zusammengehörigen Gateways in der Graphdatenbank ist eine spezielle Anpassung der activityId erforderlich. Die übliche Zusammensetzung der activityId aus dem Aktivitätstyp und einer fortlaufenden Nummer reicht hierfür nicht aus. Stattdessen sollte die activityId folgendermaßen strukturiert werden:

Öffnendes Gateway: `opening[Gatewaytyp][Nummer X].`

Schließendes Gateway: `closing[Gatewaytyp][Nummer X].`

Diese Strukturierung ermöglicht eine klare Unterscheidung zwischen öffnenden und schließenden Gateways und berücksichtigt komplexe Szenarien mit mehreren Gateway-Paaren in einer Prozessinstanz.

Gestaltung der Formulare

Bei der Gestaltung von Formularen ist die präzise Erfassung der taskId jeder Aktivität, die ein Formular verwendet, von großer Bedeutung. Dies ist wichtig, da jede Aufgabe über den formKey einem spezifischen Formular zugeordnet ist, aber der Datensatz einer Aufgabe alle Variablen enthält, die während einer Prozessinstanz generiert werden. Diese Herausforderung verlangt eine spezifische Anpassung des Formular-Outputs.

Um die Notwendigkeit der Anpassung des Formular-Outputs zu veranschaulichen, folgt zunächst eine Gegenüberstellung der Variableneinträge einer Aufgabe. Zuerst sind die standardmäßig

verfügbaren Einträge und folgend die angepassten Einträge dargestellt.

Standard JSON-Eintrag:


```
"id": "VAR-77fd04f3-a8ed-11ee-95bb-4e347637bf48",
"name": "form_mechanikerFuerReparaturAuswaehlenForm_outcome",
"type": "string",
"scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
"scopeType": "bpmn",
"scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
"scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur",
"rawValue": "COMPLETED",
"textValue": "COMPLETED",
"textValueKeyword": "COMPLETED"
```


Angepasster JSON-Eintrag:


```
"id": "VAR-77fd04f3-a8ed-11ee-95bb-4e347637bf48",
"name": "form_mechanikerFuerReparaturAuswaehlenForm_outcome",
"type": "string",
"scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
"scopeType": "bpmn",
"scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
"scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur",
"rawValue": "mechaniker: mechaniker4;
             2024-01-01T21:33:55.164Z;
             TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48",
"textValue": "mechaniker: mechaniker4;
              2024-01-01T21:33:55.164Z;
              TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48",
"textValueKeyword": "mechaniker: mechaniker4;
                     2024-01-01T21:33:55.164Z;
                     TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48"
```


Aus dieser Gegenüberstellung wird ersichtlich, dass die in den Formularen erfassten Daten in der Standardkonfiguration weder mit dem zugehörigen Formular noch mit der zugehörigen Aktivität klar verbunden sind. Der Eintrag des Formulars, der über den formKey des Tasks einer Aktivität eindeutig zugeordnet werden kann, enthält nicht die ausgewählten Daten.
Die Anpassungen des Formular-Outputs beinhalten:

- **Integration der taskId:**
  Die Erfassung der taskId im Formular-Output ist essenziell für die Identifikation der spezifischen Task-Instanz. Dadurch wird gewährleistet, dass die gesammelten Daten genau der zugehörigen Aktivität zugeordnet werden können.
- **Erfassung der startTime der Task:**
  Die Integration der startTime definiert den zeitlichen Kontext der Datenerfassung.
- **Hinterlegen der erfassten Daten:**
  Neben der taskId selbst ist es wichtig, die erfassten Daten in der Formularausgabe zu hinterlegen, um sie einer Aktivität zuordnen zu können.

Diese Anpassungen sorgen dafür, dass die über Formulare gesammelten Daten exakt den entsprechenden Aktivitäten im Prozessmodell zugewiesen werden können.

Diese beschriebenen Anpassungen erfordern zwar einen gewissen Mehraufwand bei der Prozessdefinition, bieten jedoch signifikante Vorteile für die korrekte Implementierung und Verknüpfung von Prozess- und Entitätsdaten. Durch die präzise Konfiguration der Prozessmodelle wird eine effektive und fehlerfreie Datenintegration in die Graphdatenbank ermöglicht, was essenziell für umfassende Analyse- und Optimierungsmöglichkeiten ist.

4.3 Datenübertragung in die Graphdatenbank

Nachdem in den vorherigen Abschnitten von Kapitel 4 die Grundlagen und Vorbereitungen für den Datenimport in eine Graphdatenbank behandelt wurden, sowohl aus theoretischer als auch aus praktischer Perspektive, widmet sich das nachfolgende Kapitel der konkreten Realisierung dieser Vorarbeiten. Dieser Abschnitt stellt den Schritt dar, in dem die vorbereitenden Maßnahmen in tatsächliche Anwendungen überführt werden, und markiert somit einen wesentlichen Übergang von der Planungs- zur Ausführungsphase innerhalb des Projekts. Es bildet einen kritischen Punkt in der Arbeit, an dem die theoretischen Überlegungen und vorbereitenden Aktivitäten in praktische Ergebnisse münden.

4.3.1 Vorbereitung der Daten aus Elasticsearch für den Import

Im Kontext des Kapitels 4.3.1 richtet sich der Fokus auf die essenzielle Phase der Datenvorbereitung, die eine grundlegende Voraussetzung für den erfolgreichen Import in Neo4j darstellt. Dieser Abschnitt widmet sich detailliert der analytischen Aufbereitung der aus Elasticsearch bezogenen Daten, um sie für die Integration in eine Graphdatenbank zu qualifizieren. Diese Vorbereitung ist unerlässlich, um einen nahtlosen und effektiven Datenfluss zwischen Elasticsearch und Neo4j zu gewährleisten. Der Abschnitt erläutert systematisch die angewandten Methoden und technischen Prozesse, die zur Bewältigung dieser Herausforderung erforderlich sind.

Grundlegende Konfiguration und Initialisierung
Die Datenvorbereitung beginnt mit dem Aufbau einer Verbindung zu Elasticsearch über eine REST-API, implementiert in einem Quarkus-Service. Der `RestHighLevelClient` [vgl. Elasticsearch B.V. 2024b] ist hierbei das zentrale Element, das die Kommunikation mit Elasticsearch steuert. Für eine effiziente Datenverarbeitung werden die Einstellungen für Host und Port sowohl für Elasticsearch als auch für Neo4j definiert. Hierbei werden im Entwicklungskontext lokale Konfigurationen verwendet.

Verfeinerter Polling-Mechanismus und Datenerfassung
Der Polling-Mechanismus, implementiert durch eine `@Scheduled-Annotation`, initiiert alle zehn Sekunden eine Abfrage neuer oder aktualisierter Daten. Diese Daten, die in Form von JSON vorliegen, werden mithilfe von Java in Map-Strukturen umgewandelt. Diese Umwandlung ist kritisch,

um die Daten in ein verarbeitungsfähiges Format zu überführen und somit die Grundlage für die weitere Datenanalyse zu schaffen.

Selektive Datenerfassung und Vorfilterung
Um die Effizienz zu maximieren, werden gezielt nur relevante Datensätze aus Elasticsearch abgerufen. Dies geschieht durch eine sorgfältige Filterung von Indizes, die mit process-instances, activities oder tasks beginnen. Zusätzlich wird durch eine Sortierung nach *startTime* bzw. *createTime* sichergestellt, dass nur die Daten abgerufen werden, die nach dem zuletzt verarbeiteten Zeitpunkt erstellt wurden.

Zeitliche Sortierung und Verarbeitung
Eine wichtige Phase ist die zeitliche Sortierung der Daten. Diese Sortierung nach Zeitstempeln ist notwendig, um eine chronologische Konsistenz der Daten zu gewährleisten. Die korrekte Erfassung der Zeitstempel der verschiedenen Indextypen ist dabei von entscheidender Bedeutung.

Detailreiche strukturierte Aufbereitung für den Import
Die strukturierte Aufbereitung der Daten für den Import in Neo4j erfordert eine sorgfältige Planung. Die Daten werden zunächst in einer Map nach Prozessinstanzen, Aktivitäten und Aufgaben strukturiert. Diese Strukturierung ist entscheidend, um die Daten in einer logischen Reihenfolge zu ordnen, die die Integrität der Beziehungen in Neo4j sicherstellt. Die Sortierung erfolgt in einer Weise, dass zuerst die Prozessinstanzen, dann Aktivitäten (die keine Sequence Flows sind), gefolgt von Sequence Flows und schließlich Aufgaben verarbeitet werden. Diese Sortierung stellt sicher, dass beim Import in Neo4j zuerst die notwendigen Knoten bereits erstellt werden konnten, bevor Beziehungen hergestellt oder Attribute hinzugefügt werden.

Die gründliche Vorbereitung und strukturierte Aufbereitung der Daten bildet das Fundament für einen erfolgreichen Import in Neo4j. Die in diesem Kapitel erörterten Methoden und Verfahren gewährleisten, dass die Daten nicht nur effizient verarbeitet, sondern auch in einer Weise aufbereitet werden, die ihre strukturelle und inhaltliche Integrität wahrt. Im nächsten Kapitel wird der Fokus auf den eigentlichen Importprozess gelegt, mit Schwerpunkt auf der Anwendung von Cypher-Code zur Integration der sorgfältig vorbereiteten Daten in die Neo4j-Datenbank.

4.3.2 Erstellung und Verknüpfung der Prozess- und Entitätsdaten

In Kapitel 4.3.2 wird die praktische Umsetzung der Datenintegration in die Graphdatenbank Neo4j detailliert thematisiert. Im Mittelpunkt steht die methodische Implementierung des Datenimports, insbesondere die Erstellung und Verknüpfung der Prozess- und Entitätsdaten unter Einsatz von Cypher-Code. Dieser Abschnitt beleuchtet die Entwicklung spezifischer Skripte und Befehle, die es ermöglichen, die sorgfältig vorbereiteten Daten effizient in Neo4j einzubetten und zu verknüpfen. Die hier beschriebene Vorgehensweise ist essenziell für die erfolgreiche Implementierung der Datenstrukturen innerhalb der Graphdatenbank und stellt somit einen kritischen Aspekt des Datenimports dar.

Verbindungsaufbau
Die Übertragung der sorgfältig vorbereiteten Daten in die Neo4j-Graphdatenbank beginnt mit dem Aufbau einer Verbindung mittels des Neo4j-Java-Drivers. Diese Schnittstelle ermöglicht es, die Daten in die lokale Graphdatenbank einzuspeisen. Eine zentrale Abfrage differenziert hierbei die verschiedenen Indextypen, um entsprechende Verarbeitungsschritte zu initiieren.

Spezifische Datenaufbereitung
Bei Datensätzen vom Typ Aufgabe erfolgt eine gezielte Anpassung, bei der ausschließlich jene Attribute übertragen werden, die für die Aktivitätsknoten relevant sind. Diese Anpassung fokussiert auf die überarbeiteten Variablen der Formularausgabe und trägt damit zur Datenkonsistenz bei.

Methodik der Knoten- und Beziehungserstellung
In Neo4j erfolgt die Knoten- und Beziehungserstellung durch Cypher-Befehle. Während der Befehl `CREATE` neue Knoten ohne Rücksicht auf bereits vorhandene erstellt, ermöglicht `MERGE` eine intelligente Erstellung und Aktualisierung: Existiert ein Knoten noch nicht, wird er angelegt und mittels `ON CREATE` die entsprechenden Attribute erstellt. Bei Vorhandensein werden bestehende Attribute aktualisiert oder erweitert (`ON MATCH`). Diese Methode verhindert effektiv die Duplikation von Knoten und trägt zur Effizienz des Datenimports bei. Die Identifikation der Knoten erfolgt über eindeutige IDs, die aus Kapitel 3 entnommen werden können, um eine präzise Zuordnung zu gewährleisten.

Behandlung spezieller Knotentypen und Beziehungen
Beim Import von Prozessinstanz-Daten werden verschiedene Knotentypen wie *Application*, *Deployment*, *ProcessDefinition* und *ProcessInstance* erstellt, sofern sie nicht bereits existieren. Anschließend werden die notwendigen Beziehungen zwischen diesen Knoten etabliert. Sollte ein *ProcessInstance*-Knoten das Attribut *deleteReason* enthalten, wird diesem zusätzlich das Label *Deleted* zugewiesen, um gelöschte oder abgebrochene Prozesse zu kennzeichnen.
Für Aktivitätseinträge wird ein analoges Vorgehen angewendet. Hier wird zunächst ein *Activity*-Knoten erstellt und daraufhin überprüft, ob im Kontext der aktuellen Prozessinstanz Knoten für *parallelGateway* oder *inclusiveGateway* notwendig sind. Diese Gateways spielen eine entscheidende Rolle bei der Steuerung des Prozessflusses. Da Prozess-Engines für jeden endenden Pfad ein separates schließendes Gateway erstellen, ist es notwendig, Paare von Gateways zu identifizieren und redundante Gateways zu entfernen. Diese Maßnahme gewährleistet, dass Pfade korrekt in einem schließenden Gateway zusammengeführt werden, entsprechend der Logik des Prozessmodells. Im Folgenden ist ein Beispielcode, der in der Umsetzung genutzt wurde, um *parallelGateways* zu bearbeiten.

```
1  MATCH (activity:Activity)
2  WHERE activity.id =
3      coalesce($processDefinitionId+$processInstanceId+$pathId+$activityInstanceId)
4  WITH activity
5  MATCH (openPg:Activity
6      {type: 'parallelGateway', processInstanceId: $processInstanceId})
7  WHERE openPg.activityId STARTS WITH 'openingParallelGateway'
8  WITH activity, COLLECT(openPg) AS openPGateways
9  UNWIND openPGateways AS openPg
```

```
10      WITH activity, openPg, [nextOpenPg IN openPGateways
11      WHERE right(openPg.activityId, 1) = right(nextOpenPg.activityId, 1)
12      AND nextOpenPg.endTime > openPg.endTime][0] AS nextOpenPg
13      MATCH (closePg:Activity
14          {type: 'parallelGateway', processInstanceId: openPg.processInstanceId})
15      WHERE closePg.activityId STARTS WITH 'closingParallelGateway'
16      AND right(closePg.activityId, 1) = right(openPg.activityId, 1)
17      AND closePg.startTime > openPg.endTime
18      AND (nextOpenPg IS NULL OR closePg.startTime < nextOpenPg.endTime)
19      WITH activity, openPg, closePg
20      ORDER BY closePg.endTime DESC
21      WITH activity, openPg, COLLECT(closePg) AS closePGateways
22      WITH activity, openPg, head(closePGateways) AS newestClosePg,
23          tail(closePGateways) AS olderClosePGateways
24      SET openPg:ParallelGateway, newestClosePg:ParallelGateway
25      WITH olderClosePGateways
26      UNWIND olderClosePGateways AS oldClosePg
27      DETACH DELETE oldClosePg
```

Erstellung der Beziehungen mit Sequence Flows Ein wesentlicher Schritt ist die Erstellung von Beziehungen zwischen Aktivitäten über Sequence Flows. Diese werden durch die vorher festgelegte Reihenfolge der Datenübertragung ermöglicht. Aktivitäten werden nach ihrer *startTime* und *endTime* sortiert, um die korrekte Zuordnung der Sequence Flows zu den entsprechenden Knoten sicherzustellen.
Bei der Zuordnung der Sequence Flows zu den Aktivitäten müssen zwei wichtige Sonderfälle beachtet werden:

- **Mehrere Aktivitäten mit identischer *endTime*:** Hier wird der Knoten mit der späteren *startTime* als Ausgangspunkt der Beziehung (a1) gewählt. Eine zusätzliche Überprüfung stellt sicher, dass der richtige Knoten ausgewählt wurde. Falls erforderlich, wird die Auswahl angepasst, um fehlerhafte Beziehungen zu vermeiden.

```
1       CALL apoc.do.when(
2           (NOT EXISTS {()-[:SEQUENCE_FLOW]->(a1)})
3           AND a1.startTime = $startTime AND NOT a1.type = 'startEvent',
4           'WITH a1
5           MATCH (before:Activity)
6               WHERE before.endTime <= a1.startTime
7                   AND before.processInstanceId = $processInstanceId
8                   AND a1.processInstanceId = $processInstanceId
9                   AND (before.pathId = $pathId OR a1.pathId = $pathId)
10                  AND before<>a1
11          WITH before, a1
12          ORDER BY
13              before.endTime DESC, a1.endTime ASC, a1.endTime DESC
14          LIMIT 1
15          RETURN {a1: before, a2: a1} AS result',
16          'RETURN {a1: a1, a2: a2} AS result',
17          {a1: a1,
18          a2: a2,
19          pathId: $pathId,
```

```
        processInstanceId: $processInstanceId,
        startTime: $startTime}
    ) YIELD value

    WITH value.result.a1 AS a1, value.result.a2 AS a2, sub

    CALL apoc.do.when(
        NOT EXISTS {(a1)-[seq:SEQUENCE_FLOW]->(a2)
                    WHERE
                        seq.startTime = $startTime AND
                        seq.endTime = $endTime AND
                        seq.pathId = $pathId
                },
        'RETURN {a1: a1, a2: a2} AS result',
        'MATCH (a3:Activity), (a4:Activity)
         WHERE a3.endTime = $startTime AND a3.startTime <= $startTime
           AND a4.startTime > $startTime
           AND a3.processInstanceId = $processInstanceId
           AND a4.processInstanceId = $processInstanceId
           AND (a3.pathId = $pathId OR a4.pathId = $pathId)
           AND a3 <> a4
           AND a3.endTime <= a4.startTime
           AND (sub IS NULL
            OR NOT (a2.startTime >= sub.startTime
                    AND a2.endTime < sub.endTime
                    AND a1.startTime < sub.startTime)
            )
         WITH a3, a4
         ORDER BY a3.endTime DESC, a4.endTime ASC, a4.startTime ASC
         LIMIT 1
         RETURN {a1: a3, a2: a4} AS result',
       {a1: a1,
        a2: a2,
        sub: sub,
        startTime: $startTime,
        endTime: $endTime,
        processInstanceId: $processInstanceId,
        pathId: $pathId}
    ) YIELD value
```

- **'Activity-SubProcess'-Knoten:** Diese Knoten repräsentieren Subprozesse und sind nicht direkt in den normalen Prozessablauf eingebunden, sondern nur mit dem *SubProcess*-Knoten verbunden, der die Aktivitäten vor und nach dem Subprozess miteinander verknüpft.

Abschluss des Datenimports Nach der Verknüpfung aller Aktivitäten werden aufgabenspezifische Attribute zu den zugehörigen Aktivitätsknoten hinzugefügt. Diese Attribute, wie *objectType*, *objectValue* und *taskStartTime*, stammen aus den Aufgabendaten und ermöglichen die Verknüpfung mit den Entitätsdaten.

Erstellung der Entitätsdaten Für die Generierung der Entitätsdaten wurden spezifische Comma-Separated Values (CSV)-Dateien für jeden Entitätstyp angefertigt und importiert. Diese Arbeit verzichtet auf eine detaillierte Darstellung des Importprozesses dieser Entitäten. Der Grund dafür

liegt in der Annahme, dass in einer praktischen Anwendungsumgebung die erforderlichen Daten direkt aus den Unternehmensanwendungen oder aus einer konsolidierten Datenbank bezogen werden würden. Somit stellt der in dieser Arbeit implementierte Importvorgang keine relevante Praxisanwendung dar und wird nicht weiter thematisiert.

Verknüpfung der Entitäts- und Aktivitätsknoten Die Verwendung des Labels *Entity* und eines einheitlichen *id*-Attributs für alle Entitäten erleichtert diese Zuordnung erheblich. Durch eine effiziente Cypher-Abfrage werden die relevanten Entitäten identifiziert und mit den Aktivitätsknoten verknüpft. Dieser letzte Schritt vervollständigt den Prozess der Datenintegration und ermöglicht eine umfassende Darstellung der Prozessdaten in der Graphdatenbank, entsprechend dem in Kapitel 3 definierten Graphschema.

```
1    MATCH (a:Activity), (e:Entity)
2    WHERE a.objectValue = e.id
3    MERGE (a)<-[r:IS_INVOLVED_IN]-(e)
4    SET r.created = datetime()
```

4.4 Exemplarische Auswertungen auf der entstandenen Datenbank

Im folgenden Kapitel 4.4 werden beispielhafte Auswertungen auf einer neu erstellten Graphdatenbank vorgestellt. Diese Auswertungen illustrieren eindrücklich, dass durch die Integration von Prozess- und Entitätsdaten nicht nur effiziente, sondern auch tiefgreifende Analysen ermöglicht werden, die zuvor entweder gar nicht möglich waren oder nur mit erheblichem manuellem Aufwand realisiert werden konnten. Insbesondere übertreffen diese Analysen die Grenzen der traditionellen Datenanalyse, wie sie bei der isolierten Betrachtung von Daten aus einzelnen Unternehmensanwendungen oder separaten Prozessinstanzen üblich ist. Der Schwerpunkt dieses Kapitels liegt auf der Demonstration, wie die strukturierte und vernetzte Datenhaltung in einer Graphdatenbank die Extraktion von wertvollen Einsichten nicht nur vereinfacht, sondern auch in bisher unerreichte Bereiche erweitert.

Zunächst wird die visuelle Repräsentation einer Prozessinstanz in der Graphdatenbank vorgestellt. Diese Visualisierung umfasst verschiedene Aspekte des Maschinenreparaturprozesses, von der Applikation bis hin zu den damit verbundenen Aktivitäten und Entitäten. Diese Darstellung (Abbildung 6) ist essenziell, um die Komplexität und die Verknüpfungen innerhalb der Prozessinstanz zu verstehen. Eine vergrößerte Darstellung findet sich im Anhang (A8).

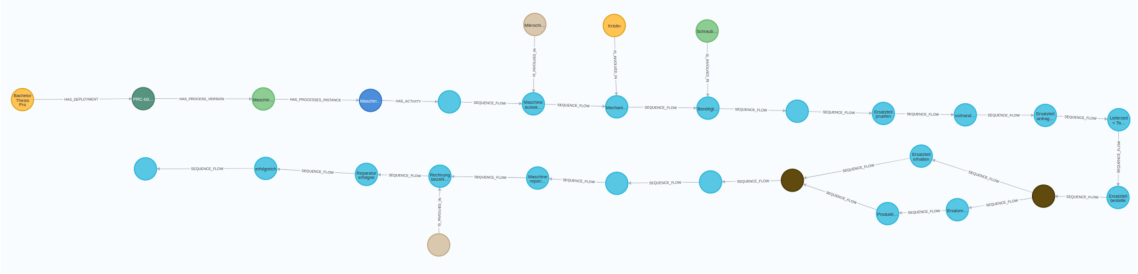


Abbildung 6 Ausgabe einer Prozessinstanz des Maschinenreparaturprozesses mit allen zugehörigen Knoten und Beziehungen

Anschließend erfolgen mehrere spezifische Auswertungen mittels Cypher-Abfragen. Die erste Abfrage zielt darauf ab, die durchschnittliche Dauer von Subprozessen pro Prozessdefinition zu ermitteln. Dies ermöglicht eine tiefergehende Analyse der Prozesseffizienz und identifiziert potenzielle Verbesserungsbereiche. Die Ergebnisse dieser Abfrage sind in der nachfolgenden Tabelle 2 dargestellt.

Durchschnittliche Dauer eines Subprozesses pro Prozessdefinition:

```
1  MATCH (pd:ProcessDefinition),(sp:SubProcess)
2  WHERE pd.processDefinitionId = sp.processDefinitionId
3  WITH pd.name AS ProcessDefinitionName,
4       sp.name AS SubProcessName,
5       sp.duration AS DauerInMillisekunden
6  RETURN ProcessDefinitionName, SubProcessName,
7         AVG(DauerInMillisekunden) / 1000 AS DurchschnittlicheDauer
```

ProcessDefinitionName	SubProcessName	DurchschnittlicheDauer
Standortwechsel; Version: 1	Maschine fuer Transport vorbereiten	11.0665

Tabelle 2 Auswertung der Durchschnittsdauer von Subprozessen pro Prozessdefinition

Diese Analyse fokussiert sich auf die Untersuchung der Lieferzeiten für verschiedene Reparaturmaterialien. Die nachfolgende Cypher-Abfrage wurde speziell entworfen, um aus der Graphdatenbank die durchschnittlichen Lieferzeiten für jedes Reparaturmaterial zu extrahieren. Diese Daten sind von großer Bedeutung, da sie Aufschluss über die Effizienz der Lieferketten und die Verfügbarkeit von Schlüsselkomponenten im Reparaturprozess geben. Die Ergebnisse dieser Abfrage (Tabelle 3), dargestellt in der nachstehenden Tabelle, bieten einen umfassenden Überblick über die durchschnittlichen Lieferzeiten der verschiedenen Materialien. Diese Informationen sind entscheidend, um Optimierungsmöglichkeiten in der Beschaffungs- und Lagerhaltungsstrategie zu identifizieren und letztendlich die Effizienz des gesamten Reparaturprozesses zu steigern. Diese Analyse kann langfristig aufzeigen, dass das teurere Schmiermittel trotz höherer Kosten aufgrund seiner kürzeren Lieferzeiten möglicherweise eine empfehlenswertere Option darstellt als das preisgünstigere Schmiermittel.

Durchschnittliche Lieferzeit pro Ersatzteil:

```
1  MATCH (m:Material)-[:IS_INVOLVED_IN]-(a:Activity {objectType: "ersatzteil"})
2  MATCH (a)-[:SEQUENCE_FLOW*]->(a2:Activity {objectType: "lieferzeit"})
```

```
    WHERE a.processInstanceId = a2.processInstanceId
    WITH m.id AS MaterialID, m.name AS MaterialName,
         toFloat(a2.objectValue) AS Lieferzeit
    WITH MaterialID,
         MaterialName,
         COLLECT(Lieferzeit) AS Lieferzeiten
    UNWIND Lieferzeiten AS lz
    WITH MaterialID,
         MaterialName,
         SUM(lz) AS SummeLieferzeit,
         COUNT(lz) AS AnzahlLieferzeiten
    RETURN MaterialName,
           SummeLieferzeit / AnzahlLieferzeiten AS DurchschnittlicheLieferzeit
    ORDER BY MaterialID
```

MaterialName	DurchschnittlicheLieferzeit
Schmiermittel guenstig	12.0
Schmiermittel teuer	1.29
Schmiermittel mittel	5.75
Schrauben Typ1	5.75
Schrauben Typ2	5.78
Schrauben Typ3	2.43
Kabel Typ1	3.5
Kabel Typ2	4.57
Kabel Typ3	4.33

Tabelle 3 Auswertung der durchschnittlichen Lieferzeit pro Ersatzteil

In diesem Teil der Analyse liegt der Fokus auf der Ermittlung der Anzahl von Reparaturen pro Maschine. Die folgende Cypher-Abfrage wurde entwickelt, um aus der Graphdatenbank präzise Daten zur Häufigkeit von Reparaturen an verschiedenen Maschinentypen zu extrahieren. Durch diese Abfrage werden die Beziehungen zwischen Maschinen und den zugehörigen Reparaturaktivitäten untersucht, wobei jede Reparaturinstanz gezählt wird. Das Ergebnis (Tabelle 4) bietet wertvolle Einblicke in die Reparaturanfälligkeit der verschiedenen Maschinentypen. Eine hohe Anzahl von Reparaturen bei bestimmten Maschinen könnte auf potenzielle Schwachstellen oder erhöhten Reparaturbedarf hinweisen. Die in der nachstehenden Tabelle dargestellten Informationen sind somit entscheidend für die Optimierung der Reparaturstrategien und können dabei helfen, langfristige Instandhaltungskosten zu reduzieren.

Anzahl der Reparaturen pro Maschine:

```
    MATCH (e:Machine)-[:IS_INVOLVED_IN]-(a:Activity)
    MATCH (pi:ProcessInstance {name: 'Maschinenreparatur'})
    WHERE a.processInstanceId = pi.processInstanceId
    WITH e.name AS MaschinenName,
         COUNT(DISTINCT pi) AS AnzahlReparaturen
    RETURN MaschinenName,
           AnzahlReparaturen
    ORDER BY AnzahlReparaturen DESC
```

MaschinenName	AnzahlReparaturen
Mikrochipproduktionsmaschine 1.2	11
Mikrochipproduktionsmaschine 2.1	8
Mikrochipproduktionsmaschine 1.1	6
Mikrochipproduktionsmaschine 2.2	4
Platinenproduktionsmaschine 2.2	3
Mikrochipproduktionsmaschine 3.1	2
Mikrochipproduktionsmaschine 3.2	2
Kabelproduktionsmaschine 1.1	2
Kabelproduktionsmaschine 1.2	2
Kabelproduktionsmaschine 2.1	2
Kabelproduktionsmaschine 3.1	2
Platinenproduktionsmaschine 3.1	2
Platinenproduktionsmaschine 1.1	2
Platinenproduktionsmaschine 1.2	2
Platinenproduktionsmaschine 2.1	2
Platinenproduktionsmaschine 3.2	2
Kabelproduktionsmaschine 3.2	1

Tabelle 4 Auswertung der Anzahl an Reparaturen pro Maschine

Ein weiterer entscheidender Aspekt der Analyse betrifft die Gesamtwartungskosten pro Maschine. Die nachfolgende Cypher-Abfrage wurde konzipiert, um aus der Graphdatenbank die kumulierten Wartungskosten für jede Maschine zu berechnen. Durch die Verbindung von Maschinen mit den dazugehörigen Wartungsaktivitäten und -kosten bietet diese Auswertung einen umfassenden Überblick über die finanziellen Auswirkungen der Instandhaltung verschiedener Maschinentypen. Höhere Wartungskosten bei bestimmten Maschinen könnten auf häufigeren Wartungsbedarf, teurere Ersatzteile oder komplexere Wartungsanforderungen hinweisen. Diese Information ist von großer Bedeutung für die strategische Planung der Wartungsressourcen und kann dabei helfen, langfristige Betriebskosten zu optimieren. Die Ergebnisse dieser Abfrage (Tabelle 5), präsentiert in der anschließenden Tabelle, zeigen die Gesamtwartungskosten, sortiert nach Maschinentyp, und liefern wichtige Einsichten für effiziente Wartungsentscheidungen.

Gesamtwartungskosten pro Maschine:

```
MATCH (m:Machine)-[:IS_INVOLVED_IN]-(a_m:Activity),
      (pi:ProcessInstance {name: 'Maschinenwartung'}),
      (a_main:Activity)-[:IS_INVOLVED_IN]-(main:Maintenance)
WHERE a_m.processInstanceId = pi.processInstanceId
      AND a_main.processInstanceId = pi.processInstanceId
WITH m.name AS MaschinenName,
     main.total AS WartungskostenText
WITH MaschinenName,
     TOFLOAT(WartungskostenText)
     AS Wartungskosten
RETURN MaschinenName,
       SUM(Wartungskosten) AS Gesamtwartungskosten
ORDER BY Gesamtwartungskosten DESC
```

MaschinenName	Gesamtwartungskosten
Mikrochipproduktionsmaschine 3.2	1400.0
Kabelproduktionsmaschine 1.2	1280.0
Kabelproduktionsmaschine 1.1	1260.0
Kabelproduktionsmaschine 2.1	1000.0
Mikrochipproduktionsmaschine 3.1	960.0
Mikrochipproduktionsmaschine 1.2	900.0
Kabelproduktionsmaschine 2.2	690.0
Kabelproduktionsmaschine 3.1	660.0
Mikrochipproduktionsmaschine 2.2	630.0
Kabelproduktionsmaschine 3.2	630.0
Mikrochipproduktionsmaschine 1.1	600.0
Mikrochipproduktionsmaschine 2.1	600.0
Platinenproduktionsmaschine 2.2	600.0
Platinenproduktionsmaschine 3.2	400.0
Platinenproduktionsmaschine 3.1	380.0
Platinenproduktionsmaschine 1.1	360.0
Platinenproduktionsmaschine 1.2	360.0
Platinenproduktionsmaschine 2.1	180.0

Tabelle 5 Auswertung der Gesamtwartungskosten pro Maschine

Dieser Teil der Analyse konzentriert sich auf die erfolglosen Reparaturen und deren Zuordnung zu einzelnen Mitarbeitern. Die dargestellte Cypher-Abfrage zielt darauf ab, die Anzahl der erfolglosen Reparaturversuche pro Mitarbeiter zu identifizieren, indem Maschinen, Prozessinstanzen und Mitarbeiterdaten miteinander verknüpft werden. Dies ermöglicht eine differenzierte Betrachtung der Effektivität einzelner Techniker im Reparaturprozess. Eine hohe Anzahl an erfolglosen Reparaturen bei einem bestimmten Mitarbeiter könnte auf Schulungsbedarf, mangelnde Ressourcen oder andere Herausforderungen im Arbeitsumfeld hinweisen. Die Ergebnisse dieser Abfrage (Tabelle 6), die in der folgenden Tabelle zusammengefasst sind, bieten wertvolle Einblicke in die Leistung einzelner Mechaniker und unterstützen die Optimierung von Personalentwicklungsmaßnahmen sowie die Verbesserung der Gesamtqualität des Reparaturservices.

Erfolglose Reparaturen pro Mitarbeiter:

```
MATCH (e:Machine)-[:IS_INVOLVED_IN]-(a:Activity)
MATCH (pi:ProcessInstance {name: 'Maschinenreparatur'})
    WHERE pi.processInstanceId = a.processInstanceId
MATCH (gw:Activity {name: 'erfolgreich'})
    WHERE pi.processInstanceId = gw.processInstanceId
WITH e.name AS MaschinenName,
     pi.processInstanceId AS pid,
     COUNT(gw) AS GatewayCount
    WHERE GatewayCount > 1
MATCH (a_emp:Activity)-[:IS_INVOLVED_IN]-(emp:Employee)
    WHERE a_emp.processInstanceId = pid
WITH MaschinenName,
     pid,
     (emp.name + " " + emp.surname) AS MechanikerName
RETURN MaschinenName,
       MechanikerName,
       COUNT(DISTINCT pid) AS ErfolgloseReparaturen
ORDER BY AnzahlNichtErfolgreicheReparaturen DESC,
         MaschinenName,
         MechanikerName
```

MaschinenName	MechanikerName	ErfolgloseReparaturen
Kabelproduktionsmaschine 1.1	Melanie Johnson	1
Mikrochipproduktionsmaschine 1.1	Gared Wick	1

Tabelle 6 Auswertung der erfolglosen Reparaturen pro Mechaniker

Die in diesem Kapitel vorgestellten Auswertungen illustrieren, wie die Nutzung einer Graphdatenbank für die Verarbeitung und Analyse von Prozess- und Entitätsdaten weitreichende und präzise Einblicke in operative Abläufe ermöglicht. Die Effizienz und Flexibilität solcher Analysen stellen einen signifikanten Mehrwert gegenüber traditionellen, weniger vernetzten Datenanalysesystemen dar. Dies unterstreicht das Potenzial von Graphdatenbanken für die Vernetzung von Prozess- und Entitätsdaten, um tiefergehende und umfassendere Auswertungen in Unternehmenskontexten zu realisieren.

5 Bewertung und Ausblick des erarbeiteten Ansatzes

In diesem Kapitel steht die kritische Bewertung und die eingehende Diskussion zur Anwendbarkeit des vorgestellten Konzepts im Vordergrund. Nach einer umfassenden Darstellung der vorbereitenden Schritte und der Implementierung des Konzepts wird nun dessen Praxistauglichkeit einer gründlichen Prüfung unterzogen. Hierbei werden sowohl die Stärken als auch die Limitationen und die verschiedenen Einsatzmöglichkeiten des Konzepts in unterschiedlichen Unternehmenskontexten beleuchtet. Die Abschnitte 5.1 und 5.2 befassen sich speziell mit der Untersuchung der praktischen Relevanz und der Vielfalt an Anwendungsszenarien. Dabei wird sowohl die direkte Umsetzbarkeit als auch die langfristigen Entwicklungsperspektiven des Konzepts berücksichtigt.

5.1 Kritische Betrachtung der praktischen Umsetzbarkeit des Konzepts

Die vorliegende Arbeit demonstriert erfolgreich, dass die Darstellung und Analyse von Prozess- und Entitätsdaten in einer Graphdatenbank nicht nur möglich, sondern auch wertvoll ist. Das entwickelte prototypische Graphschema und dessen Integration in eine Graphdatenbank stellen einen bedeutenden Schritt in dieser Richtung dar. Trotz des Erfolgs dieses Ansatzes sind jedoch mehrere Herausforderungen und Einschränkungen zu beachten, die im Folgenden diskutiert werden.

Prozessmodellierung
Einer der kritischen Aspekte in der Prozessmodellierung ist die Identifizierung und korrekte Implementierung der BPMN-Elemente. Obwohl in diesem Konzept wesentliche BPMN-Elemente berücksichtigt wurden, gibt es eine Vielzahl weiterer Elemente, die eine spezielle Behandlung erfordern könnten, ähnlich den bereits behandelten Gateways. Daraus ergibt sich, dass dieses Konzept eher als Richtlinie zur Erstellung einer Basisstruktur dient und nicht alle möglichen Sonderfälle abdeckt. Diese Einschränkung ist besonders relevant, wenn es um die praktische Anwendung in komplexen Geschäftsprozessen geht.

Entitätsdaten und Datenkonsolidierung
In der Unternehmenspraxis stellen redundante Daten aus verschiedenen Anwendungssystemen eine signifikante Herausforderung dar. Besonders problematisch wird es, wenn diese Datensätze inkonsistent sind. Vor dem Import von Entitätsdaten aus diversen Unternehmensanwendungen ist daher eine umfassende Konsolidierung und Zentralisierung der Datenbasis erforderlich. Dieser Prozess kann aufgrund seiner Komplexität, insbesondere in größeren Unternehmen, zu einem erheblichen Aufwand führen und potenziell die Umsetzung des Konzepts erschweren. Eine mögliche Lösung könnte in der Fokussierung auf spezifische Unternehmenseinheiten liegen, um die Datenstrukturierung zu vereinfachen und den Aufwand für die Datenbereinigung zu minimieren.

Management von Entitätstypen
Ein weiterer zu berücksichtigender Aspekt ist die Vielzahl an Entitätstypen, die in großen Unternehmensnetzwerken existieren können. Für jeden Entitätstyp muss eine spezifische Datenstruktur entwickelt werden, was einen beträchtlichen manuellen Aufwand bedeutet. Obwohl dieser Auf-

wand grundsätzlich einmalig ist und sich bei zukünftiger Nutzung des Konzepts lediglich auf Erweiterungen beschränkt, bleibt er eine nicht zu unterschätzende Herausforderung.

Datenschutz und Mitarbeiterleistung
Ein weiteres wesentliches Element, das besondere Aufmerksamkeit erfordert, ist der Datenschutz, besonders wenn personenbezogene Daten involviert sind. Die Zugriffsberechtigungen auf die Graphdatenbank und die Art der in den Entitätstypen *Mitarbeiter* enthaltenen Daten müssen sorgfältig geprüft werden. Zudem birgt die Möglichkeit, Leistungsauswertungen von Mitarbeitern durchzuführen, potenzielle Konflikte mit dem Betriebsrat und erfordert daher eine sensible Handhabung im Einklang mit Datenschutzvorschriften und betrieblichen Vereinbarungen.

Schlussfolgerung
Insgesamt zeigt diese kritische Betrachtung, dass, obwohl das entwickelte Konzept vielversprechende Ansätze und Methoden für die Darstellung und Analyse von Prozessdaten in Graphdatenbanken bietet, die Umsetzung in der Praxis mit verschiedenen Herausforderungen verbunden ist. Diese Herausforderungen reichen von technischen Aspekten der Prozessmodellierung über datenbezogene Probleme bis hin zu datenschutzrechtlichen Überlegungen. Es wird deutlich, dass für eine erfolgreiche Implementierung des Konzepts in der Unternehmenspraxis eine ganzheitliche Betrachtung und Berücksichtigung dieser Faktoren unerlässlich ist.

5.2 Ausblick und Erweiterungspotentiale des Konzepts

In diesem Kapitel wird das Potenzial des Konzepts zur Erweiterung und Optimierung für den praktischen Einsatz ausführlich erörtert. Das Konzept, das in seiner Grundform bereits eine breite Palette an Anwendungen in der Prozessautomatisierung abdeckt, ist besonders für Unternehmen aller Größenordnungen von Bedeutung. Von kleinen und mittleren Unternehmen bis hin zu multinationalen Konzernen bietet es eine universelle Anwendbarkeit.

Die Integration von Natural Language Processing (NLP) stellt eine zentrale Erweiterungsmöglichkeit dar. NLP ermöglicht es Nutzern ohne spezifische Vorkenntnisse in Cypher, effektiv mit Graphdatenbanken zu interagieren. Dies eröffnet eine neue Dimension der Datenanalyse. Beispielsweise könnte durch einfache Spracheingaben ermittelt werden, welcher Mitarbeiter bei der Reparatur von Maschinen am effizientesten ist, welche Maschinen die höchsten Wartungskosten verursachen oder an welchem Standort sich die meisten Maschinen befinden. Solche Fragen könnten zu wertvollen Einsichten führen, die die Unternehmensführung in strategischen Entscheidungen unterstützen.

Ein weiterer wichtiger Aspekt ist die Nutzung von Predictive Analytics. Durch die Kombination von Prozess- und Entitätsdaten lassen sich Muster erkennen und Vorhersagen über zukünftige Ereignisse treffen. Dies könnte beispielsweise dazu dienen, vorauszusagen, welche Maschine wahrscheinlich in einem bestimmten Zeitraum ausfallen wird, wie lange der Ausfall dauern könnte oder wie langwierig ein Standortwechsel sein wird. Diese Art von Analytik könnte Unternehmen helfen, proaktiver zu agieren und potenzielle Probleme zu antizipieren, bevor sie eintreten.

Abschließend lässt sich feststellen, dass das Konzept, trotz der Herausforderungen bei der erstmaligen Implementierung und des damit verbundenen Aufwands, ein hohes Potenzial für die Zukunft in der Welt der Datenanalyse besitzt. Die in dieser Arbeit gestellten Forschungsfragen wurden umfassend und zufriedenstellend beantwortet. Das Konzept bietet nicht nur eine solide Basis für zukünftige technische Erweiterungen, sondern trägt auch wesentlich zur Erweiterung der Möglichkeiten in der Datenanalyse bei. Dies spiegelt sich in den vielfältigen Anwendungsmöglichkeiten und dem Mehrwert, den es für Unternehmen jeder Größe bietet, wider.

Anhang

A1 Process Instance JSON-Datei


```
{
  "_index": "process-instances-20220201-0848-52-31732230",
  "_type": "_doc",
  "_id": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
  "_version": 1704144868388,
  "_score": 1,
  "_source": {
    "__flowableVersion": 9,
    "processDefinitionName": "Maschinenreparatur",
    "activityId": "endNoneEvent1",
    "deploymentId": "PRC-b346524b-a8ae-11ee-95bb-4e347637bf48",
    "id": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
    "processDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
    "processDefinitionCategory": "http://flowable.org/test",
    "processDefinitionVersion": 1,
    "durationInMillis": 44175,
    "processDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur",
    "startUserId": "admin",
    "startActivityId": "startnoneevent1",
    "startTime": "2024-01-01T21:33:44.175Z",
    "involvedUsers": [
      "admin"
    ],
    "variables": [
      {
        "id": "VAR-733a73c6-a8ed-11ee-95bb-4e347637bf48",
        "name": "maschine",
        "type": "string",
        "scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
        "scopeType": "bpmn",
        "scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
        "scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur
            ",
        "rawValue": "mc1_2",
        "textValue": "mc1_2",
        "textValueKeyword": "mc1_2"
      },
      {
        "id": "VAR-7aaf2cb0-a8ed-11ee-95bb-4e347637bf48",
        "name": "form_ersatzteileAuswaehlenForm_outcome",
        "type": "string",
        "scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
        "scopeType": "bpmn",
        "scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
        "scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur
            ",
        "rawValue": "ersatzteil: ersatz6; 2024-01-01T21:34:00.584Z; TSK-77fdc848-a8ed-11
            ee-95bb-4e347637bf48",
```

```
        "textValue": "ersatzteil: ersatz6; 2024-01-01T21:34:00.584Z; TSK-77fdc848-a8ed-11ee-95bb-4e347637bf48",
        "textValueKeyword": "ersatzteil: ersatz6; 2024-01-01T21:34:00.584Z; TSK-77fdc848-a8ed-11ee-95bb-4e347637bf48"
      },
      {
        "id": "VAR-7aaf2cb1-a8ed-11ee-95bb-4e347637bf48",
        "name": "ersatzteile",
        "type": "string",
        "scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
        "scopeType": "bpmn",
        "scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
        "scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur",
        "rawValue": "ersatz6",
        "textValue": "ersatz6",
        "textValueKeyword": "ersatz6"
      },
      {
        "id": "VAR-8ab9ec44-a8ed-11ee-95bb-4e347637bf48",
        "name": "erfolgreich",
        "type": "boolean",
        "scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
        "scopeType": "bpmn",
        "scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
        "scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur",
        "rawValue": "true",
        "booleanValue": true
      },
    [...]
  }
}
```

A2 Activity JSON-Datei


```
{
  "_index": "activities-20220201-0848-47-31727645",
  "_type": "_doc",
  "_id": "PRC-733bfa6a-a8ed-11ee-95bb-4e347637bf48",
  "_version": 1704144836929,
  "_score": 1,
  "_source": {
    "__flowableVersion": 1,
    "processDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
    "processInstanceId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
    "runtimeActivityInstanceId": "PRC-733bfa6a-a8ed-11ee-95bb-4e347637bf48",
    "durationInMillis": 7979,
    "activityName": "Mechaniker auswaehlen",
    "activityId": "formTask2",
    "executionId": "PRC-7069a076-a8ed-11ee-95bb-4e347637bf48",
    "tenantId": "",
    "startTime": "2024-01-01T21:33:48.909Z",
    "assignee": "admin",
    "endTime": "2024-01-01T21:33:56.888Z",
    "activityType": "userTask",
    "taskId": "TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48"
  },
  [...]
  }
}
```

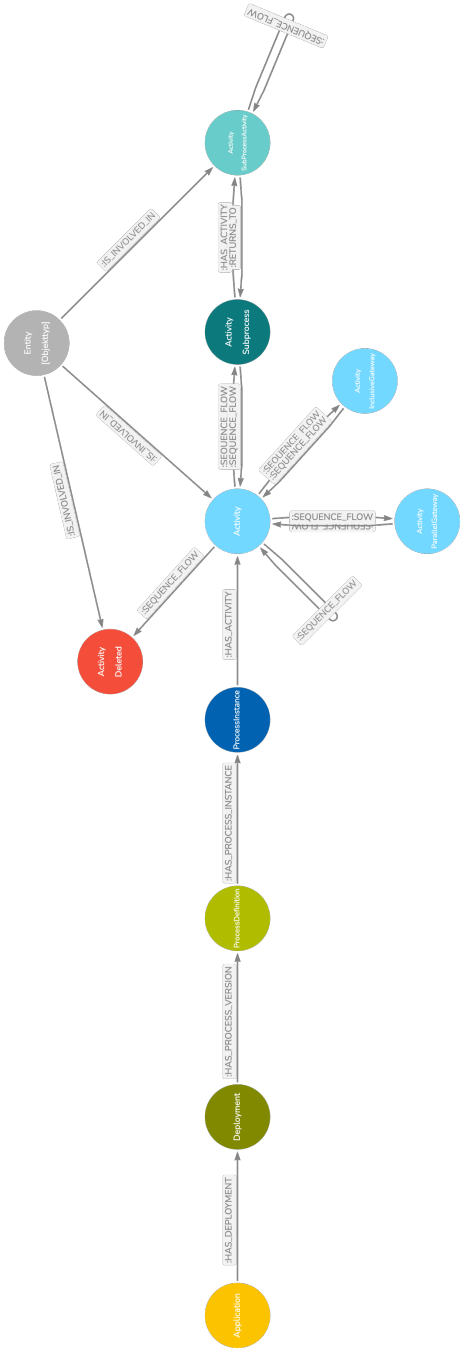
A3 Task JSON-Datei


```
{
  "_index": "tasks-20220201-0848-52-31732998",
  "_type": "_doc",
  "_id": "TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48",
  "_version": 1704144868388,
  "_score": 1,
  "_source": {
    "__flowableVersion": 8,
    "rootScopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
    "dueDate": null,
    "processDefinitionName": "Maschinenreparatur",
    "scopeDefinitionName": "Maschinenreparatur",
    "id": "TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48",
    "processDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
    "processInstanceId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
    "scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
    "priority": 50,
    "executionId": "PRC-7069a076-a8ed-11ee-95bb-4e347637bf48",
    "taskDefinitionKey": "formTask2",
    "scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
    "scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur",
    "parentScopeDefinitionCategory": "http://flowable.org/test",
    "rootScopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur
        ",
    "processDefinitionCategory": "http://flowable.org/test",
    "name": "Mechaniker auswaehlen",
    "assignee": "admin",
    "durationInMillis": 7976,
    "description": null,
    "scopeDefinitionCategory": "http://flowable.org/test",
    "parentScopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
    "processDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur",
    "rootScopeType": "bpmn",
    "parentScopeType": "bpmn",
    "scopeType": "bpmn",
    "rootScopeDefinitionCategory": "http://flowable.org/test",
    "involvedUsers": [
      "admin"
    ],
    "parentScopeDefinitionName": "Maschinenreparatur",
    "owner": null,
    "variables": [
      [...],
      {
        "id": "VAR-77fd04f3-a8ed-11ee-95bb-4e347637bf48",
        "name": "form_mechanikerFuerReparaturAuswaehlenForm_outcome",
        "type": "string",
        "scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
        "scopeType": "bpmn",
        "scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
        "scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur
            ",
```

```
        "rawValue": "mechaniker: mechaniker4; 2024-01-01T21:33:55.164Z; TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48",
        "textValue": "mechaniker: mechaniker4; 2024-01-01T21:33:55.164Z; TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48",
        "textValueKeyword": "mechaniker: mechaniker4; 2024-01-01T21:33:55.164Z; TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48"
      },
      {
        "id": "VAR-706a15b1-a8ed-11ee-95bb-4e347637bf48",
        "name": "taskId",
        "type": "string",
        "scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
        "scopeType": "bpmn",
        "scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
        "scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur",
        "rawValue": "TSK-884b362c-a8ed-11ee-95bb-4e347637bf48",
        "textValue": "TSK-884b362c-a8ed-11ee-95bb-4e347637bf48",
        "textValueKeyword": "TSK-884b362c-a8ed-11ee-95bb-4e347637bf48"
      },
      [...]
      {
        "id": "VAR-77fd04f4-a8ed-11ee-95bb-4e347637bf48",
        "name": "mechaniker",
        "type": "string",
        "scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
        "scopeType": "bpmn",
        "scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
        "scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur",
        "rawValue": "mechaniker4",
        "textValue": "mechaniker4",
        "textValueKeyword": "mechaniker4"
      },
      [...]
      {
        "id": "VAR-77fd04f3-a8ed-11ee-95bb-4e347637bf48",
        "name": "form_mechanikerFuerReparaturAuswaehlenForm_outcome",
        "type": "string",
        "scopeId": "PRC-7069a073-a8ed-11ee-95bb-4e347637bf48",
        "scopeType": "bpmn",
        "scopeHierarchyType": "root",
        "scopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
        "scopeDefinitionKey": "org.exentra.bachelor_thesis_processes.maschinenreparatur",
        "rawValue": "mechaniker: mechaniker4; 2024-01-01T21:33:55.164Z; TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48",
        "textValue": "mechaniker: mechaniker4; 2024-01-01T21:33:55.164Z; TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48",
        "textValueKeyword": "mechaniker: mechaniker4; 2024-01-01T21:33:55.164Z; TSK-733bfa6b-a8ed-11ee-95bb-4e347637bf48"
      },
      [...]
    ],
```

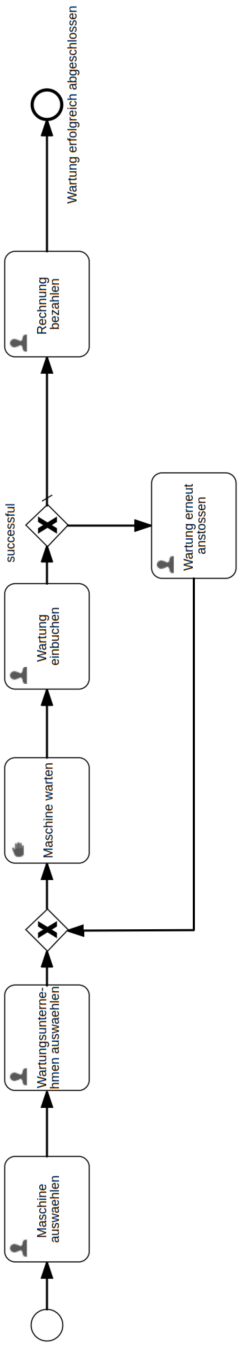
```
96      "parentScopeDefinitionId": "PRC-b34ae62f-a8ae-11ee-95bb-4e347637bf48",
97      "formKey": "mechanikerFuerReparaturAuswaehlenForm",
98    [...]
99    }
100 }
```

A4 Konzeptionelles Graphschema



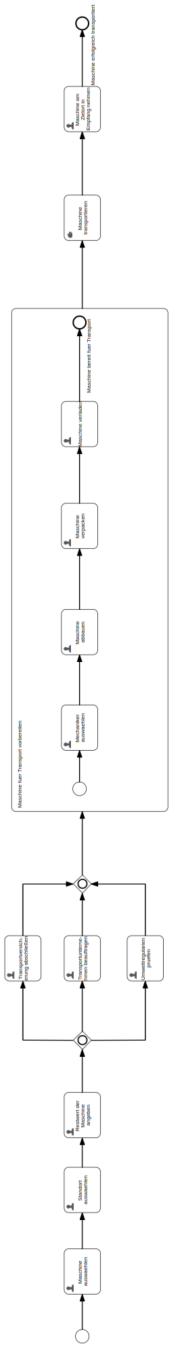
Erstelltes Graphschema mit Nutzung der erstellten Transformationsregeln

A5 Maschinenwartungsprozess



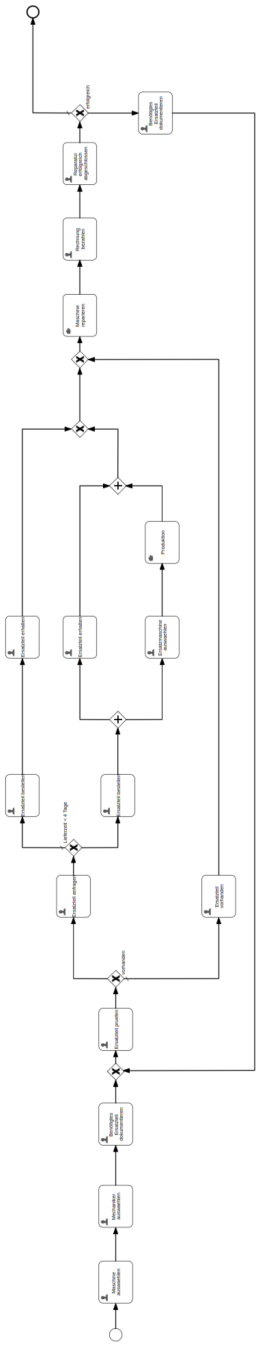
Prozessmodell des Maschinenwartungsprozesses

A6 Standortwechselprozess



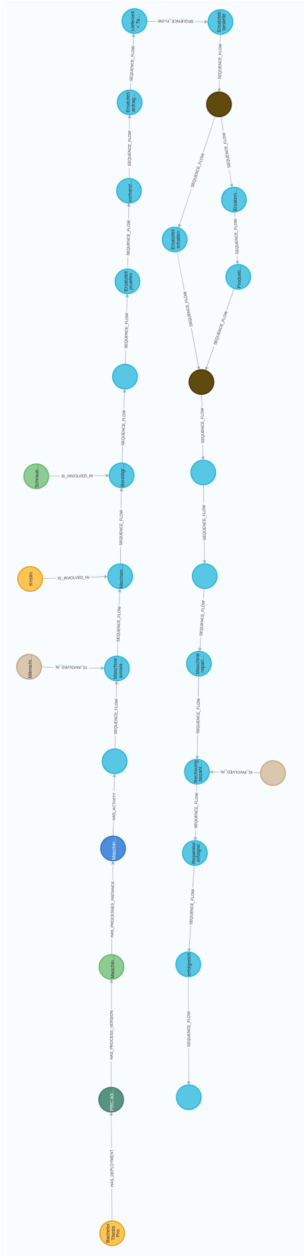
Prozessmodell des Standortwechselprozesses

A7 Maschinenreparaturprozess



Prozessmodell des Maschinenreparaturprozesses

A8 Prozessinstanzausgabe in Neo4j Browser



Ausgabe einer Prozessinstanz des Maschinenreparaturprozesses mit allen zugehörigen Knoten und Beziehungen

Literaturverzeichnis

Angles, Renzo und Claudio Gutierrez: „An Introduction to Graph Data Management“, in: *Cornell University Library, arXiv.org* (2017), DOI: 10.1007/978-3-319-96193-4_1, https://doi.org/10.1007/978-3-319-96193-4_1, zuletzt aufgerufen 13. 01. 2024.

Anthapu, Ravindranatha: Graph data processing with cypher: a practical guide to building graph traversal queries using the Cypher syntax on Neo4j, 1. Aufl., PDF e-book, Birmingham und Mumbai: Packt Publishing, 2022, https://portal-igpublish-com.thi.idm.oclc.org/iglibrary/search/PACKT0006565.html, zuletzt aufgerufen 13. 01. 2024.

Docker Inc.: Docker overview, 2024, https://docs.docker.com/get-started/overview/#docker-architecture, zuletzt aufgerufen 13. 01. 2024.

Elasticsearch B.V.: Data in: documents and indices, 2024, https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html, zuletzt aufgerufen 13. 01. 2024.

Elasticsearch B.V.: Java High Level REST Client, 2024, https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/java-rest-high.html, zuletzt aufgerufen 14. 01. 2024.

Elasticsearch B.V.: Kibana—your window into Elastic, 2024, https://www.elastic.co/guide/en/kibana/current/introduction.html#introduction, zuletzt aufgerufen 13. 01. 2024.

Elasticsearch B.V.: What is Elasticsearch?, 2024, https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html, zuletzt aufgerufen 13. 01. 2024.

Flowable: Add a Flow-App with Flowable Design: Adding a new Design Flow-App, 2023, https://documentation.flowable.com/latest/develop/fe/flow-app, zuletzt aufgerufen 13. 01. 2024.

Flowable: BPMN 2.0 Constructs, 2023, https://www.flowable.com/open-source/docs/bpmn/ch07b-BPMN-Constructs, zuletzt aufgerufen 11. 01. 2024.

Flowable: Flowable User Guides, 2023, https://documentation.flowable.com/latest/user/user-introduction, zuletzt aufgerufen 13. 01. 2024.

Flowable: Getting Started, 2023, https://www.flowable.com/open-source/docs/bpmn/ch02-GettingStarted/, zuletzt aufgerufen 13. 01. 2024.

Forschungsteam PDA-RobE: „PDA-RobE Zwischenbericht 2022 Langfassung“, Interner Zwischenbericht, Pfaffenhofen und Erlangen, 2022.

Kassem, Gamal und Klaus Turowski: „Matching of Business Data in a Generic Business Process Warehousing“, in: *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, Veröffentlichungsdatum in IEEE Xplore: 02 Januar 2020, Elektronische ISBN: 978-1-7281-1360-9, Print on Demand (PoD) ISBN: 978-1-7281-1361-6, Las Vegas, NV, USA: IEEE, Dez. 2018, S. 284–289, DOI: 10.1109/CSCI46756.2018.00062.

Leymann, Frank und David Schumm: Business Process Model and Notation (BPMN), 2018, https://wirtschaftslexikon.gabler.de/definition/business-process-model-and-notation-bpmn-52689/version-275807, zuletzt aufgerufen 13. 01. 2024.

Leymann, Frank und David Schumm: Geschäftsprozesstechnologie, 2018, https://wirtschaftslexikon.gabler.de/definition/geschaeftsprozesstechnologie-52694/version-275812, zuletzt aufgerufen 13. 01. 2024.

Neo4j Inc.: Data types and mapping to Cypher types, 2023, https://neo4j.com/docs/python-manual/current/data-types/, zuletzt aufgerufen 14. 01. 2024.

Neo4j Inc.: Welcome to Neo4j, 2023, https://neo4j.com/docs/getting-started/, zuletzt aufgerufen 13. 01. 2024.

Nolan, Deborah und Duncan Temple Lang: „JavaScript Object Notation“, in: *XML and Web Technologies for Data Sciences with R*, New York, NY: Springer New York, 2014, S. 227–253, DOI: 10.1007/978-1-4614-7900-0_7, https://doi.org/10.1007/978-1-4614-7900-0_7,

The Kubernetes Authors: Was ist Kubernetes?, 2024, https://kubernetes.io/de/docs/concepts/overview/what-is-kubernetes/, zuletzt aufgerufen 13. 01. 2024.

Trotha, Christian von, Tobias Kleinert und Ulrich Epple: „Ein Konzept zur Kontextualisierung von Prozessdaten: Der erste Schritt auf dem Weg zur datenzentrierten Handlung“, in: *atp magazin* 62.10 (2020), S. 68–76, ISSN: 2190-4111, https://doi.org/10.17560/atp.v62i10.2502, zuletzt aufgerufen 12. 01. 2024.

Ye, Sheng u. a.: „Recovering Latent Data Flow from Business Process Model Automatically“, in: *Wireless communications and mobile computing* 2022 (2022), S. 1–11, ISSN: 1530-8669, http://dx.doi.org.thi.idm.oclc.org/10.1155/2022/7579515, zuletzt aufgerufen 13. 01. 2024.

Eidesstattliche Erklärung zur Bachelorarbeit

Ich erkläre hiermit, dass ich die Arbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ingolstadt, den 15. Januar 2024

Daria Öhlund