ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

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

Konzept und Implementierung eines Frameworks zur Verwaltung von Digital Twins

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Sven Bihlmaier
sven.bihlmaier@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Klaus Kammerer

2017

Fassung 3. August 2018

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Die fortschreitende Digitalisierung und damit verbundene Themen wie Industrie 4.0 werden für Unternehmen immer wichtiger. So sollen zukünftig beispielsweise digitale Abbilder, sogenannte Digital Twins, von Produkten helfen, deren Lebenszyklus zu analysieren. Die hierfür erforderlichen Daten müssen erhoben und gespeichert werden. Diese können auch für andere Aufgaben bereitgestellt werden. Die Daten liegen aber meist in unterschiedlichen Formaten und an verschiedenen Endpunkten bereit. Deren Erhebung und Integration in bestehende Informationssysteme ist aufgrund der meist großen Datenmengen zudem aufwendig und meist nur mit großem Aufwand verwaltbar.

Im Rahmen dieser Bachelorarbeit wird ein graphbasiertes Konzept zur Verwaltung von Digital Twins am Beispiel von Pharmaverpackungsmaschinen vorgestellt. Zusätzlich wird die Machbarkeit mit einer prototypischem Implementierung aufgezeigt.

Danksagung

Zunächst gilt mein Dank Herrn Prof. Dr. Manfred Reichert für die Bereitstellung des Themas der Bachelorarbeit und den Ressourcen, dieses zu bearbeiten. Einen besonderen Dank widme ich meinem Betreuer Klaus Kammerer, der mich während der gesamten Zeit fachlich und persönlich unterstützt und motiviert hat.

Außerdem möchte ich meiner Familie danken, die mich tatkräftig moralisch unterstützt hat. Weiter danke ich meinen Freunden, insbesondere Adrian und Fabian, die mich sowohl fachlich unterstützt als auch motiviert haben.

Mein Dank gilt zudem Sarah, die mich während der Arbeitsphase immer wieder auf den Boden zurückgeholt hat.

Glossar

API Application Programming Interface. Eine API ist eine Schnittstelle, mit deren Hilfe Systeme miteinander kommunizieren können.

CRUD Das Akronym steht für Create, Read, Update, Delete. Dies sind die üblichen Methoden, um Daten in persistenten Speichern zu verwalten.

Framework Ein Framework ist eine Rahmenstruktur für ein Softwareprojekt. Es bietet wichtige Funktionalitäten für die gewünschten Einsatzgebiete.

HTTP Hypertext Transfer Protocol. HTTP ist ein Protokoll zur Übertragung von Daten zwischen Systemen.

JSON Die JavaScript Object Notation ist ein Datentransport-Format.

REST Representational State Transfer bezeichnet die Möglichkeit, Methoden über eine Web-Schnittstelle aufzurufen.

URL Ein Uniform Resource Locator ist ein einheitlicher Ressourcenzeiger in einem Computernetzwerk. Meist auch umgangssprachlich als "Internetadresse" genutzt, können mit URLs Ressourcen eindeutig identifiziert werden.

XML Mit der Extensible Markup Language können hierarchisch strukturierte Daten abgebildet werden. XML ist ein standardisiertes Format und kann dazu genutzt werden, Daten zwischen Systemen zu transportieren.

Inhaltsverzeichnis

1

Einleitung

1.1 Problemstellung

Aufgrund der fortschreitenden Digitalisierung sehen sich Unternehmen gezwungen, immer mehr Daten zu speichern und zu verarbeiten. Ein Nutzen hiervon besteht beispielsweise darin, genauere Einblicke über die Nutzung ihrer Produkte zu erhalten, um diese Produkte weiter verbessern zu können. Dies kann beispielsweise eine Einsparung in der Produktion oder eine Erhöhung der Qualität des Produkts sein.

Um schneller und präziser Daten wie beispielsweise Maschinenauslastung, Warenbestand oder Auslastung des Fuhrparks zu erhalten, ist es daher ratsam, digitale Repräsentationen dieser Maschinen zu erstellen. Die Menge an Daten um ein solches Cyber-physisches System wird auch digitaler Zwilling (englisch: “Digital Twin”) genannt [1]. Daten eines digitalen Zwillings sind meist über mehrere Systeme verteilt und beinhalten digitale Repräsentationen von Hardware und Sensoren [2].

Bei der Erstellung von Digital Twins treten verschiedene Herausforderungen auf. Zum einen kommen Daten meist in heterogenen Datenmodellen vor; Datenströme von Sensoren und Dokumente zur Beschreibung eine Maschine liegen beispielsweise nicht im selben Format vor. Daher werden verschiedene Softwarekomponenten benötigt, um Dokumente und Sensordaten in ein gemeinsames Format zu überführen und so speichern zu können. Des Weiteren müssen auch Daten über die physikalische Struktur einer Maschine hinterlegt werden. Beispielsweise können CAD-Zeichnungen Teil eines Digital Twins sein.

Außerdem gilt es, erhobene Daten in semantische Beziehung zu setzen, da viele Daten umso mehr Aussagekraft bekommen, je mehr sie mit anderen Daten verbunden werden. In ihrer Gesamtheit sollen die Daten beispielsweise Aussagen über die Funktionalität des physischen Objekts zulassen, die dem Digital Twin zugrunde liegt. Je mehr Daten gesammelt werden und je mehr sie vernetzt werden können, desto genauer beschreiben die Daten das Produkt und seine Funktionalität. Zusätzlich gilt es, das gesamte Lifecycle-Management eines Produktes abdecken zu können. Daten müssen also gesammelt und umgewandelt werden können und den jeweils aktuellen Stand repräsentieren.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Erstellung eines Konzeptes, um Daten von Maschinen in die Form eines Digital Twins zu überführen. Ein weiterer Aspekt dieser Arbeit ist das Entwickeln eines einheitlichen Werkzeugs zur Erstellung und Verwaltung von Digital Twins. Durch ein solches Werkzeug können verschiedene Unternehmen Digital Twins erzeugen, die alle ein gemeinsames Format besitzen. So hat nicht jedes Unternehmen eine bestimmte Art von Datenformaten, sondern mehrere Unternehmen ein gemeinsames Datenformat. In Folge dessen wird die Wartbarkeit der Software erheblich gesteigert. Aus dieser einfacheren Wartbarkeit folgt beispielsweise, dass sich Nutzer nicht mit verschiedenen Modellen auseinandersetzen müssen oder diese in eine andere Form überführen müssen.

Ein Digital Twin kann daher beispielsweise sowohl für Wartungs-, als auch zu Analysezwecken genutzt werden, da dieser auch Messdaten interner Sensoren beinhalten kann. Die Erstellung von Digital Twins soll so generisch wie möglich erfolgen, um möglichst viele Einsatzfälle abdecken zu können. Heterogene Datensätze in einem einheitlichen Format bilden die Grundlage für digitale Dienste. Außerdem sollen Beziehungen zwischen Messdaten und Daten über Maschinenkomponenten modellierbar sein, um komplexe logische Zusammenhänge zwischen diesen darstellen zu können.

1.3 Struktur der Arbeit

Kapitel 2 beschreibt Anforderungen an die Arbeit. Kapitel 3 führt Konzepte ein, die für das Verständnis dieser Arbeit notwendig sind. In Kapitel 4 wird das basierend auf den erhobenen Anforderungen abgeleitete Konzept vorgestellt. Kapitel 5 beschreibt die prototypische Umsetzung des Konzepts. Kapitel 6 zeigt, inwiefern der Prototyp den Anforderungen entspricht. Kapitel 7 benennt einige verwandte Konzepte. Kapitel 8 gibt eine Zusammenfassung der Arbeit und einen Ausblick über zukünftige Entwicklungen.

2

Anforderungen und Analyse

In diesem Kapitel werden verschiedene Anwendungsbeispiele für einen Digital Twin aufgezeigt. Anhand dieser Beispiele werden anschließend funktionale und nicht-funktionale Anforderungen an das Konzept abgeleitet.

2.1 Use Cases

Im Folgenden werden verschiedene Szenarien beschrieben, in denen ein Digital Twin eingesetzt werden kann, um verschiedene Aufgaben zu erleichtern.

2.1.1 Use Case 1: Produktionsmaschinen

Unternehmen sind in der Lage, große Datenmengen über ihre Produkte und Kunden zu sammeln, ohne genau zu wissen, wie sie einen Mehrwert daraus generieren können. Sie haben meist sowohl Daten über den physikalischen Aufbau ihrer Produkte als auch Sensormesswerte. Jedoch werden diese Daten meist nicht zusammen gespeichert oder in Verbindung gebracht.

Cyber-physische Systeme bestehen meist aus einer Vielzahl von Modulen und Sensoren mit heterogenen Datenmodellen. Ein Beispiel für ein solches System ist das geplante intelligente Stromnetz, das Stromversorger und Stromnutzer näher zueinander führen und so viele Probleme (z.B. schlechte geografische Verteilung der Energiemenge) lösen soll [3]. Daten für ein Cyber-physisches System werden somit von unterschiedlichen Quellen bereitgestellt und müssen dennoch einheitlich verarbeitet werden. Momentan

werden Daten aus verschiedenen Quellen in ein System geleitet und dort in ihrem jeweiligen Format abgespeichert. Um diese Daten dann vergleichen zu können, müssen die Datensätze zuerst analysiert werden, um die gewünschten Daten in einem nativen Format extrahieren zu können (siehe Abbildung 2.1).

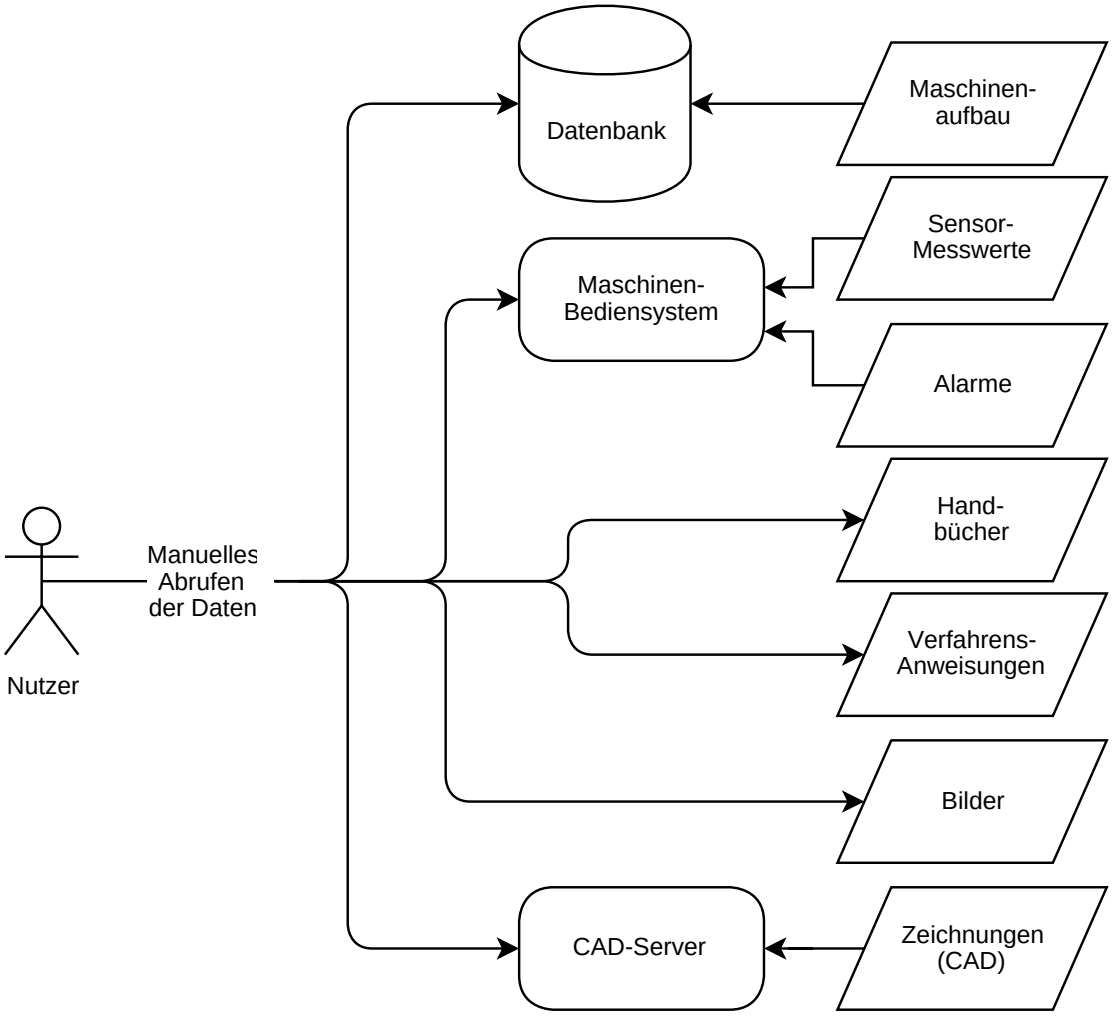

Abbildung 2.1: Anwendungsfalldiagramm Produktionsmaschinen

Die verschiedenen Daten werden außerdem meist unstrukturiert in einer Datenbank gespeichert, sodass bei einer Suche meist die gesamte Datenmenge durchsucht werden muss, um einen bestimmten Datensatz zu finden.

Soll dann auf die Daten zugegriffen oder sollen die Daten verändert werden, muss für jedes Datenformat eine andere Bearbeitungsmethode gewählt werden. Meist können verschiedene Datenformate nur auf bestimmten Maschinen manipuliert werden. Es gibt demnach keine einheitliche Schnittstelle, die von den Mitarbeitern genutzt werden kann. Bei einem Ausfall einer solchen Maschine muss ein externer Techniker hinzugezogen werden, der durch die Auftragsplanung, das Kennenlernen der Maschine, die Lokalisierung des Problems und schließlich die Problemlösung einen langen Stillstand der Maschine verursacht, in dem diese nicht arbeiten kann. Zudem wird der Wartungsprozess meist nur unzureichend dokumentiert [4]. Diese Dokumentationen werden dann wiederum in einer Datenbank hinterlegt und dort nur im Fehlerfall noch einmal eingesehen. Wichtige Statistiken, die aus diesen Dokumentationen erhoben werden können und sich positiv auf die Wartungsintensität von Systemen auswirken, können daher nicht effektiv ausgeführt werden. Wenn ein Techniker Einsicht in die verschiedenen Daten einer Maschine bekommen sollte, muss gewährleistet sein, dass er nur freigegebene Daten einsehen darf. Beispielsweise sind für einen Techniker Daten über den physikalischen Aufbau einer Maschine relevant. Andere Daten wiederum können für die Wartung irrelevant sein und sollten daher nicht von einem Techniker eingesehen werden.

2.1.2 Use Case 2: PKW

Ähnlich wie bei Maschinen im vorhergehenden Anwendungsfall bestehen auch PKW aus vielen Einzelteilen, Komponenten und Sensoren, die wiederum unterschiedliche Eigenschaften aufweisen und Messdaten generieren können.

Die Daten über den physikalischen oder logischen Aufbau eines PKW können innerhalb der Produktion von Nutzen sein, da PKW oft in vielen verschiedenen Konfigurationen und mit speziellen Extraausstattungen produziert werden und die digitale Abbildung hierbei mit einer einheitlichen Repräsentation hilfreich sein kann.

Besitzer eines PKW müssen diesen oftmals wartungsbedingt in eine Werkstatt bringen und ihn auf Fahrtauglichkeit oder Schäden und Fehler untersuchen lassen. Je nach Zustand des PKW ist diese Überprüfung mehr oder weniger aufwendig und nimmt entsprechend mehr oder weniger Zeit und somit Kosten in Anspruch. Oftmals fehlt eine

genaue Beschreibung des Fahrzeugzustands vor der eigentlichen Untersuchung. Hierdurch kann es zu unerwünschten Fehleinschätzungen bezüglich des Wartungsaufwands kommen.

Anhand im PKW gemessener Sensordaten können außerdem beispielsweise auch Informationen über das Fahrverhalten gewonnen werden. Diese Informationen sind momentan zum Beispiel für Versicherungen interessant. So bieten manche Versicherungen an, dem Versicherten in einer Werkstatt ein kleines Gerät in den PKW einbauen zu lassen, mit dem bestimmte Werte wie Seitenbeschleunigung (die beispielsweise auf harte Lenkmanöver hinweisen) oder Beschleunigung (die hartes Bremsen aufzeigen kann) gemessen werden, um im Nachhinein die Versicherungstarife an das Fahrverhalten des Fahrers anzupassen: je sicherer ein Fahrer fährt, desto günstiger kann sein Versicherungstarif eingestuft werden [5]. Die hierbei gesammelten Daten sind momentan nicht einsehbar und stehen ausschließlich der Versicherung zur Verfügung. Diese Daten werden nicht voll ausgenutzt, da sie viele Informationen enthalten, die auch in anderen Teilsystemen Relevanz haben. So können beispielsweise die vom Gerät gemessenen Sensordaten in das Fahrzeugsystem eingegeben werden, um so die Nachrüstung von Fahrerassistenzsystemen in alten PKW zu ermöglichen.

Allgemein ist das Sammeln und übersichtliche Darstellen dieser Fahrzeugdaten ein wichtiger Aspekt. Über diese Informationen können viele Aussagen bezüglich der Wartung von PKW gesammelt und getroffen werden. Anhand der ADAC Pannenstatistik wird ersichtlich, dass die Wartung von PKW eine große Rolle spielt und viel Aufwand mit sich bringt [6]. PKW-Hersteller sollten demnach mehr in eine höhere Wartbarkeit investieren. Es ist momentan zwar möglich, aber dennoch sehr schwierig, Probleme am eigenen PKW als fachfremde Person zu beheben. Dies flächendeckend zu realisieren wäre für viele PKW-Besitzer eine markante Ersparnis.

Nicht nur PKW, sondern auch LKW können von der Technik profitieren. Über das interne Sammeln der Messdaten können hierbei wichtige Informationen wie Ruhezeiten oder Mautgebühren zentral gespeichert und an die jeweilige Spedition weitergegeben werden. Natürlich betrifft auch LKW der Vorteil der verbesserten Wartbarkeit.

Daten, die bei den verschiedenen Messungen anfallen oder die bei der Produktion eines PKW bereits hinterlegt werden, können zentral in einer digitalen Repräsentation eines PKW gespeichert werden, sodass alle Daten über diesen Digital Twin auf einmal abgerufen werden können. Insgesamt kann mit dem Digital Twin der Pfad in Richtung Wartungsverbesserung, Telematik und Fahrsicherheit weiter geebnet und ausgebaut werden. Abbildung 2.2 stellt einen Auszug aus einer Übersicht über die verschiedenen Daten dar, die von einem Nutzer manuell abgefragt werden können.

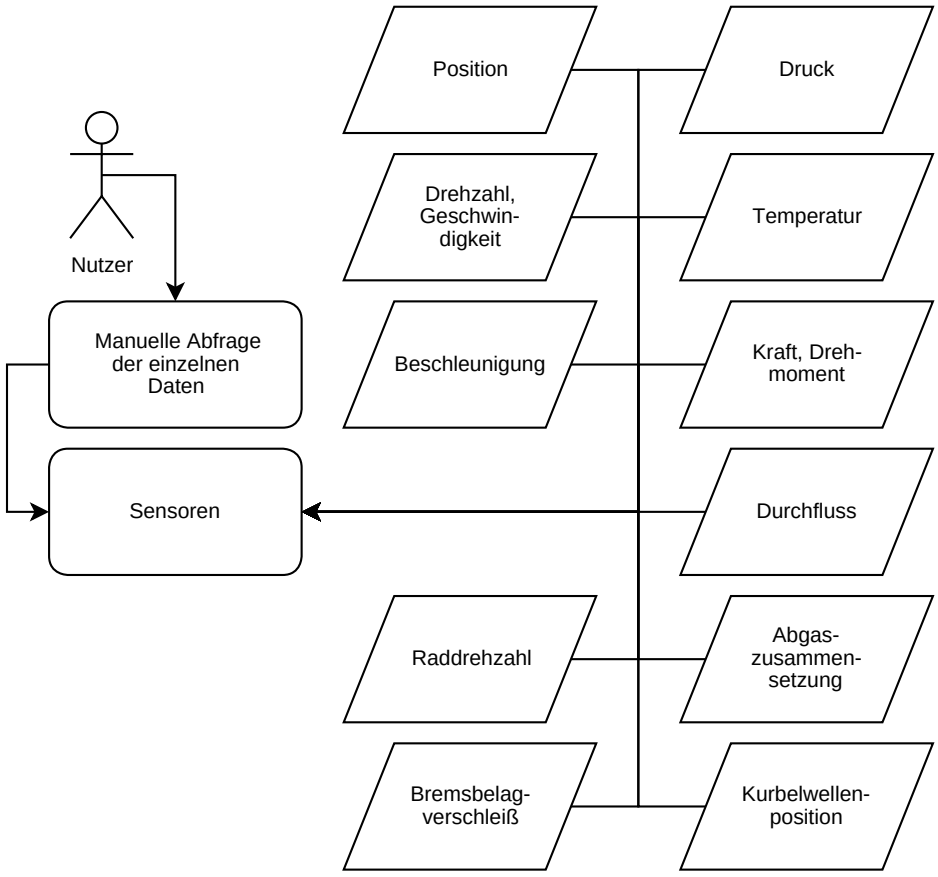

Abbildung 2.2: Auszug der Datenstruktur eines PKW

2.1.3 Use Case 3: Softwareabhängigkeiten

Bei der Softwareentwicklung können Teile anderer Software wiederverwendet werden, sofern die zugrundeliegende Architektur dies zulässt. Softwareentwickler profitieren somit von bereitgestellten Frameworks zur Erstellung komplexer Softwaresysteme. Diese vielen verschiedenen Frameworks können auch Abhängigkeiten untereinander aufweisen. Um eine korrekte Auflösung dieser Abhängigkeiten gewährleisten zu können, muss sichergestellt werden, dass die einzelnen Versionen der Frameworks dies erlauben. Um die Verwaltung solcher Frameworks und ihrer Versionsanforderungen zu automatisieren und zu vereinfachen, werden viele unterschiedliche Dependency Management Systeme angeboten (beispielsweise NuGet für .NET [7] oder Maven beziehungsweise Gradle für Java [8, 9]. Solche Systeme überwachen, sofern möglich, alle vorhandenen sogenannten ”Dependencies” (Abhängigkeiten zu anderer Software) und sorgen dafür, dass diese auf dem benötigten und möglichst neuesten Stand sind. Diese Systeme schlagen dem Nutzer vor, welche Versionen benötigt werden und laden diese (sofern dies erlaubt ist) automatisch in das Softwareprojekt. Durch die meist gegenseitige Abhängigkeit vieler verschiedener Dependencies kann dieses Netz aus Abhängigkeiten als Graph dargestellt werden, dem sogenannten *Dependency Graph*.

Auch für die beiden bereits genannten Einsatzzwecke (Maschine und PKW) und deren Software werden immer auch Frameworks genutzt, die entweder vom Hersteller selbst oder von Zulieferern entwickelt werden.

Abbildung 2.3 zeigt ein Beispiel für einen solchen Dependency Graph für eine Graph-Algorithmik-Software. Diese benötigt Frameworks, um beispielsweise einzelne Knoten eines Graphen verwalten zu können oder komplexe mathematische Berechnungen durchführen zu können. Sowohl Klassen zur Erstellung eines Graphen als auch die mathematisch-logischen Pakete werden durch solche Frameworks bereitgestellt.

Ein Dependency Graph kann mit Dependency Management Software erstellt werden. So können wichtige Kernkomponenten, mehrfach genutzte Pakete oder Abhängigkeiten sofort erkannt werden.

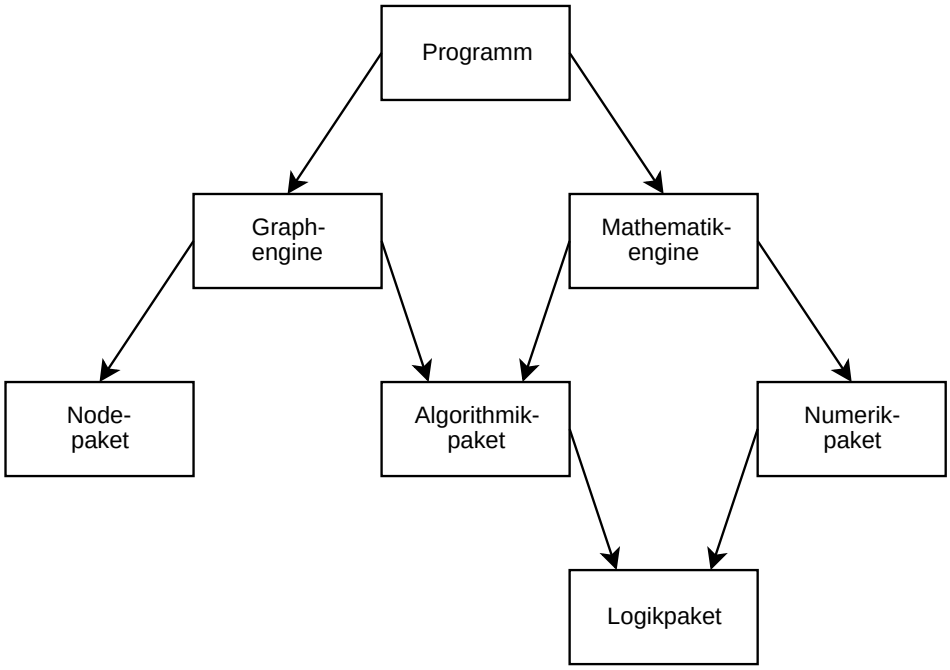

Abbildung 2.3: Beispielhafter Dependency Graph

2.2 Anforderungen

Die aufgezeigten Use Cases behandeln verschiedene Problemstellungen und Themengebiete. Das zu erstellende Konzept soll Digital Twins der vorgestellten Use Cases ermöglichen. Zusätzlich wird ein Managementsystem für die einzelnen Digital Twins benötigt, um diese zentral speichern und verwalten zu können.

Die einzelnen Anforderungen bezüglich der Funktion der Software werden in Tabelle 2.1 aufgelistet. Des Weiteren ergeben sich nicht-funktionale Anforderungen, die in Tabelle 2.2 beschrieben werden.

Tabelle 2.1: Funktionale Anforderungen

F1	Heterogene Datenmodelle	Daten sollen in verschiedenen Formaten eintreffen können und so verarbeitet werden, dass sie nachher ein einheitliches Format besitzen.
F2	Unterschiedliche Datenquellen	Daten sollen aus verschiedenen Quellen eintreffen können und weiterverarbeitet werden.
F3	Beziehung zwischen Datensätzen	Daten über die physikalische Struktur eines Objekts sollen mit Messwerten der Sensoren dieses Objekts verbunden werden können, um so weitere Aussagen treffen zu können.
F4	Gruppierung von Daten	Messdaten sollen direkt an die entsprechenden Module, in denen sie aufgenommen wurden, gekoppelt werden können.
F5	Dynamische Akquisition von Daten	Die Software soll selbstständig während der Laufzeit Daten sammeln, verarbeiten und abspeichern können.
F6	Effiziente Suchfunktionen	Die Software soll so aufgebaut sein, dass Daten strukturiert abgespeichert werden, sodass Suchen auf den Daten möglichst schnell sind.
F7	Standardisierte Schnittstellen	Mit der Software soll möglichst agil kommuniziert werden können. Über eine Schnittstelle soll auf die gesamte Software zugegriffen werden können.
F8	Authentifizierung	Der Zugriff auf Daten soll beschränkt werden können, sodass für den Zugriff eine bestimmte Authentifizierungsstufe nötig ist.

Tabelle 2.2: Nicht-funktionale Anforderungen

NF1	Data Awareness	Unternehmen sollen sich der Daten bewusst sein, die sie sammeln und der Möglichkeiten, die sie mittels dem Sammeln der Daten haben. Dafür sollen unter anderem Modelle der Daten bereitgestellt werden.
NF2	Generischer Aufbau	Das Konzept zur Erstellung von Digital Twins soll so generisch wie möglich werden, um möglichst viele Use Cases abdecken zu können.
NF3	Geschwindigkeit	Alle Suchen und das Laden der Daten sollen so schnell wie möglich (möglichst unter einer Minute) ablaufen.
NF4	Wiederverwendbarkeit	Die Software soll so erstellt werden, dass große Teile der Software wiederverwendet werden können (Stichwort Schnittstellen).
NF5	Wartbarkeit	Die Software soll einfach verständlich und gut dokumentiert sein, sodass sie einfach zu warten ist.
NF6	Skalierbarkeit	Die Software soll hoch skalierbar sein. Das heißt es sollen Tools genutzt werden, die eine möglichst hohe Skalierbarkeit gewährleisten können.

3

Grundlagen

3.1 Digitaler Zwilling

Ein Digitaler Zwilling ist ein Modell, das eine physikalische Maschine hinsichtlich ihrer Funktion und ihrer Daten digital abzubilden versucht [10]. So können ganze Maschinen mitsamt ihrer Funktionalität und ihren physikalischen Gegebenheiten digitalisiert werden. Zur Unterstützung dessen werden der Maschine eigene Sensor- oder Metadaten (z.B. Standort, Anschaffungsjahr, letzte Wartung) angehängt. Der Mehrwert für das Unternehmen, das den digitalen Zwilling einsetzt, und den Anwender des Systems liegt hier in der permanenten Verfügbarkeit. So besitzt die Firma durch den digitalen Zwilling einer Maschine an einem gegebenenfalls fernen Standort über diese Maschine trotz der Distanz alle nötigen Informationen. Beispielsweise kann so präventiv eine Wartung angesetzt werden, wenn Messdaten von Sensoren von den bisher gemessenen Werten stark abweichen. Dies kann das Unternehmen nutzen, um die Wartung der eigenen Maschinen zu überwachen und zu optimieren.

Zudem kann im Nachhinein über den standardisierten Aufbau der digitalen Zwillinge einfacher Datenaustausch mit externen Unternehmen gewährleistet werden. Hierbei können unter anderem den Mechanikern vor Ort direkt Daten zugänglich gemacht werden, sodass diese mit dem Aufbau einer Maschine bereit vertraut sind, bevor sie die Maschine in der Firma gesehen haben. Ein solcher Digitaler Zwilling mit einigen Komponenten und Messwerten zeigt Abbildung 3.1.

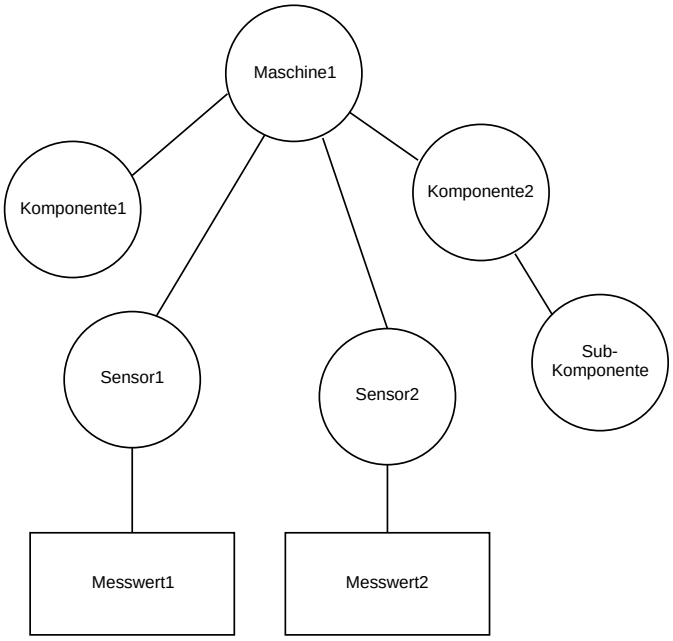

Abbildung 3.1: Beispielhafter Digitaler Schatten einer Maschine

3.2 Datenbankmanagementsysteme

Um die enorme Menge an Daten, die bei kontinuierlichen Sensormessungen entsteht, verarbeiten zu können, können Daten in Datenbankmanagementsystemen gespeichert werden. Diese Systeme ermöglichen, je nach Spezifikation, dass Daten strukturiert abgespeichert werden und effizient nach verschiedenen Kriterien durchsuchbar gemacht werden können. In Abbildung 3.2 ist der Aufbau eines solchen Datenbankmanagementsystems dargestellt. Hier ist es der Benutzer, der eine Anfrage (z.B. zur Speicherung eines Datums) an ein Datenbankmanagementsystem sendet. Dieses nimmt die Anfrage entgegen und sorgt im Hintergrund dafür, dass bestimmte Prozesse angestoßen werden, die das Speichern eines Datums in der gekoppelten Datenbank ausführen.

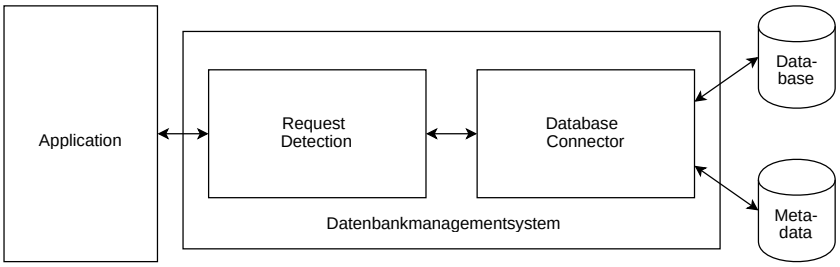

Abbildung 3.2: Aufbau Datenbankmanagementsystem

Es gibt mehrere Arten der Speicherung von Daten in Datenbanken. Manche speichern Daten relational in Tabellen ab (z.B. MySQL). Diese Datenbanken werden oft für relationale (tabellenförmige) Datensätze, beispielsweise Mitarbeiterdaten (Namen, Adressen, Beziehungsstatus, etc.), genutzt (siehe Tabelle 3.1). Relationale Datenbanken sind darauf spezialisiert, große Datenmengen zu speichern und durchsuchbar zu machen. Dabei kann immer mindestens ein Datensatz (oder auch eine Zeile in einer Tabelle) als Datum aufgefasst werden.

Tabelle 3.1: Relationaler Beispieldatensatz

ID	**Vorname**	**Nachname**	**Wohnort**	**Straße**	**Hausnummer**
241	Richard	Mustermann	Musterstadt	Musterstraße	4
242	Paul	Musterschmidt	Musterstadt	Musterweg	8
243	Patrick	Mustermaier	Musterstadt	Musterstraße	15

Datenbanken können aber auch auf andere Arten Daten speichern, beispielsweise können sie hierarchisch gespeichert werden. Dabei werden Datensätze hierarchisch aufgebaut und Datensätze demnach verschachtelt abgespeichert, sodass in einem Oberbegriff mehrere Datensätze eines Unterbegriffs gespeichert werden können. Ein Beispiel hierfür wäre die Speicherung von Daten in einem XML-Format.

Des weiteren können Daten in einer Netzwerkstruktur abgespeichert werden. In dieser Graphenform werden Daten miteinander verknüpft, um Abhängigkeiten von Daten in der Datenbank darstellbar zu machen. Daten in einer solchen Form abzuspeichern

bietet den großen Vorteil, dass Relationen von Daten, die nicht zusammen an einem Ort gespeichert werden (z.B. in einer Tabelle in einer relationalen Datenbank) dennoch in dem Datenspeicher gehalten werden können.

Außerdem können Daten dokumentorientiert gespeichert werden. Ein Beispiel hierfür bietet die Elasticsearch-Datenbank, in der Daten beispielsweise in JSON-Dokumenten gespeichert werden können. Diese Dokumente können einfach erstellt und in die Datenbank gespeichert oder einfach aus der Datenbank gelesen und in Java Objekte umgewandelt werden. Das Ansprechen der Datenbank über Programmcode ist dank verschiedenen Frameworks einfach möglich (siehe Kapitel 3.4.2).

3.2.1 Neo4j Graph Database

Neo4j ist eine Java-basierte Open-Source-Graphdatenbank, die im Gegensatz zu relationalen Datenbanken Daten graphenbasiert speichert. Daten werden in Knoten unterteilt und können über Kanten in Relation gebracht werden können (siehe Abbildung 3.3). Dieser Graph kann mit Hilfe verschiedener Algorithmen durchsucht werden, um Daten und deren Verbindungen zueinander zu analysieren, beispielsweise mit einem Centrality-Algorithmus, der einen Wert für jeden Knoten in dem Graphen ermittelt, wie mittig dieser Knoten im Graph liegt oder einem Likeliness-Algorithmus, der einen Wert für einen Graphen ermittelt, wie sehr dieser einem anderen Graphen ähnelt. Aus diesen Analysen können nachher Aussagen über den Graphen oder den digitalen Schatten getroffen werden.

Für weitere Informationen zum Erstellen einer Neo4j-Anwendung oder das Aufsetzen einer Neo4j-Graphendatenbank kann das Buch “Neo4j Graph Data Modeling” von M. Lal hinzugezogen werden [11].

3.2.2 Elasticsearch

Elasticsearch ist eine Suchmaschine, deren Hauptaufgabe es ist, bestimmte Suchen auf einem gegebenen Datensatz auszuführen [12]. Elasticsearch kann Daten im JSON-

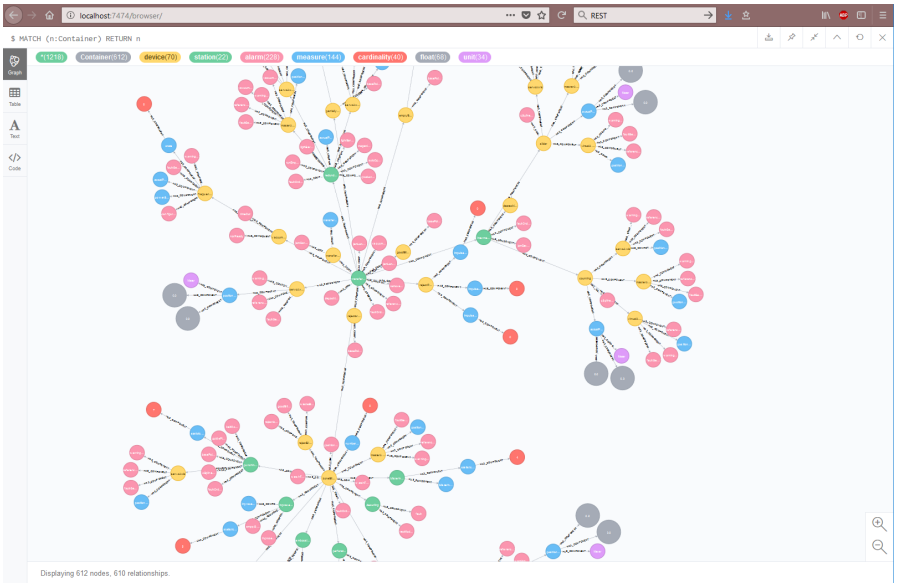

Abbildung 3.3: Beispielgraph

Format speichern, was eine effektive Möglichkeit ist, generisch erzeugte Objekte in einer Datenbank zu hinterlegen.

Die Spezialisierung auf das Ausführen von Suchen macht Elasticsearch zu einer hochperformanten und skalierbaren Lösung für die Analyse von in Objekten gespeicherten Datensätzen.

Zudem bietet Elasticsearch eine Weboberfläche namens Kibana, mit der sogenannte Dashboards (Sammlung von Diagrammen, die die Ergebnisse der Suche repräsentieren und strukturieren) erstellt werden können (siehe Abbildung 3.4).

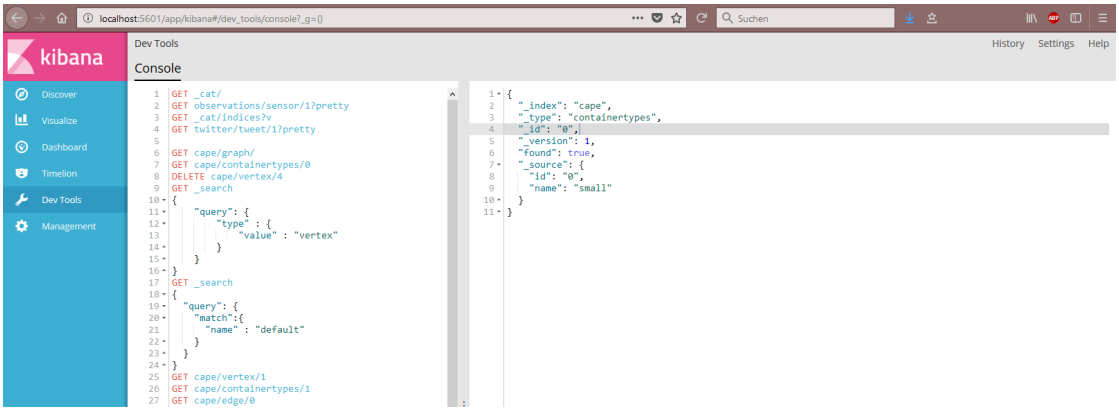

Abbildung 3.4: Beispiel der Kibana Oberfläche

3.3 Web Services

Ein Web Service ist ein Softwaresystem, das die funktionale Maschine-zu-Maschine-Interaktion über ein Netzwerk gewährleisten soll [13]. Über Web Services können diese Geräte über das Internet kommunizieren. Dafür wird die Netzwerktechnologie, wie beispielsweise HTTP, so verändert, dass maschinenlesbare Datenformate übertragen werden können (z.B. JSON oder XML).

3.3.1 Austauschformate

XML

XML ist eine strukturierte Beschreibungssprache, in der Daten hierarchisch strukturiert werden können und durch ein XML-Dokument übermittelt werden können [14]. Die Daten können intern in Container eingespeist werden, die dann wiederum mit Parametern versehen werden können. XML ist dabei universell verständlich und kann daher auf verschiedensten Systemen eingesetzt werden.
Ein Beispiel für ein XML Dokument, das ein Fahrzeug mit seinen Komponenten in hierarchischer Reihenfolge beschreibt, kann wie folgt aussehen:

```
<schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema>
    <Fahrzeug baujahr="2003", farbe="blau">
        <Motor anzahlZylinder="2">
            <Zylinder1 material="Stahl">
                <Kolben1 durchmesser="2.3"/>
            </Zylinder1>
            <Zylinder2 material="Stahl">
                <Kolben2 durchmesser="2.3"/>
            </Zylinder2>
        </Motor>
    </Fahrzeug>
</schema>
```

Listing 3.1: XML Dokument

JSON

JSON ist ein kompaktes Datenformat, mit dem es möglich ist, Daten und Objekte sowohl für Maschinen als auch für Menschen einfach lesbar abzuspeichern oder zu übermitteln [15]. JSON wurde für JavaScript entwickelt und ist auch als solches lesbar. Es ist demnach einfach, mittels JSON Java-Objekte zu serialisieren, sodass diese dann an andere Rechner oder Methoden gesendet werden können.
JSON bietet zudem den Vorteil, dass es weniger Rechenaufwand als XML benötigt, um Objekte zu (de-)serialisieren. XML erfordert einen externer Parser, der die Objekte während der Laufzeit erstellt. Da der JSON-Parser nahezu nativ arbeitet, wird viel Aufwand bei der (De-)Serialisierung eingespart.
Ein Beispiel für ein JSON-Dokument, das ein Auto mit seinen Komponenten beschreibt, könnte wie folgt aussehen:

```
{
    "Hersteller": "Mazda",
    "Modell": "BK-03",
    "Baujahr": 2003,
    "Beschreibung": "Schraegheckklimousine",
    "Motor":{
        "Leistung": "103PS",
        "Zylinderanzahl": 6,
    },
    "Raeder":{
        "Groesse": "17Zoll",
        "Saison": "Winter"
    }
}
```

Listing 3.2: JSON Dokument

3.3.2 Schematisierung

JSONSchema

JSONSchema ist eine Beschreibungssprache für Schemata von JSON-Dokumenten [16]. Mit JSONSchema kann beispielsweise festgelegt werden, dass in einem Feld für eine Jahreszahl kein Buchstabe vorkommen darf. JSONSchema kann verwendet werden, wenn Objekte aus Datensätzen generiert werden und sorgt dafür, dass diese den vorgegebenen Regeln entsprechen.

OpenAPI Interface Specification

Um REST-Methoden und deren Aufruf-URLs zu dokumentieren, bietet sich die OpenAPI Interface Specification an [17]. Diese bietet, unter Verwendung entsprechender Frameworks, eine Sammlung aller über eine REST-Schnittstelle aufrufbaren Methoden an (siehe Abbildung 3.5).

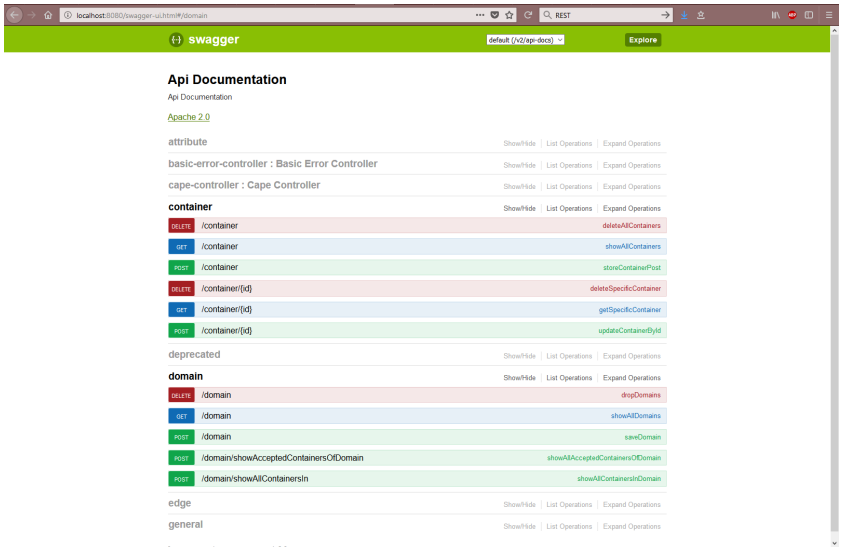

Abbildung 3.5: OpenAPI Beispiel

Hierbei bietet die OpenAPI Interface Specification verschiedene Möglichkeiten und Tools an, um eigene APIs zu verwalten. Es können von Grund auf neue APIs erschaffen

oder bereits bestehende editiert werden. Zudem bietet Swagger die Möglichkeit, APIs zu schreiben und dann aus diesen APIs direkt Code zu generieren. Swagger bietet auch eine interaktive Dokumentation zu den APIs an, die dem Endnutzer zur Verfügung gestellt werden kann, sodass dieser sich einfacher in dem Programm zurechtfindet. Des Weiteren bietet Swagger eine Testumgebung für APIs an, mit denen eigene REST-Methoden aufgerufen und getestet werden können. Swagger kann zudem eine Einsicht in die Methoden gewähren und zeigt beispielsweise das erforderte JSON Format an, in dem Daten an einen bestimmten Endpunkt gesendet werden müssen. Aus diesem Schema des Aufrufs kann das Datenmodell hergeleitet, da diese Informationen meist auch den Aufbau bestimmter Datenklassen beinhalten.
Weiterhin bietet Swagger auch die Option, Rückgabewerte an die eigenen Bedürfnisse anzupassen. So können HTTP Fehlercodes auf die eigenen Methoden und Fehlerfälle zugeschnitten genutzt werden, um dem Nutzer ein Gefühl dafür zu geben, wo der Fehler beim Aufruf lag.

3.3.3 URL

Eine URL identifiziert und lokalisiert eine Ressource. Ein bekanntes Beispiel hierfür ist das adressieren von Seiten im Internet. Über die URL ist somit ein Pfad angegeben, wo die angezeigte Ressource (Website) liegt.
Natürlich lassen sich mit URLs weit mehr als nur Internetadressen identifizieren. Durch Strukturierung bestimmter Teilkomponenten können beispielsweise genaue Identifikatioren zugewiesen werden. Für ein Auto gäbe es dann beispielsweise eine URL für den Motor: Auto://Motorinnenraum/Motor und eine URL für das rechte, hintere Rad: Auto://Räder/hinten/rechts. So können verschiedene Teile eines großen Ganzen genau lokalisiert und strukturiert werden. Beispielsweise kann sich ein Nutzer über die URL Auto://Räder alle Räder des Autos ansehen, da alle Räder diesem Pfad untergeordnet sind.

3.3.4 Representational State Transfer (REST)

REST bezeichnet ein Programmierparadigma für verteilte Systeme [18]. REST soll hierbei einen Architekturstil schaffen, der die Anforderungen des modernen Web besser darstellt. REST kann genutzt werden, um über das Internet Daten und Objekte zwischen weit entfernten Rechnern zu transferieren. Zum Beispiel kann über das HTTP-Protokoll mittels REST auf Funktionen auf anderen Rechnern zugegriffen werden. Diesen sogenannten REST-Calls kann dann noch ein Payload angehängt werden, der dann automatisch an das Zielsystem übertragen wird. So können beispielsweise mit JSON erstellte Datensätze zwischen Rechnern über das Internet ausgetauscht werden. Zudem bietet REST eine weitere Abstraktionsebene zwischen Client und Server (siehe Abbildung 3.6), sodass bei Codeänderungen eine Seite der Schnittstelle beibehalten werden kann, was einen angenehm wartungsfreundlichen Vorteil bietet.
Da REST auf HTTP basiert, kann auf jeden Aufruf, beziehungsweise jede Anfrage eine Antwort vom anderen Rechner folgen, um den Ablauf der Aktion zu kommentieren. Beispiele für solche Antworten oder “Statuscodes” sind 200-OK oder 404-NOT FOUND, wenn die gesuchte Ressource, zum Beispiel ein Datum in einer Datenbank, nicht gefunden wurde.

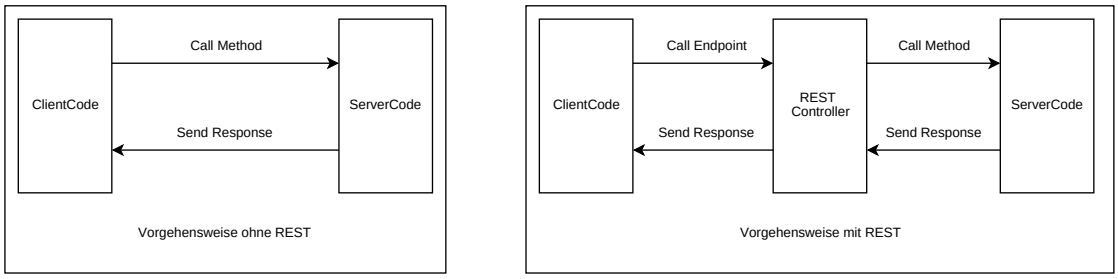

Abbildung 3.6: Ablauf eines REST-Aufrufs

3.4 Spring Framework

Das Spring Framework ist ein quelloffenes Dependency-Injection-Framework für Java, das die Modularisierung von Softwarekomponenten unterstützt [19]. Das Spring Fra-

mework bietet Hilfestellung bei dem gesamten Lifecycle einer Applikation. Spring Boot ermöglicht es, eine Applikation so schnell wie möglich und ohne großartige Konfiguration vorab lauffähig zu machen. Spring bietet hierbei einfache Möglichkeiten, beispielsweise mit wenig Aufwand einen REST-Service zu erstellen. Des weiteren bietet Spring einfache Werkzeuge, um eine SQL- oder eine NoSQL-Datenbank mit der eigenen Applikation zu verknüpfen. Spring enthält auch Hilfsmittel, um Applikationen direkt in einer Cloud mittels microservice-style zu entwickeln. Zudem bietet Spring Werkzeuge an, die eigene Applikation mit Mobilgeräten oder einfachen Sensoren zu verbinden, sodass diese kommunizieren können. Dependency-Injection ist das Schlüsselwort, um die Parameter und Abhängigkeiten eines Objekt während der Laufzeit zu bestimmen. Dieses Framework bietet also die Möglichkeit, Objekten, die bei der Erstellung ein anderes Objekt benötigen, ein zentral hinterlegtes Objekt als Abhängigkeit zu übergeben, sodass das Objekt erstellt werden kann. So wird die Verantwortlichkeit für das Verwalten des Aufbaus der Abhängigkeiten zwischen den verschiedenen Java-Objekten an eine zentrale Komponente übergeben. Das Spring Framework bietet eine Vielzahl von Methodiken und Techniken, um verschiedene Java-Einsatzgebiete abzudecken [20]. Zudem bietet das Spring Framework einige Erweiterungen an, die für noch speziellere Fälle einfach zu dem bestehenden Framework hinzufügen werden können.

Die verschiedenen Erweiterungen für das Spring Framework sind in Abbildung 3.7 dargestellt. Im Folgenden wird kurz aufgezeigt, wofür die einzelnen Erweiterungen genutzt werden können.

Spring AMQP bietet Hilfsmittel, um AMQP-basierte Messaging-Lösungen zu entwerfen. Spring Batch bietet Möglichkeiten, robuste Batch-Programme zu entwickeln. Spring BeanDoc ist eine Erweiterung, die Spring Bean-Factories dokumentiert und Graphen auf den Daten der Beans erstellt. Spring Boot bietet einfache Werkzeuge um alleinstehende Applikationen zu programmieren, die so schnell wie möglich ausgeführt werden können. Spring Extensions bietet die Möglichkeit, eigene Spring-Erweiterungen einzubinden. Spring IDE ist ein Plugin für die Eclipse IDE, das bereits vorgefertigte Plugins für Eclipse enthält. Der Sinn hinter Spring Data ist es, ein einheitliches, Spring-basiertes Programmmodell zu entwickeln, mit dem der Zugriff auf Daten in einer Datenbank er-

leichtert wird. Spring LDAP bietet Hilfsmittel, mit Spring Applikationen zu entwickeln, die das Lightweight Directory Access Protocol nutzen. Spring Integration erweitert das Spring-Framework um eine Möglichkeit, Applikationen zu entwickeln, die Enterprise Integration Patterns nutzen. Spring OSGi bietet Werkzeuge, Spring Applikationen zu entwickeln, die das OSGi-Framework nutzen. Spring Social bietet eine Vereinfachung für den Zugriff auf verschiedene Social Networks. Spring Web Services hilft bei der Erstellung von Contract-First-Webservices. Spring Roo hilft bei der raschen Generierung von Spring-basierten Enterpriseanwendungen. Spring MVC bietet Möglichkeiten zur Erstellung von Webanwendungen. Spring Security (ehemals Spring Acegi) bietet Werkzeuge zur Absicherung von Java-Anwendungen und Webseiten. ColdSpring ColdFusion ist die Portierung des Spring-Frameworks auf die ColdFusion-Plattform. Spring for Android ist eine Erweiterung, die das Erstellen von nativen Android-Applikationen erleichtern soll. Spring Web Flow bietet Hilfe bei der Implementierung von Abläufen auf einer Webseite. Spring Rich-Client hilft bei der Erstellung von Rich Clients auf Basis des Spring-Frameworks. Spring BlazeDS ist die Open-Source-Lösung zur Erstellung von Spring-unterstützten RIA-Anwendungen mit Adobe Flex. Letztendlich bietet Spring .NET eine Portierung des Spring-Frameworks auf die .NET Plattform.

Die beiden implementierten Erweiterungen werden im Folgenden genauer erläutert.

3.4.1 Spring Boot

Spring Boot ist eine Erweiterung des Spring Frameworks, das den Anwender in der Programmierung selbstständig laufender Programme unterstützen soll. Solche Programme werden initial einmal angestoßen, laufen dann selbstständig weiter und warten beispielsweise auf Befehle oder Daten für die Verarbeitung. Mit solchen Programmen können auch REST-Schnittstellen betrieben werden, welche dauerhaft darauf warten, dass über die Schnittstelle Befehle eintreffen, die dann bearbeitet werden können.

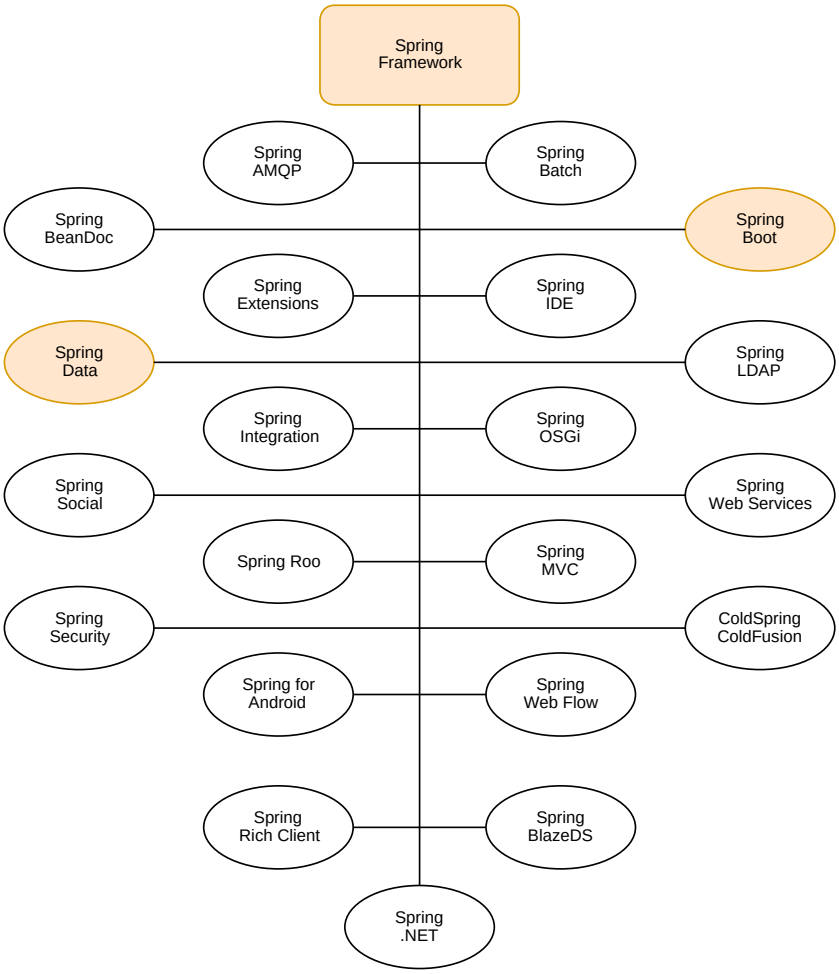

Abbildung 3.7: Übersicht der Spring Framework Extensions

3.4.2 Spring Data

Spring Data ist eine Erweiterung des Spring Frameworks, das die Anbindung an eine Vielzahl verschiedener Datenbanken über Java-Code ermöglicht. Mit Spring Data können Datenbanken direkt gesteuert werden und ihnen Befehle für Speichervorgänge oder die Suche auf Daten übermittelt werden. Aufgrund der einfachen Anbindung von Java-Code an die Datenbanken können alle wichtigen Vorgänge direkt im Java-Code gesteuert oder auch Datenbankoperationen in REST-Schnittstellen aufgelistet werden, sodass direkt über einen Befehl im Browser Änderungen in der Datenbank vollzogen werden.

4

Lösungskonzept

Dieses Kapitel stellt ein Konzept zur Erstellung von Digital Twins unter Berücksichtigung der zuvor erhobenen Anforderungen vor. Diese Anforderungen werden zunächst ausgewertet, um einen genauen Überblick über das zu erstellende System zu erhalten.

4.1 Anforderungsauswertung

Aus den bisher erfassten Anforderungen können bestimmte Funktionen abgeleitet werden, die das Projekt zur Erfüllung der Anforderungen enthalten muss.

F1 Das System benötigt mehrere Parser, die die eingespeisten Daten zunächst in ein einheitliches Format überführen (hier z.B. JSON).

F2 Es muss eine Möglichkeit geben, das System über Schnittstellen ansprechen zu können, sodass mehrere verschiedene Quellen Daten an das selbe System senden können.

F3 Um die Daten des Digitalen Schattens, also Messwerte und Metadaten, verbinden zu können, wird eine Datenbank benötigt, die diese Beziehungen darstellen kann (hier z.B. Neo4j).

F4 Um die Messdaten direkt an Module anheften zu können, wird eine generische Darstellung benötigt, die Daten frei verbinden kann. Dies kann direkt in der Datenbank geschehen.

F5 Um Daten selbstständig sammeln zu können, wird eine Schnittstelle benötigt, die im Programmcode angesprochen werden und empfangene Daten automatisch weiterverarbeiten kann.

F6 Eine Datenbank muss gewisse Voraussetzungen erfüllen, um Daten strukturiert abspeichern zu können. Zudem wäre es von Vorteil, wenn die Datenbank selbst bereits Suchfunktionen auf den enthaltenen Daten anbietet (trifft beides auf Neo4j zu).

F7 Eine Schnittstelle, mit der möglichst dynamisch gearbeitet werden kann, ist REST, daher wäre dies perfekt geeignet, um auch von mehreren Personen und Systemen gleichzeitig angesprochen zu werden.

F8 Eine bereits integrierte Authentifizierungstechnologie ist in Neo4j enthalten.

4.2 Architektur

In diesem Kapitel wird zunächst ein typischer Ablauf im System beschrieben. Daraufhin folgt eine Beschreibung der einzelnen Systemkomponenten.

4.2.1 Systemablauf

Eine Maschine produziert während ihrer Laufzeit meist dauerhaft Sensordaten (z.B. Messwerte). Diese Sensordaten werden über einen Webservice an den Server gesendet. Der Server empfängt dann die Sensordaten und konvertiert diese in ein einheitlich benutzbares Format (z.B. JSON). Die Sensordaten werden dann nach der Konvertierung über den Datenbankconnector an die angeschlossene Datenbank gesendet. Diese Speichert die Sensordaten so ab, dass sie schnell und effektiv durchsucht werden und mit anderen, bereits bestehenden Datensätzen in Verbindung gebracht werden können. Ein Nutzer des Systems kann dann über einen Webservice die verknüpften Datensätze in einer Datenbank einsehen und Analysen und Suchen auf den Datensätzen ausführen. Nutzer können zudem auch manuell Daten über einen Webservice in die Datenbank einfügen. So können bestimmte Metadaten an z.B. Messwerte angefügt werden.

4.2.2 Datenbankaufbau

Im Folgenden werden die einzelnen Komponenten der Datenbankverbindung dargestellt.

Datenschnittstelle

Über die Datenschnittstelle kann der Code auf dem Server angesprochen werden (siehe Abbildung 4.1).

Möchte ein Nutzer eine gewisse Funktion auf einem Server ausführen, muss dieser zunächst den Befehl an den Server übermitteln. Diesem Befehl müssen gegebenenfalls noch einige Nutzdaten mitgesendet werden, die der Server dann verarbeiten kann. Um dem Server einen Befehl zu übermitteln, muss der Nutzer mit diesem über eine Datenschnittstelle kommunizieren. Der Nutzer setzt zunächst einen Request (Anfrage) über die Datenschnittstelle ab. Dieser Request wird auf dem Server angenommen und weiterverarbeitet. Der Server setzt nun eine Anfrage ab, die vom Database-Connector empfangen wird. Dieser führt die geforderte Aktion auf der Datenbank aus und empfängt die Daten von der Datenbank. Diese Daten werden dann vom Server aufgenommen und über die REST-Schnittstelle an den Nutzer zurückgegeben.

Neo4j-Datenmodell

Mit Hilfe der graphenbasierten Datenbank Neo4j lassen sich Daten in einem graphbasierten Format abspeichern. Dabei wird das Datum als Knoten gehalten und kann mit jedem anderen Datum verbunden werden. So können beliebig komplexe Graphen aufgebaut werden, die nicht nur die Daten selbst, sondern auch ihre Beziehungen zueinander darstellen. Daten werden im System in einem Container gehalten. Dieser kann alle Daten darstellen, egal ob Messwert, Modul oder Sensor. Container werden anhand ihrer URL oder URI gekennzeichnet. Zudem können Container mit einem Label versehen werden, um darstellen zu können, worum es sich bei dem Datencontainer handelt (z.B. Messwert). Container sind untereinander mittels Relationships verbunden. Diese Relationships stellen dar, welche Container inwiefern zu einander in Relation stehen.

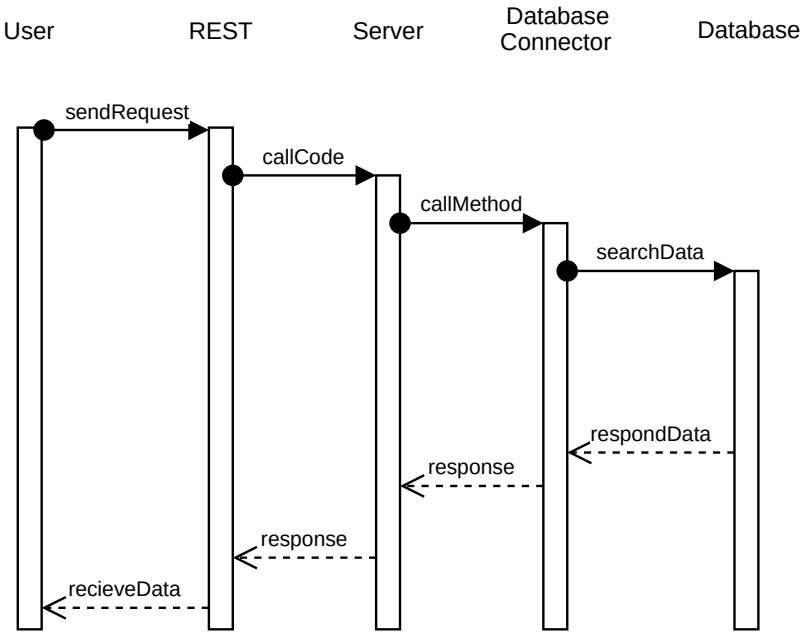

Abbildung 4.1: REST-Sequenzdiagramm

Relationships haben eine bestimmte Gültigkeit. Sie enthalten sowohl den Start- und Endcontainer der Beziehung als auch einen Zeitraum, in dem die Relationship gültig ist. Liegt dieser Zeitraum in der Vergangenheit, wird die Beziehung dementsprechend markiert. Die aktuellste Beziehung wird immer mit dem Label ”latest” gekennzeichnet. Des weiteren können Container und ihre Beziehungen nach bestimmten Kriterien geordnet werden. Container können dann in sogenannten ”Domains” strukturiert werden. Beispielsweise kann es eine ”Parkplatz”-Domain geben, in der nur bestimmte Containertypen erlaubt sind. Der Messwert eines Schrittzählers auf dem Mobiltelefon gehört demnach nicht in die ”Parkplatz”-Domain. Abbildung 4.2 zeigt das erarbeitete Datenmodell.

Datenbankconnector

Der Datenbankconnector wird von dem Spring Framework-Add-on Spring Data bereitgestellt. Dieses Add-on sorgt dafür, dass sogenannte Repositories Methoden zur Verfügung stellen, um mit der Datenbank zu kommunizieren. Diese Methoden sind von Haus aus mitgeliefert und bieten zum einen die üblichen CRUD-Methoden, als auch eine

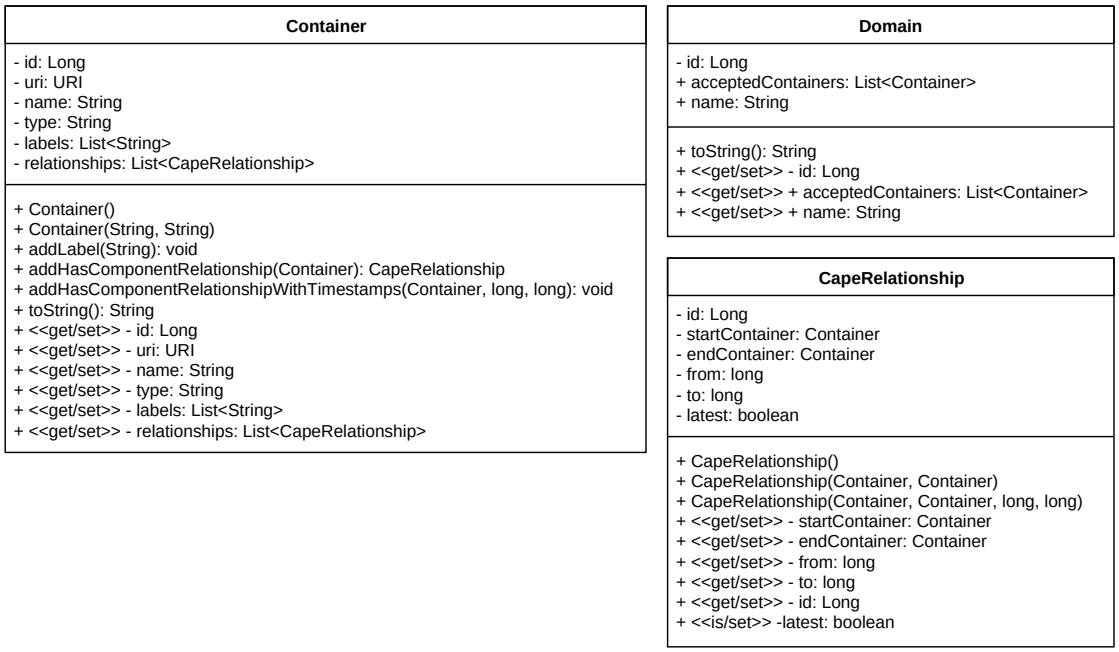

Abbildung 4.2: Datenmodell

Möglichkeit, eigene Methoden zu definieren (z.B. `findById()`). Die Kommunikation mit der Datenbank kann dadurch wie folgt dargestellt werden (Abbildung 4.3). Hierbei werden die Daten, die über den REST-Controller empfangen werden, von einem Service entgegengenommen. Dieser stößt dann die entsprechende Methode im Repository an. An einem Repository hängt immer die zugrundeliegende Datenklasse, die dem Repository eine genaue Beschreibung über den Aufbau und der Daten liefert. Das Repository führt dann die gewünschte Aktion aus und liefert die Ergebnisse an den Service zurück, über den diese dann weiterverarbeitet oder wieder an den Controller und schlussendlich an den Nutzer zurückgegeben werden können.

4.2.3 Systemaufbau

Nachfolgend wird der Systemaufbau des Projekts beschrieben.

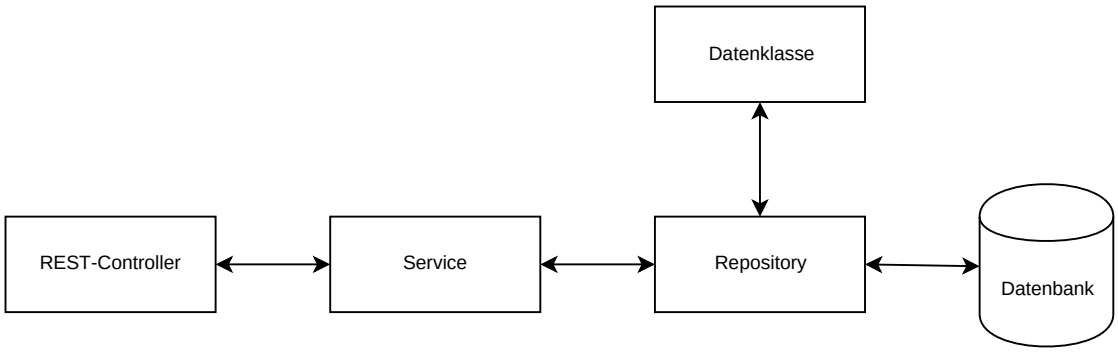

Abbildung 4.3: Datenbankconnector

Systemarchitektur - Überblick

Abbildung 4.4 zeigt die Systemarchitektur des erarbeiteten Konzepts. Hier sind die einzelnen Systemkomponenten und ihre Kooperation dargestellt.

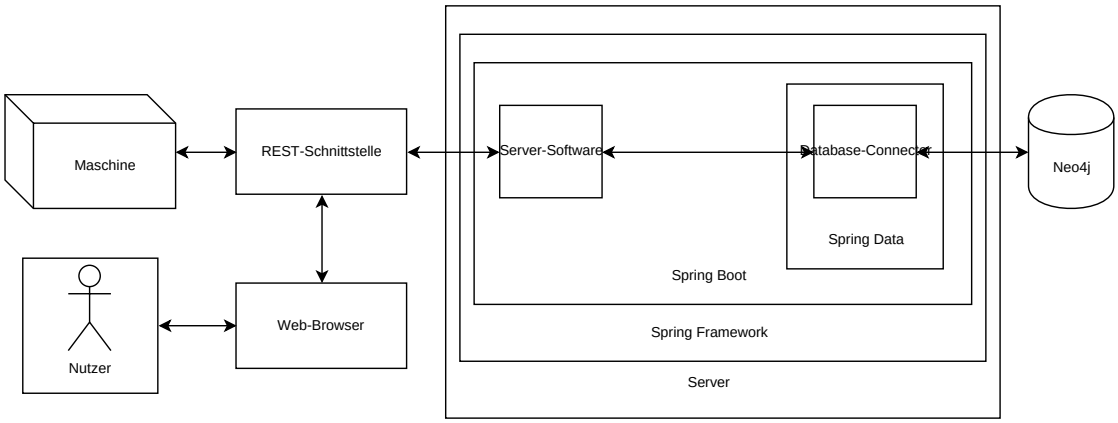

Abbildung 4.4: Systemarchitektur

Business Logic

Das Herzstück des System ist die verwaltende Geschäftslogik (engl.: Business Logic). Diese wird durch mehrere Komponenten implementiert, die nachfolgend erläutert werden.

Daten werden über den Web Service in das System eingegeben. Dort werden die Daten von einem Controller empfangen, der, je nach angesprochenem Endpunkt, weiteren Code ausführt und die empfangenen Daten weiterleitet. Dieser Code wird von einem Service zur Verfügung gestellt, der wieder über eine Schnittstelle an den Controller angebunden ist. In diesem Service werden die Daten dann mittels Algorithmen weiterverarbeitet und gefiltert. Diese Algorithmen können zudem auf die vom Spring-Framework erstellten Repositories zugreifen, um Daten in der Datenbank auslesen oder manipulieren zu können. So können im Zusammenspiel beispielsweise Algorithmen über die REST-Schnittstelle angestoßen werden, um bestimmte Statistiken auf den Daten zu berechnen. Aus diesen einzelnen Aufgabengebieten und der Kommunikation der Komponenten über Schnittstellen lässt sich wiederum folgendes Komponentendiagramm ableiten (Abbildung 4.5).

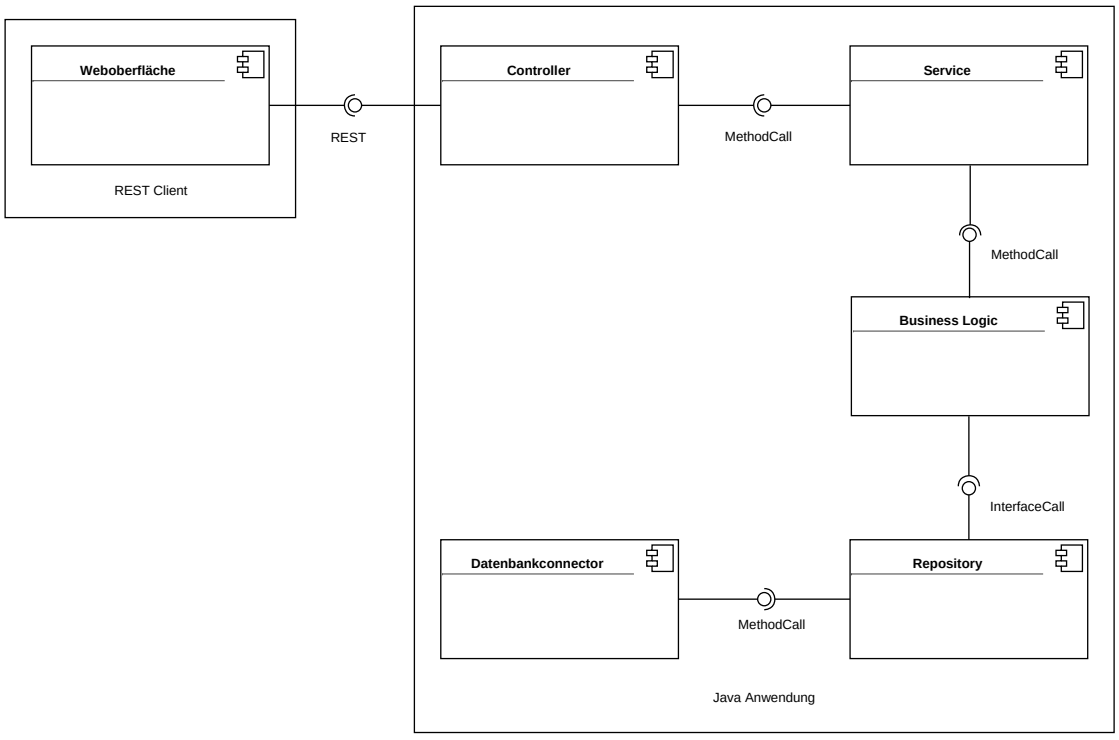

Abbildung 4.5: Komponentendiagramm

Klassendiagramm

Mithilfe dieses Schemas der einzelnen Komponenten lässt sich dann folgendes Klassendiagramm aufbauen, das den Aufbau des Projektes beschreibt (Abbildung 4.6):

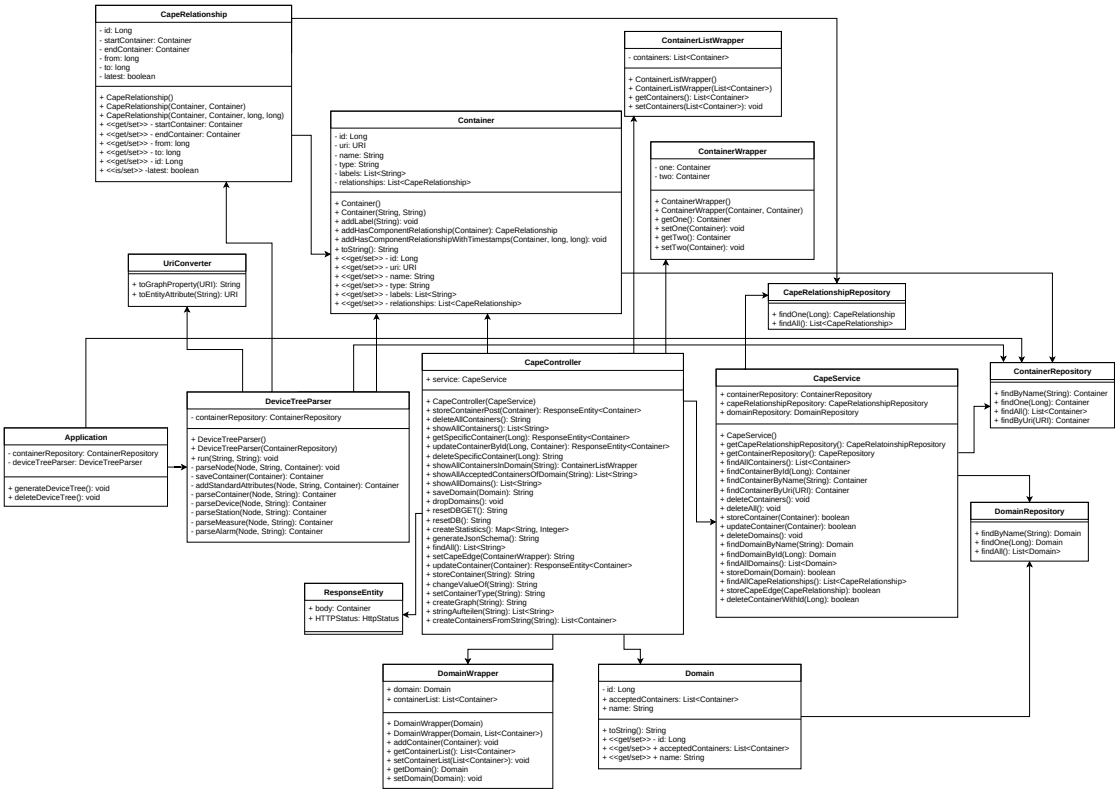

Abbildung 4.6: Vollständiges Klassendiagramm des Systems

Weiteres Hintergrundwissen zu den Themen Frameworks oder Architektur kann dem Buch “Moderne Enterprise Architekturen” von D. Masak entnommen werden [21].

4.3 Funktionalität

Um den vorher dargestellten Systemablauf implementieren zu können, wurden die Anforderungen an die Funktionalität des Systems analysiert und aus den Anforderungen und dem Systemablauf ein Modell gebildet, das sowohl den Ablauf widerspiegeln als auch

alle Funktionalitäten abdecken kann. Dieses Systemmodell wird zunächst beschrieben und in Abbildung 4.7 dargestellt.

Durch einen auf dem Server laufenden Parser wird gewährleistet, dass verschiedene Formate (F1) und verschiedene Quellen (F2) beim Verarbeiten der Daten genutzt werden können. Der Parser überführt alle Daten in ein einheitliches Format. Zusätzlich kann der Server dynamisch Daten von einer Maschine empfangen (F6). Der Server hat intern die Möglichkeit, Datensätze in der Neo4j-Datenbank zu verknüpfen (F3) und an bestimmte Knoten (in diesem Fall Module) zu koppeln (F4). Durch Neo4j kann auch eine Authentifizierung durchgeführt werden, die sicherstellt, dass nur autorisierte Personen Zugriff auf die Daten erhalten (F5). Die Daten können mithilfe von Neo4j schnell durchsucht werden, da sie bereits strukturiert gespeichert werden (F7). Eine Weboberfläche mit einer REST-Schnittstelle sorgt zudem dafür, dass Nutzer von verschiedenen Geräten aus auf die Daten zugreifen können (F8).

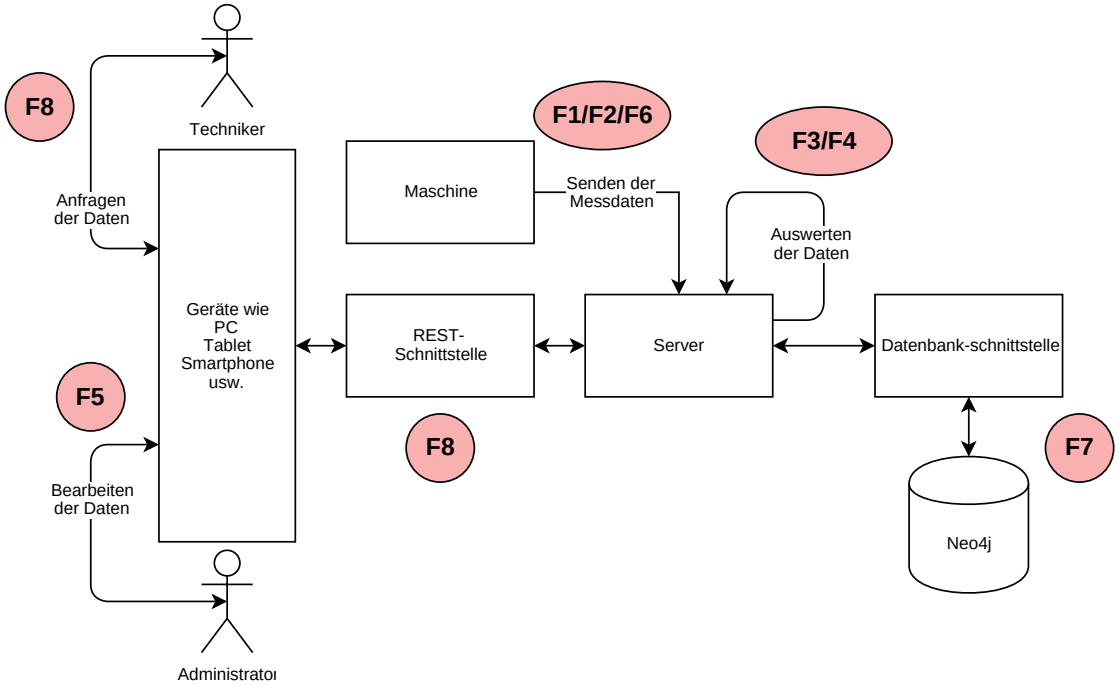

Abbildung 4.7: Lösungskonzept Use Case 1

5

Realisierung

Dieses Kapitel führt die Realisierung der vorher angegebenen Anforderungen auf und wie diese implementiert wurden.

5.1 Spring Class Management

Das Spring-Framework bietet die Möglichkeit, bestimmte Java-Klassen mit verschiedenen Annotations zu markieren, um ihren Zweck in dem System festzulegen. Zum Beispiel bietet Spring Boot das Werkzeug, einen Entry-Point für das Programm zu bestimmen (siehe Listing 5.1).

```
1 @SpringBootApplication
```

Listing 5.1: Entry Point Annotation

Der Builder weiß somit genau, wo die Start-Klasse ist, bei der initial Code ausgeführt werden muss. Diese Klasse stößt dann mit ihren Methoden sämtliche anderen Prozesse im System an. Dabei kann auch festgelegt werden, welche Methoden zum initialen Build gehören, und welche erst nach dem Build ausgeführt werden sollen (siehe Listing 5.2).

```
1 @PostConstruct
2 public void generateDeviceTree() throws IOException, URISyntaxException{
```

Listing 5.2: Bestimmung ob Code vor oder nach dem Build ausgeführt werden soll

Des Weiteren kann Spring Objekte aus einem zentralen Speicher nutzen, bevor sie im Code initiiert werden. Somit können Objekte mittels folgendem Code (siehe Listing 5.3) direkt genutzt werden und müssen nicht vorher über einen Konstruktor initiiert werden, für den vielleicht noch nicht alle Daten bereitliegen.

```
1 @AutoWired
2 ContainerRepository containerRepository;
```

Listing 5.3: Spring Annotation für REST Schnittstelle

5.2 REST-Schnittstelle

Die REST-Schnittstelle wurde mit Hilfe von Spring und einigen OpenAPI-Annotations realisiert. Spring bietet hierbei mehrere Möglichkeiten zur Strukturierung einer solchen REST-Schnittstelle. Zunächst kann die Klasse, die die REST-Schnittstelle beinhaltet, mit einer simplen Annotation für den Spring-Controller markiert werden (siehe Listing 5.4).

```
1 @RestController
2 public class CapeController {
```

Listing 5.4: Spring Annotation für REST Schnittstelle

Die einzelnen Methoden in dem Controller werden dann mit einer Annotation versehen, um einige Parameter festzulegen. So können sowohl der anzusteuernde Endpunkt als auch der Methodentyp im HTTP-Format festgelegt werden. Zusätzlich können den Methoden auch Daten in der URL oder als Payload mitgeschickt werden (siehe Kapitel 5.5).

```
1 @RequestMapping(value = "/container/{id}", method = RequestMethod.GET)
2 public ResponseEntity<Container> getSpecificContainer(@PathVariable("id")Long id){...}
3
4 @RequestMapping(value = "/container",method = RequestMethod.POST)
5 public ResponseEntity<Container> storeContainerPost(@RequestBody Container container){...}
```

Listing 5.5: REST-Methoden-Annotation

Die übertragenen Daten können dann entweder aus dem Pfad extrahiert werden (siehe Listing 5.6).

```
1 @PathVariable("id") Long id
```

Listing 5.6: Spring Annotation für REST Schnittstelle

oder im Format einer Java-Klasse direkt aus dem Body des HTTP-Calls (siehe Listing 5.7)

```
1 @RequestBody Container container
```

Listing 5.7: Spring Annotation für REST Schnittstelle

5.3 Neo4j-Datenbank

Danach wird die Datenbank einfach über eine Konsole initialisiert und kann dann über eine Weboberfläche, die standardmäßig über die URL localhost:7474 erreicht werden kann, genutzt werden.

Auf dieser Weboberfläche können dann entweder die Daten selbst oder die Änderungen an den Daten eingesehen werden. Visualisiert werden können die Daten hierbei bereits als dynamischen Graph mit allen Knoten und Kanten. Diese Knoten können außerdem nach Belieben farblich markiert werden, um eine einfachere Übersicht zu erreichen. In der Weboberfläche können zudem auch Queries ausgeführt werden, um beispielsweise den Status der Datenbank abzufragen oder Daten hinzuzufügen oder zu manipulieren.

5.4 Datenbankschnittstelle mit Spring

Spring bietet zusätzlich zu den bereits oben genannten Annotations weitere Markierungen, um beispielsweise die Datenklassen des Systems zu markieren. So können unter Anderem mit (siehe Listing 5.8)

```
1 @EnableNeo4jRepositories
```

Listing 5.8: Repositories initiieren

die vorliegenden Repositories geladen werden, um der Neo4j-Datenbank bereits die Struktur den eintreffenden Daten übergeben zu können. Außerdem können sich die Datenklassen, die hierfür in einem separaten Package liegen sollten, mit der Annotation (siehe Listing 5.9).

```
1 @EntityScan("cape.neo4j.model")
```

Listing 5.9: Scan nach Datenklassen

selbstständig scannen lassen, sodass Neo4j und Spring bereits die datenbeinhaltenden Klassen von den rein funktionalen Klassen unterscheiden können.
Die bereits erwähnten Repositories sind vorgefertigte Interfaces und bieten die Methoden,

die genutzt werden, um mit der Datenbank zu kommunizieren. Sie werden mit der folgenden Annotation versehen (siehe Listing 5.10), sodass über den REST-Client der Status eines Repositories abgefragt werden kann. In diesem Beispiel wird markiert, dass es sich um das Repository handelt, das die Datenklasse ”Container” verwaltet.

```
1 @RepositoryRestResource(path = "containers")
```

Listing 5.10: Repository Annotation

Ein gesamtes Repository ist in Abbildung 5.11 dargestellt.

```
1  @RepositoryRestResource(path = "containers")
2  public interface ContainerRepository extends PagingAndSortingRepository<Container, Long> {
3      Container findByName(String name);
4
5      Container findOne(Long id);
6
7      List<Container> findAll();
8
9      Container findByUri(URI uri);
10
11     @Query("CALL algo.betweenness.stream('{containerType}','{edgeType}',{direction:'{direction}'})\n" +
12             "YIELD nodeId as id1, centrality\n" +
13             "MATCH (n:Container)\n" +
14             "WHERE ID(n) = id1\n" +
15             "RETURN n,centrality order by centrality desc limit 20;")
16     Iterable<Map<String, Integer>> calculateBetweenness(String containerType, String edgeType, String direction);
17
18     @Query("MATCH (n:Container) WHERE NOT (n)<-[:HAS_COMPONENT]-() RETURN n")
19     List<Container> findRootNodes();
20 }
```

Listing 5.11: Repository

Hierbei ist zu sehen, dass das Repository von einem sogenannten ”PagingAndSortingRepository” abstammt. Dieses bietet Neo4j die Möglichkeit, Daten seitenweise sortiert zurückzugeben (siehe Listing 5.12).

```
1 public interface ContainerRepository extends PagingAndSortingRepository<Container,Long>{
```

Listing 5.12: PagingAndSortingRepository

Des Weiteren sind hier die von Spring Data vorgegebenen Standardmethoden zu sehen, wie ”findByName” oder ”findOne”. Diese Methoden werden von Spring erkannt und ohne weiteren Aufwand für den Programmierer passend auf die Daten verlinkt (siehe Listing 5.13).

```
1 Container findByName(String name);
2 Container findOne(Long id);
```

Listing 5.13: Standardmethoden von Spring Data

Sollten diese Methoden nicht genügen oder sollten bestimmte Algorithmen ausgeführt werden müssen, bietet das Framework die Werkzeuge, eigene Methoden aus Queries zu bauen. So finden sich in dem Repository Methoden mit der Annotation ”@Query” wieder, welche die in Neo4j auszuführende Query enthalten (siehe Listing 5.14).

```
1 @Query("MATCH (n:Container) WHERE NOT (n)<-[:HAS_COMPONENT]-() RETURN n")
2 List<Container> findRootNodes();
```

Listing 5.14: Query Annotation

So muss vom Programmierer nur der Rückgabetyp und der Name der Methode festgelegt werden, um das gewünschte Ergebnis zu erzielen. Aus solchen Queries heraus können auch in Neo4j gespeicherte Algorithmen ausgeführt werden, wie in diesem Beispiel der ”betweenness”-Algorithmus. Diesem werden im Methodenaufruf auch direkt Parameter übergeben, die dann automatisch in die Query eingesetzt werden (siehe Listing 5.15).

```
1 @Query("CALL algo.betweenness.stream('{containerType}','{edgeType}',{direction:'{direction}'})\n" +
2             "YIELD nodeId as id1, centrality\n" +
3             "MATCH (n:Container)\n" +
4             "WHERE ID(n) = id1\n" +
5             "RETURN n,centrality order by centrality desc limit 20;")
6     Iterable<Map<String, Integer>> calculateBetweenness(String containerType, String edgeType, String direction);
```

Listing 5.15: Query Annotation für Algorithmen

5.5 API-Übersicht mit OpenAPI

OpenAPI bietet die Möglichkeit, die bereits annotierten REST-Methoden mit einigen Parametern zu beschreiben um daraus eine API-Dokumentation generieren zu lassen (siehe Listing 5.16).

```
1 @ApiOperation(value = "showAllContainers", notes = "Prints out all containers in the database",
2 response = String.class, responseContainer = "List", tags = "container")
```

Listing 5.16: OpenAPI Annotation

Hierbei lassen sich einige Werte wie der dargestellte Methodenname (value), ein eigener Kommentar (notes), den Rückgabetyp direkt als Java-Class (response), bestimmte Modifikationen des Rückgabetyps, z.B. eine Liste oder ein Array (responseContainer) und Tags festlegen, aus denen dann eine vollständig dokumentierte API generiert werden kann.

Um zu beeinflussen, wie die API nachher dargestellt wird, lässt sich dies mit Hilfe einer Config-Klasse verändern (siehe Listing 5.17).

```
1  @Configuration
2  @EnableSwagger2
3  public class SwaggerConfig {
4
5      @Bean
6      public Docket api(){
7          return new Docket(DocumentationType.SWAGGER_2)
8                  .select()
9                  .apis(RequestHandlerSelectors.any())
10                 .paths(PathSelectors.any())
11                 .build();
12     }
13
14     private ApiInfo apiInfo() {
15         return new ApiInfo(
16                 "CaPe_Neo4j_Rest_API",
17                 "This is the API of the CaPe_Neo4j-project",
18                 "0.1",
19                 "These are some custom TermsOfService",
20                 new Contact("John Doe", "www.example.com", "myeaddress@company.com"),
21                 "License of API", "API license URL", Collections.emptyList());
22     }
23 }
```

Listing 5.17: OpenAPI Config

Hierbei kann der Documentation-Type verändert werden oder Informationen über die API, das System und den Herausgeber hinzugefügt werden.

5.6 Zusammenfassung

Der in Kapitel 4 konzipierte Lösungsansatz konnte mit Hilfe des Spring-Frameworks prototypisch umgesetzt werden. Spring Data bietet eine große Hilfestellung bei der Kommunikation eines Systems mit einer Datenbank. Zusätzlich ist das Spring-Framework im Bezug auf die Programmierung von REST-Schnittstellen sehr nützlich. Hier bietet die

OpenAPI zudem viele Möglichkeiten, die genutzten REST-Methoden zu dokumentieren und im Programmcode zu strukturieren.

6

Evaluierung

In diesem Kapitel folgt die Gegenüberstellung der Problemstellung und der Use Cases mit dem fertigen Prototypen. Inwiefern das Projekt die geforderten Funktionalitäten umsetzt, wird anhand einer Tabelle aufgeführt.

Tabellen 6.1 und 6.2 führen alle Anforderungen auf, die das erarbeitete Systemkonzept umsetzen soll (siehe Kapitel 2).

Diese funktionalen Anforderungen wurden in dem System wie in Tabelle 6.3 umgesetzt, die nicht-funktionalen Anforderungen wie in Tabelle 6.4.

F1	Heterogene Datenmodelle	Daten sollen in verschiedenen Formaten eintreffen und so verarbeitet werden, dass sie nachher ein einheitliches Format besitzen.
F2	Unterschiedliche Datenquellen	Daten sollen aus verschiedenen Quellen eintreffen und verarbeitet werden können.
F3	Beziehung zwischen Entitäten	Daten über die physikalische Struktur eines Systems sollen mit Messwerten verbunden werden können, um so weitere Aussagen treffen zu können.
F4	Gruppierung von Daten	Messdaten sollen direkt an die entsprechenden Module in denen sie aufgenommen wurden, gekoppelt werden können.
F5	Dynamische Akquisition von Daten	Das System soll selbstständig während der Laufzeit Daten sammeln, verarbeiten und abspeichern können.
F6	Effiziente Suchfunktionen	Das System soll so aufgebaut sein, dass Daten strukturiert abgespeichert werden, sodass Suchen auf den Daten möglichst schnell ablaufen.
F7	Standardisierte Schnittstellen	Personen sollen möglichst agil mit dem System kommunizieren können und über eine einzige Schnittstelle Zugriff auf alle Daten gleichzeitig bekommen können.
F8	Authentifizierung	Personen, die Daten abrufen oder bearbeiten sollen nur Zugriff auf die Daten haben, für die sie auch autorisiert sind.

Tabelle 6.1: Funktionale Anforderungen an das System

NF1	Data Awareness	Unternehmen sollen sich der Daten bewusst sein, die sie sammeln und der Möglichkeiten, die sie mittels dem Sammeln der Daten haben.
NF2	Generischer Aufbau	Der Digital Twin soll so generisch wie möglich werden, um möglichst viele Einsatzgebiete abdecken zu können.
NF3	Geschwindigkeit	Alle Auswertungen, Suchen und das Laden der Daten sollen so schnell wie möglich (möglichst unter einer Minute) ablaufen.
NF4	Wiederverwendbarkeit	Der Code soll so geschrieben werden, dass große Teile des Codes wieder-verwendet werden können.
NF5	Wartbarkeit	Der Code soll einfach verständlich und gut dokumentiert sein, sodass er einfach zu warten ist.
NF6	Skalierbarkeit	Das Projekt soll hoch skalierbar sein. Das heißt es sollen Tools genutzt werden, die eine möglichst hohe Skalierbarkeit gewährleisten können.

Tabelle 6.2: Nicht-funktionale Anforderungen an das System

F1	Heterogene Datenmodelle	Daten können im JSON- oder XML- Format übertragen werden und werden so verarbeitet, dass sie nachher einheitlich in einem Containerformat vorliegen.
F2	Unterschiedliche Datenquellen	Daten können von verschiedenen Quellen aus über eine REST-Schnittstelle an das System übertragen werden.
F3	Beziehung zwischen Datensätzen	Das Verbinden der Daten geschieht durch das Verknüpfen mit Kanten in Neo4j automatisch. Diese Kanten können typisiert werden, um beliebige semantische Zugehörigkeiten darstellen zu können.
F4	Gruppierung von Daten	Messdaten können in Neo4j durch die Verbindung von Knoten direkt an Module gekoppelt werden. Hierbei werden neue Messwerte direkt mit neuen Kanten mit bestehenden Knoten verbunden.
F5	Dynamische Akquisition von Daten	Das System kann Daten, die über eine REST-Schnittstelle empfangen werden direkt und während der Laufzeit verarbeiten und abspeichern.
F6	Effiziente Suchfunktionen	Da die Daten in Neo4j in einem Graphen abgespeichert werden, sind die Daten automatisch strukturiert. Aufgrund dieser Tatsache können Suchen und Algorithmen schnell und effizient durchgeführt werden.
F7	Standardisierte Schnittstellen	Nutzer des Systems können mit der REST -Schnittstelle kommunizieren um zusammen auf dem gleichen, aktuellen Datensatz arbeiten zu können. Dabei kann die REST-Schnittstelle von verschiedenen Systemen aus (Mobiltelefon, Tablet, Desktop) angesprochen werden.
F8	Authentifizierung	Neo4j bietet eine bereits bestehende Nutzerverwaltung, die genutzt werden kann, um den Zugriff auf die Daten zu beschränken.

Tabelle 6.3: Umsetzung der funktionalen Anforderungen

NF1	Data Awareness	Die anfallenden Daten (z.B. Messwerte) einer Maschine, die im Betrieb entstehen, können mit dem System empfangen und abgespeichert werden. So können Unternehmen einen einfachen und schnellen Überblick über die Daten bekommen, die bei der Produktion auftreten.
NF2	Generischer Aufbau	Durch den Aufbau aus Containern ist das System sehr generisch. Sämtliche Daten können in Containern abgespeichert werden, sodass verschiedene Systeme in der Datenbank abgebildet werden können.
NF3	Geschwindigkeit	Das Suchen und Auswerten der Daten ist sehr schnell. Das initiale Laden wird durch die Parser begrenzt, welche hierbei einen Flaschenhals darstellen.
NF4	Wiederverwendbarkeit	Der Code kann durch eine ausführliche Code-Dokumentation und durch die Nutzung vieler Schnittstellen einfach wiederverwendet werden.
NF5	Wartbarkeit	Der Code ist sowohl mit einer externen API-Spezifikation als auch mit einer Code-internen JavaDoc dokumentiert.
NF6	Skalierbarkeit	Dem System können beliebig viele Daten zugeführt werden. Der Flaschenhals ist zum momentanen Zeitpunkt der Parser.

Tabelle 6.4: Umsetzung der nicht-funktionalen Anforderungen

7

Verwandte Konzepte

Nachfolgend werden verwandte Konzepte und Softwarelösungen vorgestellt.

7.1 Cumulocity

Cumulocity ist eine Device Cloud Plattform, deren Ziel es ist, Geräte miteinander zu vernetzen [22]. Maschinen und Geräte sollen so verknüpft werden, dass sie zentral und über eine Cloud angesteuert werden können und durch das Überprüfen der von den Maschinen gesendeten Messdaten überwacht werden können. Die Geräte werden hierbei mit einer Software gekoppelt, die in einer Cloud läuft. Die einzelnen Geräte senden sowohl Metainformationen, wie die Identität, den Standort oder die Softwareversion, als auch dynamische Daten wie Events, Alarme und Fehlermeldungen. Nutzer können sich von einem beliebigen Standort aus mit dem Cumulocity-Server verbinden und die Daten sowie den Status der einzelnen Geräte einsehen. Des Weiteren kann über die Software auf Events eingegangen oder Fehlerfälle behoben werden (sofern dies über die Distanz möglich ist). Dafür stellt Cumulocity einige Tools zu Verfügung, die über Webservices bedient werden können. Es können auch Standardregeln erstellt werden, die im Falle eines Fehlers genutzt werden. So können beispielsweise bei verschiedenen Fehlern (z.B. fehlende Messwerte, Slowdown des Systems etc.) verschiedene Alarme erstellt werden und an verschiedene Positionen gesendet werden. Außerdem ist es möglich, das Gerät aus der Distanz neu zu starten oder dem Gerät einfache Befehle wie z.B. ein Firmwareupdate zu senden.

Um Aufwand zu sparen, stellt Cumulocity eigene Connectors, sogenannte ”Agents” zur Verfügung, die diverse Maschinen mit unterschiedlichen Firmwares, also verschiedenen

Softwares, die die Bedienung einer Maschine ermöglichen, miteinander verbinden können.

Das Unternehmen Cumulocity sorgt mit seiner Implementierung dafür, dass die Daten geschützt werden und für Analysen aufbereitet werden können. Cumulocity bietet bereits integrierte Real-Time-Streaming Analysen, die ein effektives Werkzeug für z.B. Predictive Maintenance darstellen können.

7.2 Watson IoT

Die Firma IBM hat mit Hilfe ihres KI-Tools Watson eine eigene Implemetierung für eine IoT-Lösung entwickelt [23]. Die IoT-Lösung von IBM enthält unter anderem Teile wie die Erstellung eines Digitalen Zwillings oder den Einbau einer Blockchain für zusätzliche Sicherheit. Durch diese Blockchain können beispielsweise Möglichkeiten realisiert werden, wie Geschäftspartner auf IoT-Daten einer nutzenden Firma zugreifen können, ohne eine zentrale Verwaltungs- oder Managementstelle zu benötigen.

Durch das Einbinden von IBM Watson bietet diese IoT-Lösung auch KI-getriebene Ansätze wie die kognitiv unterstützte Defektsuche bei produzierten Teilen. So kann diese Lösung über das Vergleichen von optischen Sensorwerten mit zuvor eingespeisten Mustern Defekte in Produkten erkennen, die einem Menschen nur schwer auffallen würden.

7.3 AWS IoT

Amazon bietet auf Basis der Amazon Web Services (AWS) eigene IoT-Lösungen an [24]. Diese können entweder als Infrastructure as a Service (IaaS-), Platform as a Service (PaaS-) oder als Software as a Service (SaaS-) Lösungen bezogen werden (siehe Abbildung 7.1).

Das Produkt AWS IoT Core von Amazon kann die Kommunikation von verschiedenen Geräten ermöglichen, selbst wenn diese unterschiedliche Protokolle zur Datenübertragung

nutzen. Dabei kann AWS IoT Messages von verschiedenen Inputs entgegennehmen, diese auswerten und an verschiedene Outputs weiterleiten. Dort können dann andere Dienste von AWS wie zum Beispiel AWS Lambda die Nachrichten entgegennehmen und anhand dieser bestimmte Funktionen ausführen. So kann mit AWS IoT schnell und einfach eine simple Lüftersteuerung mit eingehenden Temperaturdaten realisiert werden [24].

Zusätzlich bietet Amazon mit AWS Greengrass die Möglichkeit, Geräte auch ohne Verbindung zum Internet über ein lokales Netzwerk kommunizieren zu lassen, um so schnell und unabhängig auf lokale Events reagieren zu können.

Außerdem können eventuell gespeicherte Daten mit Amazon S3 mit einer hohen Beständigkeit gespeichert und sicher und direkt abgefragt werden. Amazon S3 bietet unter anderem die Möglichkeit, komplexe Big-Data-Analysen auf den Daten ausführen zu können, ohne die Daten in ein separates Analysesystem verschieben zu müssen. Die Daten in Amazon S3 können mittels Amazon Athena on-demand mit SQL abgefragt werden. Amazon S3 Select bietet zusätzlich die Möglichkeit, nur Teildatensätze aus der Datenbank abzurufen, um so die Leistung der meisten Anwendungen, die häufig auf Daten in S3 zugreifen, um bis zu 400% zu steigern.

Für weitere Informationen zu AWS oder für Hilfe bei der Implementierung eigener AWS kann das Werk “Programming Amazon Web Services” von James Murty hinzugezogen werden [25].

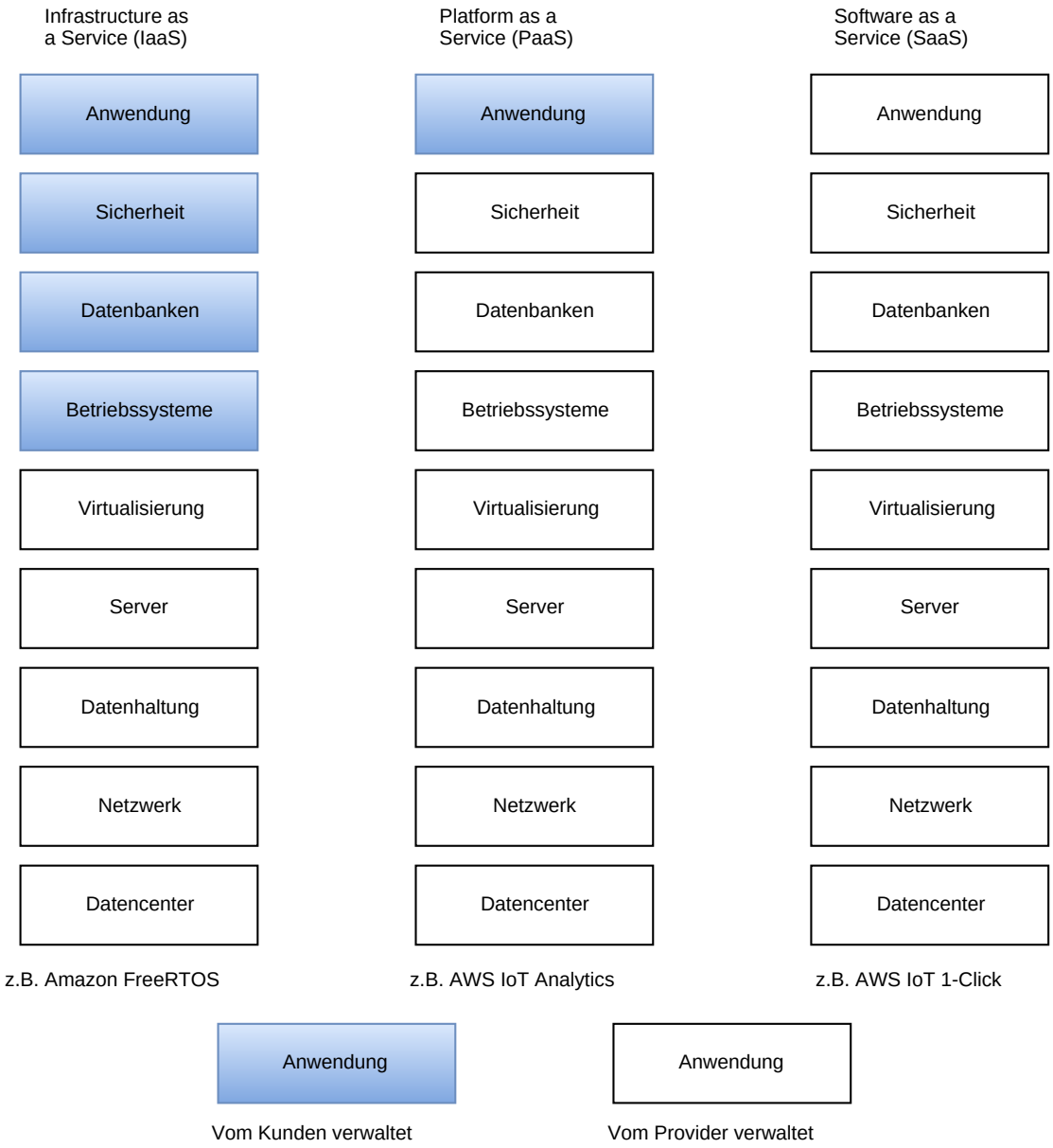

Abbildung 7.1: Übersicht der AWS-Produkte

8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Die Problemstellung umfasste das Problem, dass physische Objekte digitalisiert werden sollten. Dieser digitale Zwilling soll dann mit Messdaten und Metadaten des Objekts angereichert werden, um eine möglichst funktionale und eine möglichst realitätsnahe Abbildung des Objektes zu realisieren.

Das erarbeitete Konzept beinhaltet die Möglichkeit, mittels einer Java-Anwendung und einer Datenbank einen digitalen Zwilling zu erzeugen und zugehörige Mess- und Metadaten zu speichern. Dazu ist eine einfache Kommunikation mit dem System über eine Webservice-Schnittstelle möglich. So können sowohl Maschinen ihre Messdaten selbstständig über die Webservice-Schnittstelle an das System übermitteln, als auch Nutzer mit dem System kommunizieren, um administrative Aufgaben auszuführen oder Daten manuell hinzufügen oder löschen zu können.

Der erstellte Prototyp setzt die Anforderungen des Konzeptes und der Problemstellung um. Der Prototyp nutzt ein Spring-Framework, um eine effektive Ausführbarkeit und eine einfache Anbindung an die Datenbank zu realisieren. Als Datenbank wurde die graphorientierte Neo4j-Datenbank verwendet, um die Daten in der Datenbank in Relation setzen zu können und so beispielsweise Messdaten mit ihren jeweils zugehörigen Sensoren verbinden zu können. Mit dem Prototyp kann zusätzlich über eine REST-Schnittstelle kommuniziert werden, um Daten hinzuzufügen, zu ändern, zu löschen oder Algorithmen (z.B. zur Zentralität der einzelnen Knoten im Graphen) auf den Datensätzen ausführen zu können.

8.2 Ausblick

Das System setzt die bisherigen Anforderungen gut um. Jedoch werden in der Zukunft weitere Funktionen von dem System gefordert. Demnach ist das System noch nicht zu 100% ausgebaut. Des Weiteren gibt es noch manche Punkte, die eventueller Nachbesserung bedürfen. So können sicherlich noch einige Performanceverbesserungen am Code vorgenommen werden.

Zusätzlich wäre es von Vorteil, wenn zunächst eine performantere Lösung für die Parser gefunden werden könnte, da diese momentan einen Flaschenhals (engl.: Bottleneck) darstellen: Das initiale Laden der Daten läuft nicht so schnell ab, wie initial gewünscht war. Des Weiteren sollten die Parser auf mehr verschiedene Systeme und Formate ausgeweitet werden, um so viele Formate wie möglich verarbeiten zu können.

Eine andere Erweiterung stellt die Möglichkeit dar, die zugrundeliegenden Graphen für Analysezwecke gewichten zu können. So können Graphen für verschiedene Analysezwecke nach unterschiedlichen Vorgaben gewichtet werden, um ein präziseres Ergebnis zu erzielen.

Literaturverzeichnis

[1] Jeschke, S., Brecher, C., Song, H., Rawat, D.: Industrial Internet of Things: Cybermanufacturing Systems. Springer Series in Wireless Technology. Springer International Publishing (2016)

[2] Hehenberger, P., Bradley, D.: Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and their Designers. Springer International Publishing (2016)

[3] Farhangi, H.: The Path of the Smart Grid. IEEE Power and Energy Magazine **8** (2010) 18–28

[4] Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer (2012)

[5] Maier, S.C., Todte, H.: Telematik–eine Revolution in der Kfz-Versicherung. Zeitschrift für Versicherungswesen **23** (2013) 776–782

[6] ADAC: ADAC Pannenstatistik 2017. `https://www.adac.de/infotestrat/unfall-schaeden-und-panne/pannenstatistik/` (2017) Zugriffsdatum: 07.03.2018.

[7] Dakić, D., Arh, D.: Nuget2 Essentials. Packt Publishing (2013)

[8] Media, S.: Maven: The Definitive Guide. O'Reilly (2009)

[9] McCullough, M., Berglund, T.: Building and Testing with Gradle. O'Reilly (2011)

[10] Siemens: Der Digitale Zwilling. `https://www.siemens.com/customer-magazine/de/home/industrie/digitalisierung-im-maschinenbau/der-digitale-zwilling.html` (2017) Zugriffsdatum: 27.02.2018.

[11] Lal, M.: Neo4j Graph Data Modeling. Packt Publishing (2015)

[12] Chhajed, S.: Learning ELK Stack. Packt Publishing (2015)

[13] Haas, B.: W3C Web-Services. Website (2004)

[14] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible Markup Language (XML). World Wide Web Journal **2** (1997) 27–66

[15] Severance, C.: Discovering JavaScript Object Notation. Computer **45** (2012) 6–8

[16] Pezoa, F., Reutter, J.L., Suarez, F., Ugarte, M., Vrgoč, D.: Foundations of JSON Schema. In: Proceedings of the 25th International Conference on World Wide Web. WWW '16, International World Wide Web Conferences Steering Committee (2016) 263–273

[17] Mulligan, C.: Open API Standardisation for the NGN Platform. In: 2008 First ITU-T Kaleidoscope Academic Conference - Innovations in NGN: Future Network and Services. (2008) 25–32

[18] Balachandar, B.: RESTful Java Web Services: A pragmatic guide to designing and building RESTful APIs using Java. Packt Publishing (2017)

[19] Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R., Risberg, T., Arendsen, A., Davison, D., Kopylenko, D., Pollack, M., et al.: The Spring Framework–Reference Documentation. Interface **21** (2004) 27

[20] Johnson, R., Höller, J., Arendsen, A., Risberg, T., Sampaleanu, C.: Professional Java Development with the Spring Framework. Wiley (2007)

[21] Masak, D.: Moderne Enterprise Architekturen. Xpert.press. Springer (2006)

[22] SoftwareAG: Cumulocity Homepage. `https://www.cumulocity.com` (2017) Zugriffsdatum: 08.04.2018.

[23] IBM: IBM Watson IoT Cognitive Processes. `https://www.ibm.com/internet-of-things/industries/iot-manufacturing/cognitive-process` (2018) Zugriffsdatum: 18.05.2018.

[24] Amazon: AWS IoT. `https://aws.amazon.com/de/iot/` (2018) Zugriffsdatum: 18.05.2018.

[25] Murty, J.: Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB. O'Reilly Series. O'Reilly (2008)

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Sven Bihlmaier Matrikelnummer: 827650

Erklärung

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

Ulm, den .....

Sven Bihlmaier