Carl von Ossietzky Universität Oldenburg
OFFIS - Institut für Informatik

LAOTSE
Abschlussbericht

PG SESAdata
Dennis Nowak, Kristian Bruns, Mark Milster
Lina Spiekermann, Uwe Griese

31. März 2016

Zusammenfassung

UG

Neben den schadhaften Emissionen von Kohlekraft, treiben die Endlichkeit und Preissteigerungen fossiler Energieträger, die Energiewirtschaft weltweit in Richtung der Nutzung von erneuerbaren Energien, wie z.B. Windkraft und Solarenergie. Um die Versorgungssicherheit der Verbraucher trotz der Schwankungen dieser natürlichen Quellen zu garantieren, soll das Stromnetz hin zu einer dezentralen intelligenten Lösung umgerüstet werden - ein sogenanntes Smart Grid.
Damit das Grid wirklich *smart* wird, sollte es sich auf das Verhalten von Erzeugern und Verbrauchern einstellen und auf plötzliche Abweichungen reagieren können. Für den Wandel der Energiesysteme ist die Entwicklung und Einführung neuer, intelligenter Komponenten zwingend notwendig. Das Ziel dieser Simulation ist es das Prototyping neuer Komponenten effizient und kostenünstig zu ermöglichen, bevor aufwändige und teure Feldversuche durchgeführt werden. Ein naheliegendes Vorgehen zur Voraussage dieser Werte ist eine virtuelle Simulation eines Smart Grids mit all seinen Bestandteilen, während gleichzeitig Daten von realen Sensoren gesammelt und ausgewertet werden. Eine solche Simulation kann am OFFIS Oldenburg im Smart Energy Simulation and Automation Laboratory (SESA-Lab) umfassend durchgeführt werden und produziert eine sehr große Menge an Messwerten in extrem kurzer Zeit. Für die Verarbeitung und Analyse dieser Datenmengen sowie die Administration der Datenquellen benötigt das SESA-Lab eine Erweiterung, die diese Aufgaben erfüllt.
Die Projektgruppe SESAdata hat das System LAOTSE entwickelt um hochfrequente Datenstromquellen via Livemonitoring zu verwalten und deren Messwerte langfristig zu archivieren. Des Weiteren stellt LAOTSE sowohl ein Analysetool für die Durchführung als auch ein Graphical User Interface (GUI) für die Visualisierung von Analysen auf diesen Langzeitdaten zur Verfügung. Für die Lösung dieser Aufgabe wird das Datenstrommanagementsystem (DSMS) Odysseus genutzt, welches die Daten anschließend an eine Datenbank übergibt, die für die längerfristige Speicherung zuständig ist. Neben der Mamutaufgabe der verlustfreien Speicherung der zum Teil hochfrequenten Massendaten, ist die Analyse- und Visualisierungsmöglichkeit der Daten essenziell. Dazu ermöglicht die integrierte Analysekomponente des LAOTSE-Systems die komfortable und flexible Datenanalyse. Darüber hinaus werden Standardmetriken (MIN, MAX, AVERAGE und RANGE) sowie Betrachtungsfunktionen (Dateneingang der Sensoren oder Sensorausfälle) angeboten, welche über ein Dashboard visualisiert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

UG

Informationen sind in der heutigen mit Informations- und Kommunikationstechnologien durchdrungenen Welt eine der wichtigsten Triebfedern und gelangen zu immer größerem Wert. Die Industrie, Unternehmen und Forschungseinrichtungen haben ein großes Interesse, auf der Grundlage von zuverlässigen Informationen, Planungs- und Entscheidungssicherheit zu erlangen.

1.1. Motivation

UG

Mit zunehmender Verbreitung der Informations- und Kommunikationstechnologie wurde die Ressource Information immer bedeutender und führte zur heutigen Informationsgesellschaft. Informationen sind in der heutigen Zeit allgegenwärtig und haben sich in vielen Bereichen zu einem bedeutsamen Wertschöpfungsfaktor etabliert. So ist es auch im Bereich der Energiesysteme. Mit deren Wandel zu intelligenten Smart Grids steigt das Datenaufkommen, durch den Einsatz von moderner IKT mit unzähliger Sensorik und Aktorik, rasant und erfordert zunehmend die Berücksichtigung des Themengebietes Big Data (vgl. [Sma]).
Gemäß der Definition des Anwendungsgebiets Big Data (siehe Abschnitt 2.4.4) resultiert das „Big“ aus der Datenquellenvielzahl „variety“, der zum Teil sehr hohen Übertragungsgeschwindigkeit im kHz-Bereich „velocity“ und des daraus entstehenden Datenvolumens „volume“.
Das SESA-Lab im OFFIS bietet eine Testumgebung für Smart-Grid-Technologien, in der diese kostengünstig erprobt werden können. Dabei können u.a. Funktionsweisen von experimentellen Technologien simuliert werden. In dem betrachteten Projektumfeld sollen das Datenaufkommen dieser Simulationen sowie bis zu 1000 teilweise heterogene Sensoren mit maximalen Übertragungsgeschwindigkeiten von 30 kHz berücksichtigt werden. Eine erste Abschätzung des Mengengerüstes der anfallenden Daten verdeutlicht schnell, warum „Big Data“ produziert werden.

Nach fachlicher Meinung sind derzeit Simulationen mit bis zu 50.000 Konten, welche sekündlich Messwerte liefern, durchaus denkbar. Bei einer Simulationsdurchführung in Realzeit, werden jede Sekunde 50.000 Messwerte erzeugt. Im Falle einer Messwertübertragung mittels JSON-String an das Zielsystem und einer Datensatzgröße (Sensorname 12 Zeichen, Messwert 10 Zeichen und Overhead 5 Zeichen) von 27 Zeichen, wird jeder Messwert mit einer Größe von 27 Byte kodiert (UTF-8-Kodierung 1 Byte pro Zeichen). Demzufolge würde ein Datenvolumen von **1.350.000 Byte/s** $\approx$ **1,29 MB/s** anfallen. Eine einzelne Simulation erzeugt in 24 Stunden **108,63 GB Daten und 4,32 Mrd. Messwerte**.

Angenommen 1000 Sensoren erzeugen mit einer durchschnittlichen Übertragungsrate von 500 Hz konstant Daten. Somit würden bei einer Speichergröße von 128 Bit (64 Bit Zeitstempel und

64 Bit Messwert) für jeden erfassten Messwert, **8 Mio. Byte/s ≈ 7,63 MB/s** erzeugt werden. An nur einem einzigen Tag entstünden **643,73 GB Daten und 43,2 Mrd. Messwerte**.

Mit Zunahme der Datenquellen und Übertragungsraten steigt der Umfang und damit die Anforderungen an die Speicherung und Analyse der Daten.
Die Leitfrage lautet, wie sich die gewünschten Informationen aus den unzähligen Datenquellen ermitteln lassen, um die erhofften Erkenntnisse zu erlangen.
Das Stichwort lautet Datenanalyse. Vor diesem Hintergrund befinden sich viele Unternehmen, Institutionen und Forschungseinrichtungen vor einer großen Herausforderung. Gewaltige Datenmengen, stetiger und extrem rasanter Datenzuwachs, vielschichtige IT-Systeme, sowie die Datenquellenvielfalt verhindern einen schnellen und umfassenden Einblick.
Das Resultat sind Massendaten, die zu groß und komplex sind oder sich zu schnell ändern, um mit manuellen oder klassischen Methoden der Datenverarbeitung gespeichert und ausgewertet werden zu können. Daraus haben sich die Einsatzgebiete Big Data und Datenmanagement gebildet, die sich primär mit Technologien und Konzepten zum Sammeln, Speichern und Auswerten dieser Massendaten beschäftigen. Wünschenswert wäre es, diese großen Datenmengen zeitnah und effizient verarbeiten zu können und dabei umfassenden Zugriff sowie Analysemöglichkeiten auf die anfallenden Datenmengen zu bekommen.

1.2. Projektziel

Ziel des Projektes SESAdata ist die Erstellung einer Datenmanagementumgebung, die dazu dient Forschungsprojekten im Bereich Smart Grid eine Plattform zur Speicherung von Energiedaten bereitzustellen. Dafür müssen die zum Teil sehr hochfrequenten und von unzähligen Sensoren stammenden Daten mit Zeitstempeln verlustfrei verarbeitet und persistent gespeichert werden.
Darüber hinaus sind Analysen und Visualisierungen sowohl auf Echtzeit- als auch auf gespeicherten Langzeitdaten zu ermöglichen. Die größte technische Herausforderung liegt dabei in der hochfrequenten Verarbeitung und verlustfreien Speicherung der Messwerte mit ihren genauen Messzeitpunkten.
Das Projektszenario für die Projektgruppe SESAdata umfasst die persistente Speicherung und Analyse verschiedener Messdaten. Als Datenquellen sind dabei drei Kernbereiche zu integrieren:

1. Reale Messwerte von Sensoren in Windparks die beispielsweise Schallpegel, Temperaturen oder Einspeisespannungen messen.

2. Simulierte Messwerte aus der Co-Simulationsplattform mosaik.

3. Über Fernwirkprotokolle angebundene externe Komplettsysteme, wie der Blockheizkraftwerk (BHKW)-Container in der Lesumstraße in Oldenburg.

Zur Verarbeitung der Datenströme ist das, für andere Projekte bereits erfolgreich eingesetzte, DSMS Odysseus zu verwenden.

1.3. Aufbau

LS

Der vorliegende Abschlussbericht beschreibt die endgültigen Ergebnisse des Projektes SESAdata. Um dem Leser einen guten Einstieg in das Projekt, sowie eine gute Nachvollziehbarkeit zu ermöglichen, werden in Kapitel 2 zunächst die Grundlagen der Hauptbestandteile des Projektes erklärt.
Kapitel 3 beschreibt die erste Vision des Zielsystems und stellt mit dem Grobkonzept den ersten Lösungsansatz einschließlich der Projektanforderungen vor. Nachdem der Leser mit dem Lösungsansatz und der Spezifikation des Zielsystems vertraut ist, stellt das Kapitel Projektmanagement 3, nach der Vorstellung des Vorgehensmodells und der Projektorganisation, den Projektablauf detailliert da.
Im folgenden Kapitel 5 wird mit der Darstellung der Realisierung begonnen, indem die eingesetzten Technologien erläutert werden. Das Kapitel Systemdesign beschreibt den Konzeptionsprozess und die Ergebnisse bezüglich aller Systemkomponenten von LAOTSE mit Hard- und Softwaredesign sowie der genutzten Datenbanken.
Es folgt eine Erläuterung der genauen Zusammenhänge in der Implementierung von LAOTSE in Abschnitt 7.
Das vorletzte Kapitel 8 beschreibt die Vorgänge zur Qualitätssicherung für das zu erstellende Produkt, indem die eingesetzten Tools, Vorgehensweisen und Ergebnisse beschrieben werden.
Es folgt die Evaluation von LAOTSE in Kapitel 9 bezüglich des Projektziels und der gestellten Anforderungen.
Zurückblickend auf den Projektverlauf findet sich in Kapitel 10 ein Resumee.
Kapitel 11 bietet einen Ausblick auf zukünftige Einsatzgebiete und Optimierungsmöglichkeiten von LAOTSE.

2. Grundlagen

Dieses Kapitel beschreibt grundlegende Techniken, die für das Verständnis dieser Arbeit relevant sind.

2.1. SESA-Lab

UG

In der Energiebranche hat bereits vor einigen Jahren ein radikaler Wandel begonnen. Mit zunehmender Verbreitung der Informations- und Telekommunikationstechnologien, der zunehmenden Auslastung der Netze, sowie der flexiblen Energieeinspeisung, wurde die Betriebsführung elektrischer Transport und Verteilernetze zunehmend dynamischer und intelligenter. Der Wandel führt zu immer mehr Informationsaustausch, Komplexität und Automatisierung (vgl. [Sma])
Um diese Veränderungen zu unterstützen und vorantreiben zu können, ist eine Weiterentwicklung der IKT-Infrastruktur unumgänglich. Die Energieinformatik beschäftigt seit einigen Jahren mit der Modellierung, Simulation und Optimierung von intelligenten Stromnetzen (sogenannten Smart Grids). Bisher hatten die virtuellen Modelle und Tests ihre Restriktionen und konnten nicht alle denkbaren Szenarien, zeitkritische Effekte oder Fehlersituationen in elektrotechnischen sowie mechanischen Systemteilen zufriedenstellend berücksichtigen.
Abhilfe soll eine parallel ausgeführte Soft- und Hardwareumgebung schaffen, die neuartige Smart Grid Lösungen realitätsnah testen kann. Das SESA-Lab besteht zum einen aus Hardware zur Anbindung einzelner Komponenten des Smart Grids, wie beispielsweise einem OPAL-RT Echtzeitsimulator, sowie zum anderen aus dem Co-Simulationsframework mosaik, welches Netze oder Teile davon virtuell simuliert. Diese Laborumgebung ist vom OFFIS in Zusammenarbeit mit der Universität Oldenburg geschaffen worden.
Das Testlabor ermöglicht es verschiedene Komponenten komplexer Energieversorgungssysteme, z.B. Steuer- und Regelungstechnologien, unter realistischen Bedingungen in einem virtuellen System realitätsnah zu testen. Das ermöglicht neuartige IKT-Lösungen intelligenter Energiesysteme vor ihrem Einsatz umfassender zu evaluieren. Somit ist ein risikoreduzierter, kostensparender und schnellerer Ausbau der intelligenten Energienetze möglich (vgl. [Mü+13], [Ses]).

2.2. Smart Grid Simulationen mit mosaik

LS

Mosaik ist ein mächtiges Simulationsframework mit dem auf relativ einfache Weise große Smart Grid Simulationen ausgeführt werden können.
In mosaik können Simulatoren gestartet werden, die Modelle der Teilnehmer eines Smart Grids

enthalten. Diese können anschließend verbunden, in einem selbst geschriebenen Szenario ausgeführt und ihre Stromauslastung etc. beobachtet werden.
Mosaik ist in Python programmiert und ist open-source verfügbar. Im Folgenden werden die Hauptbestandteile von Mosaik nach [Mos] detaillierter beschrieben.

Mosaik API Die Mosaik API ist für die Kommunikation zwischen mosaik und den angefügten Simulatoren zuständig. Dabei gibt es für den Nutzer zwei verschiedene Möglichkeiten diese zu implementieren, wie zu sehen in Abbildung 2.1:

1. Low-Level API: In dieser Variante muss der Entwickler die Verbindung inklusive JSON-Parser etc. selbst individuell anpassen.
2. High-Level API: Mosaik bietet bereits eine fertige Implementierung dieser JSON-Verbindung an, bei der ein neuer Simulator nach der Implementierung des `Base-Interfaces` sofort genutzt werden kann. Natürlich gibt es dabei weniger Spielraum für den Entwickler, da er den Anforderungen des Interfaces gerecht werden muss.

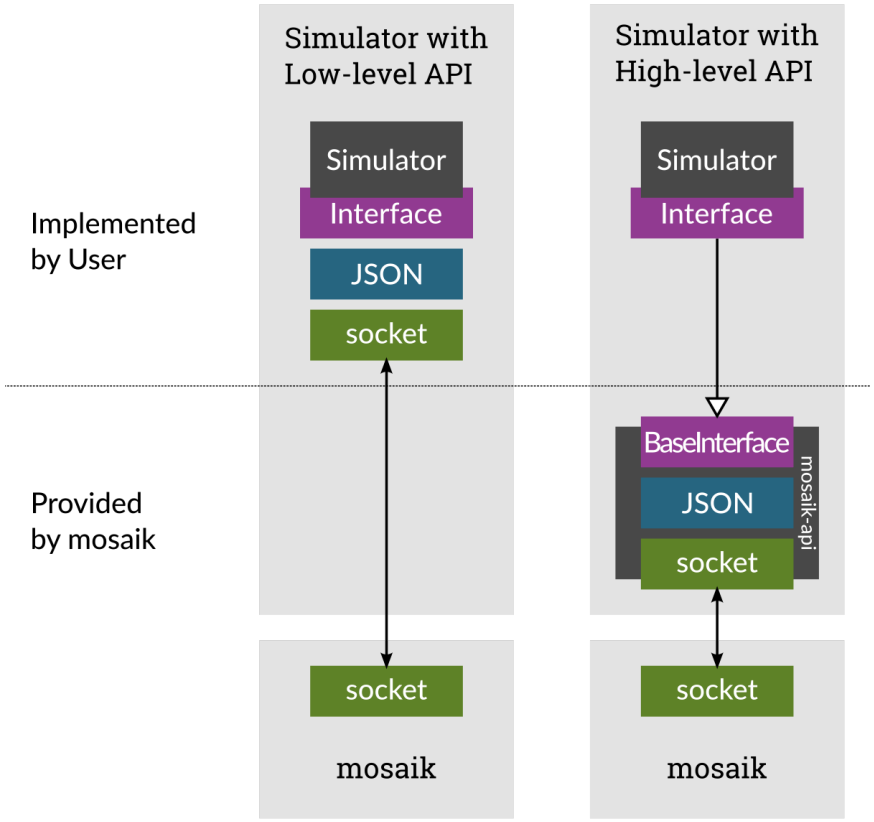


Abbildung 2.1.: Kommunikationsvarianten [Mos]

Szenario API Dieser Teil von Mosaik ist für den Aufbau und die Ausführung von Simulationen zuständig.

Zunächst muss das Szenario erstellt und geladen werden. Dieses wird aus verschiedenen Entitäten von Simulationsmodellen gebildet, welche die Komponenten der realen Umwelt repräsentieren und beispielsweise das Verhalten eines reales Ortsnetzes simulieren. In diesem Szenario können die *Entitäten* Stromverbraucher (Haushalte, E-Autos etc.) oder

Stromerzeuger (Kraftwerke, Photovoltaikanlagen etc.) repräsentieren. Die Topologie des zu simulierenden Netzes wird über die Verbindungen der einzelnen Entitäten gebildet, welche in diesem Szenario über „ Stromleitungen“ hergestellt wird

Ist das Szenario soweit erstellt, kann die Simulation gestartet werden.

Simulator Manager In mosaik können mehrere Simulationsprozesse gleichzeitig ausgeführt werden und dabei sogar untereinander Daten austauschen. Dies ist vor allem sinnvoll, da eine Simulation sehr speicherintensiv sein kann, obwohl viele Berechnungen parallel ausführbar sind. Solche Simulationen können prinzipiell in jeder Programmiersprache geschrieben werden, aber nur in Python 3 direkt importiert werden. Ansonsten funktioniert die Kommunikation über Network sockets.

Scheduling Das Scheduling von mosaik kommt besonders bei parallelen Simulatoren zum Tragen, wenn diese eine gegenseitige Datenverbindung besitzen. Ist dies der Fall müssen sie ggf. auf die Berechnung eines Datums warten um fortfahren zu können. Insbesondere kann es sein, dass verschiedene Simulatoren nicht im gleichen Takt arbeiten und so ein Simulator mehrere Schritte machen darf während ein anderer noch wartet.

2.3. Automatisierungsprotokolle

DN

In der Energieinfrastruktur werden eine Vielzahl an Protokollen eingesetzt, um Daten zwischen einzelnen Geräten zu übertragen. LAOTSE soll Daten von verteilten Sensoren sammeln und speichern können. Als Kommunikationsmedium steht Ethernet zur Verfügung. Es gibt zahlreiche Protokolle, um Daten über Ethernet zu übermitteln. Von der IEC wurden Standards zur Kommunikation innerhalb der Energieinfrastruktur veröffentlicht. LAOTSE soll über die gängigen Standards Daten von den Sensoren abrufen können. Da Odysseus beliebige Eingabequellen einbinden kann, sollten prinzipiell auch beliebige Standards unterstützt werden können. LAOTSE sollte mindestens ein Standardprotokoll unterstützen. Die zurzeit üblichen Standards werden in den folgenden Abschnitten beschrieben.

2.3.1. OPC UA

Open Platform Communications Unified Architecture (OPC UA) wurde von der OPC Foundation entwickelt. Es basiert auf Ethernet und ermöglicht eine Client/Server-Kommunikation. Es beinhaltet Komponenten zum Auslesen von Daten (Data Access), Auslesen von historischen Daten (Historical Data Access), Bereitstellung von Meldungen (Alarms & Subscriptions). Dabei wird beschrieben, welche Services die Server zur Verfügung stellen. Klienten können bspw. bestimmte Daten abonnieren, damit neue Daten automatisch übertragen werden.

2.3.2. IEC 61850

International Electrotechnical Commission (IEC) definiert eine Datenstruktur, in der die Daten von Sensoren eingeordnet werden. Die Geräte in einem physikalischen Energienetz werden nach IEC 61850 in logische Geräte eingeteilt. Diese logischen Geräte stellen zum einen definierte

Dienste zur Verfügung, beispielsweise die Übermittlung von Messwerten, zum anderen ist aber auch definiert, welche Art von Messwerten ein bestimmtes logisches Gerät bereithalten kann.
IEC 61850 kann auf unterschiedlichen Kommunikationsprotokollen realisiert werden. Im eigentlichen Standard wird ein Mapping auf die Manufactoring Message Specification (MMS) vorgesehen. Es gibt auch die Möglichkeit, ein Mapping auf OPC UA einzusetzen.
Neben den Mappings auf Client/Server-Protokolle stehen weitere Übertragungsmedien zur Verfügung. Generic Object Oriented Substation Events (GOOSE) kann dazu genutzt werden, bestimmte Werte an mehrere Adressen zu senden.
Die Konfiguration wird mit SCALA realisiert. Es wäre denkbar, dass die Daten aus SCL-Dateien dazu genutzt werden um Anfragen zu generieren, die die Datenquellen in Odysseus einbinden.test

2.4. Datenbanktechnologien

MM

In diesem Abschnitt werden einige Grundlagen zu verschiedenen Technologien von Datenbanken kurz erläutert, die in LAOTSE genutzt werden oder zur Auswahl stehen und daher mit anderen Datenbanken verglichen werden müssen.

2.4.1. Datenstrommanagementsystem

MM

Für die Überwachung von technischen Anlagen ist die Verarbeitung von teilweise hochfrequenten Messwerten notwendig. Diese müssen ausgewertet, bearbeitet, normiert und aggregiert werden, um eine sichere Betriebsführung zu gewährleisten. Anschließend sollen die Daten häufig visualisiert und einige dieser Daten persistent in einer Datenbank gespeichert werden. Dadurch können auch später Analysen auf historischen Daten vorgenommen werden [Hau08].
Eine Reihe solcher Daten, die in zeitlicher Abfolge erhoben werden, wird als Datenstrom bezeichnet. Es ist gefordert, dass diese Datenströme mit verschiedenen Operationen bearbeitet werden können. Für eine wiederverwendbare und flexible Softwarearchitektur sollte die Visualisierung bzw. sonstige Endanwendung der Daten von der Verarbeitung der Datenströme getrennt werden. Für die Verarbeitung der Datenströme ist ein DSMS geeignet, das Schnittstellen für den Eingang und den Ausgang von Daten bereitstellt. Anwendungsprogramme und Sensoren, die Messwerte erzeugen, nutzen Schnittstellen eines DSMS [Gra14].
Für die oben genannten Zwecke der Verarbeitung von Datenströmen ist ein DSMS im Gegensatz zu einem Datenbankmanagementsystem (DBMS) besser geeignet. Ein DBMS kann als Ergänzung genutzt werden, um selektierte Daten persistent zu speichern und diese historischen Messwerte zu verwalten. Der Vorteil eines DSMS ist, dass dieses die Daten nicht alle speichert und somit nahezu in Echtzeit verarbeitet. Dies ist besonders bei hochfrequenten Messungen, wie sie teilweise bei der Überwachung technischer Anlagen auftreten, sinnvoll.
Hingegen werden bei einem DBMS alle Daten persistent in einer Datenbank gespeichert. Das DBMS steuert, genau wie ein DSMS Anfragen an diese Daten. Bei einem DBMS wird allerdings, im Gegensatz zu einem DSMS, auf zuvor gespeicherte Datensätze zurückgegriffen. Daher eignet sich ein DBMS besser zur Archivierung und ein DSMS besser zur kurzzeitigen Betrachtung und

Verarbeitung von Daten, die in einem Strom auftreten, der teilweise sehr hochfrequent und flexibel sein kann.

2.4.2. In-Memory- und Langzeitdatenbanken

MM

In-Memory Datenbanken halten alle Daten im Hauptspeicher. Dieser ist sehr viel schneller als ein Langzeitspeicher, wie eine Festplatte, ist dafür aber nur in einem kleineren Umfang verfügbar. Da die Speicherkapazität stark begrenzt ist, können Daten in dem Umfang, wie sie bei LAOTSE auftreten, nur kurzfristig gespeichert werden.
Langzeitdatenbanken speichern die Daten auf einer externen Festplatte. Zur Verarbeitung müssen diese in den Hauptspeicher geladen werden. Deshalb werden sie in LAOTSE zur Speicherung der historischen Daten genutzt. Eine der Hauptanforderungen ist es, dass die Daten aus der In-Memory Datenbank schnell genug und in großem Umfang gespeichert werden können. Auch müssen die sehr großen Datenbestände möglichst performant nach bestimmten Datensätzen durchsucht und analysiert werden können.

2.4.3. Datenbankschemata

MM

Relationale Datenbanken sind die am weitesten verbreiteten und bekanntesten Datenbanken. Sie speichern Daten in einer tabellenartigen Struktur.
NoSQL Datenbanken sind im Gegensatz zu relationalen Datenbanken auf Skalierbarkeit und damit auf eine gut erweiterbare Performanz ausgelegt. Das geht häufig zu Lasten von Funktionalitäten von SQL-Anfragen und Konsistenz. Da in LAOTSE die Skalierbarkeit gefordert ist, liegt es nahe eine Datenbank aus dem Gebiet der NoSQL Datenbanken auszuwählen. NewSQL bietet die Möglichkeit die Skalierbarkeit und Performanz von NoSQL Datenbanken mit SQL-Abfragen und der Konsistenz von relationalen Datenbanken zu verbinden (vgl. [Glu15]). Also sind auch NewSQL Datenbanken für LAOTSE zu betrachten. Eine Unterart von NoSQL Datenbanken sind so genannte zeitreihenbasierte Datenbanken. Diese sind darauf optimiert Datentupel nach Zeitstempeln abzulegen, um später komplexe Analysen auf diesen ausführen zu können. Außerdem stehen hier die Einfügegeschwindigkeit und die Skalierbarkeit durch Verteilung im Vordergrund (vgl. [DF15]). Damit ist diese Art von Datenbanken für LAOTSE gut geeignet. Es wurden aus jedem dieser Bereiche einige Datenbanken anhand einer Recherche ausgewählt, um diese in einem ersten Testdurchlauf zu evaluieren. Dabei wurde auf die Erfüllung der Anforderungen von LAOTSE geachtet. Es wurden auch einige vielversprechende Konzepte ausgewählt, die zum Teil erweiterte Funktionen bieten oder die Architektur und Implementierung von LAOTSE vereinfachen könnten.

2.4.4. Big Data

UG

Gemäß verbreiteter Definitionen bezieht sich das „Big“ grundsätzlich auf „volume“ (Datenvolumen, Umfang), „velocity“ (Erstellungs- und Übertragungsgeschwindigkeit) und „variety“ (Vielzahl der Datentypen und -quellen). Oftmals wird diese um „value“ und „validity“ ergänzt, die den unternehmerischen Mehrwert und die Sicherstellung der Datenqualität berücksichtigen sollen (vgl. [Gar], [BGK14, S.27 ff.]). Der Begriff „Big Data“ unterliegt aufgrund der stetigen Erweiterung des Anwendungsgebietes einem Wandel. Im Allgemeinen deckt dieses Themenfeld große digitale Datenmengen, aber auch deren Analyse und Auswertung ab. Ergänzend wird die Betrachtung der Technologien, welche zum Sammeln und Auswerten dieser Datenmengen verwendet werden, berücksichtigt (vgl. [Rei14, S.10 ff.]).
Dieses Anwendungsfeld wird i.d.R. bei großen Datenmengen, die zu groß und komplex sind oder sich zu schnell ändern, sowie schwach strukturiert sind, um mit manuellen oder klassischen Methoden der Datenverarbeitung gespeichert und ausgewertet werden zu können, eingesetzt (vgl. [Cra, S.12 ff.]).
So ist es auch im Bereich der Energiesysteme. Mit deren Wandel zu intelligenten Smart Grids steigt das Datenvolumen und die Vielzahl der Datentypen und -quellen rasant an. Insbesondere im Bereich der Forschung stellt sich die Frage, wie sich die gewünschten Informationen aus den unzähligen Datenquellen ermitteln lassen, um die erhofften Erkenntnisse zu erlangen. Dabei kann das Anwendungsfeld Big Data unterstützen, welches sich primär mit Technologien und Konzepten zum Sammeln, Speichern und Auswerten dieser Massendaten beschäftigen.

3. Projektkonzept

UG

Konzepte sind Entscheidungsgrundlagen und Handlungsvorgaben, welche bei der Umsetzung komplexer Software-Projekte unverzichtbar sind. Gerade bei Projekten, bei denen zu Beginn oft nicht mehr als eine vage Vorstellung bekannt ist, ist ein Grobkonzept, oder auch Lastenheft, unabdingbar.
Bevor mit der Umsetzung des Projektes begonnen werden kann, muss ein gemeinsames Verständnis über das Zielsystem zwischen dem Auftraggeber und der Projektgruppe erarbeitet werden. Dabei soll ein Grundverständnis über Anforderungen, Prozessablauf und Umsetzungsalternativen hergestellt werden. Daher wird in der ersten Phase des Projektes, nach dem offiziellen Start durch das Projektkickoff, mit dem Requirements Engineering begonnen, um die Anforderungen des Zielsystems und das Grobkonzept für die Realisierung zu erarbeiten.
Das Grobkonzept umfasst eine erste Beschreibung des Zielsystems auf der Grundlage der erhobenen Anforderungen und Einschränkungen. Damit das zu erstellende Produkt die geforderten Eigenschaften des Auftraggebers erfüllt und die vom Projektteam erbrachten Leistungen überprüfbar sind, werden die Anforderungen an das System gemeinsam erarbeitet und schriftlich fixiert. Dabei erfolgt die Unterteilung in funktionale und nicht-funktionale Anforderungen.

3.1. Grobkonzept

UG

Dieses Grobkonzept zeigt die erste Vision des Zielsystems, gibt die Richtung für das Softwareprojekt vor und steckt fachlich sowie technisch den Rahmen des Projektes ab.
Damit das Konzept einen bleibenden und positiven Eindruck auf die Auftraggeber hinterlässt, wird zunächst ein geeigneter Name für das Zielsystem bestimmt. Dabei wurde im Kollektiv der deskriptive Name **L**IVE **A**NALYSIS AND L**O**NG-**T**ERM **S**TORAGE **E**NVIRONMENT (LAOTSE) bestimmt.
Das Konzept beginnt mit der Beschreibung der Zielbestimmung, sowie der Darstellung des Anwendungsbereiches. Dabei wird ein erster theoretischer Überblick der Systembestandteile und des Umfeldes gegeben. Um die Aufgabenstellung und Projektbestandteile weiter zu konkretisieren, ist im nächsten Abschnitt ein erster globaler Überblick über die Funktionalitäten des Gesamtsystems gegeben. Anschließend erfolgt die Darstellung der Akteure und der Systemumgebung, deren Interaktionen mittels Use-Case Diagrammen veranschaulicht wird. Die geplanten Komponenten des Zielsystems mit deren Beziehungen untereinander werden übersichtlich mit Hilfe eines Komponentendiagramms erläutert. Dabei erfolgt auch die Darstellung der geplanten Schnittstellen. Nachdem der erste Ansatz des Zielsystems erläutert ist, wird in der Technologieauswahl auf die erforderlichen Komponenten eingegangen. Im vorletzten Kapitel ist ein erster Entwurf der

erforderlichen Datenstruktur dargestellt. Weitere identifizierte Rahmenbedingungen werden im Abschluss des Konzeptes zusammengefasst.

3.1.1. Zielbestimmung

UG

Im Rahmen des Projektes soll das SESA-Lab des OFFIS erweitert werden. Um das Projektumfeld besser einordnen zu können, erfolgt zunächst eine kurze Beschreibung des SESA-Lab.
Das SESA-Lab besteht zum einen aus Hardware zur Anbindung einzelner Komponenten des Smart Grids, wie beispielsweise einem OPAL-RT Echtzeitsimulator, sowie zum anderen aus dem Co-Simulationsframework mosaik, welches Netze oder Teile davon virtuell simuliert. Das Testlabor ist vom OFFIS in Zusammenarbeit mit der Universität Oldenburg geschaffen worden und ermöglicht die Komponenten komplexer Energieversorgungssysteme unter realitätsnahen Bedingungen zu testen.
Ziel des Projektes ist die Schaffung einer Datenmanagementumgebung für das SESA-Lab, bei der die zum Teil sehr hochfrequenten und von unzähligen Sensoren stammenden Daten, mit genauen Zeitstempeln, in Echtzeit verarbeitet und persistent gespeichert werden können. Darüber hinaus sollen Analysen und Visualisierungen sowohl auf die Echtzeitdaten als auch auf die gespeicherten Langzeitdaten ermöglicht werden. Dabei liegt die größte technische Herausforderung in der hochfrequenten Einspeisung der Daten in eine persistente Datenbank.

3.1.2. Systemeinsatz

UG

Die Zielgruppe des Systems sind fachkundige wissenschaftliche Mitarbeiter des OFFIS aus dem Bereich Energie. Zunächst sollen für Forschungszwecke Analysen und Visualisierungen sowohl auf Echtzeitdaten, als auch auf gespeicherten Langzeitdaten ermöglicht werden. Die zukünftige wirtschaftliche Nutzung sollte aber bedacht werden.
Die Datenquellen sind bisher Simulationsdaten des SESA-Lab, welche über Mosaik bereitgestellt werden, und Messwerte von Raspberry PIs. Darüber hinaus sollen über Fernwirkprotokolle angebundene externe Systeme, wie der BHKW-Container in der Lesumstraße in Oldenburg, angebunden werden. Neben der Einbindung dieser drei vorhandenen Datenquellen, soll das System zukünftig anhand gängiger Standards erweiterbar sein.
Dabei sollen neben der Anzeige von Metadaten der Quellen, die Rohdaten wie Einzelmesswerte aber auch aggregierte Daten über beispielsweise Zeitreihen ermöglicht werden. Es sind sinnvolle und hilfreiche Analysen (Durchschnitt, Verbrauchsspitzen, etc.) und Visualisierungen zu entwickeln, welche aber individuell ausgewählt und ggf. angepasst werden können. Da die Umgebung zukünftig erweiterbar sein soll, wäre es hilfreich auch neue Analysen und Visualisierungen bereitzustellen und ggf. vorhandene zu modifizieren. Die Erstellung von Warnmeldungen und Reports bei bestimmten Ereignissen, wie beispielsweise der Ausfall eines Sensors, wäre eine sehr hilfreiche Funktion.
Als DSMS ist Odysseus zu verwenden. Da dieses System eine Eigenentwicklung der Uni Oldenburg ist, können erforderliche Erweiterungen und Änderungen im Fachbereich vorgenommen werden. Bei der Verarbeitung hochfrequenter Daten, im kHz Bereich, sind die Grenzen von Odysseus nicht genau bekannt. Diese gilt es herauszufinden und zu validieren in wieweit die Datenma-

nagementumgebung bei der Vielzahl der Datenquellen zuverlässig und stabil eingesetzt werden kann.
Die In-Memory-Datenbank und die Langzeitarchivierung sind entsprechend der Anforderungen frei wähl- und gestaltbar. Dabei ist ein Mittelweg zwischen der Schreib- und Lesegeschwindigkeit zu wählen.
Die folgende Grafik aus dem Projektkickoff, zu sehen in Abbildung 3.1, liefert den Einstieg in das Projekt. Dargestellt ist der theoretische Datenfluss und die Aufgaben der Projektgruppe.

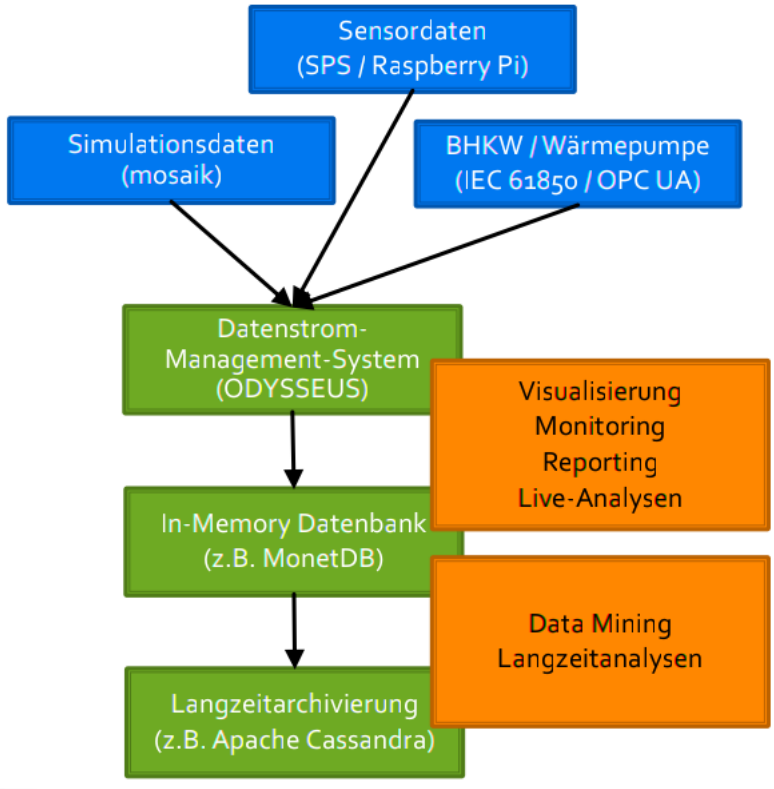


Abbildung 3.1.: SESAdata grobe Datenflussbeschreibung

3.1.3. Hauptfunktionen

UG

In den ersten Meetings sind zusammen mit den Auftraggebern, die Anforderungen erhoben worden. Diese wurden mit der Ideenfindungstechnik Brainstorming gemeinsam erarbeitet und priorisiert.
In der Folgenden Tabelle 3.1 sind die Teilziele mit den dazugehörigen Ergebnissen aufgelistet.
Die Ziele lässt sich in sechs voneinander abhängige Hauptaufgaben überführen, die folgend kurz dargestellt werden:

Datenquellen einbinden: Diese Aufgabe gehört zu den Kernfunktionen und besteht aus der Einbindung der Raspberry PIs und über Fernwirkprotokolle angebundene externe Komponenten) in die Datenstrommanagementumgebung.

Prio	Teilziele	Ergebnisse
hoch	Eingangsdaten verarbeiten (kHz, 1000 Sensoren)	Metainformationen, Messwerte
hoch	Odysseus einbinden	Datenstrom der Quellen verarbeiten
hoch	Zeitstempel mitführen	Synchronisierte Zeitreihen
hoch	Anbindung Mosaik	Simulationsdaten aus Mosaik
hoch	Daten persistent speichern	Rohdaten, aggregierte Daten
hoch	Erweiterbarkeit um Standardprotokolle	weitere Datenquellen können hinzugefügt werden.
mittel	Modularität/Erweiter-barkeit	Komponenten können ausgetauscht werden
mittel	Übersicht vorhandener Sensoren	Liveanzeige + Langzeitdatenbank
mittel	Anzeige/Analyse einzelner Sensoren	Auswahlmöglichkeit bei der Darstellung
niedrig	Anbindung eines BHKW	Verarbeitung Sensordaten BHKW-Dummys
niedrig	Reports über Ausfälle/Störungen	Meldungen über Sensorausfall oder ganzer Datenbank
niedrig	Analyse in Echtzeit und Langzeitdaten	Reports aus Odysseus und Langzeitdatenbank
niedrig	Kompression von Daten	geringere Speicherausnutzung

Tabelle 3.1.: SESAdata Hauptfunktionen

Datenströme verarbeiten: Mit dieser Kernfunktion sollen die Datenströme der Datenquellen verarbeitet werden. Primär geht es um die möglichst schnelle Zusammenführung der Daten (Sensordaten und Metadaten) als synchronisierte Zeitreihen für die Langzeitdatenbank. Darüber hinaus können eingehende Daten für Analysen und Visualisierungen genutzt werden.

Daten persistent speichern: Diese Kernfunktion umfasst die Überführung der eingehenden Sensordaten in eine geeignete Datenstruktur, sowie deren dauerhafte Speicherung. Dabei ist ein Mittelweg zwischen der Schreib- und Lesegeschwindigkeit zu wählen.

Analysen: Diese optionale Funktion beinhaltet die Bereitstellung von einigen nützlichen Analysen, die sowohl die Langzeitdaten als auch eingehende Datenströme berücksichtigen müssen.

Visualisierungen: Mit dieser optionalen Funktion soll eine ansprechende, übersichtliche und geeignete Darstellung der Langzeitdaten und der eingehenden Datenströme ermöglicht werden. Dabei sind nützliche Daten und ggf. Datenreihen zu bestimmen sowie nützliche Darstellungsformen zu wählen.

Erweiterbarkeit: Das System soll zukünftig erweiterbar sein, womit das Hinzufügen weiterer Quellen und deren Datenverarbeitung ermöglicht werden muss. Bei steigendem Leistungsbedarf, muss das System um weitere Hardware- und Softwarekomponenten wie Server und Datenbanken erweiterbar sein. Das Hinzufügen von Analysen und Visualisierungen ist

nicht vorrangig und daher als Nebenfunktion anzusehen. Alle Erweiterungen des Systems, sollen zur Laufzeit ohne Unterbrechung möglich sein.

3.1.4. Systemakteure

UG

Bei den Akteuren handelt es sich um alle Personen, Organisationen, Geräte, Systeme oder Dienste, welche mit dem System in Verbindung stehen und damit interagieren. Tabelle 3.2 zeigt die Akteure des Systems LAOTSE mit ihren Aufgaben.

Akteur	**Art**	**Beschreibung**
Einfacher Anwender	Person	Bedient das zu entwickelnde System. Kann Analysen und Visualisierungen auswählen sowie ein- und ausblenden.
Fortgeschrittener Anwender	Person	Kann, über die Funktionalitäten des einfachen Anwenders hinaus, neue Datenquellen anlegen.
Administrator	Person	Administriert das zu entwickelnde Datenmanagementsystem. Dazu gehört neben der Aufrechterhaltung des Betriebes, auch die fortlaufende Systemerweiterung um einen stabilen Betrieb zu gewährleisten. Darüber hinaus legt er bei Bedarf neue Analysen und Visualisierungen an oder modifiziert vorhandene.
Entwickler	Person	Wird nicht betrachtet.
Externe Datenquellen (Raspberry PI, SESA-Lab, mosaik,BHKW-Container)	Datenquelle	Liefert Meta- sowie Sensordaten, welche vom Datenmanagementsystem verarbeitet werden. Pro Sensor können Einzelwerte übertagen werden. Die Übertragungsgeschwindigkeit geht bis in den kHz Bereich.

Tabelle 3.2.: Tabelle der Systemakteure

Das Diagramm in Abbildung 3.2 zeigt eine grafische Darstellung der Akteure in Beziehung zum System.

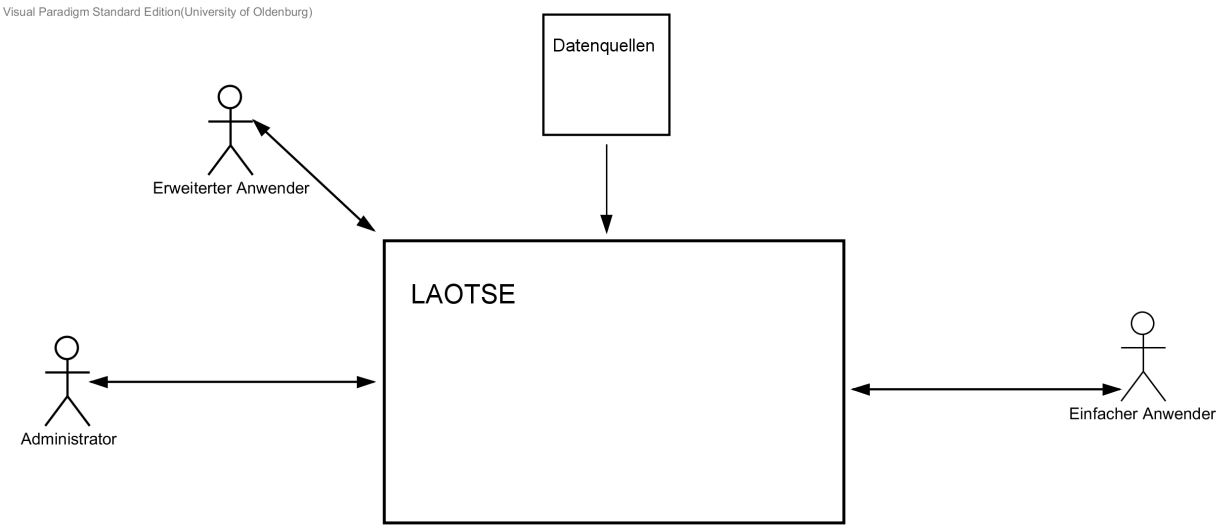


Abbildung 3.2.: LAOTSE Umweltdiagramm

3.1.5. Use-Cases

Alle

Im folgenden Use-Case-Diagramm werden die wesentlichen Komponenten des zu entwickelnden Systems und die Interaktionen der in Kapitel 2 beschriebenen Akteure dargestellt.

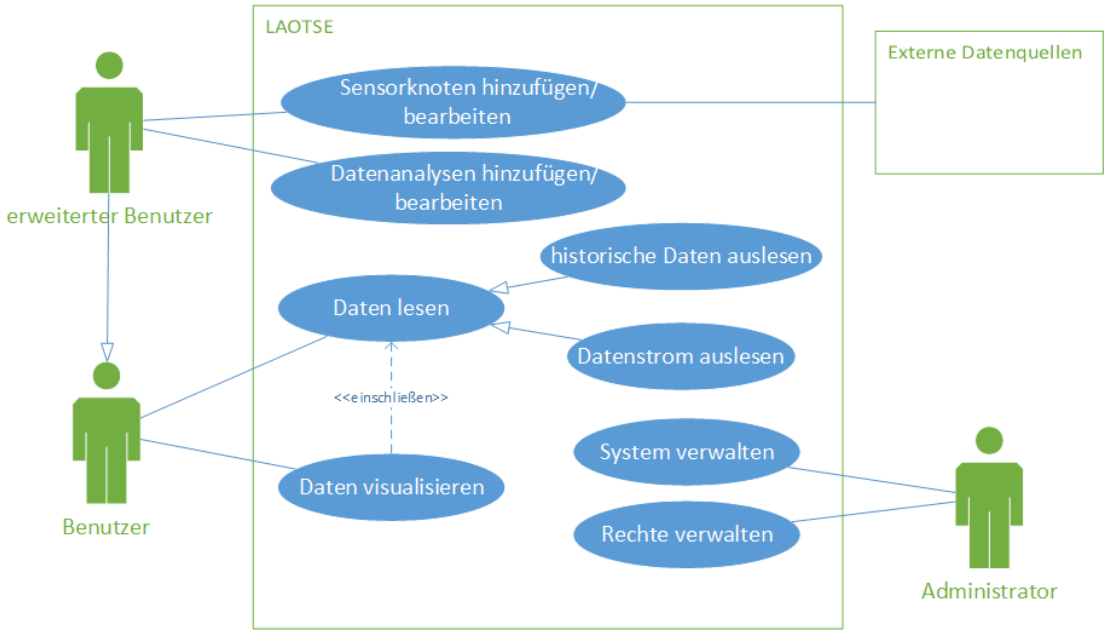


Abbildung 3.3.: Use-Case Diagramm zu LAOTSE

Titel	**Sensorknoten hinzufügen/bearbeiten**/entfernen
Kurzbeschreibung	Der erweiterte Benutzer fügt einen Sensorknoten hinzu oder bearbeitet (entfernt) einen bereits vorhandenen Sensorknoten.
Akteure	erweiterter Benutzer
Auslöser	Ein Sensor soll hinzugefügt oder bearbeitet (entfernt) werden.
Vorbedingung	Der erweiterte Benutzer ist im System angemeldet und das System ist in einem konsistenten Zustand.
Nachbedingung	Das System befindet sich in einem konsistenten Zustand.
Ergebnis	Ein Sensor wurde hinzugefügt oder bearbeitet (entfernt).
genereller Ablauf	1. Der erweiterte Benutzer benutzt die Odysseus-GUI um einen Sensor hinzuzufügen/bearbeiten/entfernen. 2. Der erweiterte Benutzer gibt Metadaten bzgl. des Sensors ein. 3. Der erweiterte Benutzer wählt Analysen auf den Messdaten des Sensors aus. 4. Das erfolgreiche Hinzufügen/Bearbeiten/Entfernen wird dem Benutzer angezeigt.

Titel	**historische Daten auslesen**
Kurzbeschreibung	Der Benutzer liest (aggregierte) Messwerte aus dem System aus.
Akteure	Benutzer
Auslöser	Der Benutzer hat einen Informationsbedarf.
Vorbedingung	Der Benutzer ist im System angemeldet und das System ist in einem konsistenten Zustand.
Nachbedingung	Das System befindet sich in einem konsistenten Zustand.
Ergebnis	Daten wurden ausgegeben.
genereller Ablauf	1. Der Benutzer benutzt die LAOTSE-GUI um eine Anfrage auszuwählen. 2. Der Benutzer gibt die Parameter der Anfrage ein. 3. Das System gibt das Ergebnis der Anfrage aus.

Titel	**Datenstrom auslesen**
Kurzbeschreibung	Der Benutzer liest (aggregierte) Messwerte aus dem System aus.

Akteure	Benutzer
Auslöser	Der Benutzer hat einen Informationsbedarf.
Vorbedingung	Der Benutzer ist im System angemeldet und das System ist in einem konsistenten Zustand.
Nachbedingung	Das System befindet sich in einem konsistenten Zustand.
Ergebnis	Daten wurden ausgegeben.
genereller Ablauf	1. Der Benutzer benutzt die Odysseus-GUI um eine Anfrage auszuwählen. 2. Der Benutzer gibt die Parameter der Anfrage ein. 3. Das System gibt das Ergebnis der Anfrage aus.

Titel	**Daten visualisieren**
Kurzbeschreibung	Der Benutzer wählt einen Betrachtungsgegenstand und ggf. einen zugehörigen Zeitraum aus.
Akteure	Benutzer
Auslöser	Der Benutzer möchte Daten in visueller Form betrachten.
Vorbedingung	Es sind visualisierbare Daten vorhanden.
Nachbedingung	Das System bleibt unverändert.
Ergebnis	Daten wurden in visueller Form angezeigt.
genereller Ablauf	1. Der Benutzer benutzt die LAOTSEGUI um einen Betrachtungsgegenstand auszuwählen. 2. Das System gibt dem Benutzer ggf. die Möglichkeit eine Visualisierungsart auszuwählen. 3. Dem Benutzer werden die Daten visualisiert angezeigt.

Titel	**Datenanalyse hinzufügen/bearbeiten.**
Kurzbeschreibung	Der Benutzer (Forscher) fügt eine Datenanalyse hinzu oder bearbeitet eine bereits vorhandene Datenanalyse.
Akteure	Benutzer (Forscher)
Auslöser	Eine Datenanalyse soll hinzugefügt oder bearbeitet werden.
Vorbedingung	Der Benutzer (Forscher) ist im System angemeldet und das System ist in einem konsistenten Zustand.
Nachbedingung	Das System befindet sich in einem konsistenten Zustand.
Ergebnis	Eine Datenanalyse wurde hinzugefügt oder bearbeitet.

genereller Ablauf	1. Eine Datenanalyse wird vom Entwickler im Java-Code hinzugefügt/bearbeitet. 2. Der Entwickler stellt dies dem Administrator zur Verfügung.

Titel	**System verwalten**
Kurzbeschreibung	Der Administrator verwaltet das System durch Rechte-/Infrastruktur-Anpassung.
Akteure	Administrator
Auslöser	Das System erfordert Veränderungen.
Vorbedingung	Das System befindet sich in einem konsistenten Zustand.
Nachbedingung	Das System befindet sich in einem konsistenten Zustand.
Ergebnis	Das System wurde verwaltet.
genereller Ablauf	1. Der Administrator nutzt seine GUI, um entsprechende Systemänderungen durchzuführen. 2. Die Änderungen treten in Kraft.

3.1.6. Komponenten

MM, UG

Das LAOTSE System wird ein sehr komplexes System werden, bei dem viele verschiedene Komponenten miteinander agieren müssen. Um alle Komponenten und ihre Zusammenhänge darstellen zu können, werden UML Komponentendiagramme eingesetzt.
Ein solches Diagramm ist in Abbildung 3.4 dargestellt. Es enthält alle geplanten Komponenten des LAOTSE Systems und welche Beziehungen die Komponenten untereinander haben.
Die nachfolgenden Ausführungen sollen dabei noch einmal genauer die einzelnen Komponenten und ihre Funktion innerhalb des Systems darstellen und erläutern.
Die Komponenten lassen sich in drei Kernbereiche unterteilen. Zum einen gibt es Komponenten, die sich mit der Datenhaltung beschäftigen. Auf diese Komponenten greifen dann die Komponenten aus dem Bereich der Logik zu, die für die Datenverarbeitung und Datenanalyse zuständig sind. Im letzten Schritt sollen diese Daten für den Benutzer visualisiert werden. Dafür sind die Komponenten aus dem Bereich Darstellung zuständig, welche die Benutzeroberfläche für das System darstellen.

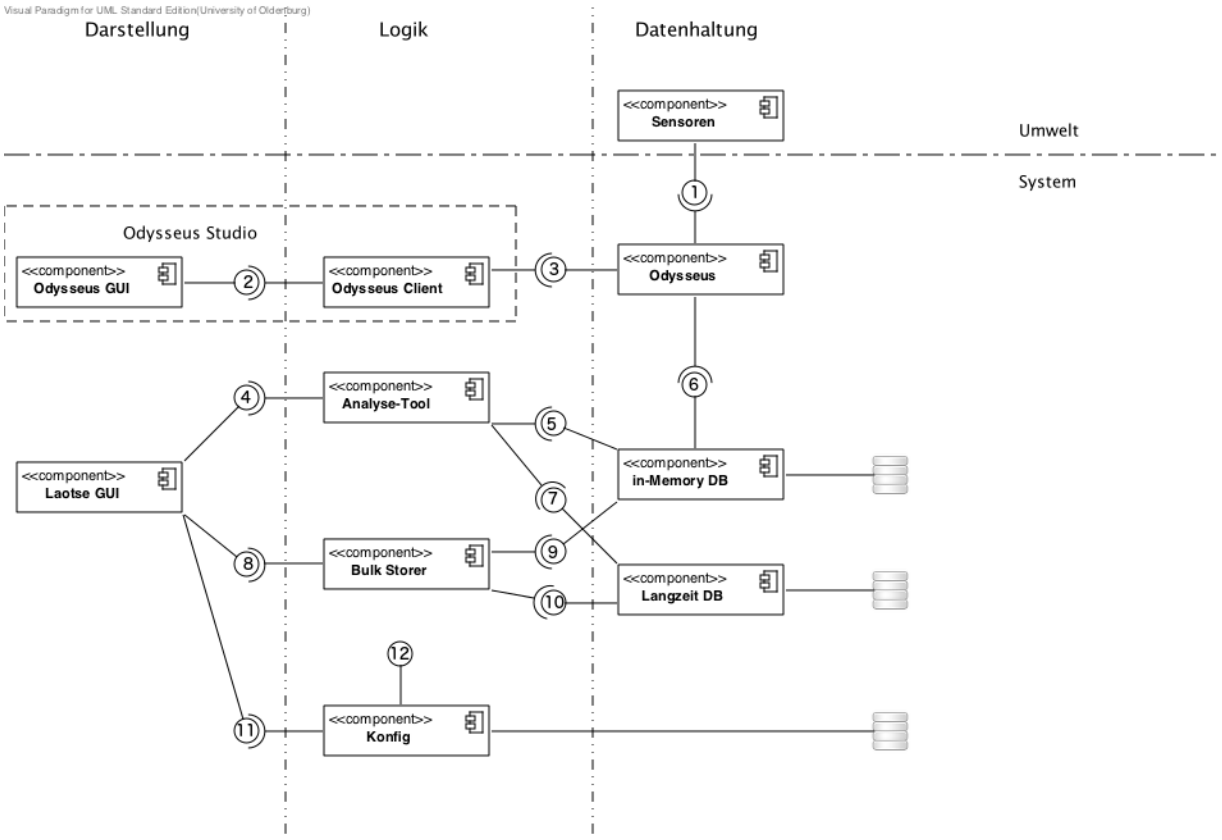


Abbildung 3.4.: Komponentendiagramm des Systems mit Darstellung der geplanten Schnittstellen

Sensoren Nicht zum LAOTSE System gehören die Sensoren, diese liegen außerhalb des Systems, sind in dem Komponentendiagramm allerdings dennoch aufgeführt, da diese die Daten für das System liefern, ohne die das System nicht arbeiten könnte und hinfällig wäre.

Odysseus Server Die erste Komponente, die dem Bereich der Datenhaltung zuzuordnen ist, ist der Odysseus Server, der die Daten, welche die Sensoren liefern, in Echtzeit verarbeiten kann. Odysseus ist also für den Datenstrom zuständig und ist somit die Grundlage für die Live-Datenanalyse.

In-Memory Datenbank Odysseus speichert die gegebenen Daten des Datenstroms in eine In-Memory Datenbank, auf die nahezu in Echtzeit zugegriffen werden kann. Diese In-Memory Datenbank hält weitere Daten für Analysen bereit, die auf den Daten ausgeführt werden sollen. Die In-Memory DB hält die Daten allerdings nur so lange, wie Analysen auf den Daten durchgeführt werden oder bis die Daten aus der In-Memory Datenbank in die Langzeit Datenbank geschrieben wurden.

Langzeit Datenbank Die Langzeit Datenbank ist zuständig für die Langzeitarchivierung der Daten und soll es ermöglichen, neue Daten mit historischen Daten vergleichen zu können. Sie ist somit dafür zuständig Daten über einen langen Zeitraum zu speichern und sie für einen späteren Zeitpunkt zur Verfügung zu stellen, damit sie dann gegebenenfalls noch analysiert werden können.

Zu beachten ist bei den beiden Komponenten, der Langzeit Datenbank und der In-Memory Datenbank, dass die in Abbildung 3.4 dargestellten Komponenten nicht die Datenbank an sich darstellen, sondern die Datenbank Handler, welche die Zugriffe auf der jeweiligen Datenbank ausführen. Die Datenbanken selbst werden durch die Symbole rechts davon dargestellt.
In-Memory Datenbank und Langzeit Datenbank organisieren die Datenhaltung innerhalb des LAOTSE Systems. Damit die gegebenen Daten jedoch auch organisiert und verarbeitet und nicht nur gespeichert werden können, müssen auch Logikkomponenten existieren, die dies umsetzen.

Konfiguration Als erste muss dabei die Konfigurationskomponente erwähnt werden, die Konfigurationsdaten für das System in einer Datenbank speichert und für die anderen Logikkomponenten zur Verfügung stellt. So enthält diese Komponente zum Beispiel die Metadaten der einzelnen angeschlossenen Sensoren. Somit kann aus diesen Daten ermittelt werden, welche Sensoren aktuell an das System angeschlossen sind und es können über diese Komponenten ebenfalls Sensoren hinzugefügt werden. Aus diesem Grund sind alle anderen Logikkomponenten des Systems Observer dieser Konfigurationskomponente, um auf Änderungen und neue Sensoren reagieren zu können und so zu gewährleisten, dass eine korrekte Analyse durchgeführt wird.

Odysseus Client Für die Odysseus Komponenten existiert dementsprechend auch eine eigene Logikkomponente. Diese Komponente ist der Odysseus Client, der auf dem Datenstrom des Odysseus Servers anfragen laufen lassen kann und auf diese Art und Weise die Daten analysieren oder aggregieren kann. Gemeinsam mit der dazugehörigen Odysseus GUI sind diese Komponenten im Odysseus Studio vereint, das Anfragen auf den Datenstrom aus Odysseus ermöglicht und analysierte Daten im nächsten Schritt visualisieren kann. Da der Odysseus Client ebenfalls ein Observer der Konfigurationskomponente ist, kann das Odysseus Studio vor allem dazu genutzt werden um zu visualisieren welche Sensoren aktuell Daten liefern und gegebenenfalls, ob Warnzustände vorliegen. Bei der Umsetzung ist aus Performance- und Austauschbarkeitsgründen nach Möglichkeit die Trennung der Clients nach Funktionalitäten vorzunehmen. Eine denkbare Unterteilung wäre:

1. Sensor-Client (write)

2. Analyse-Client (read)
3. Visualisierungs-Client (read)

Odysseus GUI Die Odysseus GUI visualisiert die im Odysseus Client ermittelten Daten und stellt somit die Komponente für die Echtzeitüberwachung dar.

Analyse-Tool Für die eigentliche Analyse der eingehenden Daten ist das Analyse-Tool zuständig. Dieses hat neben den Konfigurationsdaten sowohl Zugriff auf die In-Memory Datenbank und somit echtzeitnahe Daten, als auch auf die Langzeit Datenbank. In dieser Komponente können also alle früheren und aktuellen Daten vereint betrachtet werden und somit komplexe Analysen durchgeführt werden.

Bulk Storer Der Bulk Storer dient dazu, von Zeit zu Zeit die Daten aus der In-Memory Datenbank auszulesen und in die Langzeitdatenbank hineinzuschreiben. Damit kann die In-Memory Datenbank entlastet werden und neue Daten aufnehmen und die Langzeit Datenbank hält diese Daten für spätere Analysen weiter. Als zusätzliche Aufgabe kann der Bulk Storer der wichtigsten Darstellungskomponente, der LAOTSE GUI, Statusmeldungen senden, auf welche die LAOTSE GUI dann reagieren kann.

LAOTSE GUI Die angesprochene LAOTSE GUI ist die Haupt-Benutzerschnittstelle des Systems. Sie ermöglicht es dem Benutzer beispielsweise Analysen auf dem Analyse-Tool auszuführen oder neue Analysen hinzuzufügen, die dann in der Konfigurationsdatenbank gespeichert werden und zu einem späteren Zeitpunkt erneut ausgeführt werden können. Ebenso bietet die LAOTSE GUI dem Benutzer die Möglichkeit neue Sensoren im System zu registrieren oder bestehende Sensoren zu bearbeiten, damit diese für spätere Analysen genutzt werden können. Dabei gibt die LAOTSE GUI die neuen Informationen an die Konfigurationskomponente, die diese Daten dann speichert und die anderen Logikkomponenten bekommen Nachricht über die Änderung und können sich die neuen Daten holen und damit arbeiten. Ebenso ist die LAOTSE GUI aber auch dazu da, die Ergebnisse der durchgeführten Analysen darzustellen und sie dem Benutzer so auf eine geeignete Art und Weise anschaulich zu machen.

3.1.7. Schnittstellen

MM

In Abbildung 3.4 ist das Komponentendiagramm des Systems mit seinen Schnittstellen dargestellt. Es werden verschiedene Schnittstellen eingesetzt. Dadurch können heterogene Komponenten wie z.B. verschiedene Sensoren standardisiert eingebunden und genutzt werden. Auch wird dadurch Flexibilität sichergestellt und es können Komponenten ausgetauscht werden, ohne andere Komponenten anpassen zu müssen. Die Schnittstellen sind fortlaufend nummeriert und werden in diesem Abschnitt beschrieben:

1. Die Sensoren implementieren eine Schnittstelle, die Metadaten und ggf. Datenformate der Messwerte beschreibt. Über diese Schnittstelle können Daten von den Sensoren an das System übermittelt werden und das System kann Metainformationen über die Sensoren abfragen.

2. Die grafische Benutzeroberfläche des Odysseus-Studio implementiert eine Schnittstelle, die grafische Funktionen bereitstellt. Es werden Benutzereingaben an Odysseus und Visualisierungen von Odysseus verarbeitet.

3. Der Odysseus-Server implementiert eine Schnittstelle, die vom Client des Odysseus-Studio genutzt wird. Über diese Schnittstelle lässt sich der Server steuern und z.B. Abfragen an die Datenströme erzeugen, starten und stoppen.

4. Analog zu Schnittstelle 2 implementiert die LAOTSE GUI drei Schnittstellen, die von der Logik zur Visualisierung von Daten genutzt werden können. Dabei stellt jede Schnittstelle nur die für die jeweilige Komponente notwendigen Funktionen bereit.

5. Die Komponente der In-Memory DB implementiert eine Schnittstelle, die vom Analyse-Tool genutzt werden kann, um auf die Daten der Datenbank zuzugreifen.

6. Analog zu Schnittstelle 5 können über diese Schnittstelle Daten in die In-Memory DB geschrieben werden.

7. Analog zu Schnittstelle 5 implementiert die Langzeit-DB eine Schnittstelle, um die Daten zur Analyse bereit zu stellen.

8. Siehe Schnittstelle 4

9. Siehe Schnittstelle 5

10. Analog zu Schnittstelle 6 können hier Daten in die Langzeit DB geschrieben werden.

11. Siehe Schnittstelle 4

12. Die Konfigurationskomponente stellt eine Schnittstelle bereit, die von beliebigen anderen Komponenten genutzt werden kann. Die anderen Komponenten werden dann über Änderungen in der Konfiguration benachrichtigt. Dies ist z.B. über das Observer-Pattern umsetzbar, indem die Konfiguration ein Observable darstellt und die anderen Komponenten sich als Beobachter registrieren.

3.1.8. Technologieauswahl

UG, KB

Dieser Abschnitt beschäftigt sich mit den ausgewählten Technologien die für die Realisierung des Systems eingesetzt werden sollen. Dabei wird zwischen den Technologien, zur Erstellung des Systems während des Projektes und denen die in das System eingehen, unterschieden.

Projektumgebung

Für die Steuerung und Organisation des Projektes sollen die folgenden Werkzeuge verwendet werden, welche ausführlich im Kapitel 4.3 beschrieben sind.

- Versionsverwaltung SVN
- JIRA

- Confluence
- LaTeX

Eclipse (Projektumgebung)

Bei der Wahl der Entwicklungsumgebung für Java fällt die Entscheidung meist auf eine der beiden bekanntesten Entwicklungsumgebungen eclipse und NetBeans. Da für die Umsetzung des Projektes Odysseus verwendet werden soll und dieses auf dem OSGi-Framework Equinox basiert, ist eclipse geeigneter. Eclipse basiert nämlich auch auf Equinox, wodurch eine bessere Nutzung und Entwicklung von Odysseus möglich ist. Darüber hinaus arbeiten nahezu alle Projektteilnehmer hauptsächlich mit eclipse, was zusätzlich für die Verwendung spricht. Insgesamt ist eclipse eine Entwicklungsumgebung, die nahezu alle Möglichkeiten mit Hilfe von Plug-ins bietet. So kann in eclipse auch gleichzeitig die Subversion (SVN) Anbindung gewährleistet und dadurch ein besserer Arbeitsablauf geboten werden.

Odysseus (System)

Wie schon im Abschnitt zu eclipse erwähnt soll in diesem Projekt das Datenstrommanagementsystem (DSMS) Odysseus verwendet werden. Odysseus ist dabei dafür zuständig die Daten, die in Echtzeit von den Sensoren geliefert werden zu verarbeiten und in eine Datenbank zu schreiben. Ebenso soll Odysseus die Möglichkeit bieten bestimmte Dinge in Echtzeit anzuzeigen. Die Wahl des DSMS ist auf Odysseus gefallen, da Odysseus an der Universität in Oldenburg entwickelt wird und somit für das Projekt eine gute Unterstützung und schneller Support bei auftretenden Fehlern gewährleistet werden kann. Außerdem wurde Odysseus als einzusetzendes Tool von dem Auftraggeber vorgegeben.

Java (System)

Als Programmiersprache für die Entwicklung des Systems ist die Entscheidung auf die Programmiersprache Java gefallen, da hiermit eine plattformübergreifende Entwicklung gewährleistet werden kann. Außerdem haben alle Projektteilnehmer die meisten Erfahrungen mit Java gemacht und das DSMS Odysseus wurde ebenfalls in Java entwickelt. Da Odysseus im Zuge dieses Projektes weiterentwickelt werden soll, muss zwangsläufig Java eingesetzt werden. Da die Entwicklung mit zwei verschiedenen Programmiersprachen nicht förderlich für das Projekt ist, wird deshalb für das gesamte System Java als Programmiersprache verwendet.

mosaik (System)

Mosaik ist ein Smart Grid Simulationsframework. Es dient also dazu Smart Grids zu simulieren und die Simulationsdaten für eine Analyse zur Verfügung zu stellen. Die Sensordaten des SESA-Lab werden häufig mit mosaik erweitert, sodass ein großes Netz simuliert wird. Mosaik ist dabei als Simulationsframework von dem Auftraggeber vorgeschrieben worden. Da es aber auch eine Java Anbindung bietet, ist es für dieses Projekt sehr gut einsetzbar und da es bereits mit dem SESA-Lab zusammenarbeitet, ist keine Integration notwendig und mosaik das richtige Tool für den Einsatz im zu entwickelnden System.

Raspberry PIs (System)

Raspberry PIs werden zur Anbindung der Sensoren und der Verarbeitung der Messwerte benötigt. Diese sollen als Knotenpunkte innerhalb des Netzes fungieren und somit einzelne Sensoren darstellen und simulieren. Zum Beispiel könnte ein Raspberry PI dazu genutzt werden ein BHKW zu simulieren, um die spätere Nutzung eines solchen zu gewährleisten. Die entwickelten Raspberry PIs werden dann im SESA-Lab integriert und dienen als Datenquelle für das zu entwickelnde System.

virtuelle Maschine (VM) (System)

Das später fertige System muss auf einer oder mehreren VMs laufen. Dementsprechend werden VMs auf Linux Basis verwendet, auf denen das lauffähige System kompiliert werden kann. Ebenso wird eine VM für die Entwicklung des Systems verwendet, auf der zum Beispiel SVN läuft. Diese VMs sind also unverzichtbar, sowohl für die Entwicklung des Systems, als auch für das später fertige System.

In-Memory Datenbank (System)

Eine geeignete In-Memory Datenbank wird im Projektverlauf nach ausführlicher Aufstellung von Technologie-Alternativen ausgewählt.

Persistente Datenhaltung (System)

Eine geeignete Lösung für persistente Datenhaltung wird im Projektverlauf nach ausführlicher Aufstellung von Technologie-Alternativen ausgewählt.

3.1.9. Datenstruktur

MM, LS

Dieser Abschnitt gibt einen ersten groben Überblick über die Datenstruktur des Systems. Dazu wird zunächst ein Entwurf des Datenflusses dargestellt. Des Weiteren wird kurz auf die internen Datenformate und der zur Speicherung eingegangen.

Datenflussbeschreibung

In Abbildung 3.5 ist der Datenfluss der Messwerte dargestellt, die von Sensoren an LAOTSE übertragen werden. Im folgenden sind diese Datenflüsse beschrieben und dabei verweise auf die nummerischen Bezeichner angegeben.
Zuerst treffen sie als Datenstrom im DSMS Odysseus ein (1) und werden dort live verarbeitet. Wenn es gewünscht ist, können Ergebnisse einfacher live-Analysen oder die Rohdaten an das Odysseus-Studio gesendet (2) und dort visualisiert (3) werden.
Anschließende werden alle Rohdaten in die in-Memory DB geschrieben (4). Dabei puffert Odysseus ggf. den Datenstrom entsprechend der Verarbeitungsgeschwindigkeit der Datenbank und überträgt die Daten Stückweise.
Die In-Memory DB Komponente handhabt, das reale Schreiben (5) und Lesen (6) aus der Datenbank. Die entsprechende Komponente der Langzeit DB arbeitet analog (9 und 10).

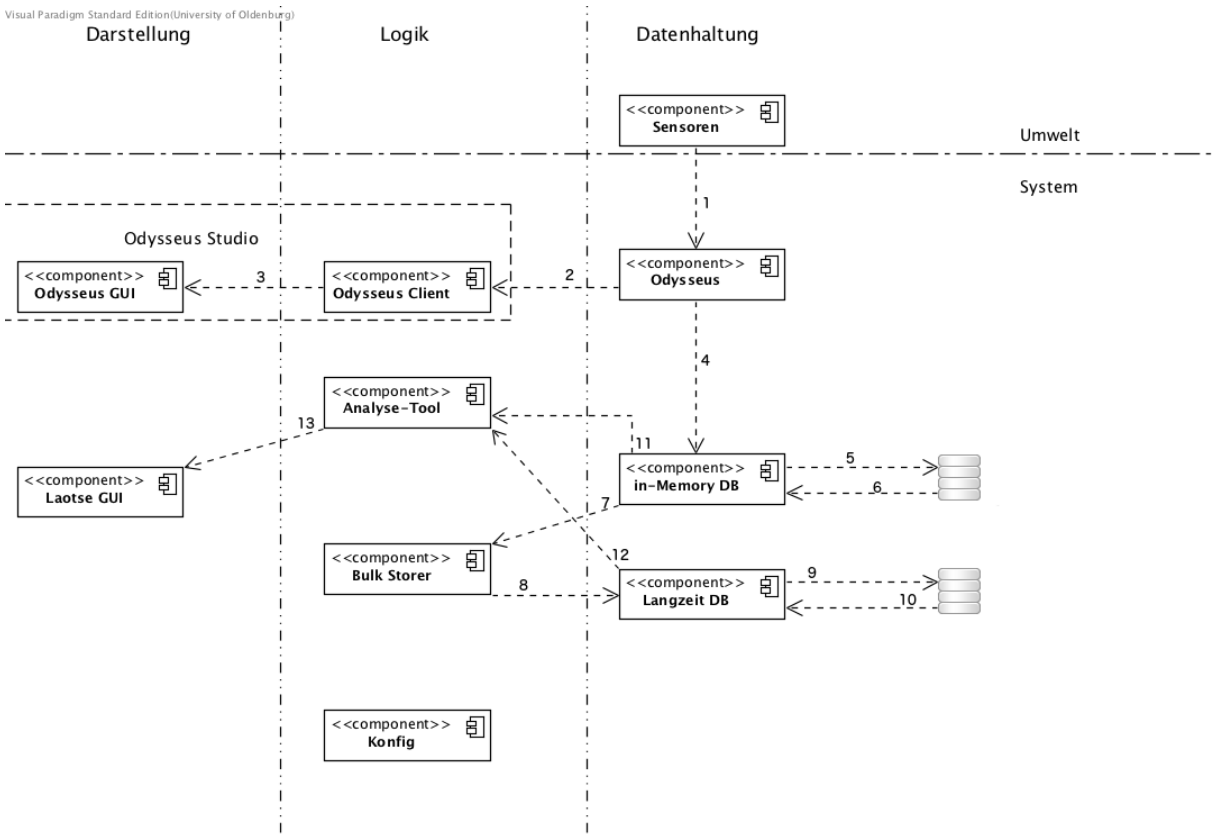


Abbildung 3.5.: Komponentendiagramm des Systems mit Darstellung des geplanten Datenflusses der rohen und verarbeiteten Messwerte der Sensoren

Die Messwerte werden aus der In-Memory DB über die Komponente in definierten Intervallen in den Bulk Storer geladen (7) und von diesem in die Langzeit DB geschrieben (8). Dadurch wird die Langzeit DB, die große Datenmengen verarbeitet, allerdings langsamer als die In-Memory DB arbeitet nicht überlastet und entsprechend mit den Messwerten versorgt.
Wenn Analysen auf den aufgezeichneten Daten durchgeführt werden sollen, werden die gewünschten Daten aus der In-Memory DB (11) und Langzeit DB (12) in das Analyse-Tool geladen. Die Analyseergebnisse können in der LAOTSE GUI visualisiert (13) werden.

Datenformate intern

Die intern genutzten Datenformate sollten nach Möglichkeit auf Java-Standarddatentypen aufsetzen, die auch von Odysseus unterstützt werden. Dies erleichtert zukünftige Erweiterungen oder den Austausch von Komponenten, die auf die betroffenen Schnittstellen zugreifen.

Datenformate zur Speicherung

Die Datenformate zur Speicherung, insbesondere im Zusammenhang mit der Langzeitdatenhaltung, sollten nach Möglichkeit 2 verschiedenen Anforderungen genügen:

1. Sie sollten möglichst schnell in die DB ein- bzw. aus ihr auszulesen sein.

2. Sie sollten aufgrund des hohen Datenvolumens möglichst komprimiert abgespeichert werden können.

Der zweite Punkt würde eine auf den eingehenden Daten anwendbare bijektive Komprimierungsfunktion voraussetzen, deren Durchführung wiederum Zeit kosten wird, was einen Widerspruch zur ersten Anforderung darstellt. Aus diesem Grund muss ein Kompromiss zwischen den beiden Anforderungen gefunden werden. Hierbei liegt die Priorität eindeutig auf der Schnelligkeit des schreibenden/lesenden Zugriffs auf die DB. Da Speicherplatz praktisch in unbegrenztem Maße zur Verfügung steht ist die zweite Anforderung weniger zu berücksichtigen.

3.1.10. Skalierbarkeit

MM

Das System muss große Datenmengen, die in einer hohen Frequenz eingespeist werden verarbeiten können. Dafür wurden entsprechende Anforderungen erhoben, die z.B. die Rechenleistung und die Speicherkapazität betreffen. Beides muss entsprechend hoch sein um die Aufgabe zu erfüllen. Es ist gewünscht, dass die Architektur erweiterbar ist.
Eine Möglichkeit diese Anforderungen zu erfüllen ist es, eine horizontale Skalierbarkeit im System umzusetzen. Das bedeutet, dass eine Lastverteilung auf verschiedene Ressourcen erfolgt. Wenn die Kapazität des Systems der anfallenden Last in Form von Rechenleistung für den Datenstrom oder Speicherkapazität für die Datenmengen nicht mehr gerecht werden kann, sollen weitere Knoten integriert werden können, die weitere anfallende Last aufnehmen können. Die Fähigkeit eines Systems zur Integration weiterer Knoten und Lastverteilung auf diese, charakterisiert die horizontale Skalierbarkeit. Der Vorteil gegenüber einer vertikalen Skalierbarkeit, bei der vorhandene Knoten erweitert werden würden, ist, dass der horizontalen Skalierbarkeit theoretisch keine Grenzen gesetzt sind. Voraussetzung dafür ist, dass die Implementierung dies unterstützt.
Konkret muss zur Lastverteilung der Verarbeitung der Datenströme Odysseus horizontal skalierbar sein. Für die Lastverteilung der Datenmenge muss ein skalierbares DBMS für die Komponenten der Langzeitdatenbank und der In-Memory-Datenbank gewählt werden.
Durch die Skalierung kann gewährleistet werden, dass das System sowohl bei geringem Datenaufkommen, als auch bei hohem Datenaufkommen zuverlässig in den geforderten Bearbeitungszeiten arbeitet.

3.2. Funktionale Anforderungen

Alle

Die funktionalen Anforderungen beziehen sich auf die konkreten Funktionalitäten, die das System anbieten soll. Sie bilden die Grundlage für den Leistungsumfang des Zielsystems und sind daher widerspruchsfrei und überprüfbar formuliert.
In der folgenden Tabelle sind die Funktionalen Anforderungen an LAOTSE aufgelistet.

ID	Prio	Anforderung
L-FA1	A	Das System soll die eingehenden Sensordaten (Roh- und Metadaten) persistent speichern können.

L-FA2	A	Das System soll die eingehenden Daten verlustfrei verarbeiten können.
L-FA3	A	Das System soll den Messzeitpunkt (Timestamp) zu den gespeicherten Daten speichern können.
L-FA4	A	Das System soll Sensordaten von Raspberry PIs verarbeiten können.
L-FA5	A	Das System soll die Möglichkeit bieten weitere Quellen, über aktuell verbreitete Standards, hinzufügen zu können.
L-FA6	A	Das System muss Sensordaten aus dem SESA-Lab, die über mosaik bereitgestellt werden, verarbeiten können.
L-FA7	A	Das System muss Datenströme mit Hilfe des vorhandenen DSMS Odysseus verarbeiten können.
L-FA8	B	Das System soll die Möglichkeit, bieten Systemerweiterungen (Hardware, Software) im laufenden Betrieb ermöglichen zu können.
L-FA9	B	Wenn Datenquellen ausfallen, muss das System fähig sein, unterbrechungsfrei weiterlaufen zu können.
L-FA10	B	Wenn ausgefallene Datenquellen wieder verfügbar sind, soll das System diese automatisch wieder einbinden können.
L-FA11	B	Das System soll Analysen auf den gespeicherten Langzeitdaten durchführen können.
L-FA12	B	Das System soll Liveanalysen auf den Datenströmen durchführen können.
L-FA13	B	Das System soll fünf vorimplementierte Standardanalysen, bereitstellen können.
L-FA14	B	Das System soll um weitere Analysen, durch den Benutzer (Forscher), ergänzt werden können.
L-FA15	B	Das System soll die Möglichkeit bieten, Datenströme visualisieren zu können (Beispielsweise die aktuell erzeugte Leistung eines Windparks).
L-FA16	B	Das System soll die Möglichkeit bieten, Langzeitdaten visualisieren zu können (bspw. die erzeuge Leistung des letzten Jahres).
L-FA17	B	Das System soll die Möglichkeit bieten, Metadaten visualisieren zu können (bspw. welche Sensoren aktuell Daten an das System senden).
L-FA18	B	Das System soll die Möglichkeit bieten, Analyseergebnisse aus den Langzeitdaten und Datenströmen, visualisieren zu können (z.B. Mittelwert der letzten 3 Wochen).
L-FA19	B	Das System soll die Möglichkeit bieten, Visualisierungen ein- und ausblenden zu können.
L-FA20	B	Das System soll um weitere Visualisierungen, durch den Benutzer (Forscher), ergänzt werden können.
L-FA21	C	Das System soll die Daten, des über Fernwirkprotokolle angeschlossenen BHKW-Containers in der Lesumstraße in Oldenburg, verarbeiten können. (Wenn dieser nicht rechtzeitig in Betrieb genommen wird, erfolgt die Einbindung eines bereit gestellten Dummy-BHKWs.)
L-FA22	C	Das System soll den Ausfall von Sensoren melden können (z.B. als visuelle Hervorhebung oder Nachricht).
L-FA23	C	Das System soll Meldungen unterschrittener Schwellwerte ausgeben können (z.B. als visuelle Hervorhebung oder Nachricht).

L-FA24	C	Das System soll um weitere Meldungen erweitert werden können.
L-FA25	C	Das System soll die zu speichernden Daten komprimieren können.

3.3. Nicht-funktionale Anforderungen

Alle

Nicht-funktionale Anforderungen beziehen sich hingegen auf die Eigenschaften des Systems und der Projektdurchführung. Diese Anforderungen sind ebenfalls verpflichtend für den Leistungsumfang des Zielsystems und dadurch widerspruchsfrei und überprüfbar formuliert.

3.3.1. Projektanforderungen / Anforderungen an Durchführung und Entwicklung

Die folgenden Anforderungen beziehen sich auf die Projektdurchführung.

ID	Prio	Kategorie	Anforderung
NFA1	A	Budget	Wenn Investitionen zur Realisierung des Projektes erforderlich sind, muss das Projektteam Angebote zur Auswahl bereitstellen.
NFA2	A	Lieferumfang und Termine	Das Projektteam muss nach der Hälfte der Projektzeit einen Zwischenbericht, welcher neben dem Grobkonzept, das Feinkonzept enthält, vorlegen.
NFA3	A	Lieferumfang und Termine	Das Projektteam soll nach der Hälfte der Projektzeit einen Prototyp bereitstellen.
NFA4	A	Lieferumfang und Termine	Das Projektteam soll am 31.03.2016 den Abschlussbericht, welcher neben dem Zwischenbericht, das Feinkonzept, ein Installations- sowie Benutzerhandbuch enthält, bereitstellen.
NFA5	C	Lieferumfang und Termine	Bei allen Dokumenten soll nach Möglichkeit der Autor hervorgehen.
NFA6	A	Lieferumfang und Termine	Das Projektteam führt monatliche Reviews mit dem Auftraggeber durch.
NFA7	B	Testkonzept	Das Testkonzept soll von Beginn an erstellt und angemessen dokumentiert werden.
NFA8	B	Testkonzept	Das Testkonzept kann bei Bedarf und in Absprache mit den Betreuern erweitert werden.
NFA9	B	Evaluierung	Systemkomponenten sollen mit umfangreicher Alternativenbetrachtung, Begründung und Dokumentation ausgewählt werden.
NFA10	A	Doku	Die Dokumentation muss fortlaufend und präzise während der gesamten Projektzeit erfolgen.
NFA11	A	Handbücher	Die Handbücher müssen für Fachleute und in der deutschen Sprache erstellt werden.

3.3.2. Rechtliche Anforderungen

Die folgende Tabelle zeigt die rechtlichen Anforderungen.

ID	Prio	Kategorie	Anforderung
NFA12	A	Datenschutz	Die bereitgestellten Daten und Systeme müssen vertraulich, ohne absichtliche Weitergabe an Außenstehende, behandelt werden.
NFA13	B	Rechtliches	Urheber- und Lizenzrechte müssen gewahrt werden.
NFA14	B	Rechtliches	Anforderungsänderungen können nur durch Zustimmung des Auftraggebers durchgeführt werden.
NFA15	B	Rechtliches	Weitere rechtliche Rahmenbedingungen müssen nicht beachtet werden.

3.3.3. Technische Anforderungen

Die folgende Tabelle zeigt die technischen Anforderungen.

ID	Prio	Kategorie	Anforderung
NFA16	A	Technologie	Die Technologie kann grundsätzlich frei gewählt werden, außer Vorgaben oder vorhandene Systeme widersprechen dem.
NFA17	B	Technologie	Die Technologieauswahl muss immer ordnungsgemäß evaluiert und dokumentiert werden.
NFA18	A	Technologie	Die Technologie soll unter Berücksichtigung der Ausbaufähigkeit und des langfristigen Einsatzes ausgewählt werden.
NFA19	B	Systemaufbau	Raspberry PIs sollen als Hardwareaufbau realisiert und in das System eingebunden werden.
NFA20	A	Systemaufbau	Die Systemkomponenten des DSMS sollen auf bereitgestellten VM-Instanzen realisiert werden.
NFA21	B	Schnittstellen	Wenn Standards für die Schnittstellen vorhanden sind, sollen diese nach Möglichkeit verwendet werden.
NFA22	A	Schnittstellen	Wenn Schnittstellen bereitgestellt werden, muss deren Spezifikation und Beschreibung erstellt werden.
NFA23	C	Programmiersprache	Die Programmiersprache kann frei gewählt werden.
NFA24	B	Programmiersprache	Codekonventionen sollen eingehalten werden.
NFA25	C	Speicherkapazität	Die Speicherkapazität für die Langzeitspeicherung steht aufgrund der immer weiter sinkenden Kosten ausreichend zur Verfügung.
NFA26	B	Speicherkapazität	Wenn eine Kompression möglich ist, soll diese aus Performance- und Speicherauslastungsgründen implementiert werden,

NFA27	C	Hardware- und Softwarekomponenten	Das Projektteam muss den Ausfall einzelner Systemkomponenten wie DB-Ausfälle oder sonstige Katastrophen nicht berücksichtigen.
NFA28	B	Hardware- und Softwarekomponenten	Das System soll nach Möglichkeit modular aufgebaut werden, um System- und Kapazitätserweiterungen zur Laufzeit zu ermöglichen.
NFA29	A	Hardware- und Softwarekomponenten	Das System soll die Möglichkeit bieten Daten, von bis zu 1000 Sensoren, zu verarbeiten.
NFA30	A	Hardware- und Softwarekomponenten	Das System soll fähig sein hochfrequente Daten, von bis zu 30 kHz, zu verarbeiten.

3.3.4. Qualitätsanforderungen

Die folgende Tabelle zeigt die Qualitätsanforderungen.

ID	**Prio**	**Kategorie**	**Anforderung**
NFA31	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb hinzugefügt werden können.
NFA32	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb modifiziert werden können.
NFA33	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb deaktiviert werden können.
NFA34	C	Verfügbarkeit	Analysen sollen im laufenden Betrieb möglich sein.
NFA35	C	Verfügbarkeit	Visualisierungen sollen im laufenden Betrieb möglich sein.
NFA36	C	Verfügbarkeit	Kapazität- und Systemerweiterungen sollen im laufenden Betrieb möglich sein.
NFA37	C	Bedienbarkeit	Das System soll von Fachleuten bedient werden können.
NFA38	C	Bedienbarkeit	Das System soll möglichst intuitiv bedienbar sein.
NFA39	B	Bedienbarkeit	Analysen sollen durch den Anwender auswählbar sein.
NFA40	B	Bedienbarkeit	Visualisierungen sollen durch den Anwender auswählbar sein.
NFA41	C	Bedienbarkeit	Das GUI kann frei gestaltet werden.
NFA42	A	Bedienbarkeit	Quellen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.
NFA43	C	Bedienbarkeit	Analysen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.
NFA44	C	Bedienbarkeit	Visualisierungen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.
NFA45	B	Performanz	Das System soll auf Anfragen, die einen Zeitraum von maximal zwei Jahren umfassen, innerhalb einer Stunde das Ergebnis liefern können.

4. Projektmanagement

UG

Projektmanagement ist eines der kritischen Erfolgsfaktoren für die erfolgreiche Projektarbeit. Die Vernachlässigung oder Missachtung gehören zu den Hauptgründen für das Scheitern von Projekten. In diesem Kapitel erfolgt die Darstellung des Projektmanagements der Projektgruppe, welches das verwendete Vorgehensmodell, die Projektorganisation, die Toolunterstützung und den Projektverlauf beschreibt.

4.1. Vorgehen im Projekt

UG

In diesem Kapitel wird das Vorgehen der Projektgruppe während der Projektbearbeitung beschrieben. Für den erfolgreichen Projektverlauf sind das Projektmanagement und die Projektorganisation von entscheidender Bedeutung. Zunächst wird das Vorgehensmodell der Projektgruppe beschrieben, das zu Beginn des Projektes festgelegt wurde. Die darauf basierende Projektorganisation, wird im nächsten Abschnitt dargestellt. Für das effiziente Management und die Organisation des Projektes wurden verschiedene Tools und Methoden verwendet, welche kurz im Abschnitt Toolunterstützung aufgeführt sind. Abschließend gibt es einen Überblick über den zeitlichen Verlauf des Projektes und mögliche Planabweichungen werden analysiert.

4.1.1. Vorgehensmodell Scrum

Bevor mit der Arbeit an einem Projekt begonnen werden kann, muss ein geeignetes Vorgehensmodell gewählt werden. Das richtige Vorgehen ist entscheidend für den erfolgreichen Verlauf des Projektes und ist von verschiedenen Faktoren wie Projektumfeld, Projektgröße sowie Projektgegenstand abhängig. Im Laufe der Jahre haben sich zum Teil recht unterschiedliche Vorgehensmodelle etabliert. Für die Auswahl lieferte die sehr umfangreiche Seminararbeit zum Thema Projektmanagement und Vorgehensmodelle ausführliche Erkenntnisse und Empfehlungen. Aufgrund der geringen Projektgröße erwies sich ein agiles Vorgehensmodell am geeignetsten, welches um wichtige Teilaspekte klassischer Vorgehensmodelle ergänzt wurde. Scrum wurde aus mehreren Gründen ausgewählt. Zum einen ist Scrum für Softwareprojekte dieser Art mit dynamischem Umfeld konzipiert. Wenn also zu erwarten ist, dass Anforderungen vorab nicht vollständig beschrieben werden können, sich Rahmenbedingungen oder ganze Projektziele während des Projektes ändern, sind agile Vorgehensweisen wie Scrum vorteilhaft. Darüber hinaus ist Scrum durch die weite Verbreitung (siehe Studie „Status Quo Agile“ [Stu]) sehr ausgereift und in vielen Projektmanagementanwendungen wie JIRA von Atlassian integrierter Bestandteil. Weiteres und zugleich ausschlaggebendes Kriterium war die geringe Projektgröße, bei der Scrum besonders gute Ergebnisse liefert. Entgegengesetzt der klassischen Vorgehensmodelle kann nach der Einrichtung

der Projektumgebung direkt mit der Realisierung begonnen werden, ohne das gesamte Projekt vorzuplanen. Darüber hinaus ist der Projektmanagementaufwand nicht so umfangreich wie von klassischen Modellen, so dass man nicht in Gefahr läuft das Projektergebnis zu vernachlässigen. Darüber hinaus ist bei Srcum die funktionierende Software wichtiger als umfangreiche Dokumentation und die Zusammenarbeit mit den Projektbetroffenen bedeutsamer als die Verfolgung eines festgelegten Plans.

Funktionsweise Scrum

Der folgende Abschnitt gibt zunächst einen Überblick über die grundlegende Funktionsweise von Scrum, um die individuellen Anpassungen in diesem Projekt besser abgrenzen zu können (vgl. [Tie10] [Wir11a] [Wir11b] [Scr]). Das Framework stellt ein einfaches Prozessmodell für die Entwicklung von Software dar und basiert auf den Grundsätzen des Agile Manifesto, in dem 2001 durch eine Gruppe von Entwicklern Prinzipien zur Arbeit an Software formuliert wurden. Diese „agilen Werte“ (vgl. [Agi])

- „Individuals and interactions over processes and tools“
- „Working software over comprehensive documentation“
- „Customer collaboration over contract negotiation“
- „Responding to change over following a plan“ [Agi]

Sprints als Kernelement von Scrum

Scrum-Projekte werden inkrementell entwickelt. Der Entwicklungsprozess ist in mehrere Iterationen unterteilt, die als Sprints bezeichnet werden. Innerhalb jedes Sprints soll das Team nach Möglichkeit ein fertiges und potentiell einsetzbares Inkrement erzeugen, welches dem Auftraggeber vorgestellt wird. Am Sprintende steht die lauffähige Komponente zur Verfügung. Die Sprints sollen eine feste Länge haben, die nicht mehr als einen Monat beträgt. Zum Ende eines jeden Sprints werden die Ergebnisse vorgestellt und abgenommen oder zur Verbesserung in den Nächsten Sprint überführt.

Anforderungen in Form von User Stories

Anforderungen bilden die Grundlage von Scrum und werden zu Beginn des Projektes erhoben und in einer Liste, dem so genannten Product Backlog, gesammelt. Diese werden nach und nach in den einzelnen Sprints umgesetzt und die Funktionalitäten bereitgestellt. Scrum definiert keine inhaltlichen Kriterien an Anforderungen, sondern besagt lediglich, dass allein der Product Owner für das Anforderungsmanagement verantwortlich ist. In Scrum-Projekten ist die Definition der Anforderungen in Form von User Stories verbreitet, welche die Funktionalitäten des Systems aus Sicht des Benutzers beschreiben. User Stories werden bewusst offen formuliert, da diese im Verlauf des Projektes präzisiert werden und sich ggf. noch ändern können. Nachdem eine oder mehrere Anforderungen einem Sprint zugeordnet sind, erfolgt zu Beginn die Präzisierung und Ableitung der erforderlichen Arbeitspakete.

Für die Bearbeitung eines Projektes unterscheidet Scrum zwischen drei zentralen Rollen, den Scrum Master, den Product Owner und das Entwicklungs-Team.

Product Owner

Der Product Owner verwaltet die Anforderungen an die Entwicklung und ist der Repräsentant des Kunden. Er integriert den Kunden in den Entwicklungsprozess und entscheidet, welche Funktionalitäten umgesetzt werden sollen. Der Product Owner sorgt dafür, dass die Anforderungen vom Team verstanden und mit dem größten Kundennutzen realisiert werden. Die Release-Planung gehört genauso, wie die Abnahme der Produktinkrements zu seinen Aufgaben. Die Anforderungen an das System werden im Sprint Backlogs festgehalten, die in den einzelnen Sprints ausgewählt und abgearbeitet werden. Der Poduct Owner hat erheblichen Einfluss auf den Erfolg oder Misserfolg des Projektes und muss wichtige Entscheidungen treffen. Darum sollte diese Rolle durch eine Person mit entsprechender Entscheidungsbefugnis eingenommen werden.

Team

Dieses besteht im Idealfall aus fünf bis neun Mitgliedern, welche sich selbstorganisieren und eigenverantwortlich das System in den Sprints entwickeln. Ein harmonisches, technisch versiertes Team mit Programmier- und Softwareentwicklungskenntnissen ist maßgeblich für den Erfolg. Darüber hinaus sollte es sich durch Respekt, Verständnis, gegenseitige Unterstützung und enge Zusammenarbeit auszeichnen. Im Rahmen der Selbstorganisation erfolgt auch der Umgang mit Konfliktsituationen eigenverantwortlich und wird bei Bedarf vom Scrum Master unterstützt.

Scrum Master

Übergeordnet ist er für die Implementierung und Umsetzung des Vorgehensmodells verantwortlich. Im eigentlichen Entwicklungsprozess hat er die Aufgabe, die Werte und Regeln während des Projektes zu wahren und das Team beim Formen, Zusammenwachsen und Entwickeln zu unterstützen. Die Moderation der Meetings gehört ebenfalls zu seinen Aufgaben. Hindernisse sind zu beseitigen, damit sich das Team ganz auf die Entwicklung des Systems konzentrieren kann. Er ist die Schnittstelle zur Außenwelt und zuständig für die Kommunikation mit Nichtteammitgliedern, um das Team von dieser Aufgabe zu befreien und dient als zentrale Anlaufstelle. Er hat ebenso dafür zu sorgen, dass die nötige Infrastruktur zur Verfügung steht. Der Scrum Master besitzt keine Weisungskompetenz und ist nicht höher als das Team gestellt.

Für den erfolgreichen Projektablauf ist der Informationsaustausch zwischen den projektbeteiligten Rollen besonders wichtig. Aus diesem Grund sieht Scrum einige Arten von Meetings vor, die jeweils einem bestimmten Zweck dienen und zu definierten Zeitpunkten im Scrum-Prozeß stattfinden. Die einzelnen Meetings werden folgend vorgestellt.

Sprint Planning Meeting

Das Sprint Planning Meeting findet zu Beginn eines Sprints statt und dient zu dessen Planung. Zunächst werden die Ziele für den kommenden Sprint bestimmt. Dazu werden die umzusetzenden Anforderungen gemeinsam vom Product Owner und dem Team aus dem Product Backlog ausgewählt, welche in Form von User Stories vorliegen. Im zweiten Teil des Meetings werden die ausgewählten User Stories genauer spezifiziert, indem eine Story in mehrere Tasks zerlegt wird, um die einzelnen Arbeitspakete zu ermitteln.

Daily Scrum

Bezeichnet kurze tägliche Meetings die sprintbegleitend zu einem festen Zeitpunkt stattfinden. Neben dem Scrum Team nimmt der Scrum Master an den Meetings teil. Die Teammitglieder sollen dabei kurz die folgenden Fragen beantworten:

- Was habe ich seit dem letzten Meeting erreicht?
- Was will ich bis zum nächsten Meeting erreichen?
- Was steht mir dabei im Weg?

Ziel ist dabei den Entwicklungsprozess zu beobachten, um ggf. weitere Schritte planen zu können und Probleme rechtzeitig zu erkennen. Auf Grundlage dieser Erkenntnisse können Anpassungen vorgenommen werden, um das Ziel wie geplant zu erreichen.

Sprint Review

Zum Abschluss eines Sprints findet das sogenannte Sprint Review Meeting statt, indem die Ergebnisse des Sprints dargestellt und besprochen werden. Nach Möglichkeit sollen die implementierten Funktionalitäten präsentiert werden. Dabei ist eine Livedemo anstelle von Präsentationsfolien vorzuziehen. An dem Meeting nehmen das Team, Product Owner und Scrum Master und ggf. weitere Stakeholder des Projekts teil. Die Ergebnisse werden gemeinsam besprochen und es können Änderungswünsche berücksichtigt werden. Der Product Owner entscheidet, ob die entwickelten Aufgaben/Stories abgenommen werden oder ob in kommenden Sprints weiter daran gearbeitet werden muss. Der Entwicklungsprozess wird für den Kunden transparent gestaltet und er hat die Möglichkeit der intensiven Mitgestaltung. Anforderungsänderungen können so frühzeitig in den Entwicklungsprozess integriert werden.

Sprint Retrospektive

Dieses Teamtreffen findet nach dem Sprint Review satt und dient der Analyse des bisherigen Entwicklungsprozesses. Es sollen sowohl positive als auch negative Ereignisse des vorherigen Sprints festgehalten werden, um Verbesserungsmöglichkeiten zu identifizieren, die im kommenden Sprint angewandt werden können. Es ist sozusagen eine Metadiskussion über den eigentlichen Entwicklungsprozess zur Ableitung von Verbesserungsmaßnahmen

Bevor der Prozessablauf betrachtet werden kann, werden noch die Artefakte (Erzeugnisse) erläutert. Dabei handelt es sich um Ergebnisse des jeweiligen Entwicklungsprozesses. Neben den Ergebnissen der Programmierung gehören dazu auch weitere für die Projektarbeit erforderliche Ergebnisse, wie die Erstellung der Anforderungen und Aufgabenpakete.

Product Backlog

Dieses Erzeugnis ist das zentrale Anforderungsdokument in einem Scrum-Projekt. Die Anforderungen werden in Form von User Stories festgehalten, die in einer bestimmten Reihenfolge angeordnet sind. Die Reihenfolge soll dabei ihrem geschäftlichen Mehrwert entsprechen und wird vom Product Owner festgelegt. Zweck dieser Anordnung ist es diejenigen Anforderungen zuerst umzusetzen, die den höchsten Mehrwert für den Kunden bieten. Neben der Priorität sollte der

Realisierungsaufwand der jeweiligen Story angegeben werden. Das Product Backlog gibt somit einen Überblick über die wesentlichen Features des zu entwickelnden Systems. Die Präzisierung der User Stories erfolgt in den Sprint Planning Meetings, so dass der Planungsaufwand auf mehrere Sprints verteilt wird.

Sprint Backlog

Das Sprint Backlog ist das Ergebnis des Sprint Planning Meetings und enthält die Aufgaben/User Stories, die im kommenden Sprint abgearbeitet werden sollen. Die einzelnen User Stories werden in Tasks aufgesplittet, um festzulegen welche Teilaufgaben für die Implementierung erledigt werden müssen.

Produktinkrement

Das Inkrement ist das Ergebnis aller bisherigen Sprints, also die implementierte Software im jeweiligen Entwicklungsstadium. Nach jedem Sprint soll sich das Inkrement in einem nutzbaren, getesteten, dokumentierten Zustand befinden und den Anforderungen aus dem Product Backlog genügen. Nach dem letzten Sprint entspricht das Inkrement dem fertigen System.

Im Anschluss der Präzisierung der Komponenten wird Scrum folgend als Prozess dargestellt: Der

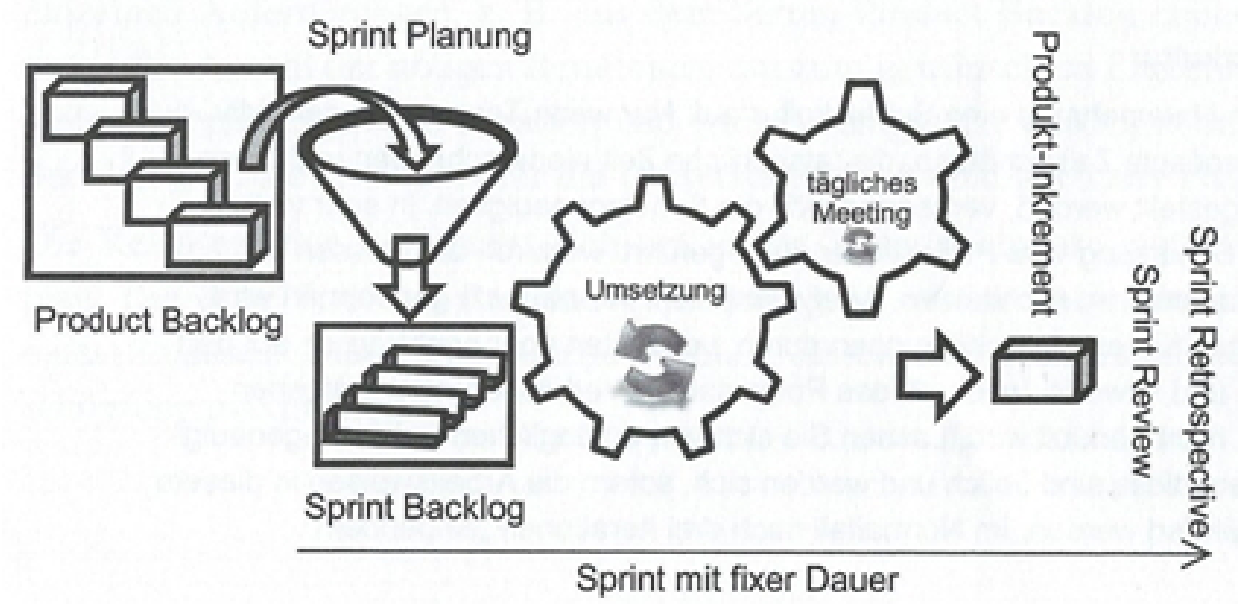


Abbildung 4.1.: Vorgehensmodell Scrum [Tie10]

Product Owner führt zu Projektbeginn mit dem Kunden eine Anforderungsanalyse durch, und erstellt auf der Grundlage das Product Backlog, wie zu sehen in Abbildung 4.1. Es ist ein lebendes Dokument, welches laufend verändert und weiterentwickelt wird. Dabei können Anforderungen hinzukommen, sich ändern oder gelöscht werden.
Der Scrum Master moderiert das Sprint Planning Meeting, in der dem Team vom Product Owner einen Vorschlag über die umzusetzenden Anforderungen (User Stories) gemacht wird. Nach der detaillierten Vorstellung durch den Product Owner, zerlegt das Team die User Stories, identifiziert Aufgaben mit den erforderlichen Tasks und Aktivitäten, die zur technischen Umsetzung

nötig sind. Danach wird der Aufwand vom Team in Arbeitseinheiten, die einen Viertel Tag entsprechen, geschätzt und daraus ein priorisiertes Sprint Backlog erstellt. Dieses enthält neben den Prioritäten, die Anforderungen an das System mit den dazugehörigen Aktivitäten. Gleichzeitig werden die Abnahmekriterien definiert. Am Ende der Sitzung beginnt der sogenannte Sprint und leitet die Umsetzung der definierten Aufgaben ein. Dieser sollte maximal 30 Tage dauern und am Ende ein funktionsfähiges Inkrement (Produkt) ergeben. Während der Umsetzung wird am Ende eines jeden Arbeitstages der aktuelle Fortschritt betrachtet, sowie der Restaufwand abgeschätzt und dargestellt. In den täglichen 15 minütigen Daily Scrum Meetings besprechen das Team, der Product Owner (nur als Zuhörer) und der Scrum Master die Probleme, den Fortschritt sowie die tagesaktuellen Vorhaben. Das Sprint Review wird zum Schluss eines Sprints durchgeführt. Dabei stellt das Team dem Kunden die Arbeitsergebnisse mittels einer Livedemonstration vor. Der Kunde kann dann das Ergebnis über den Product Owner abnehmen oder bei Unvollständigkeiten sowie Fehlern ablehnen. Die Moderation wird vom Scrum Master übernommen.
Der letzte Schritt eines jeden Sprints ist die Sprint Retrospective, mit dem Ziel die Zusammenarbeit im Team und den Prozess zu verbessern. Die Verbesserungsvorschläge müssen die SMART-Kriterien erfüllen.
Scrum eignet sich für kleine Teams mit fünf bis neun Mitgliedern und kann auch in großen verteilten Teams eingesetzt werden. Mit steigender Mitgliederzahl nimmt der Kommunikations- sowie Dokumentationsaufwand stark zu. Genauso wie viele andere agile Modelle, hat Scrum eine besonders gute Eignung für große, komplexe Softwareprojekte mit neuen Technologien. Da eine detaillierte Planung zu Beginn nicht existiert, ist Flexibilität besonders wichtig. Das tägliche Arbeiten mit Scrum kann durch angebotene Managementtools erheblich vereinfacht werden.

4.1.2. Vorgehensmodell in der Projektgruppe

Im Rahmen der Projektgruppe ist das agile Vorgehensmodell Scrum entsprechend den Anforderungen der Projektgruppe angepasst und um Aspekte der klassischen Vorgehensmodelle erweitert worden.
Dieses wird folgend beschrieben und in Abbildung 4.2 veranschaulicht: Die wesentlichen Kriterien, die bei der Erstellung des Vorgehensmodells zu beachten galten, waren die geringeren wöchentliche Arbeitszeit von 16 Stunden und die minimale Anzahl von fünf Projektmitgliedern. Aufgrund der geringeren wöchentlichen Arbeitszeit wurde die Anzahl der Arbeitstreffen auf zwei bis drei Tage pro Woche reduziert. Die Länge der Sprints ist trotzdem bei der empfohlen Länge von 4 Wochen geblieben, um den großen Vorteil der Dynamik zu wahren. Somit ist das rechtzeitige Erkennen von Fehlern, die Einbringen von Verbesserungen sowie die Überprüfung des richtigen Weges frühzeitig gegeben. Dabei ist die Gefahr der Übernahme von nicht erledigten Aufgaben in den nächsten Sprint gestiegen, welches aber aufgrund der überwiegenden Vorteile bewusst in Kauf genommen wurde.
Bei den Anforderungen wurde auf die Vorteile der klassischen Vorgehensmodelle gesetzt. Anstelle der von Scrum empfohlenen unscharfen User-Stories sind die Anforderungen detailliert in einem Lastenheft konkretisiert worden. Dieses ermöglicht die genaue Überprüfung der Projektergebnisse, die frühzeitige Konkretisierung der Testfälle und stellt sicher, dass von Anfang an in die richtige Richtung entwickelt wird. Da die Anforderungen die Grundlage des Projektes bilden, wurden diese zu Beginn im ersten Sprint ausgiebig erhoben und klassisch in Funktionale- und Nicht-Funktionale Anforderungen unterteilt. Die Erarbeitung der Anforderungen erfolgte nach Scrum zwischen dem Entwicklungsteam und dem Product Owner und wurde nach der Abnahme in einer Liste, dem so genannten Product Backlog, überführt. Entgegengesetzt der klassischen

Ansätze sind diese Anforderungen nicht verbindlich oder fest vorgegeben und können während des Projektverlaufs ggf. ergänzt oder verändert werden.
Neben den Rollen von Scrum sind noch weitere Rollen definiert worden, um die im Projekt anfallenden Aufgaben besser organisieren und strukturieren zu können. Diese sind im folgenden Kapitel Projektorganisation ausführlich beschrieben. Die von Scrum vorgegebenen Rollen wurden in unserem Vorgehensmodel weitestgehend übernommen und teilweise modifiziert.
Der Product Owner ist durch drei Personen, den Betreuern des Projektes, besetzt. Das Team nimmt neben den Entwicklungsaufgaben auch Teilaufgaben des Scrum Masters, die der Infrastrukturbereitstellung, wahr. Diese ist auf weitere Rollen innerhalb des Teams, wie beispielsweise des Administrators, verteilt. Darüber hinaus ist der Scrum Master entgegengesetzt der Scrum-Vorgaben auch an der Projektarbeit beteiligt und erledigt Aufgaben des klassischen Projektmanagers.

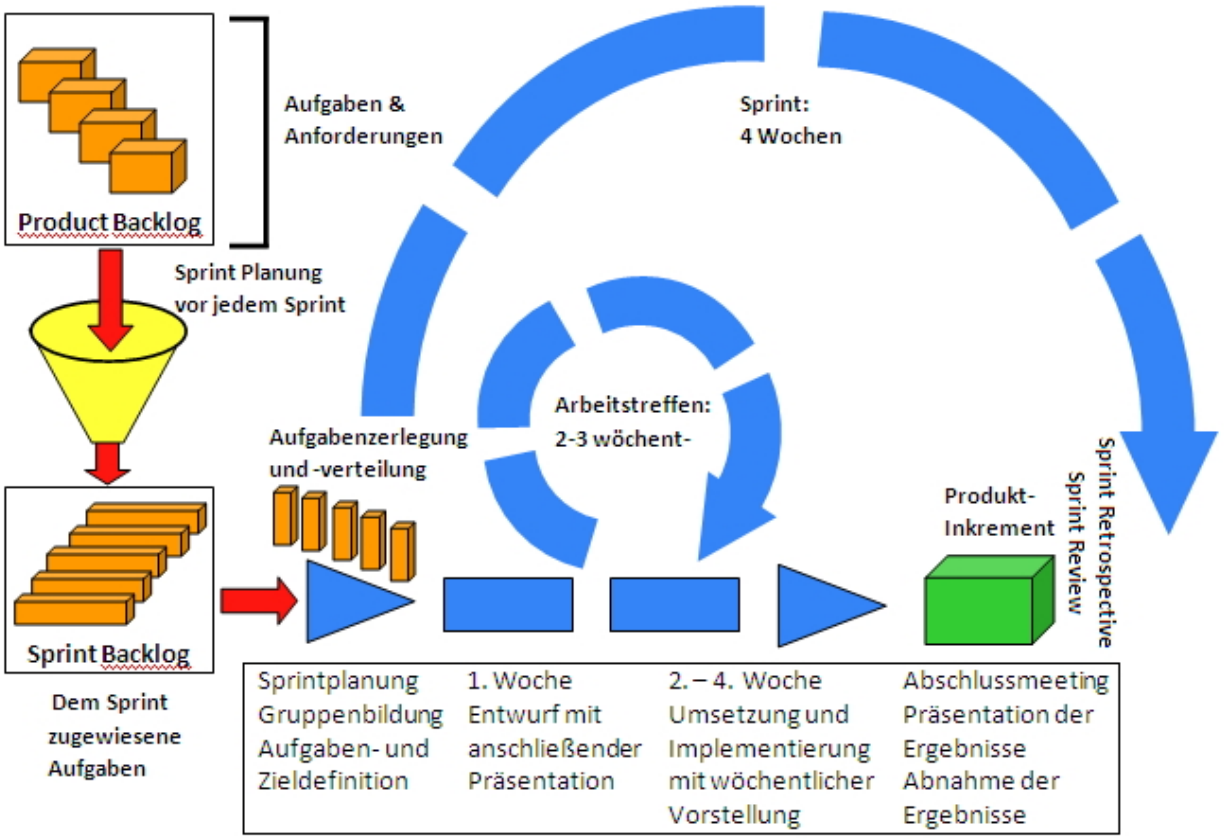


Abbildung 4.2.: Vorgehensmodell LAOTSE angelehnt an [Tie10]

Die von Scrum empfohlenen Meetings wurden weitestgehend übernommen und teilweise an die Bedürfnisse angepasst. Zunächst ist ein Weekly Scrum Meeting eingeführt worden, in dem die Ergebnisse des jeweiligen Sprints wöchentlich den Product Ownern vorgestellt und organisatorische Themen geklärt wurden. Dabei sind zentral folgende Fragen beantwortet worden:

- Was wurde seit dem letzten Meeting erreicht?
- Was soll bis zum nächsten Meeting erreicht werden?
- Was steht dabei im Weg?

Ziel ist dabei den Entwicklungsprozess kontinuierlich zu beobachten, um ggf. weitere Schritte planen zu können und Probleme rechtzeitig zu erkennen. Auf Grundlage dieser Erkenntnisse können Anpassungen so frühzeitig in den Entwicklungsprozess integriert werden. Die einzelnen Meetings konnten aus zeitlichen und organisatorischen Gründen nicht immer exakt zu den von Scrum geforderten Zeiten durchgeführt werden. Insbesondere wurden das Sprint Planning und Sprint Review Meeting nicht exakt nach der Beendigung und dem Beginn der Sprints, sondern im Rahmen der Weekly Scrum Meetings abgehalten.
Die Artefakte (Ergebnisse) entsprechen, bis auf einer kleinen Anpassung im Product Backlog, denen der von Scrum. Die Anforderungen werden anstelle der von Scrum empfohlenen unscharfen User Stories genau präzisiert und überprüfbar formuliert.
Um die Vorteile der klassischen Projektmanagement-Methoden im Projekt zu berücksichtigen, wird das agile Vorgehensmodell um klassische Methoden erweitert. Dabei werden klare Strukturen zur detaillierten Planung und Spezifikation berücksichtigt. Die klaren Strukturen umfassen die gründliche Analyse und Definition der Entwicklungsumgebung der Projektgruppe, sowie die Festlegung klarer Rollen in der Projektgruppe.
In der detaillierten Planung geht es vorrangig um die Strukturierung des Projektes in Teilaufgaben und Arbeitspakete, welche das Projekt plan- und kontrollierbar machen. Eine Spezifikation umfasst die genaue Beschreibung des zu erstellenden Systems durch die Auflistung der Anforderungen. Ziel ist dabei die Anforderungen detailliert und überprüfbar zu definieren und in einem Lastenheft schriftlich zu fixieren.

4.2. Rollen

UG

Zum Projektstart wurden wichtige, zentrale Projektrollen festgelegt, um organisatorische Aspekte innerhalb des Projektes besser und zielführender umsetzen zu können. Dazu sind die folgenden Rollen definiert worden, welche über die gesamte Projektdauer beibehalten werden und abhängig von der Projektphase unterschiedlichen Aufwand umfassen.

4.2.1. Scrum Master / Projektmanager

Die Rolle des Scrum Masters wird von Uwe Griese wahrgenommen. In dieser Rolle ist er in der Projektgruppe für die Umsetzung des Vorgehensmodells nach Scrum zuständig. Dazu wird zu Beginn des Projektes das Vorgehensmodell, unter Berücksichtigung der Gegebenheiten und Bedürfnisse der Projektgruppe, erarbeitet und eingeführt. Im Projektverlauf hat er die Aufgabe das Vorgehensmodell fortlaufend zu überprüfen, die Einhaltung der Regeln zu wahren und das Team beim Zusammenwachsen und Entwickeln zu unterstützen. Des Weiteren sind Hindernisse, wie Probleme im Projektteam oder Störungen von außen, zu beseitigen, damit sich das Projektteam ganz auf die Entwicklung des Systems konzentrieren kann. Er ist die Schnittstelle zur Außenwelt und moderiert die Meetings, um das Team zu entlasten und dient als zentrale Anlaufstelle. Ebenso hat er dafür zu sorgen, dass die nötige Infrastruktur zur Verfügung steht. Der Scrum Master besitzt keine Weisungskompetenz und ist nicht höher als das Team gestellt.
Um die Vorteile der klassischen Projektmanagement-Methoden im Vorgehensmodell zu berücksichtigen, wird das Aufgabenfeld des Scrum Masters um klassische Methoden erweitert. In der Projektmanagertätigkeit ist er für die Festlegung klarer Strukturen, der detaillierten Planung, der

Spezifikation sowie der Anforderungsanalyse und -erhebung zuständig. Darüber hinaus gehört die Auswahl und Pflege der PM-Unterstützungswerkzeuge zu seinen Aufgaben, welche im Abschnitt **??** erläutert werden. Die Rolle des Scrum Masters wird von Uwe Griese wahrgenommen. In dieser Rolle ist er in der Projektgruppe für die Implementierung und Umsetzung des Vorgehensmodells nach Scrum zuständig. Dazu wird zu Beginn des Projektes das Vorgehensmodell, unter Berücksichtigung der Gegebenheiten und Bedürfnisse der Projektgruppe, erarbeitet und eingeführt. Im Projektverlauf hat er die Aufgabe das Vorgehensmodell fortlaufend zu überprüfen, die Einhaltung der Regeln zu wahren und das Team beim Zusammenwachsen und Entwickeln zu unterstützen. Des Weiteren sind Hindernisse, wie Probleme im Projektteam oder Störungen von außen, zu beseitigen, damit sich das Projektteam ganz auf die Entwicklung des Systems konzentrieren kann. Er ist die Schnittstelle zur Außenwelt und moderiert die Meetings, um das Team zu entlasten und dient als zentrale Anlaufstelle. Ebenso hat er dafür zu sorgen, dass die nötige Infrastruktur zur Verfügung steht. Der Scrum Master besitzt keine Weisungskompetenz und ist nicht höhergestellt als das Team.
In der Projektmanagertätigkeit ist er für die Festlegung klarer Strukturen, der detaillierten Planung und der Spezifikation zuständig. Darüber hinaus ist er für die Auswahl und der Pflege der Unterstützungswerkzeuge zuständig, welche im Abschnitt **??** erläutert werden.

4.2.2. Product Owner

Die Rolle der Product Owner wird von den PG-Betreuern wahrgenommen, welche die Interessen des Auftraggebers repräsentieren. Sie sollen die fachliche Seite des Auftraggebers vertreten, Anforderungen stellen und die Umsetzung des Projektes im Hinblick auf die Funktionalität, Benutzbarkeit, Performanz und Qualität verfolgen. Sie geben eine klare Vision für das zu erstellende System vor, auf dessen Grundlage die Anforderungen und Funktionalitäten des Systems gemeinsam erfasst werden. Als Vertreter des Auftraggebers nehmen sie die Anforderungen und Funktionalitäten ab. Um dieses gewährleisten zu können, werden sie wöchentlich über den aktuellen Projektverlauf informiert und sind in der Sprintplanung und -Abnahme involviert.

4.2.3. Administration

Der Administrator ist für die Bereitstellung und Administration der benötigten Infrastruktur verantwortlich. Dazu zählen im Wesentlichen die VM-Umgebungen, benötigte Tools und Werkzeuge, sowie erforderliche Drittsoftware zur Realisierung. Die Administration und Betreuung einzelner Werkzeuge und Systeme wird im Projektverlauf, aus Aufwands- und Einarbeitungsgründen, auch von den einzelnen Projektmitgliedern vorgenommen. Die einzelnen Werkzeuge und eingesetzten Systeme werden im Abschnitt Technologie dargestellt. Darüber hinaus ist er für die Benutzerverwaltung zuständig und dient als Ansprechpartner bei Problemen. Er trägt die Verantwortung für den reibungslosen Betrieb der Infrastruktur und dient als direkte Schnittstelle zu den Projektbetreuern, welche die benötigten Ressourcen bereitstellen. Der Administrator ist für die Bereitstellung und Administration der benötigten Infrastruktur verantwortlich. Dazu zählen im Wesentlichen die VM-Umgebungen, die benötigten Projektmanagementwerkzeuge und die benötigten Systeme und Werkzeuge die zur Entwicklung des Systems benötigt werden. Die Administration und Betreuung einzelner Werkzeuge und Systeme wird im Projektverlauf, aus Aufwands- und Einarbeitungsgründen, auch von den einzelnen Projektmitgliedern vorgenommen. Die einzelnen Werkzeuge und eingesetzten Systeme werden im Abschnitt Toolunterstützung, mit deren Verantwortlichkeiten, dargestellt. Darüber hinaus ist er für die Benutzerverwaltung zuständig und

dient als Ansprechpartner bei Problemen. Er trägt die Verantwortung für den reibungslosen Betrieb der Infrastruktur und dient als direkte Schnittstelle zu den Projektbetreuern, welche die benötigten Ressourcen bereitstellen. Diese Rolle wird im Projekt von Dennis Nowak wahrgenommen.

4.2.4. Testbeauftragter / Qualitätsmanagement

Das Test- und Qualitätsmanagement wird von Lina Spiekermann übernommen. Inhalt des Testmanagements ist die Definition, Koordination und Kontrolle der Testaktivitäten. Dazu ist zunächst die Planung und Umsetzung der Testumgebung erforderlich, welche im Kapitel 5 Technologien dargestellt wird.
Im Rahmen der Definition werden die Vorgaben bezüglich der Qualitätssicherung und Tests festgelegt. Dabei sind unter anderem die Testmethoden und -werkzeuge, der Grad der Testabdeckung sowie die Art der Dokumentation der Testfälle und deren Ergebnisse zu bestimmen. Ziel der Koordination und Kontrolle ist die Einhaltung aller definierten Testvorgaben um den Grad der Testabdeckung zu erreichen.
Die Erstellung, Durchführung und Dokumentation der Testfälle erfolgt durch die Entwickler innerhalb der Entwicklungsphasen. Das Qualitätsmanagement befasst sich mit der Planung, Steuerung und Überprüfung der Qualität des zu entwickelnden Systems und des Entwicklungsprozesses. Die produktorientierte Perspektive zielt auf die Einhaltung der Anforderungen an das zu entwickelnde System ab, welche im Lastenheft fixiert sind. Bei der prozessorientierten Perspektive werden die Anforderungen an den Prozess der Softwareentwicklung bestimmt, welche als Vorgaben für die Gestaltung und Implementierung dienen.

4.2.5. GUI-Beauftragter

Die Rolle des GUI-Beauftragten, wahrgenommen durch Mark Milster, ist für die allgemeine Festlegung der Design-Grundlagen, sowie deren Einhaltung zuständig. Dabei steht die Abbildung der Funktionen unter Berücksichtigung von Usabilitygesichtspunkten im Vordergrund.
Zusätzlich soll die Effektivität und Effizienz der GUI auf einem hohen Niveau gehalten werden. Die Fehlertoleranz, Bedienbarkeit und Individualisierbarkeit werden im Laufe des Projektes immer wieder kritisch betrachtet. Der GUI-Beauftragte dient als Ansprechpartner für die Projektgruppe, der bei Designfragen unterstützt und die GUI-Qualitätssicherung übernimmt. Designentscheidungen sind im Kollektiv zu bestimmen.

4.2.6. Visualisierungsbeauftragter

Diese Rolle wird durch Kristian Bruns wahrgenommen. In dieser Rolle beschäftigt er sich mit der Darstellung der projektrelevanten Themen durch geeignete grafische Mittel. Er trägt die Verantwortung für das Layout, Format und der einheitlichen Darstellung der Grafiken, Diagramme und Präsentationen. Darüber hinaus ist er für das Design des Projektgruppenlogos verantwortlich. Dabei ist er für jede Art von Grafiken zuständig. Das bedeutet, sobald eine Grafik für das Projekt benötigt wird ist der Visualisierungsbeauftragte dafür zuständig, diese Grafik zu erstellen. So ist der Visualisierungsbeauftragte zum Beispiel dafür zuständig, das Logo oder Systemdiagramme zu erstellen. Gerade bei Systemdiagrammen ist es allerdings so, dass diese schnell veraltet sind. Dementsprechend ist der Visualisierungsbeauftragte auch dafür zuständig, diese Diagramme auf

dem aktuellen Stand zu halten, sodass immer das korrekte System abgebildet wird. Seine Aufgabe ist es also alle Arten von Diagrammen und Grafiken zu erstellen und diese während des Projektverlaufs auf dem aktuellen Stand zu halten.

4.2.7. Dokumentations-Manager

Der Dokumentations-Manager trägt die Verantwortung für die Erstellung und Pflege der projektrelevanten Dokumente, Protokolle und Berichte. Dazu wurden zu Beginn gemeinsam die Dokumentationsregeln in der Projektgruppe festgelegt. Als Grundlage für die Erstellung der Dokumente wurde LaTeX festgelegt und die Dokumentationsstruktur, auf dem mittels SVN gesicherten Arbeitsbereich der Projektgruppe, angelegt.
Darüber hinaus ist im Confluence ein Dokumentationsbereich für die projektbegleitende Dokumentation angelegt worden. Dort sind sämtliche Ergebnisse, Anleitungen sowie die Bearbeitung der Arbeitspakete festgehalten. Für die Inhalte und Erstellung ist nicht alleine der Dokumentations-Manager zuständig, diese werden im Kollektiv erarbeitet. Neben der regelmäßigen Überwachung der ordnungsgemäßen Dokumentation ist er für die Integration aller Inhalte in den Zwischen- und Abschlussbericht zuständig, damit diese strukturiert und übersichtlich zur Verfügung stehen. Diese Rolle wird durch Uwe Griese wahrgenommen.

4.2.8. LaTeX-Beauftragter

Der LaTeX-Beauftragte ist für die Erstellung und die Pflege aller LaTeX-Dokumenten zuständig. Des Weiteren für die Erstellung aller relevanten Ordnerstrukturen und der Recherche von Formatierungsmitteln. Diese Rolle wird von Lina Spiekermann wahrgenommen.

4.2.9. Chefentwickler

Der Aufgabenbereich des technischen Projektleiters oder auch Chefentwicklers liegt in der Koordination und Anleitung aller Entwickler auf dem Weg zu einem Softwareprodukt von möglichst hoher Qualität. Hierfür sollte er in Zusammenarbeit mit dem Entwicklerteam geeignete Codingstandards erarbeiten, die in einer übersichtlichen und leicht zu wartenden Softwarelösung resultieren. Dabei sollte er stets einen Blick auf den Erhalt einer übersichtlichen Gesamtarchitektur halten, um beispielsweise den Einbau moderner Techniken und Standards zu unterstützen, zu deren Aneignung er auch die Entwickler regelmäßig anzuregen hat. Diese Rolle wird von Lina Spiekermann wahrgenommen. Sie trägt die Verantwortung für die fachliche Softwareentwicklung, welche im Wesentlichen das Design/Architektur und die Implementierung umfasst. Zu den essentiellen Aufgaben gehören die Sicherstellung der fachlichen Qualität der Entwicklungsumgebung, die Entwicklungsplanung und Koordinierung der Entwicklung.

4.3. Toolunterstützung

UG

Während der Projektarbeit wurden verschiedene Werkzeuge eingesetzt, die der Entwicklung des Systems, der Organisation und der Steuerung des Projektes dienten. Nachfolgend werden die ausgewählten Werkzeuge und Systeme kurz aufgeführt und erläutert.

4.3.1. Confluence

Confluence ist eine Wiki-Software, die für die zentrale Speicherung von Informationen, wie zum Beispiel Protokollen genutzt werden kann. Die Wahl des Wikis ist dabei, wie auch schon bei JIRA, auf Confluence gefallen, da die Universität Oldenburg bereits eine Lizenz besitzt und so eine einfache Erstellung des Wikis gewährleistet war. Ebenfalls hat Confluence aber den Vorteil gegenüber anderenWikis, dass Confluence und JIRA integriert werden können und so eine Verzahnung der beiden Tools gewährleistet ist. In unserem Projekt wird Confluence für die Speicherung von Sitzungsprotokollen und die Erstellung von Stundenzetteln benötigt, da jeder Entwickler dokumentieren soll, wie viele Stunden er an dem Projekt gearbeitet hat.

4.3.2. JIRA

Je komplexer die Projekte und je höher die Anforderungen an die Projektmitarbeiter sind, desto umfangreicher und flexibler müssen die Werkzeuge sein, die in der Software-Entwicklung und im Aufgabenmanagement zum Einsatz kommen.
Darüber hinaus ist die Installation und webbasierte Administration zeitsparend und JIRA durch einfaches Einfügen von Plugins erweiterbar. JIRA unterstützt das agile Vorgehensmodell Scrum, welches die Grundlage des Vorgehensmodells der Projektgruppe ist. Über das Scrum Board ist das Projekt übersichtlich und strukturiert darstellbar und steuerbar. Darüber hinaus werden verschiedene Charts angeboten, die einen schnellen und unkomplizierten Überblick über den Status des Projektes geben und so einen übersichtlichen Plan- Ist-Vergleich ermöglichen.
Die geplanten Aufgaben des Projektes werden übersichtlich und zentral im Product Backlog verwaltet und den jeweiligen Sprints zur Bearbeitung zugeordnet. Dort werden sie weiter unterteilt, präzisiert und bearbeitet. Nicht erledigte Aufgaben können problemlos in andere Sprints übernommen werden. Das komfortable Zuweisen, Dokumentieren und Abschließen der Arbeitspakete ermöglicht eine verlässliche, multinutzerorientierte, steuerbare und überprüfbare Bearbeitung der Arbeitspakete. Durch die Protokollierung der geplanten und tatsächlich benötigten Arbeitszeit der einzelnen Arbeitspakete ist eine automatische Zeiterfassung möglich und der Projektfortschritt fortlaufend überprüfbar. Darüber hinaus ermöglicht JIRA die Integration und das Anbinden von weiteren wichtigen Funktionalitäten und Anwendungen wie der Zeiterfassung und dem Wiki-System Confluence. Vorteilhaft sind auch die Erfahrungen der Projektgruppe mit JIRA , aus dem Softwareprojekt, wodurch zusätzlicher Einarbeitungsaufwand gespart wird.

4.3.3. LaTeX

Dieses Textverarbeitungsprogramm wird zur Dokumentation und Erstellung von Projektdokumenten verwendet. Dabei wird die Distribution MiKTEX oder TEXLive und eine Entwicklungsumgebung wie TEXnicCenter oder Texmaker verwendet. Der Vorteil gegenüber anderen verbreiteten Textverarbeitungsprogrammen wie Microsoft-Office ist, das sich Dokumente in einzelne, separat bearbeitbare Abschnitte unterteilen lassen, welches eine multinutzerorientierte Bearbeitung ermöglicht. Somit kann parallel, ohne Versionskonflikte an verschiedenen Textpassagen gearbeitet werden. Das Zusammenführen der Abschnitte geschieht automatisch von der Entwicklungsumgebung, mittels sogenannter Inputbefehle der .tex-Dateien, in der Hauptdatei.
Als weiterer Vorteil ist die Trennung von Inhalt und Layout zu nennen, welche eine zentrale Formatierung mittels der Dokumentenklasse ermöglicht. Dies gestattet eine datenorientierte Arbeit, bei der eine Dokumentenvorlage erstellt und universell verwendet werden kann. Darüber hinaus

bietet LaTeXein sehr gutes Textbild und eine effiziente Quellenverwaltung. Neben diesen Gründen trugen die Erfahrungen des Projektteams mit dem Umgang von LaTeXzur Entscheidung für die Nutzung bei.

4.3.4. Versionsverwaltung SVN

Für die Versionsverwaltung wurde Subversion (SVN) gewählt. SVN zählt zusammen mit Git zu den bekanntesten Versionsverwaltungstools. Die Entscheidung ist für dieses Projekt auf SVN gefallen, da die meisten der Projektteilnehmer sicherer im Umgang mit SVN als mit Git sind. Da beide Tools nahezu den identischen Umfang bieten war dies der ausschlaggebende Punkt für Subversion.

4.4. Projektablauf

UG

Dieses Kapitel stellt den Zeitplan des Projektes sowie deren Verlauf in den einzelnen Sprints dar. Gemäß des LAOTSE-Vorgehensmodells ist die gesamte Projektlaufzeit in Sprints, mit einer Länge von einem Monat, unterteilt. Jeder dieser Sprints umfasst eine Reihe von Zielen, die für die Erstellung des Zielsystems erforderlich sind. Da in dem Vorgehensmodell auch wichtige Aspekte klassischer Vorgehensmodelle integriert sind, ist das Projekt zusätzlich in Phasen unterteilt. Diese Phasen bauen aufeinander auf und repräsentieren die wichtigsten Aufgabenbereiche, die zur Erstellung des Zielsystems erforderlich sind Abbildung 4.3. Folgend werden zunächst die Hauptphasen des Projektes erläutert, bevor der geplante Projektablauf dargestellt und anschließend inhaltlich der Ablauf der einzelnen Sprints dargestellt wird.

4.4.1. Projektphasen

Das Projekt begann mit der Seminarphase, in der die Projektteilnehmer zu projektrelevanten Themen eine Seminararbeit erarbeiteten. Die Themen boten den ersten Einstig in die Thematik des Projektes und deckten dabei die wesentlichen Schwerpunkte ab. Diese Erkenntnisse waren besonders Hilfreich für die Einrichtung der Projektmanagementumgebung und verschafften einen ersten Überblick über die Aufgabenstellung und Schwerpunkte des Projektes. Das war besonders hilfreich, um eine erste Vorstellung des Zielsystems zu bekommen und die Anforderungsanalyse im folgenden Schritt durchführen zu können.
In der zweiten Phase begann das Projekt offiziell mit dem ersten Sprint, in der die Anforderungsanalyse und die Einrichtung der Projektumgebung erfolgten. Ziel war es neben der ersten Vision des Zielsystems, die konkreten Anforderungen zu erarbeiten, welche in der folgenden Phase im Lastenheft verbindlich fixiert werde sollten. Darüber hinaus sollte das Vorgehensmodell und die daraus resultierende Projektmanagementumgebung eingerichtet werden.
Die dritte Phase diente der Erstellung eines verbindlichen Lastenheftes, welches präzise die Anforderungen beschreibt und als vertragliche Grundlage für die Erstellung des Zielsystems dient. Des Weiteren wurde in dieser Phase das Grobkonzept erstellt, welches ergänzend zum Lastenheft die wesentlichen Charakteristika des Zielsystems beschreibt. Dadurch sollte ein gemeinsames Verständnis von Anforderungen, Prozessablauf und Umsetzungsalternativen des Zielsystems geschaffen werden.

Die Dokumentation ist eine Querschnittsphase, welche projektbegleitend durchgeführt wird. Dabei sind fortlaufend alle projektrelevanten Informationen und Ergebnisse mittels geeigneter Dokumente, Protokolle, Berichte oder Anleitungen zu dokumentieren.
Das Testen ist essenziell für die Implementierung und überprüft die umgesetzten Funktionalitäten auf Erfüllung der Anforderungen an das Zielsystem, sowie den Programmcode auf Korrektheit und Funktionalität. Dazu wird zu Beginn ein Testkonzept mit einer entsprechenden Testumgebung erstellt, welche fortlaufend angewendet und ggf. erweitert wird. Den Abschluss dieser Phase bildet der Abschlusstest, welcher einer Abnahme durch den Auftraggeber unterliegt.
In der Implementierung werden die Funktionalitäten des Zielsystems gemäß der Anforderungen und Zielvorgaben erstellt. Dieser Bereich umfasst neben der Programmierung, die Einrichtung von Hard- und Software, die für das Zielsystem erforderlich ist.
Die beiden Berichtsphasen dienen der Erstellung des Zwischen- und Abschlussberichts, welche den Verlauf und die Ergebnisse des Projektes widerspiegeln. Da diese beiden Berichte eine wichtige Grundlage des Projektes darstellen, sind für die umfangreichere Erarbeitung die beiden Berichtsphasen eingeplant worden. Der Zwischenbericht ist nach der Hälfte des Projektes und der Abschlussbericht zum Ende bereitzustellen.

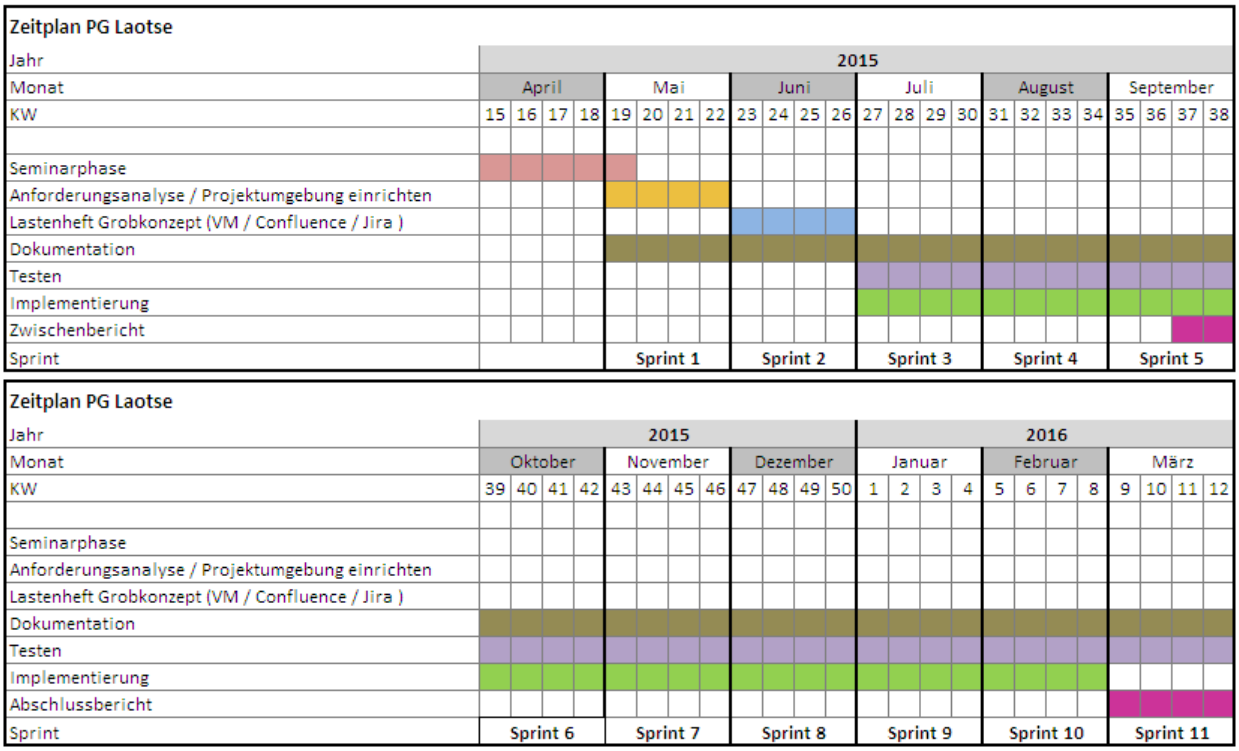


Abbildung 4.3.: Zeitplan LAOTSE

4.4.2. Projektsprints

In diesem Abschnitt sollen die einzelnen Sprints inhaltlich näher betrachtet und die wesentlichen Ergebnisse dargestellt werden. Für die einzelnen Sprints wurde im voraus eine Planung vorgenommen. Diese wird im Anhang im Kapitel B beschrieben. Zur Beschreibung dient zunächst die Darstellung der Arbeitspakete der jeweiligen Sprints, um den ersten inhaltlichen Überblick zu

verschaffen. Darauf aufbauend erfolgt die Darstellung des Sprintverlaufs mit der Vorstellung der wichtigsten Ergebnisse.

Sprint 1 Anforderungsanalyse / Projektumgebung einrichten – Mai 2015 Der erste Sprint diente im Wesentlichen der Anforderungsanalyse und der Einrichtung der Projektumgebung.

Bezeichner	Arbeitspaket
S1-AP01	Erfassung des Anwendungsgebietes
S1-AP02	Abgrenzung des Leistungsumfangs
S1-AP03	Projekteinschränkungen ermitteln
S1-AP04	Anforderungen identifizieren
S1-AP05	Anforderungen spezifiziern u. validieren
S1-AP06	Anforderungen formulieren
S1-AP07	Vorgehensmodell Projektmgmt. festlegen
S1-AP08	Zeitplan erstellen
S1-AP09	Arbeitspakete festlegen und grob umschreiben
S1-AP10	Sprints vorplanen und grob beschreiben

Zu Beginn des Projektes lag uns nur eine vage Vision des Zielsystems vor. Bevor mit der Ermittlung der Anforderungen begonnen werden konnte, musste zunächst ein gemeinsames Verständnis über das Zielsystem zwischen dem Auftraggeber und der Projektgruppe erarbeitet werden. Dazu wurde im ersten Treffen gemeinsam das Anwendungsgebiet erfasst. Darauf aufbauend erfolgte die Abgrenzung des Leistungsumfangs und der Ermittlung der Projekteinschränkungen, welche für die Anforderungsanalyse wichtige Informationen darstellen. Nachdem sich die erste Vision des Zielsystems in unseren Köpfen gebildet hatte, wurden in einem weiteren Treffen die Benutzer- und Systemanforderungen gemeinsam identifiziert. Anschließend erfolgte die intensive Validierung und Spezifizierung der Anforderungen, um diese detailliert zu bestimmen. Abschließend wurden die Anforderungen widerspruchsfrei und überprüfbar formuliert. Weitere wichtige Aufgabe war die Erarbeitung des Vorgehensmodells für die Projektgruppe. Aufgrund der geringen Projektgröße und der vagen Vision des Zielsystems war das agile Vorgehensmodell Scrum am geeignetsten. Dieses wurde noch um wichtige Aspekte klassischer Vorgehensmodelle erweitert und an die Bedürfnisse der Projektgruppe angepasst. Weitere Aufgabe der Projekteinrichtung war die Erstellung eines Zeitplans, in dem die wesentlichen Projektphasen abgebildet und auf die Sprints verteilt werden. Dieser ermöglicht die fortlaufende Überprüfung des Projektfortschritts und dient als Grundlage für die Planung und Fertigstellung von Teilergebnissen Abbildung 4.3. Nachdem der Zeitplan erstellt war, wurden die Arbeitspakete aus den Anforderungen an das Zielsystem ermittelt und im Product-Backlog fixiert. Auf der Grundlage des Zeitplans und der Arbeitspakete wurden der kommende Sprint 2 detailliert und die übrigen Sprints grob vorgeplant.

Sprint 2 Lastenheft / Grobkonzept – Juni 2015 Im zweiten Sprint bestanden die Hauptaufgaben aus der Erstellung des verbindlichen Lastenheftes sowie des Grobkonzeptes des Zielsystems und der Einrichtung der Projektumgebung. Aus Übersichtlichkeitsgründen ist das Lastenheft in das Grobkonzept integriert worden.

Bezeichner	Arbeitspaket
S2-AP01	Aufgabenstellung beschreiben

S2-AP02	Systemumgebung und Akteure beschreiben
S2-AP03	Funktionale Anforderungen formulieren
S2-AP04	Nicht-Funktionale Anforderungen formulieren
S2-AP05	Technologieauswahl beschreiben
S2-AP06	Systemdaten beschreiben
S2-AP07	VM-Umgebung einrichten
S2-AP08	Confluence einrichten u. Wiki aufbauen
S2-AP09	JIRA einrichten und Projekt anlegen
S2-AP10	Sprints und Backlog vorplanen
S2-AP11	Lastenheft / Grobkonzept erstellen
S2-AP12	Abnahme Lastenheft / Anforderungen
S2-AP13	Lastenheft in LaTex überführen
S2-AP14	Komponentenauswahl
S2-AP15	Datenmodell / weitere Rahmenbedingungen

Als erstes wurde die Aufgabenstellung des Projektes ausführlich beschrieben, um den Lesern des Grobkonzeptes einen ersten Überblick über das zu entwickelnde Zielsystem zu geben. Danach erfolgte die Beschreibung der Systemumgebung sowie deren Akteure. Dabei wurde das System nach außen abgegrenzt und die Akteure des Systems näher betrachtet. Abgeschlossen wurde das Aufgabenpaket mit der Beschreibung der Interaktion der Akteure mit dem System mittels Use-Cases. Als nächstes wurden die vorformulierten Anforderungen in funktionale und nicht-funktionale Anforderungen unterteilt und widerspruchsfrei, sowie überprüfbar formuliert. Das nächste Arbeitspaket beschäftigte sich mit der Technologieauswahl. Dabei wurden auf der Grundlage der bisherigen Erkenntnisse, die für das Zielsystem erforderlichen Komponenten und Technologien ausgewählt und beschrieben. Die Beschreibung der Systemdaten gab einen ersten Überblick über die Datenstruktur des Systems. Dabei wurde neben einem Entwurf des Datenflusses ein erster Überblick über die internen Datenformate gegeben. In dem Arbeitspaket Komponentenauswahl wurden die einzelnen Komponenten des Zielsystems identifiziert, gruppiert und die Interaktion mittels Schnittstellen dargestellt. Das Arbeitspaket „weitere Rahmenbedingungen“ beschäftigte sich mit der Skalierbarkeit des Zielsystems und verbreitete Protokolle der Datenquellen. Der Aufgabenbereich Grobkonzept / Lastenheft ist mit der Abnahme der Anforderungen und des Grobkonzeptes abgeschlossen worden (siehe Anhang Lastenheft).
Im Rahmen der Projektumgebung wurden zunächst Confluence und JIRA eingerichtet. Dazu wurde in Confluence der Arbeitsbereich der Projektgruppe (PGSESADATA) festgelegt, die Benutzer der Projektgruppe und die ersten Bereiche des Wikis angelegt. In JIRA wurden das Projekt namens PGSESADATA, die Benutzer der Projektgruppe und das Board zur Verwaltung des Projektes angelegt. Anschließend wurde das Product Backlog gefüllt, die ersten Sprints angelegt, sowie die Aufgaben und Arbeitspakete geplant.

Sprint 3 Datenflussdurchstich – Juli 2015 Im Fokus des dritten Sprints stand der erste Datenflussdurchstich, welcher die erste Datenübertragung in die Kernkomponente Odysseus ermöglichen sollte. Primär erfolge die erste Auseinandersetzung mit den Grundkomponenten Odysseus, den Sensoren und den Datenbanken.

Bezeichner	**Arbeitspaket**
S3-AP01	Odysseus aufsetzen
S3-AP02	Datengenerator einbinden

S3-AP03	erste Daten in Odysseus verarbeiten
S3-AP04	Jenkins aufsetzen
S3-AP05	In-Memory DB ermitteln
S3-AP06	Langzeit- DB ermitteln
S3-AP07	Untersuchung Datenbankarten
S3-AP08	Evaluierung Hardware
S3-AP09	Hardwareauswahl und Auflistung
S3-AP10	letzte Änderungen am Lastenheft vornehmen
S3-AP11	Hardware bestellen
S3-AP12	Odysseusinstallation dokumentieren
S3-AP13	Datengenerator dokumentieren
S3-AP14	Jenkins dokumentieren
S3-AP15	Installationsanleitungen der evaluierten DBen
S3-AP16	LAOTSE OSGi Infrastruktur

Zu Beginn erfolgte die Einarbeitung in Odysseus sowie das Aufsetzten des Odysseus Servers und Clients, um den ersten Datenstrom verarbeiten zu können. Dazu wurde ein Datengenerator eingebunden, der mit unterschiedlicher Frequenz Testdaten erzeugte, welche von Odysseus verarbeitet wurden. Nachdem der Datenstrom in Odysseus verarbeitet werden konnte, ist durch die Erhöhung der Datenerzeugungsfrequenz die Leistungsfähigkeit von Odysseus überprüft worden. Als nächstes erfolgte die Einarbeitung in die Sensoren und der erste Sensor wurde eingebunden. Dazu wurde ein vorhandener Temperatursensor an einen Raspberry PIangeschlossen und die erzeugten Messwerte mit Hilfe des Datengenerators an Odysseus übertragen. Die Abtastfrequenz betrug dabei 1 Hz und war von dem Temperatursensor vorgegeben. Des Weiteren wurde die benötigte Hardware gemäß Anforderungen evaluiert, zur Auswahl aufgelistet und gemeinsam ausgewählt (siehe Anhang Hardwareauswahl). Darüber hinaus erfolgte die Einarbeitung in die Datenbanken, um die Datenströme, nach der Verarbeitung mittels Odysseus, speichern zu können. Dazu wurden zunächst die verschiedenen Datenbankarten anhand ihres Speicherprinzips auf Eignung untersucht. Auf der Grundlage dieser Erkenntnisse ist nach geeigneten In-Memory und Langzeitdatenbanken recherchiert worden. Ein weiteres Arbeitspaket beschäftigte sich mit der Installation der zu vergleichenden Datenbanken. Als weiteres Arbeitspaket wurde mit der Einrichtung der Programmierumgebung begonnen und erste Überlegungen zur Open Services Gateway initiative (OSGi)-Infrastruktur getroffen. Außerdem wurden in diesem Sprint Jenkins aufgesetzt und LAOTSE OSGi-Infrastruktur erarbeitet. Abschließend wurden letzte Änderungen am Lastenheft vorgenommen.

Sprint 4 Datenbankauswahl – August 2015 Der vierte Sprint beschäftigte sich primär mit der Auswahl geeigneter Datenbanken für die Realisierung des Zielsystems und der Vergleichbarkeit der unterschiedlichen Alternativen. Darüber hinaus wurde mit der Einarbeitung der Sensoranbindung begonnen und der Ausbau der Programmierumgebung fortgeführt.

Bezeichner	**Arbeitspaket**
S4-AP01	Datenbanken installieren
S4-AP02	Datenbankinstallationsanleitungen erstellen
S4-AP03	DB - Testprogramm implementieren
S4-AP04	Datenbanken gemäß Testkonzept testen
S4-AP05	Datenbankauswahl und Auflistung

S4-AP06	in-Memory DB anlegen
S4-AP07	in-Memory DB Wrapper erstellen
S4-AP08	Temperatursensor einbinden
S4-AP09	Einarbeitung AD-Wandler
S4-AP10	Einarbeitung Mikrofon einbinden
S4-AP11	Einarbeitung in Protokolle (XMPP/OPC UA)
S4-AP12	LAOTSE auf Maven konvertieren
S4-AP13	Zwischenbericht

Um für das Zielsystem die geeignetste Datenbank auswählen zu können, wurde das im vorherigen Sprint erarbeitete Testkonzept weiter ausgeführt und mit der Erstellung der dafür erforderlichen Skripte begonnen. Ein anderes Arbeitspaket umfasste die Installation der zu vergleichenden Datenbanksysteme auf den VM-Instanzen und der anschließenden Dokumentation. Nach dem die Datenbanksysteme installiert waren, wurden die Tabellen gemäß Testkonzept erstellt und mit der Erstellung der DB-Wrapper begonnen. Nach der erfolgreichen Erstellung der DB-Schnittstellen, konnten für die ersten Datenbanken die Operationen (Insert, Average, Max, Min und Range) erstellt und getestet werden. Dabei wurden zunächst die In-Memory Datenbanken getestet, um auf der Grundlage später die geeignetste Langzeitdatenbank ermitteln zu können. In einem weiteren Arbeitspaket wurde die Auswahl der Datenbanken erläutert und die potentiellen Kandidaten aufgelistet. Als weiteres und grundlegendes Aufgabenfeld ist mit der Sensoranbindung in diesem Sprint begonnen worden. Aufgrund der urlaubsbedingten Verzögerung der Bearbeitung der Hardwarebestellung begann zunächst in diesem Sprint die Einarbeitung in die Sensoranbindung, um nach der Bereitstellung der Hardware die Sensoren schnellstmöglich in Betrieb nehmen zu können. Damit die Sensoren später auch an die Datenbanksysteme angebunden werden können, ist mit der Einarbeitung in die Kommunikationsprotokolle begonnen worden. Dabei wurden zunächst die beiden weit verbreiteten und als Standard geltenden Protokolle XMPP und OPCUA betrachtet. Abschließend wurde das Build-Management-Tool Maven eingerichtet, um die Programmerstellung des Zielsystems standardisieren zu können.

Sprint 5 Datenbankeinrichtung – September 2015 Der Fokus des fünften Sprints lag auf der Einrichtung des ausgewählten Datenbanksystems. Weitere Ziele waren die Erstellung der Grundstruktur des Zwischenberichtes sowie ein erstes Mockup der LAOTSE-GUI.

Bezeichner	**Arbeitspaket**
S5-AP01	In-Memory DB Wrapper erstellen
S5-AP02	Datenbanken gemäß Testkonzept testen
S5-AP03	Zieldatenbank auswählen
S5-AP04	Bulkstorer erstellen
S5-AP05	Langzeit- DB testen / auswählen
S5-AP06	Langzeit- DB anlegen
S5-AP07	Zwischenbericht
S5-AP08	Mockup GUI

Aufgrund der Urlaubssituation ist der fünfte Sprint weniger umfangreich als die übrigen ausgefallen. Zunächst wurde die letzte zu testende In-Memory Datenbank Druid angelegt und der Datenbank-Wrapper erstellt. Nachdem die Schnittstelle funktional war, konnte im nächsten Ar-

beitspaket mit dem Datenbanktest begonnen werden. Anschließend sind alle Ergebnisse zusammengetragen und den Anforderungen des Zielsystems gegenübergestellt worden.
Die Auswahl des Ziel-DBMS wurde in einem Auswahldokument festgehalten und zur Abnahme an den Auftraggeber übersandt. Nach deren Zustimmung wurde Druid als Zielsystem bestimmt. Die Auswahl ist im Wesentlichen durch die überragende Insertgeschwindigkeit und der integrierten (zugehörigen) Langzeitdatenbank begründet. Dadurch können die Daten direkt aus der In-Memory-DB in die Langzeitdatenbank übertragen werden, ohne dafür einen separaten Bulkstorer zu verwenden. Im nächsten Schritt wurde die Langzeitdatenbank angelegt und mit der Einarbeitung in die Langzeitspeicherung begonnen. Ein weiteres Arbeitspaket beschäftigte sich mit der Erstellung der Grundstruktur des Zwischenberichtes und dem Einfügen der ersten Inhalte. Mit der Erstellung des Mockups der LAOTSE GUI endete der Sprint. Dazu wurde zunächst ein Klick-Prototyp erstellt und dem Auftraggeber vorgestellt, um ein ersten visuellen Eindruck des Zielsystems zu bekommen.

Sprint 6 Datenbankoptimierung und Sensoreinbindung – Oktober 2015 Ziel des sechsten Sprints war es einen ersten funktionierenden Prototypen des Zielsystems bereitzustellen. Darüber hinaus sollten weitere Sensoren eingebunden, das Datenbanksystem optimiert und der Zwischenbericht erstellt werden.

Bezeichner	**Arbeitspaket**
S6-AP01	Langzeit- DB Wrapper erstellen
S6-AP02	In-Memory DB Wrapper optimieren
S6-AP03	DB optimieren
S6-AP04	JavaFX GUI erstellen
S6-AP05	Analysen erstellen
S6-AP06	Konzept für Protokolle
S6-AP07	Konzept für Sensormetadaten
S6-AP08	Prototyp erstellen
S6-AP09	Zwischenbericht erstellen

Damit die eingehenden Daten der Sensoren persistent gespeichert werden können, wurde im ersten Arbeitspaket der Wrapper der Langzeitdatenbank erstellt. Dabei ist zunächst die Übertragung ohne Fragmentierung (Bildung von Samples), auf einen Knoten und ohne Allokation (Verteilung) erfolgt. Nachdem die Daten von Kafka in Druid übertragen werden konnten, erfolgte die Optimierung des Datenbanksystems in einem weiteren Arbeitspaket. Dazu wurden verschiedene Fragmentierungen (Samplegrößen), Allokationen (Verteilungsfunktionen), Metriken zur Speicherung und Knotenmengen untersucht. Die Untersuchungen zeigten, dass ein Performanzanstieg durch die Optimierung der Fragmentierung und Verteilung zu erzielen ist.
Leider war die Messung der Performanzvorteile durch die geringere Anzahl der Knoten, der eingeschränkten VM-Umgebung und der noch geringen Anzahl der eingehenden Datenmengen nicht eindeutig. Da die bisherige erzielte Geschwindigkeit von ca. 179.000 Datensätzen pro Sekunde den Anforderungen genügt, ist die weitere Optimierung des Datenbanksystems weiter nach hinten gestellt worden, wenn die neuen Server zur Langzeitspeicherung vorhanden sind und mehr Datenquellen für eine realistische Testumgebung bereitstehen.
Die Optimierung des Datenbanksystems erforderte Anpassungen des In-Memory DB Wrappers, welche in einem weiteren Arbeitspaket vorgenommen wurden. Im Wesentlichen wurde dabei die Samplegröße optimiert. Ein weiteres wichtiges Arbeitspaket für die Erstellung des Prototypen,

beschäftigte sich mit der Umsetzung der LAOTSE-GUI. Diese wurde mittels JavaFX erstellt und berücksichtigt das Aussehen und die Funktionen des im vorherigen Sprint erstellten Mockups. Dabei können bisher die einzelnen Sensoren ausgewählt, Metainformationen angezeigt und deren genauen Standort, direkt in einer Karte von Google Maps anhand des Längen- und Breitengrades, angezeigt werden.
Darüber hinaus werden die bisher verfügbaren Analysen angezeigt und sind für einen ersten Sensor, dem Temperatursensor, umgesetzt. Das Arbeitspaket „Analysen erstellen“ diente in erster Linie der Erstellung der Grundstruktur zur Realisierung und Implementierung der späteren Analysen. Die für die Analysen erforderlichen DB-Abfragen werden in der zentralen LAOTSEDB gespeichert und visuell in der LAOTSE GUI integriert. Die Ergebnisse dieser Analysen können bei Bedarf wiederum in der In-Memory und Langzeitdatenbank gespeichert werden, um für weitere Analysen und Visualisierungen verwendet werden zu können. Die Modifizierung der Analysen, z.B. die Auswahl verschiedener Sensoren wird ermöglicht, um Redundanzen und Unübersichtlichkeit zu vermeiden und eine komfortable Verwendung zu gewährleisten.
In dem Arbeitspaket „Konzept für Datenübertragungsprotokolle“ wurden die Anforderungen an die Datenübertragung identifiziert und die Protokolle OPCUA, XMPP und TCP/IP näher betrachtet. Das Konzept der Sensormetadaten beschäftigte sich mit der Auswahl der relevanten Metadaten der Sensoren und der Erstellung einer Datenstruktur. Dazu wurde der Standard IEC 61815 näher betrachtet um wichtige Datenfelder für das Zielsystem zu berücksichtigen. Die Metadaten werden dabei in der LAOTSEDB für jeden Sensortyp abgespeichert und über einen eindeutigen Schlüssel referenziert. Die vorausgewählten Metadaten können später beliebig erweitert werden.
Das Arbeitspaket „Prototyp erstellen“ beschäftigte sich mit der Zusammenführung der bisher erstellten Teilkomponenten zu einem ersten lauffähigen Gesamtsystem. Dabei sollen zunächst die Messwerte, des an einem Raspberry PI angeschlossenen Temperatursensors, verarbeitet, persistent gespeichert und über die LAOTSE GUI analysiert sowie dargestellt werden. Dazu wird der Datenstrom des Temperatursensors (mit einer Übertragungsfrequenz von 1 Hz) mit Hilfe des Datengenerators von dem Raspberry PI an Odysseus übertragen. Der Datengenerator ruft (pullt) dabei die Daten vom Temperatursensor ab, fügt den Zeitstempel hinzu und überträgt die Daten mittels TCP/IP an Odysseus. Dort wird der Datenstrom verarbeitet, zu einem Sample gesammelt und an Kafka (In-Memory Datenbank) übermittelt. Kafka sammelt die eingehenden Daten ebenfalls und sendet diese anschließend an die Langzeitdatenbank Druid. Die LAOTSE GUI ermöglicht die Analyse und Darstellung der Messwerte. Die Erstellung des Zwischenberichtes schließt den sechsten Sprint ab.

Sprint 7 LAOTSE-Serveroptimierung und Sensoreinbindung – November 2015 Der siebte Sprint diente der Erstellung eines Konzeptes für das Hinzufügen weiterer Quellen, unter Berücksichtigung des Odysseus Controllers und der Instanzen der Konfigurationen. Darüber hinaus sollten der AD-Wandler eingebunden, die Zwischenpräsentation erstellt sowie gehalten und die GUI weiter entwickelt werden.

Bezeichner	Arbeitspaket
S7-AP01	Odysseus Controller
S7-AP02	Einbindung AD-Wandler
S7-AP03	GUI für Zwischenpräsentation anpassen
S7-AP04	Konzept für Protokolle
S7-AP05	Zwischenpräsentation
S7-AP06	Konzept für Instanzen der Konfiguration

S7-AP07	Konfiguration fertig stellen

Damit zukünftig die weiteren Datenquellen über den LAOTSE-Client hinzugefügt werden können, begann im ersten Arbeitspaket die Erstellung des Nachrichtenaustauschs zum Odysseus Controller.
Im zweite Arbeitspaket ist mit dem AD-Wandler (Spannungsmessmodul) der letzte Sensor in das Zielsystem eingebunden worden.
Ein weiteres Arbeitspaket beschäftigte sich mit der Implementierung der Änderungen an der GUI für die Zwischenpräsentation.
Das Arbeitspaket „Konzept für Protokolle“ diente einer Strukturierung der Vielzahl eingesetzter Übertragungsprotokolle der unterschiedlichen Datenquellen und der Auswahl eines Übermittlungsprotokolls für alle Datenquellen. Dazu wurde das für die Datengeneratoren bereits eingesetzte TCP-IP-Protokoll geprüft.
Das Arbeitspaket Zwischenpräsentation bestand neben der Erstellung der Zwischenpräsentation, aus der Vorstellung am 23.11.2015.
Das sechste Arbeitspaket beschäftigte sich mit der Erstellung des DB-Schemas der LAOTSE Konfiguration.
Die Fertigstellung der Konfiguration beinhaltet die Implementierung des DB-Schemas der LAOTSE Konfiguration.

Sprint 8 LAOTSE-Serveroptimierung und Sensoreinbindung – Dezember 2015 / Januar 2016 Aufgrund der Weihnachtsferien ist der achte Sprint bis zum 13.01.2016 verlängert worden. Im Fokus dieses Sprints stand die Anbindung der Datenquellen über das TCP-IP Übertragungsprotokoll. Des Weiteren sollten OPC UA und die LAOTSE-Datenbank eingebunden und mit der Anbindung von mosaik begonnen werden.

Bezeichner	Arbeitspaket
S8-AP01	OPC UA
S8-AP02	GUI erweitern
S8-AP03	Laotse DB anbinden
S8-AP04	Mosaik anbinden
S8-AP05	Übertragungsprotokoll
S8-AP06	Odysseus Controller

Damit die Anbindung von Datenquellen über OPC UA erfolgen kann, ist im ersten Arbeitspaket mit der Anbindung des bereitgestellten OPC UA Testservers begonnen worden. Aufgrund der DNS-Probleme im Netzwerk des SESA-Labs, konnte der in Odysseus bereitgestellte Operator für den Namenszugriff nicht verwendet werden. Um das zu umgehen, ist die Veränderung des Operators geprüft worden.
Die Erweiterungen der GUI umfassten neben der Visualisierung der Analysen mittels statischen Dashboards, die Darstellung der Aufzeichnungszeiträume der Daten (Tage, Wochen) über eine Verfügbarkeitsanzeige.
Im dritten Arbeitspaket ist die systeminterne Datenbank angebunden worden, welche die Konfiguration enthält (siehe Abschnitt 7.2.2).
Damit die über mosaik bereitgestellten Simulationsdaten des SESA-Labs verarbeitet werden können, ist zunächst die Anbindungsmethode ermittelt und das Konzept des mosaik- Starters erstellt worden. Darüber hinaus begann die Erarbeitung der mosaik-Initialisierung.

Für die Übertragung der Datenströme ist das TCP-IP Übertragungsprotokoll des Datengenerators von Odysseus angepasst und implementiert worden (siehe Abschnitt 7.7.3 Datengeneratoren).
Das letzte Arbeitspaket beschäftigte sich mit der Umsetzung der Odysseus-Query für die Anbindung des OPC UA Testservers.

Sprint 9 LAOTSE-Serveroptimierung und Sensoreinbindung – Januar 2016 / Februar 2016 Der neunte Sprint diente im Wesentlichen der Anbindung von mosaik, OPC UA und des Verteilungstests mit der bereitgestellten Serverhardware. Des Weiteren wurde die Sensorhardware im SESA-Lab eingebaut.

Bezeichner	Arbeitspaket
S9-AP01	Verteilungstest
S9-AP02	OPC UA
S9-AP03	GUI erweitern
S9-AP04	mosaik-Starter
S9-AP05	Usabiltiy Test
S9-AP06	Nachrichtendienst Laotse Server
S9-AP07	Gehäusebau und Sensoreinbau
S9-AP08	Odysseus Controller

Um das Zielsystem auf den Ende Februar bereitgestellten Servern ausgiebig testen zu können, ist zunächst das Testvorgehen zu planen. Dazu wird das Hardwarekonzept erstellt, welches neben einer Ressourcenplanung die Verteilung der einzelnen Komponenten vornimmt. Auf dieser Grundlage erfolgt die Identifizierung der Leistungsgrenzen und Engpässe des Zielsystems.
Da der Odysseus Namensoperator nicht verändert werden konnte, ist der OPC UA Operator mit dem Zugriff über eine Serverzertifikat so modifiziert worden, dass die Authentifizierung entfällt.
Die Erweiterung der GUI umfasste die Implementierung der mosaik- Viewfunktion, welche die Initialisierung und den Simulationsfortschritt visualisiert.
Der mosaik-Starter wird benötigt, um die Simulation für die Initialisierung und Datenübertragung zu starten. Dieser wurde mittels eines Nachrichtendienstes im mosaik-Controller abgebildet.
Das Arbeitspaket „Usability Test“ umfasste neben der Festlegung der Testmethoden und Testvorgehen, die Erstellung eines Testleitfadens.
Die Erweiterung des Nachrichtendienstes des LAOTSE Servers umfasst die Implementierung des mosaik-Starters und der mosaik- Viewfunktion.
Im Rahmen des Gehäuse- und Sensoreinbaus sind zunächst die Sensoren in Hutschienengehäuse integriert und anschließend in das SESA-Lab eingebaut worden.
Das abschließende Arbeitspaket „Odysseus Controller“ beschäftigte sich mit der Implementierung der mosaik- Initialisierung.

Sprint 10 Implementierungsabschluss – Februar 2016 / März 2016 Der letzte Implementierungssprint wurde für die abschließende Überprüfung der Anforderungen und Funktionalitäten sowie Umsetzung ggf. resultierender Änderungen genutzt.

Bezeichner	Arbeitspaket
S10-AP01	Verteilungstest
S10-AP02	Usability Test
S10-AP03	Abschlusstest

S10-AP04	Fehlerbehebung
S10-AP05	Code-Review
S10-AP06	Java-Doc

Mit dem Verteilungstest wurden die Engpässe und Kapazitätsgrenzen des Zielsystems untersucht und für zukünftige Skalierungen analysiert.
Um die Gebrauchs- und Funktionstauglichkeit der LAOTSE-GUI zu überprüfen, erfolgte die Durchführung und Auswertung des zuvor erstellten Usability Tests mit vier Testprobanden.
Zur Überprüfung der implementierten Funktionen und vereinbarten Anforderungen ist ein Abschlusstestkonzept erarbeitet worden, welches zum Schluss überprüft wird.
Während der Fehlerbehebung sollen die letzten Probleme und erforderlichen Änderungen behoben werden.
Nach dem Code Freeze ist abschließend der Quellcode auf Fehler überprüft und korrigiert worden.
Das letzte Arbeitspaket diente der Überarbeitung des Java-Docs, welcher den Programmcode ausreichend dokumentiert.

Sprint 11 Projektabschluss und Dokumentation – März 2016 Im letzten Sprint wurde das Projekt abgeschlossen und der Abschlussbericht erstellt.

Bezeichner	**Arbeitspaket**
S11-AP01	Optimierungen
S11-AP02	Abschlusstest
S11-AP03	Abschlussbericht

Im Rahmen der letzten Optimierungen wurden die Visualisierung der hochfrequenten Audiodaten mittels Pagination und die Absturzsicherheit der mosaik- Simulation umgesetzt.
Durch den Abschlusstest sind die Funktionalitäten und Anforderungen überprüft worden. Dieser bildet die Grundlage für die Projektabnahme durch den Auftraggeber.
Zum Projektende ist der Abschlussbericht des Projektes mit den zugehörigen Handbüchern erarbeitet worden.

5. Technologien

Dieses Kapitel beschreibt die zur Realisierung des Projektes LAOTSE verwendeten Technologien.

5.1. Werkzeuge

In diesem Abschnitt werden die während der Erstellung unterstützend eingesetzt wurden.

5.1.1. Scene Builder

MM

Der Scene Builder [Lit] ist ein Window Builder, der für die Entwicklung von FXML-Dateien für JavaFX-Applikationen dient.
Mit einem Window Builder kann mit Hilfe einer grafischen Nutzeroberfläche eine GUI erstellt werden. Dabei sieht der Nutzer des Window Builders eine grafischische Vorschau der späteren GUI. Im Scene Builder werden die FXML-Dateien erzeugt und der Entwickler braucht keinen Extensible Markup Language (XML)-Code selbst zu schreiben. Dabei können die einzelnen Elemente der GUI ausgewählt und beliebig angeordnet werden, bis das gewünschte Layout fertiggestellt ist. Auch können zahlreiche Optionen der einzelnen Elemente konfiguriert werden, die das Erscheinungsbild beeinflussen.
Die Besonderheit bei dem Scene Builder sind die Eigenschaften, die er bietet um verschiedene Funktionen von JavaFX bereits innerhalb des Scene Builders zu nutzen. So können z.B. alle GUI-Elemente mit IDs versehen werden, die später in dem FXML Controller als Java-Variable zur Verfügung stehen. Es können auch Methodennamen definiert werden, die bei bestimmten Aktionen, wie z.B. dem Klick auf einen Button ausgeführt werden sollen. Eine Exportfunktion kann aus diesen Angaben eine kompilierbare Java-Klasse erstellen, die als FXML-Controller für die GUI genutzt werden kann. Der Entwickler muss dann lediglich noch die Methoden implementieren.

5.1.2. Eclipse

LS

Um ein großes Projekt vernünftig entwickeln zu können, sollte ein Java-Integrated Developement Environment (IDE) genutzt werden, die übersichtlich und dabei unterstützend bei der Programmierarbeit mithilft. Da nahezu alle Projektmitglieder hauptsächlich in der Entwicklungsumgebung eclipse programmieren, wurde eclipse als die Entwicklungsumgebung ausgewählt. Im Nachhinein wurde dann entschieden, dass das Projekt auf dem OSGi Framework Equinox basierend, aufgebaut werden soll, da das zu nutzende DSMS Odysseus ebenfalls auf Equinox basiert

und mit Equinox eine sehr modulare Erstellung eines Projekts ermöglicht wird, sodass einzelne Komponenten im späteren Verlauf problemlos ausgetauscht werden können. Da eclipse eine Equinox-Unterstützung anbietet und NetBeans nicht, konnte die Wahl der Entwicklungsumgebung im Nachhinein gesehen nur auf eclipse fallen.

5.1.3. Maven

LS

Maven ist ein *Build-Tool* und soll als solches dafür sorgen, dass ein *Build* durchgeführt, also der Code compiliert und gelinkt (bzw. gepackaged) wird.
Vor allem aber muss dieser Vorgang organisiert werden: Für eine einzelne z.B. Java-Datei lässt sich noch gut manuell an der Konsole mit wenigen Befehlen ein ausführbarer Programmcode erzeugen; bei größeren Projekten wird dieser Prozess dann nicht nur zeitaufwändig und repetitiv sondern auch kaum noch vernünftig koordinierbar:
Es müssen bestimmte Softwareteile vor anderen gebaut und Bibliotheken geladen, sowie diverse Build-Dateien angepasst werden.
Das alles sind Dinge über die man sich als Programmierer eigentlich weniger Gedanken machen möchte. Zum Glück handelt es sich dabei um Arbeitsschritte, die sich leicht automatisieren lassen. Genau hier kommen die Build-Tools ins Spiel. Diese übernehmen die Organisation des Buildprozesses bzw. das Verfassen von Makefiles im Hintergrund.
Zusätzlich sollen moderne Build-Tools auch das automatische Bauen und die Ausführung zum Code gehöriger Unit-Tests übernehmen.

Tycho

Passende Build-Tools lassen sich in alle gängigen IDEs integrieren oder sind bereits von Haus aus eingebaut.
Im Fall dieses Projektes stellte gerade die Festlegung mancher IDEs auf bestimmte Build-Tools ein Problem dar.
So musste, um OSGi-Projekte bzw. Eclipse-Features und Eclipse-Products mit Maven bauen zu können das dazu fähige Tycho-Plugin in die Maven-Pom geladen werden.
Tycho bietet verschiedene Packaging-Möglichkeiten: Eclipse-plugin (=OSGi-Module), Eclipse-Feature, Eclipse-Product. In einem Product von Eclipse können mehrere Projekte zusammengefasst und angepasst an die jeweilige Zielumgebung ausgeliefert und gestartet werden. Dazu müssen sie vorher noch in Features zusammengefasst werden, um sie in das Produkt einfügen zu können.

5.1.4. Visual Paradigm

KB

Visual Paradigm (VP) ist eine Unified Modelling Language (UML)-Software. Das bedeutet, dass VP es ermöglicht komplexe UML-Diagramme zu modellieren und somit komplexe Strukturen zu dokumentieren. Dabei ist VP sehr umfangreich und beinhaltet nahezu alle UML-Diagrammarten, die für eine gute Dokumentation benötigt werden. Visual Paradigm ist also eine Software, die

es ermöglicht UML-Diagramme zu erstellen und diese später als Bild zu exportieren. Im Gegensatz zum Zeichnen eines UML-Diagramms per Hand besteht in diesem Fall der Vorteil, dass Diagramme im Nachhinein einfach verändert werden können.
Ebenso sind alle Komponenten, die zu einem Diagramm gehören, beim Erstellen des Diagramms verfügbar, sodass bei der Erstellung alle wichtigen Komponenten für das Diagramm berücksichtigt werden können und somit ein konsistentes UML-Diagramm entsteht.
Visual Paradigm zählt neben dem Rational Software Architect zu den bekanntesten Tools für UML-Unterstützung. Da der RSA ein sehr umfangreiches Tool ist, das noch deutlich mehr Funktionen bietet als Visual Paradigm ist für dieses Projekt die Entscheidung auf Visual Paradigm gefallen, da es übersichtlicher ist als RSA und dennoch alle Funktionen bietet, die für das Projekt notwendig sind.

5.2. Frameworks

In diesem Abschnitt wird detaillierter auf die in diesem Projekt verwendeten Frameworks eingegangen.

5.2.1. JavaFX

MM

JavaFX ist ein Java-Framework um plattformübergreifend GUI-Applikationen zu entwickeln.
Bei der Nutzung von JavaFX ergeben sich für den Entwickler zahlreiche Vorteile gegenüber anderen Technologien, wie z.B. Swing. Diese Vorteile betreffen u.a. die Komplexität der Entwicklung, die Möglichkeiten der Gestaltung der GUI und des Designs und der Kapselung verschiedener Komponenten. Ein Hauptvorteil von JavaFX ist die klare Trennung der View vom Controller. Die View wird im XML-Dialekt FXML spezifiziert. In diesen wird die Struktur der GUI festgelegt und bestimmt, welche Java-Klassen jeweils als Controller dienen. In diesen Controllern können dann Interaktionen des Nutzers mit der GUI behandelt werden und die Elemente der GUI mit Daten gefüllt werden. So sind die Logik und die Darstellung immer voneinander getrennt. Die Anordnung der GUI-Elemente in der FXML-Datei kann von einem Entwickler mithilfe des Werkzeugs „Scene Builder“ spezifiziert werden.
Ein Designer kann CSS-Datien nutzen um die GUI zu gestalten. Dadurch können hoch personalisierte und flexible Layouts, die auch zur Laufzeit angepasst werden können, standardisiert implementiert werden.

5.2.2. OSGi / Equinox

DN

Odysseus ist eine Anwendung, die auf die **OSGi**-Service-Plattform aufsetzt. In diesem Abschnitt werden das Konzept der OSGi-Service-Plattform erläutert und für die Erweiterung von Odysseus relevante Eigenschaften betrachtet.
Die OSGi-Service-Plattform ermöglicht die Entwicklung von Anwendungen in Java, die aus mehreren Modulen bestehen. Diese Module können während des laufenden Betriebs installiert, gestartet, gestoppt und deinstalliert werden.

Module werden **Bundles** genannt. Bundles stellen fachliche oder technische Einheiten dar. Sie bestehen aus Klassen und weiteren Ressourcen. Nach außen sind zunächst nur Schnittstellen sichtbar. Die Implementierung an sich ist nicht von anderen Bundles einsehbar, außer es werden explizit Java-Packages exportiert.
Bundles bieten Services an, die von anderen Bundles genutzt werden können. Dazu registrieren sie den Service in der Service-Registry. Bundles greifen über den Namen der Service Schnittstelle auf den Service zu.
Das OSGi Framework beschreibt mehrere Schichten, auf denen eine Anwendung beschrieben werden muss.

- Die Modulschicht beschriebt den statischen Aufbau der Module. So wird in einer Manifest Datei der Name des Bundles, die Version und weitere Eigenschaften festgelegt, die zur Laufzeit der Anwendung vom OSGi-Framework benötigt werden. Hier können auch Teile eines Bundles exportiert werden. Erst jetzt können andere Bundles diese Packages importieren, ebenfalls über einen Eintrag in die Manifest Datei. In OSGi-Anwendungen gibt es Versionierung. Es können gleichzeitig unterschiedliche Versionen eines Bundles eingebunden sein. Bundles können festlegen, welche Bedingungen erfüllt sein müssen, damit sie gestartet werden können. Dazu zählt das Vorhandensein von Bundles einer bestimmten Version oder eine bestimmt Java-Laufzeitumgebung.

- Die Lifecycle-Management-Schicht beschreibt die dynamischen Eigenschaften der Bundles. Hier werden die Zustände beschrieben, in denen sich ein Bundle befinden kann. Während der Laufzeit des Programms können Bundles dynamisch in die Anwendung installiert werden. Nachdem alle Abhängigkeiten aufgelöst wurden, kann das Bundle gestartet werden. Das Bundle kann gestoppt und danach wieder deinstalliert werden. Die Aktionen, die diese Übergange auslösen, sind ebenfalls in der Lifecycle-Management-Schicht beschrieben. Zum Ändern der Zustände ist im OSGi-Framework ein Management Agent integriert.

- Die Service-Schicht hält die Dienste, die von Bundles zur Verfügung gestellt werden. Die Bundles machen ihre Services der Service-Registry bekannt. Damit können andere Bundles diese Dienste systemweit in Anspruch nehmen. Da Services jedoch ständig an- und abgemeldet werden können, ist nicht sicher, dass ein Service ständig verfügbar ist. Hier müssen Bundles Lösungen bereithalten.

- Zusätzlich gibt es eine Security-Schicht, die parallel zu den anderen Schichten arbeitet. Hier können die Zugriffsberechtigungen auf Bundles eingeschränkt werden und weitere Kontrollen realisiert werden.

Das Anmelden von Services an der Service Registry wird zur Laufzeit durchgeführt. Werden viele Services registriert, so muss jedes Bundle zunächst initialisiert werden. Das kostet Zeit und Speicherplatz. Damit diese Probleme umgangen werden können, wurden Declarative Services eingeführt. Declarative Services werden durch ihr Komponentenbeschreibung im XML-Format beschrieben. In der Manifest-Datei des Bundles wird auf diese Beschreibung verwiesen. Wird ein Bundle geladen, so wird die Beschreibung von der Service Component-Runtime gelesen und Abhängigkeiten zu anderen Services aufgelöst. Der Service wird der Service Registry bekannt gemacht, auch wenn dieser noch nicht geladen wurde. Die Service Component-Runtime kann den Service dann nachladen, wenn dieser von einem Bundle angefordert wird [Wüt+08].

5.3. Fremdsoftware

In diesem Abschnitt wird auf die in diesem Projekt verwendete Fremdsoftware eingegangen. Dabei werden Open-Source Kompenenten zur Datenstrom- und Datenspeicherung verwendet, welche an die Projektbedürfnisse angepasst werden.

5.3.1. Odysseus

MM

Odysseus ist ein Framework für DSMS, das von der Abteilung Informationssysteme der Carl von Ossietzky Universität Oldenburg entwickelt wird. Es ist auf Flexibilität und Erweiterbarkeit ausgelegt und bietet die Möglichkeit auf viele verschiedene Bedürfnisse angepasst zu werden. Mit Odysseus kann ein Datenstrommanagementsystem realisiert werden, das von verschiedenen Anwendungen genutzt werden kann, um Anfragen auf kontinuierliche Datenströme auszuführen [Odya].
In diesem Abschnitt werden die weiteren Grundlagen zu Odysseus zuerst aus der Sicht des Anwenders betrachtet, der ein Datenstrommanagementsystem mit erstellen möchte. Anschließend folgt die Sichtweise des Entwicklers, der die Funktionalitäten von Odysseus erweitern möchte. Abschließend wird kurz die Architektur von Odysseus vorgestellt.

Anwendersicht auf Odysseus

Odysseus arbeitet nach dem Klienten-Server Prinzip. Der Odysseus-Server stellt dabei das eigentliche Datenstrommanagementsystem dar, in dem die Anfragen an die Datenströme ausgeführt werden. Als Klient steht das Odysseus-Studio zur Verfügung, das zur Administration des Odysseus-Server und Visualisierung der Daten genutzt werden kann. Der Server als DSMS, führt bei der Initialisierung standardmäßig keine Anfragen aus, kennt keine Datenquellen und gibt die Daten auch nicht aus. Durch das Odysseus-Studio können die Funktionen des Servers genutzt werden und beliebige Anfragen ausgeführt werden. So entstehen die individuellen Datenströme [Odyd]. In Abschnitt 5.3.1 werden die Funktionen des Servers und im Abschnitt 5.3.1 die des Klienten genauer beschrieben.

Entwicklersicht auf Odysseus

Das gesamte Odysseus Framework, also sowohl der Odysseus-Server als auch das Odysseus-Studio sind in Java nach dem Komponentenmodell mit Open Service Gateway initiative (OSGi) implementiert. OSGi ermöglicht es, eine Software komponentenweise aufzubauen und Abhängigkeiten zwischen den Komponenten zu benennen [Wüt+08]. Dadurch können beliebige Funktionen durch neue OSGi-Bundles implementiert werden und während der Laufzeit von Odysseus dynamisch geladen oder entfernt werden. Dieses Prinzip ist durch die IDE Eclipse bekannt, die ähnlich aufgebaut ist und in der genauso neue Features während der Laufzeit installiert werden können. OSGi-Bundles können untereinander Abhängigkeiten besitzen. So kann bspw. ein Bundle Funktionen eines anderen Bundles nutzen und daher nur dann genutzt werden, wenn das zweite Bundle ebenfalls zur Laufzeit von Odysseus installiert ist. Der Vorteil dieser Architektur ist, dass Odysseus so sehr flexibel und individuell anpassbar gehalten werden kann.

Architektur von Odysseus

In der Abbildung 5.1 ist die vereinfachte Architektur von Odysseus zu sehen. Dabei bilden die unterste Ebene externe Sensoren, die Daten produzieren. Der Data-Stream-Management-Level ist ein beispielhaftes Datenstrommanagementsystem, das vom Odysseus-Server implementiert wird. Auf der linken Seite wird dargestellt, dass die Daten der Sensoren als Datenquellen im Odysseus-Server abstrahiert werden. Diese Daten werden durch Pipes geleitet, wobei jede Pipe eine Funktion, wie z.B. eine Aggregation oder Selektion auf den Daten durchführt. Am Ende stehen die Daten in Sinks zur Verfügung. Sinks sind Schnittstellen des DSMS, die anderen Anwendungen ermöglichen auf die verarbeiteten Daten zuzugreifen.
Die verschiedenen Funktionen auf den Daten greifen auf Techniken, wie einen Scheduler oder Monitor (rechts) zu. Die oberste Ebene stellt eine Anwendung dar, die Anfragen an den Odysseus-Server stellt und die produzierten Daten visualisiert.
Dies kann z.B. Odysseus-Studio sein [App+12].

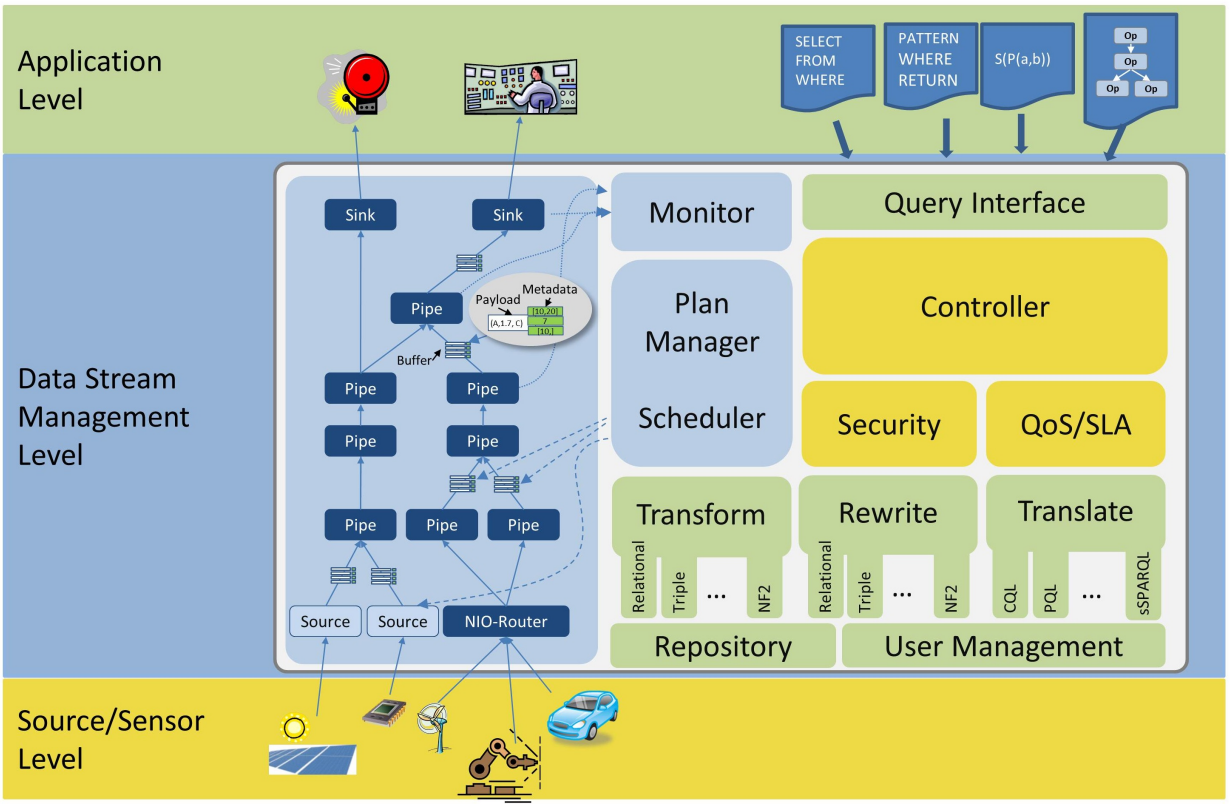


Abbildung 5.1.: Architektur von Odysseus entnommen aus [App+12]

Funktionen im Odysseus-Server

In diesem Abschnitt werden die Kernfunktionalitäten des Odysseus-Server vorgestellt. Dieser kann ein Datenstrommanagementsystem darstellen. Im Folgenden wird zuerst die Datenverarbeitung und anschließend die Anfrageverarbeitung betrachtet.

Datenverarbeitung Bei der Datenverarbeitung in einem Datenstrommanagementsystem werden die Daten, die aus heterogenen Quellen in verschiedenen, teilweise sehr hohen, Frequenzen auftreten, verarbeitet und einem Ziel zugeführt. In den folgenden Abschnitten werden zuerst die

Quellen, dann die Verarbeitung mit den Pipes und schließlich die Datensenken erläutert. Abschließend werden die datenstromtheoretischen Prinzipien der Fenster, Zeitstempel und Heartbeats und deren Verwendung in Odysseus vorgestellt.

Datenquellen Datenquellen im Odysseus-Server sind Verarbeitungsknoten, die Daten direkt aus einer externen Quelle empfangen und für die weitere Verarbeitung in Odysseus bereitstellen und entsprechend weiter geben. Sie bilden die Schnittstelle zwischen dem Odysseus-Server als Datenstrommanagementsystem und den Quellen, die in der Realität die Daten erzeugen, wie z.B. Sensoren an technischen Anlagen. Zwischen diesen können die Daten mit verschiedenen Protokollen, wie z.B. OPC UA, welches in der Automatisierung eingesetzt wird, übertragen werden. Auch können Dateien und Datenbanken eingelesen und als Datenstrom wiedergegeben werden.
Die Bereitstellung der Daten funktioniert nach dem Push-Prinzip. Es werden immer dann Daten bereitgestellt, wenn welche auflaufen und diese sofort weitergegeben [Odyd].

Pipes Als Pipe wird ein Verarbeitungsknoten im Odysseus-Server bezeichnet, der eine beliebige Operationen auf den Daten durchführt. Hier werden einige Funktionen beispielhaft vorgestellt, die in [Odyd] nachzulesen sind:

- Datentypen konvertieren
- Datumsfunktionen
- Bearbeitung von Bilddateien
- Mengenoperationen auf mehrere Messwerte auch aus verschiedenen Quellen
- Mathematische Berechnungen
- Funktionen des maschinellen Lernens auf Datenströmen

Es können beliebig viele Pipes nacheinander ausgeführt werden, so dass eine Hierarchie wie in Abbildung 5.1 entsteht [App+12].

Datensenken Nach der letzten Pipe kann eine Datensenke implementiert werden. Diese stellt eine Schnittstelle des Odysseus-Server dar und stellt die bearbeiteten Daten einem externen Programm zur Verfügung. Es kann zuvor der Datentyp beliebig verändert werden. Auch die Datensenken unterstützen zahlreiche verschiedene Protokolle. In Odysseus stehen Schnittstellen zu Datenbanken, TCP/IP-Verbindungen und Dateien zur Verfügung. Dadurch können die Daten beliebig ausgegeben und entsprechend weiter genutzt werden [Odyd].

Fenster Für einige Funktionen, wie z.B. die Ermittlung eines Maximalwertes, andere Aggregationen und Funktionen des maschinellen Lernens, sind mehrere Messwerte notwendig. Da ein Datenstrom unendlich lang ist und auch die Frequenz der Daten nicht immer gleichbleibend ist, stehen in einem Datenstrommanagemntsystem, anders als in einer Datenbank, nicht immer alle historischen Daten zur Verfügung.
Mit dem Konzept der Fenster wird eine variable Lösung geboten, auf kurzzeitig gespeicherte Werte zurückzugreifen. Indem die Daten eines Fensters genutzt werden, wird die Frage beantwortet,

welche und wie viele Daten für eine Berechnung aus dem Datenstrom herangezogen werden sollen. Bei einem Fenster werden eine bestimmte Anzahl der letzten empfangenen Datenelemente gespeichert und so in eine Berechnung mit einbezogen. In Abbildung 5.2 ist das Prinzip eines Fensters dargestellt.

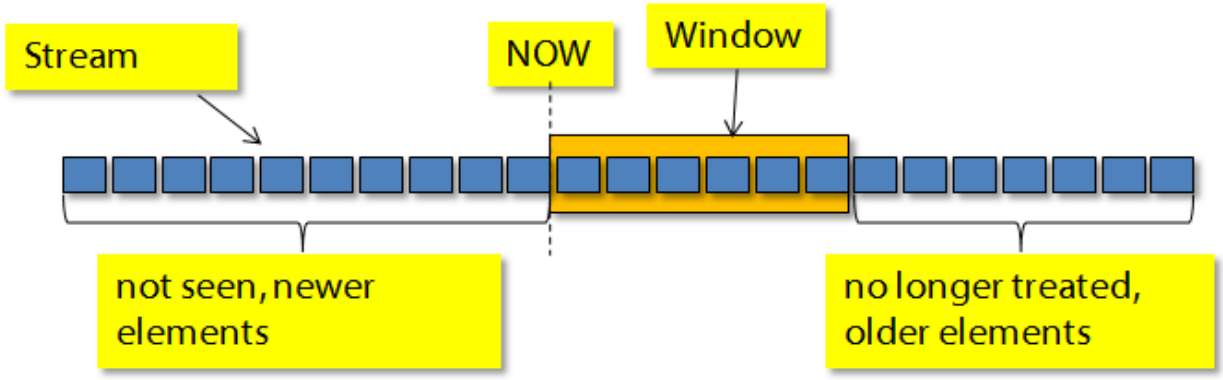


Abbildung 5.2.: Darstellung eines Fensters entnommen aus [Fen]

Um zu variieren, wie viele Elemente genutzt werden sollen, also wie groß das Fenster ist, bietet Odysseus drei Typen von Fenstern:
Bei dem **zeitbasierten** Fenster wird als Parameter eine Zeitangabe übermittelt, die das maximale Alter der Elemente im Fenster angibt. Es befinden sich nicht immer gleich viele Elemente in dem Fenster. Je nachdem wie viele Elemente in der angegebenen Zeit aufgetreten sind, können sich mehr oder weniger Elemente gleichzeitig im Fenster befinden.
Bei dem **elementbasierten** Fenster wird als Parameter eine Zahl übermittelt, die steuert, wie viele Elemente sich in dem Fenster befinden. Das Fenster umfasst immer die letzten x Elemente. Der repräsentierte Zeitraum kann also je nach Frequenz der Elemente variieren.
Bei dem **prädikatbasierten** Fenster wird als Parameter ein Prädikat übergeben. Alle Elemente, die das Prädikat erfüllen, befinden sich in dem Fenster. Das Fenster umfasst also weder immer den selben Zeitraum, noch immer die selbe Anzahl an Elementen [Fen].

Zeitstempel Ein weiteres Problem bei der Verarbeitung von Datenströmen ist, dass für einige Anwendungen sichergestellt sein muss, dass die Elemente der Datenströme in chronologischer Abfolge verarbeitet werden. Es kann vorkommen, dass mehrere Datenströme zusammengefügt werden, die in einer unterschiedlichen Frequenz arbeiten. Dafür und für die Verwendung von zeitbasierten Fenstern ist es notwendig, dass Datenelemente Datenstromübergreifend chronologisch geordnet und Identifiziert werden können. In DSMS werden dazu die Elemente der Datenströme mit Zeitstempeln angereichert, falls diese nicht bereits durch die Ausgangsdaten gegeben sind. Zeitstempel sind Metadaten, die angeben, wann z.B. ein Messwert durch einen Sensor erhoben wurde [Gra14]. Odysseus kann Zeitstempel im Odysseus-Studio anzeigen und im Server verarbeiten und manipulieren.

Heartbeats Die Frequenz, in der Elemente in einem Datenstrom gesendet werden, muss nicht immer gleichbleibend sein. Daher kann in einer Anwendung nicht eindeutig festgestellt werden, ob die Verbindung zu einem Datenstrom abgebrochen ist, oder nur über einen ungewöhnlich langen Zeitraum kein neues Datenelement eingetroffen ist. Heartbeats sind Elemente in Datenströmen, die ausschließlich Metadaten enthalten und genau wie echte Datenelemente durch alle

Instanzen der Verarbeitung geleitet werden [Gra14]. Dadurch kann jede Instanz in einem festgelegten Intervall überprüfen, ob Heartbeats eintreffen und die Verbindung zum Datenstrom noch besteht, auch wenn zur Zeit keine Datenelemente geliefert werden. Odysseus bietet das Prinzip der Heartbeats für Datenströme und im Odysseus-Studio die Möglichkeit, diese zu visualisieren.

Anfrageverarbeitung Der Odysseus-Server bietet die Möglichkeit, Anfragen von einem externen System entgegenzunehmen. Die Schnittstelle dafür implementiert das Query-Interface. Die Anfragen können z.B. von dem Odysseus-Studio stammen und sind standardmäßig in einer der drei unterstützen Sprachen: CQL, PQL und SASE, formuliert. Näheres dazu wird im nächsten Abschnitt erläutert.
Der Odysseus-Server erzeugt Anfragepläne in Baumform, indem entsprechende Pipes erstellt werden, welche die einzelnen Operationen der Anfrage ausführen. Diese Anfragepläne werden von jedem Element eines Datenstroms durchlaufen, so dass diese in der Pipe verarbeitet werden. Ein Anfrageplan kann sich auf nur einen oder auf mehrere Datenströme beziehen, indem z.B. die Daten eines Datenstroms mit Daten eines anderen Datenstroms angereichert werden. Am Ende werden die erzeugten Daten über eine Datensenke dem Programm übermittelt, welches die Anfrage gestartet und den Anfrageplan damit initiiert hat. Eine grafische Darstellung von beispielhaften Anfragepläne ist links in der Abbildung 5.1 zu sehen.
Ein Anfrageplan bleibt im Gegensatz zu einer SQL-Anfrage in einem DBMS bestehen und verarbeitet kontinuierlich weiter die Daten der entsprechenden Datenströme. Die Ergebnisse werden potentiell unendlich lange weitergegeben, bis die laufende Anfrage explizit beendet wird.
Es können zahlreiche verschiedene Anfragen quasi-parallel ablaufen. Damit deren Ausführung nicht kollidiert und um die Reihenfolge der Abarbeitungsschritte einzuteilen, gibt es in Odysseus einen Scheduler. Dieser unterteilt die Anfragen ggf. und sorgt für eine quasi-parallele Ausführung und dafür, dass die gewünschten Ergebnisse zur geforderten Zeit zur Verfügung stehen. Dazu erzeugt der Scheduler einen Ausführungsplan für die einzelnen Operatoren [App+12].

Funktionen im Odysseus-Studio

Das Odysseus-Studio ist ein Klient, der auf den Odysseus-Server zugreifen kann. Die Hauptaufgaben sind die Erstellung und Manipulation von Anfragen an das Datenstrommanagementsystem und die Visualisierung der erzeugten Daten. Diese beiden Funktionen werden in diesem Abschnitt erläutert. Odysseus-Studio bietet dazu eine grafische Benutzeroberfläche, die in Abbildung 5.3 beispielhaft zu sehen ist.
Odysseus-Studio ist, wie das gesamte Odysseus-Framework nach dem OSGi-Prinzip aufgebaut. Als grafische Benutzeroberfläche wird SWT in einer Eclipse-Application genutzt. Dadurch wird die erweiterbare und komponentenweise Architektur von Odysseus optimal unterstützt und es können beliebig viele Views geöffnet, geschlossen und neu angeordnet werden. Ein Entwickler kann auch eigene Views implementieren.

Manipulation der Anfragen Odysseus-Studio kann auf den Odysseus-Server als eine Anwendung zugreifen und Anfragen übermitteln. Dazu können die Anfragen in verschiedenen Sprachen formuliert werden [Odyd].

- **Continuous Query Language:** CQL ist eine Sprache, die auf SQL aufbaut und deklarativ aufgebaut ist. Es können direkt Datenquellen, Pipes und Datensenken im Odysseus-Server definiert werden.

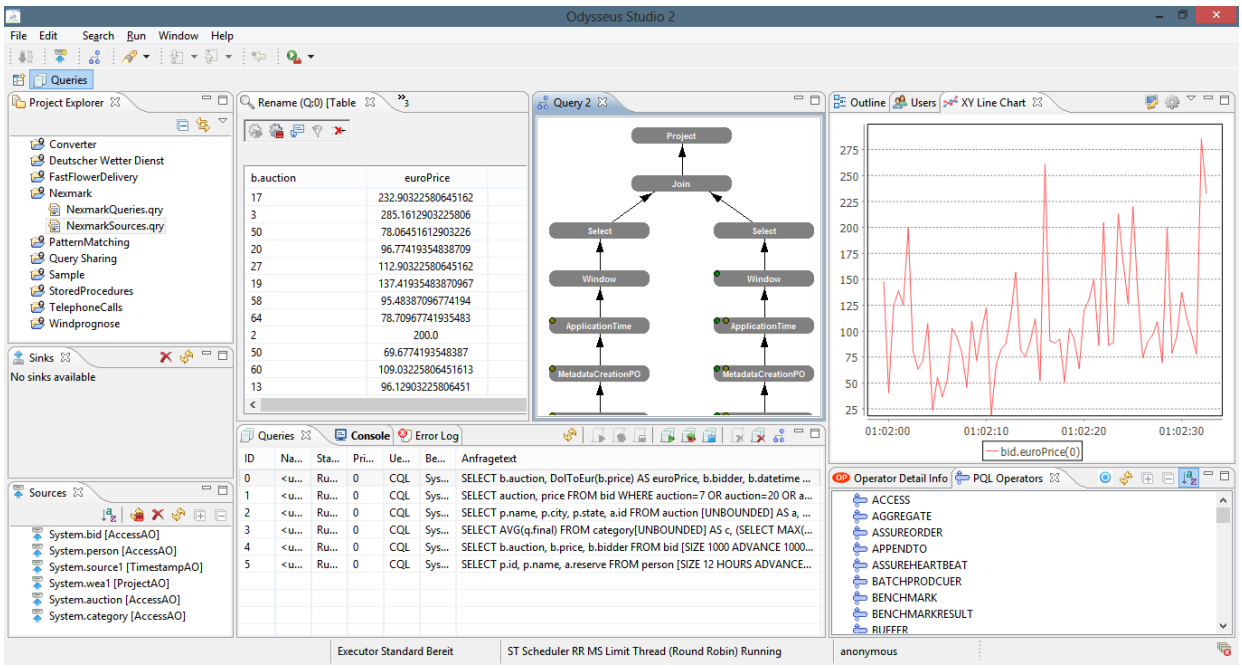


Abbildung 5.3.: Screenshot von Odysseus-Studio entnommen aus [Odya]

- **Procedural Query Language:** PQL ist eine prozedurale Sprache, die direkt Operatoren aufruft und Parameter übergeben kann.
- **SASE:** SASE ist eine zeitbasierte Sprache, die auf dem Auftreten von Events basiert.

Odysseus-Studio bietet grundsätzlich zwei Arten von Editoren um Skripte mit diesen Sprachen zu erzeugen. Zum einen gibt es einen textbasierten und zum anderen einen graphbasierten Editor. Im textbasierten Editor können die Anfragen direkt in der jeweiligen Sprache programmiert werden. Im graphbasierten Editor können die einzelnen Operationen, die zur Verfügung stehen per Drag& Drop in einem Baum angeordnet werden. Es entsteht ein sogenannter „Anfrageplan“, der die Funktionsweise komplexer Anfragen anschaulich darstellt. In Abbildung 5.4 ist ein beispielhafter simpler Anfrageplan dargestellt. Dieser wurde in [MN14] genutzt, um Daten einer Windkraftanlage für eine Visualisierung zu optimieren.

Der Compiler erzeugt daraus ein Skript in PQL. Wenn ein Skript mit Anweisungen einer dieser Sprachen ausgeführt wird, werden die entsprechenden Operationen im Odysseus-Server aufgerufen.

Überwachung der Anfragen Alle laufenden Anfragen können überwacht werden. Dazu zeigt die Queries-View alle gestarteten Anfragen mit ihrem jeweiligen Status an. Der Status kann

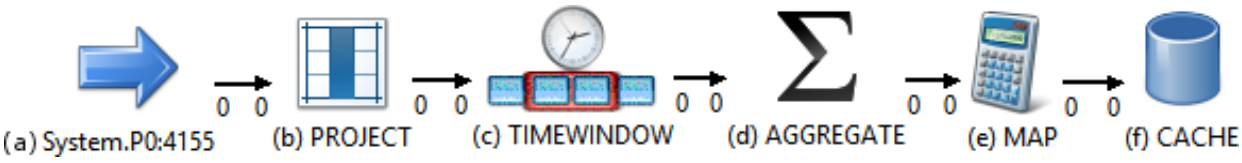


Abbildung 5.4.: Beispielhafter Anfrageplan von Odysseus entnommen aus [MN14]

laufend oder angehalten sein. In Abbildung 5.3 ist unten in der Mitte eine beispielhafte Queries-View zu sehen. Sie dient weiter noch dazu die Anfragen zu starten, zu unterbrechen oder die jeweils erzeugten Daten anzuzeigen. Im letzten Fall würde eine entsprechende weitere View geöffnet. Welche Möglichkeiten es dazu gibt wird im nächsten Abschnitt erläutert.
Neben den Anfragen gibt es eine View jeweils für die verfügbaren Datenquellen und -senken, die standardmäßig unten links angeordnet sind (vgl. Abbildung 5.3).

Visualisierung der Daten mit Dashboards Um Daten zu Visualisieren nutzt Odysseus-Studio das Konzept der Dashboards. Bei Dashboards handelt es sich um spezialisierte XML-Dateien, die grafisch dargestellt werden und aus Dashboard-Parts bestehen, die wiederum im Dashboard selbst beliebig angeordnet werden können. Als Dashboard-Part (DBP) kann jede beliebige Visualisierung implementiert werden, die benötigt wird, um Elemente aus Datenströmen darzustellen. Standardmäßig gibt es eine Reihe an variabel einsetzbaren DBPs [Odyd]:

- **Textbasierte Dashboard-Parts** werden eingesetzt um die Daten, die eine Anfrage liefert, in unveränderter Form darzustellen. Dabei werden die einzelnen Attribute horizontal und die einzelnen Werte vertikal angeordnet.
- **Tabellenbasierte Dashboad-Parts** werden eingesetzt um die Daten in einer Tabelle anzuzeigen. Dabei ähneln sie stark den textbasierten DBPs, sind aber für den Menschen einfacher lesbar. Ein Beispiel ist in Abbildung 5.3 oben links zu sehen.
- **Grafische Dashboard-Parts** bieten die größte Vielfalt und häufig kontextbezogene Darstellung der Daten. Dabei gibt es zahlreiche Darstellungsformen, wie z.B. einfache Graphen in einem Koordinatensystem und verschiedene Formen von Diagrammen. Ein Beispiel ist in Abbildung 5.3 oben rechts zu sehen.

Alle Dashboard-Parts bieten einen Controller, über den Einstellungen vorgenommen werden können. Diese Einstellungen können sowohl das Aussehen als auch die zu visualisierenden Daten beeinflussen. Beispielsweise kann bei Graphen ausgewählt werden, welche Attribute visualisiert werden sollen und welche Attribute auf welcher Achse in welchen Dimensionen dargestellt werden sollen.
Für weitere Anwendungsgebiete lassen sich hoch spezialisierte Darstellungsformen implementieren, wie z.B. die Rotordrehgeschwindigkeit eines Windkraftrades oder die Darstellung von gemessenen Fehlerzuständen an Sensoren. In [MN14] sind dafür praktische Beispiele zu finden.

Einsatzgebiete von Odysseus

Odysseus kann eingesetzt werden um ein hoch flexibles Datenstrommanagementsystem zu realisieren. Durch die Erweiterbarkeit des Quellcodes kann es außerdem in allen Bereichen, wie Datenverarbeitung, Funktionsumfang und Visualisierungsmöglichkeiten, hoch spezialisiert werden. Das haben bereits viele studentische Arbeiten an der Universität Oldenburg gezeigt. Beispielsweise wurde Odysseus in der Projektgruppe „LSOP - Liveanalysen im Sport mit Odysseus P2P “an der Universität Oldenburg zur Visualisierung der Abläufe von Ballspielen eingesetzt.
Darüber hinaus kann ein durch Odysseus umgesetztes Datenstrommanagementsystem als zentraler Teil eines SCADA-Systems (supervisory control and data acquisition) dienen. Ein SCADA-System wird zur Überwachung technischer Anlagen genutzt [RR15]. Dabei ist die Verarbeitung der anfallenden Messwerte in Echtzeit durch ein Datenstrommanagementsystem ein zentraler

Bestandteil der Aufgaben. Ergänzend werden Warnungen und Fehlermeldungen bei der Überschreitung von Grenzwerten und eine hohe konfigurierbarkeit der zu überwachenden Anlagen in SCADA-System vorausgesetzt. Auch diese Funktionen sind mit Odysseus möglich, da der Funktionsumfang entsprechend erweitert werden kann, was [MN14] gezeigt hat.

Zusammenfassung Odysseus

In diesem Abschnitt wurden die Grundlagen zu Odysseus erklärt, die für das Verständnis der Arbeit notwendig sind. Dabei ist klar geworden, dass Odysseus besonders für Entwickler geeignet ist, die den Funktionsumfang erweitern wollen, da es hoch flexibel aufgebaut ist. In Abschnitt 5.3.1 wurden einige wichtige Funktionen des Odysseus-Server vorgestellt. Der Leser hat sich mit dem Prinzip der Datenverarbeitung auf Datenströmen vertraut gemacht und es wurden wichtige Konzepte eingeführt, die in einem DSMS genutzt werden können und alle von Odysseus zur Verfügung gestellt werden. In Abschnitt 5.3.1 wurde die Benutzeroberfläche „Odysseus-Studio“ vorgestellt, die als Schnittstelle zur Bearbeitung und Nutzung des DSMS „Odysseus-Server“ dient. Es wurden Möglichkeiten aufgezeigt, Anfragepläne, die im vorangegangenen Abschnitt eingeführt wurden, mit unterschiedlichen Techniken zu bearbeiten. Auch wurde deutlich, dass Odysseus-Studio vielseitig zur Visualisierung von Daten, die aus Datenströmen gewonnen werden, eingesetzt werden kann. In Abschnitt 5.3.1 wurden verschiedene abstrakte Nutzungsszenarien von Odysseus vorgestellt.
Die Projektgruppe SESAdata an der Carl von Ossietzky Universität Oldenburg beschäftigt sich mit der Datenverarbeitung in einem Smart Grid und im speziellen im SESA-Lab, das im Institut für Informatik im Einsatz ist. Bei der Simulation eines Smart-Energy-Grid fallen, wie beim Betrieb eines normalen Kraftwerkes, viele Daten an. Diese sind aufgrund der verschiedenen simulierten Energieerzeuger und der zusätzlichen Einbeziehung echter Messwerte sehr heterogen. Auch die Frequenz ist variabel und kann u.U. schwanken. In der Einleitung wurde erklärt, dass unter diesen Umständen der Einsatz eines Datenstrommangementsystems sinnvoll ist. Dafür eignet sich in diesem Fall das DSMS Odysseus aus verschiedenen Gründen gut. Zum einen wird es ebenfalls an der Carl von Ossyietzky Universität entwickelt und es können Synergieeffekte genutzt werden. Zum anderen ist es sehr flexibel ausgelegt und bietet mit dem Odysseus-Studio eine erweiterbare GUI, die sowohl zur Steuerung von Anfragen, als auch zur Visualisierung der Daten genutzt werden kann. Das hat in einem ähnlichen Kontext, wie SESAdata bereits die Erweiterung von Odysseus als SCADA-System für Windkraftanlagen gezeigt, die in Abschnitt 5.3.1 vorgestellt wurde.

5.3.2. Druid

KB

Druid ist eine zeitreihenbasierte Datenbank, welche darauf ausgelegt ist, große Menge von Daten mit dem dazugehörigen Zeitstempel zu speichern und zu verarbeiten. Da die Daten, die in diesem Projekt verarbeitet werden, sollen einen Messzeitpunkt besitzen, ist es sinnvoll diese Daten in einer zeitreihenbasierten Datenbank zu speichern und so verschiedenste Analysen erlauben. Neben den Zeitstempeln können verschiedene „dimensions“ gespeichert werden, die verschiedene zu speichernde Daten enthalten können. Diese „dimensions“ sind vergleichbar mit Attributen in relationalen Datenbankschemata. Sie geben an unter welchem Namen die einzelnen Werte gespeichert werden sollen. Ebenso können zu diesen „dimensions“ verschiedene Metriken gespeichert

werden. Mit Hilfe dieser Metriken können später direkt aggregierte Werte ermittelt und dementsprechend analysiert werden. So ist es zum Beispiel möglich einen Mittelwert, einen Maximalwert und einen Minimalwert über die verschiedenen gespeicherten Werte zu ermitteln. Somit müssen solche Berechnungen später nicht von einer Anwendung vorgenommen werden, sondern können direkt in der Datenbank berechnet werden.
Druid ist allerdings nicht nur eine zeitreihenbasierte Datenbank, sondern besteht aus vielen verschiedenen Komponenten, die miteinander kommunizieren.
Die einzelnen Komponenten werden als Druid Cluster zusammengefasst. Innerhalb des Clusters existieren verschiedene Knoten. Diese sind der Echtzeitknoten („Realtime-Node“), der historische Knoten („Historical-Node“), der Koordinator Knoten („Coordinator-Node “) und der Broker Knoten („Broker-Node“). Diese 4 Knoten sind für die komplette Datenverarbeitung innerhalb von Druid verantwortlich.
Da der Arbeitsspeicher in der Regel begrenzt ist, bietet Druid nicht nur eine In-Memory Speicherung der Daten an, sondern kann Daten auch persistent über einen langen Zeitraum speichern. Hierfür ist der historische Knoten und der dazugehörige Deep-Storage zuständig. Liegen die Daten nicht mehr in einem festgelegten Zeitintervall für die In-Memory Speicherung, werden diese an den historischen Knoten weitergegeben und von diesem in einem sogenannten Deep Storage gespeichert. Dieser kann z.B. eine andere Datenbank sein, die dann die Daten für die Langzeitspeicherung hält. LAOTSE benutzt dafür Cassandra (Siehe [Apa15]). Der Vorteil gegenüber anderen Konstrukten ist hierbei, dass in Druid die Daten automatisiert vom Echtzeitknoten an den historischen Knoten weitergeleitet werden.

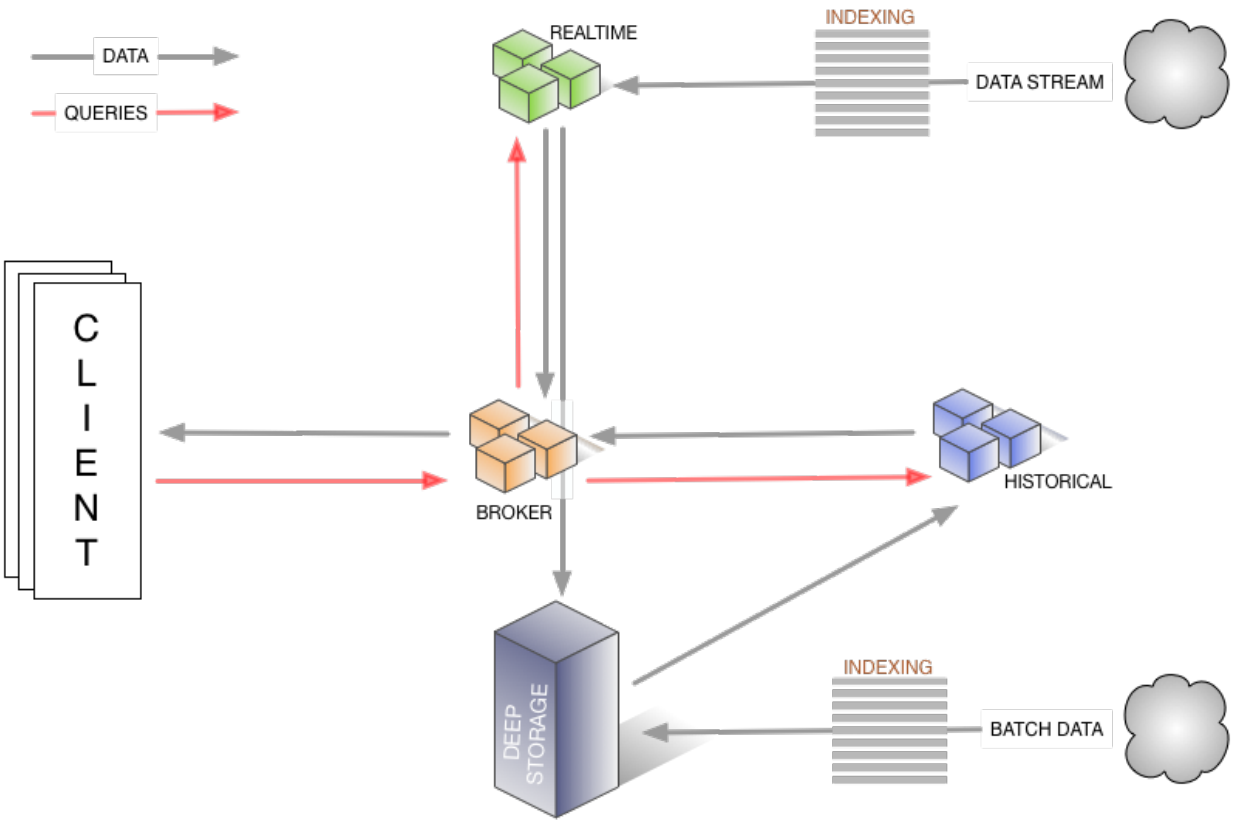


Abbildung 5.5.: Druid Datenfluss [**whatDruid**]

Die gespeicherten Daten stehen für den Benutzer also entweder im Echtzeitknoten oder im his-

torischen Knoten zur Verfügung. Da der Benutzer allerdings nicht weiß in welchem Knoten sich welche Daten befinden, weiß er auch nicht an welchen Knoten er seine Anfrage senden soll. Daher existiert der Brokerknoten, der die Anfragen des Benutzers verwaltet. Abbildung 5.5 zeigt diese Kommunikation des Benutzers mit dem Brokerknoten und der internen Verwaltung dieser Anfrage in Druid. Der Benutzer sendet also seine Anfrage an Druid an den Brokerknoten. Dieser stellt dann wiederum die Anfrage an den Echtzeitknoten und den historischen Knoten. Beide können nun Teile der Daten, die für die Anfrage benötigt werden, enthalten. Daher senden sie die Daten zurück an den Brokerknoten. Der Brokerknoten fügt die Daten aus dem Echtzeitknoten und dem historischen Knoten dann gegebenenfalls zusammen und sendet das fertige Ergebnis zurück an den Benutzer, der nun das gesuchte Ergebnis erhält, egal ob die Daten im Echtzeitknoten oder im historischen Knoten liegen.
Somit ist Druid nicht nur eine einfache zeitreihenbasierte Datenbank, sondern sie bietet die Möglichkeit den kompletten Datenfluss in einem System abzuhandeln und sich somit viele Kommunikationsschwierigkeiten zu ersparen, denn innerhalb von Druid können die Daten sowohl In-Memory als auch persistent in einer Langzeitdatenbank gespeichert werden und die komplette Kommunikation findet innerhalb von Druid statt. Da allerdings auch die Kommunikationswege innerhalb von Druid interessant sind, soll im folgenden Abschnitt noch einmal genauer auf den Datenfluss vom Einfügen der Daten bis hin zur persistenten Speicherung im historischen Knoten eingegangen werden.

Kommunikation in Druid

Wie im vorherigen Abschnitt beschrieben gibt es im Druid Cluster verschiedene Knoten, die miteinander kommunizieren müssen. Wie die Kommunikation dabei abläuft soll in diesem Abschnitt beleuchtet werden.
Druid kann Daten direkt von einem Datenstrom einlesen. Allerdings kann Druid die Daten nicht direkt aus dem Datenstrom ziehen, sondern für das Einlesen eines Datenstroms wird ein sogenanntes Messaging System benötigt. Im Falle von Druid ist dies zumeist Apache Kafka (Siehe [Apab]).
Kafka funktioniert dabei so, dass verschiedene „Producer“ existieren, die die Daten des Datenstroms an Kafka weitergeben. Innerhalb von Kafka werden diese Daten nun hinterlegt. Um an diese Daten des Datenstroms zu gelangen, werden sogenannte „Consumer“ in Kafka registriert, die informiert werden, sobald neue Daten in Kafka zur Verfügung stehen und sich diese Daten dann ziehen können. Druid kann gemäß dieses Aufbaus als Kafka Consumer registriert werden um so an die zu speichernden Daten zu gelangen. Somit wird der Datenstrom also zunächst einmal an Kafka übergeben um dann zum Druid Indexer zu gelangen. Abbildung 5.6 zeigt die Kommunikation innerhalb von Druid. Diese Daten des Datenstroms werden dann zunächst einmal im Echtzeitknoten gespeichert. Im Echtzeitknoten sollen diese Daten allerdings nur solange liegen, wie das definierte Zeitfenster vorgibt. Ist dieses Zeitfenster für den aktuellen Datensatz abgelaufen, soll dieser im historischen Knoten abgelegt und somit an die Langzeitdatenspeicherung übergeben werden.
Dazu existiert der Koordinatorknoten. Dieser ist dafür zuständig, dass die Daten vom Echtzeitknoten in den historischen Knoten übertragen werden. Dazu existiert ein sogenannter Metadatenspeicher, in den abgelaufene Daten vom Echtzeitknoten hinterlegt werden, damit diese nicht verloren gehen können. Diese Daten greift der Koordinator auf und gibt sie an Zookeeper weiter (Siehe [Zoo]). Zookeeper ist ein Synchronisationsservice. Mit Hilfe von Zookeeper findet dann also die Übertragung der Daten, die vom Koordinatorknoten aufgegriffen wurden hin zum histori-

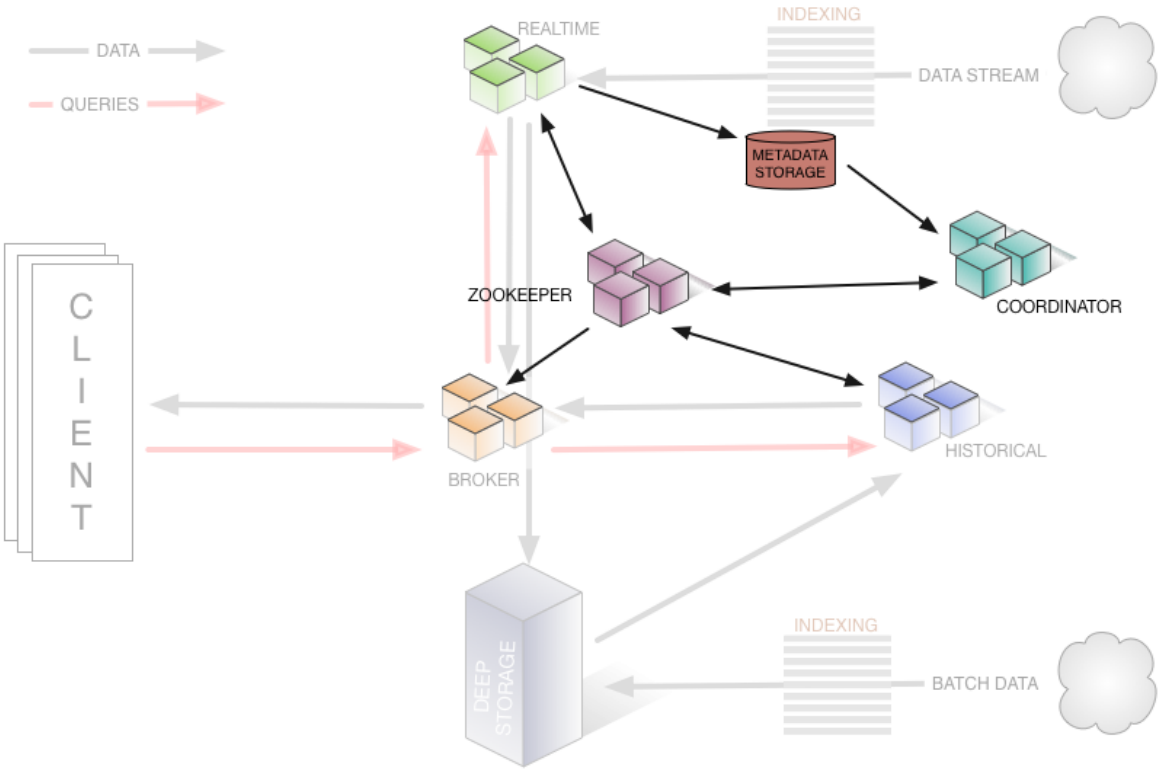


Abbildung 5.6.: Druid Kommunikation [**whatDruid**]

schen Knoten statt. Die Daten werden dann vom historischen Knoten in die definierte Datenbank für die Langzeitdatenhaltung geschrieben.
Zookeeper ist auch für die Kommunikation der einzelnen Knoten untereinander zuständig, zum Beispiel wenn eine Anfrage des Benutzers gestellt wird, ist Zookeeper dafür verantowrtlich, dass die erforderlichen Daten zum Brokerknoten gelangen.

Druid Verteilung

Um Druid hinsichtlich der Performance zu verbessern, kann Druid auf mehrere Knoten verteilt werden. Allerdings sollte auf diesen Maschinen dann nicht dar komplette Cluster gestartet werden, sondern lediglich der Echtzeitknoten und eventuell zusätzlich der historische Knoten für die persistente Speicherung. Somit sind dann im Druid Cluster weiterhin ein Brokerknoten und ein Koordinatorknoten vorhanden, die den Cluster verwalten. Der Broker- und Koordinatorknoten sollten zur bessern Ausfallsicherung repliziert werden.
Wichtig bei der Verteilung ist, dass die verschiedenen Echtzeitknoten in der gleichen Consumer-Gruppe von Kafka liegen, damit Kafka die Daten für einen einzelnen Sensor nicht auf verschieden Knoten verteilt. Um die eingehenden Daten zu verteilen, müssen verschiedenen Partitionen für das Kafka Topic definiert werden. Diese verschiedenen Partitionen können dann auf die verschiedenen Echtzeitknoten verteilt werden. Damit allerdings auch Daten in allen Kafka Partitionen vorhanden sein können, muss im Kafka Producer schon festgelegt werden, wie die Daten auf die verschiedenen Partitionen verteilt werden sollen. Dazu kann für den Kafka Producer ein Parti-

tioner definiert werden, der Regeln beinhaltet, nach denen die Daten auf die Partitionen verteilt werden. So könnte zum Beispiel eine Aufteilung nach Zeitintervallen oder Ähnlichem stattfinden. Für die Verteilung von Druid ist es also so, dass im Producer eine Regel für die Partitionierung festgelegt wird. Diese Daten werden dann in die zugewiesenen Kafka Partitionen hineingeschoben. Von dort aus werden die Daten der Partitionen auf die jeweiligen Echtzeitknoten, die als Kafka Consumer registriert sind, verteilt. Da das gesamte Datenvolumen durch Kafka läuft, kann es sinnvoll sein, auch Kafka auf mehrere Knoten zu verteilen.
Um also ein möglichst optimales Ergebnis zu erreichen sollte sowohl Kafka als auch Druid verteilt werden und eine möglichst ideale Partitionierung der Daten gefunden werden, damit eine optimale Verteilung und ebenso eine performante Abfrage zusammengehöriger Daten gewährleistet werden kann.

5.3.3. Kafka

KB

Apache Kafka ist ein open-source Mitteilungssystem. In diesem Projekt ist es deshalb von Bedeutung, weil es für die Einbindung des Datenstroms in Druid zuständig ist. Die Grundfunktionalitäten werden in diesem Abschnitt genauer beleuchtet.
Kafka ist so aufgebaut, dass die Daten nur durchlaufen und kurzzeitig zwischengespeichert werden. Für die Aufnahme dieser Daten können verschiedene Topics erstellt werden, in denen verschiedene Daten gespeichert werden können. Für jedes erstellte Topic können sich sogenannte Producer und Consumer für Kafka registrieren.
Die Producer sind die Produzenten der Daten. Das bedeutet, dass diese die Daten liefern, die dann durch Kafka hindurch laufen. Es kann aber nicht nur ein Producer pro Topic registriert werden, sondern es können auch mehrere Producer Daten für ein Kafka Topic liefern. Dementsprechend ist Kafka auch für die Aufnahme von Daten verschiedener Datenquellen geeignet.
Ähnlich läuft es auch beim ausgehenden Datenstrom aus Kafka. Für die Aufnahme von Daten, die in Kafka ankommen, registrieren sich verschiedene Consumer für die verschiedenen Topics. Auch hier gilt, dass mehrere Consumer sich für das gleiche Topic registrieren können. Die Consumer bekommen dabei in regelmäßigen Abständen die neuen Daten, die in Kafka eingegangen sind, von Kafka gesendet und können diese dann weiter verarbeiten. Dabei gilt, dass die Consumer verschiedenen Consumer-Gruppen zugeordnet werden können. Liegen mehrere Consumer in der gleichen Consumer-Gruppe, werden die Daten aus Kafka auf diese Consumer verteilt. Liegen die Consumer in verschiedenen Consumer-Gruppen, erhalten alle Consumer die gleichen Daten.
Somit ist also auch eine Verteilung mit Kafka möglich und damit eine Verbesserung der Performanz.
Kafka benötigt für die interne Kommunikation Zookeeper, der im nächsten Abschnitt beschrieben wird.

5.3.4. Zookeeper

DN

Apache Zookeeper ist ein Kommunikationsservice. Für dieses Projekt dient es dazu die Synchronisation und die Kommunikation zwischen den verschiedenen Druid Knoten und Kafka zu

organisieren. Alle für das Projekt relevanten Komponenten registrieren sich bei Zookeeper, sodass Zookeeper die Komponente ist, die alle Knoten des Systems kennt.
Darüber hinaus ist Zookeeper dafür zuständig, dass die Daten korrekt zwischen den einzelnen Knoten ausgetauscht werden und die Ergebnisdaten einer Benutzeranfrage auch an den Benutzer geliefert werden. Zookeeper organisiert also die komplette Kommunikation, sowohl zwischen Druid und Kafka, als auch in Druid intern sowie in Kafka intern und zählt damit zu den wichtigsten Komponenten dieses Projekts.

5.3.5. MySQL

MM

MySQL [Mys] ist eine relationale Datenbank und Freeware. Es kann die gängige Sprache SQL genutzt werden, um die Datenstrukturen zu manipulieren und diese abzurufen. Außerdem stehen verschiedene Treiber, z.B. ein JDBC-Treiber für Java zur Verfügung, um mit einem Client auf die Datenbank zuzugreifen. Prozeduren, Funktionen und Trigger können direkt in dem Datenbankmanagementsystem implementiert werden. Dabei ist die SQL-Sprache sehr ähnlich mit PL/SQL von Oracle [Pls].
In LAOTSE werden zwei MySQL-Datenbanken genutzt. Zum einen hält die LAOTSE-DB Metadaten der Sensoren. Auf diese wird vom LAOTSE-Server zugegriffen. Zum anderen hält der Metadatenspeicher von Druid Informationen über die Segmente wie z.B. Name, Version, Start- und Endzeitpunkt. Auf diese Instanz greift der Coordinator von Druid zu. Sie wird von Druid automatisch verwaltet und muss nur zur Verfügung gestellt und einmalig konfiguriert werden.

5.3.6. Cassandra

KB

Die Datenbank Cassandra ist eine NoSQL-Datenbank, die mit Schlüssel-Wert-Paaren arbeitet. Dadurch kann eine hohe Skalierbarkeit und eine hohe Verfügbarkeit gewährleistet werden. Durch die Speicherung in Schlüssel-Wert-Relationen kann eine große Menge an Daten abgespeichert werden und diese können durch den Schlüssel sehr einfach wieder abgerufen werden. Die Datenbank Cassandra weist wie schon erwähnt eine hohe Skalierbarkeit auf. So kann während der Laufzeit ein weiterer Cassandra-Knoten hinzugefügt werden und die Daten werden dann auch in diesem Knoten gespeichert. Um einen Cassandra-Knoten in einem Cluster zu registrieren, muss ein so genannter „Seed Provider“ angegeben werden. Dieser ist ein anderer Knoten innerhalb des Clusters. Da beim aufsetzen eines neuen Clusters noch kein Knoten im Cluster vorhanden ist, gibt der erste Knoten eines Clusters immer sich selbst als „Seed Provider“ an. Bei der Speicherung von Daten setzt Cassandra auf Tabellen mit Spalten. Cassandra ist demnach zwar eine NoSQL-Datenbank, die Schlüssel-Wert-Paare speichert, allerdings ist sie von der Struktur her dem bekannten relationalen Schema sehr ähnlich. [Apa15].

Cassandra kann für das Datenbanksystem Druid als Deep-Storage genutzt werden. Wird Cassandra als Deep Storage für Druid eingesetzt, ist Cassandra für die Langzeitdatenspeicherung zuständig. Das bedeutet, dass Druid alle Daten, die aus dem Realtime-Knoten kommen, an Cassandra weitergibt um die Daten dort persistent zu speichern. Dazu muss Druid den Cassandra Keyspace kennen, damit auf die Datenbank zugegriffen werden kann. Fallen die Daten aus dem

Realtime Knoten raus, weil sie zu lange nicht genutzt wurden, sollen sie als historische Daten gespeichert werden. Dabei ist zu beachten, dass Druid die Daten nicht in ihrer ursprünglichen Form hinterlegt, sondern auf eine verschlüsselte Art und Weise, mit Hash-Werten (vgl. [Druc]). Das bedeutet für den Benutzer wiederum, dass er die Daten nicht direkt aus Cassandra auslesen kann, sondern die Daten über Druid abrufen muss. Möchte der Benutzer also auf die Daten, die bereits als historisch betrachtet werden, zugreifen so kann er eine Anfrage an den historischen Knoten von Druid stellen und dieser liest dann die Daten aus Cassandra aus, bringt sie wieder in eine lesbare Form und gibt dem Benutzer dann das Ergebnis zurück.

5.3.7. Druid R API

LS

Für den direkten Zugriff auf Daten innerhalb von Druid kann die Open-Source RDruid Bibliothek [Rdr] verwendet werden. Diese ist in der Programmiersprache R implementiert und verwendet Http, um JSON-kodierte Anfragen an Druid zu senden. Die Ergebnisse dieser Anfragen werden von RDruid geparst und in lesbarer Form (als R-data frame) zurückgegeben.
RDruid bietet aktuell die folgenden vordefinierten Anfragen an:

TimeBoundary Liefert den ersten und den letzten Zeitpunkt, zu dem von einer bestimmten Datenquelle Daten in Druid vorhanden sind.

Timeseries Liefert Daten in einem bestimmten Intervall. Dabei können direkt beim Aufruf Metriken übergeben werden, die auf diesen Daten auszuführen sind (Summe, Anzahl etc.) sowie nach der Auswertung auszuführende Aggregationen, wie z.B. die Durchschnittsberechnung.

GroupBy Diese Anfrage arbeitet ähnlich wie die Timeseries mit einem Intervall. Allerdings gibt es hier die Möglichkeit die Daten nach angegebenen Eigenschaften gruppieren zu lassen.

TopN Diese Anfrage liefert die N größten Werte für eine bestimmte Metrik.

Weitere Abfragen müssten analog der bereits vorhandenen hinzugefügt werden.
RDruid nutzt Httr für die Http Nutzung, allerdings wurde diese R Bibliothek in der neuen Version geändert, so dass Argumente aus dem Httr Aufruf an jsonlite-Funktionen weitergeleitet werden und dort zum Absturz führen. Die Nutzung von RDruid ist demnach zurzeit nicht problemlos möglich, es wird aber erwartet, dass diese Probleme in der nahen Zukunft behoben werden.

6. Systemdesign

DN

Dieses Kapitel beschreibt den grundlegenden Aufbau des entwickelten Systems. Dabei wird zunächst auf das grobe Design mit der Verteilung der einzelnen Komponenten eingegangen. Im weiteren Verlauf dieses Kapitels sollen dann die Komponenten in ihren Einzelheiten beschrieben werden. Dabei wird vor allem zwischen der entwickelten Soft- und Hardware unterschieden. Es wird aber ebenfalls auf den Aufbau der Datenbanken und die Optimierung bzw. die Auswahl der verschiedenen Komponenten eingegangen.

6.1. Übersicht

MM

In diesem Abschnitt wird das Design des Systems grob dargestellt. Auf die einzelnen Komponenten und deren Interaktionen zueinander wird im Verlauf dieses Kapitels detaillierter eingegangen.

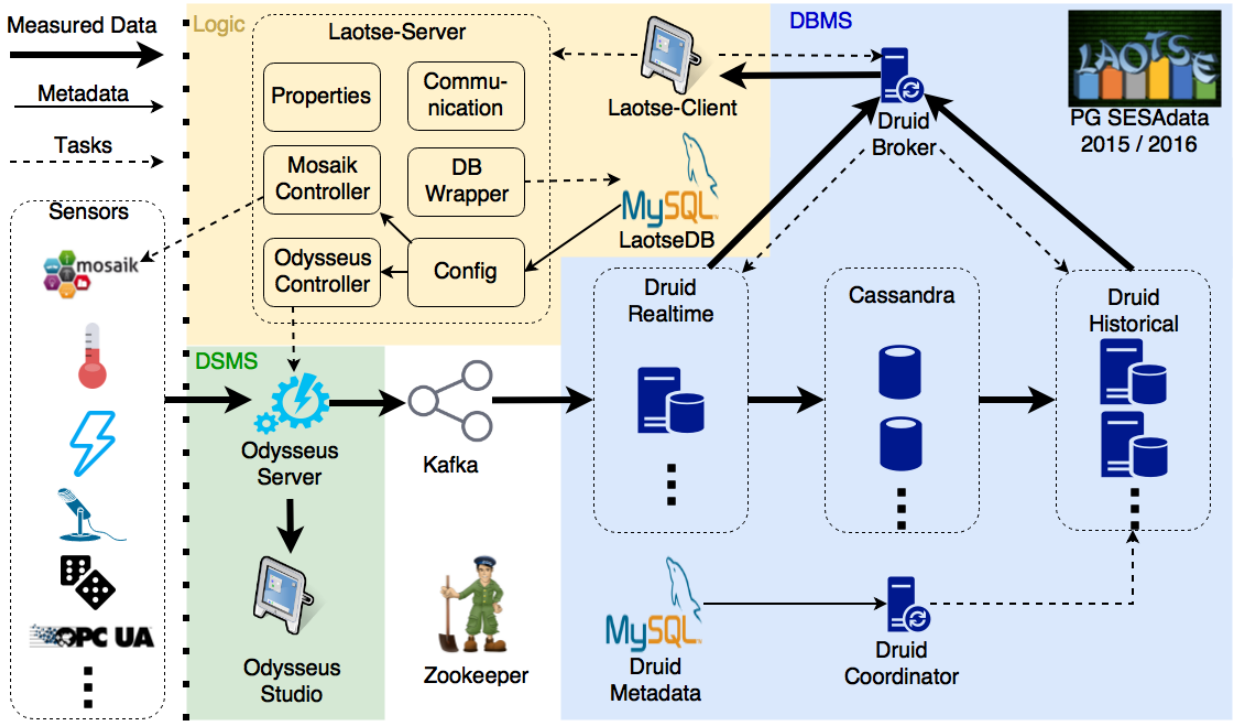


Abbildung 6.1.: Design von LAOTSE

In Abbildung 6.1 ist das Design von LAOTSE grobgranular in einer Übersicht dargestellt. Daten

aus den unterschiedlichen Sensoren laufen im Odysseus-Server zusammen, der die Datenströme behandelt. Diese können direkt mit dem Odysseus-Studio visualisiert und aggregiert werden.
Die Datenströme verlaufen durch Kafka als Schnittstelle weiter in die Datenbank Druid, in der sie persistent gespeichert werden. Dies geschieht zuerst kurzfristig im Druid-Realtime-Knoten. Dieser gibt sie weiter an den Deepstorage Cassandra, aus dem ältere Daten bei Bedarf von dem Druid-Historical-Knoten geladen werden können. Der Druid-Koordinator verwaltet die Metadaten der Daten, die in einer MySQL-Datenbank gehalten werden. Der Druid-Broker dient dazu Anfragen aus dem LAOTSE-Client entgegenzunehmen, zu verarbeiten und die entsprechenden Ergebnisse zu laden. Zusätzlich können aus dem LAOTSE-Client Nachrichten an den LAOTSE-Server gesendet werden. Dieser hat Zugriff auf die LAOTSE-Datenbank, die Metadaten über die Sensoren und Anfragen beinhaltet. Dadurch kann über Odysseus, durch den LAOTSE-Server automatisiertes, Datenstrommanagement betrieben und mosaik-Simulationen gesteuert werden. Zookeeper verwaltet die Kommunikation der einzelnen Knoten untereinander.

6.2. Verteilung

DN

In diesem Abschnitt wird die Verteilung des Systems dargestellt.
Abbildung 6.2 zeigt die einzelnen Bestandteile des Systems und wie diese miteinander interagieren. Dabei ist es grundsätzlich so, dass alle Bestandteile des Systems im SESA-Lab liegen und von außerhalb auf diese Komponenten zugegriffen werden kann. Bei diesem Projekt wird der Benutzer mittels eines Odysseus Clients, über eine VPN-Verbindung auf das System zugreifen. Die eingerichtete Entwicklungsumgebung umfasst sechs virtuelle Maschinen, zwei OPC UA Server und drei Raspberry PIs. Die Komponenten haben die internen IP-Adressen 172.20.10.130 - 172.20.10.138. Auf den virtuellen Maschinen liegen die Systemkomponenten und auf den Raspberry PIs liegen die Datengeneratoren für die einzelnen Sensoren. Die 130er Maschine enthält Tools, die bei der Verwaltung und der Entwicklung des Systems unterstützen (SVN, Jenkins, SonarQube). Über die 131er Adresse ist einer der drei Raspberry PIs angeschlossen, der die Daten des Temperatursensors überträgt. Die beiden anderen Raspberry PIs sind über die 137er und 138er Adresse an das System angeschlossen und liefern die Daten des Stromsensors und des Audiosensors. Die virtuelle Maschine mit der 132er Adresse ist eine Windows Maschine, auf die über RDP zugegriffen werden kann. Auf dieser Maschine befindet sich der Odysseus Client um die Vorgänge innerhalb von Odysseus zu überwachen und ein UaExpert Client, der einen Zugriff auf den gegebenen OPC UA Server ermöglicht. Die 133er Maschine ist die Maschine, die die Systemkomponenten für die Speicherung der Daten enthält (Kafka, Zookeeper, Druid Cluster, MySQL DB, Cassandra). Die 134er und 135er Maschine enthalten Zahlengeneratoren, die hochfrequente Daten für einen komplexen Test liefern können. Auf der 136er Maschine laufen mosaikund Odysseus. Unter der Adresse 172.20.50.50 sind zwei OPC UA Server verfügbar, von denen Daten abgerufen werden können.

6.3. Softwaredesign

In diesem Abschnitt wird die Implementierung und Vernetzung des LAOTSE-Systems im Detail beschrieben.

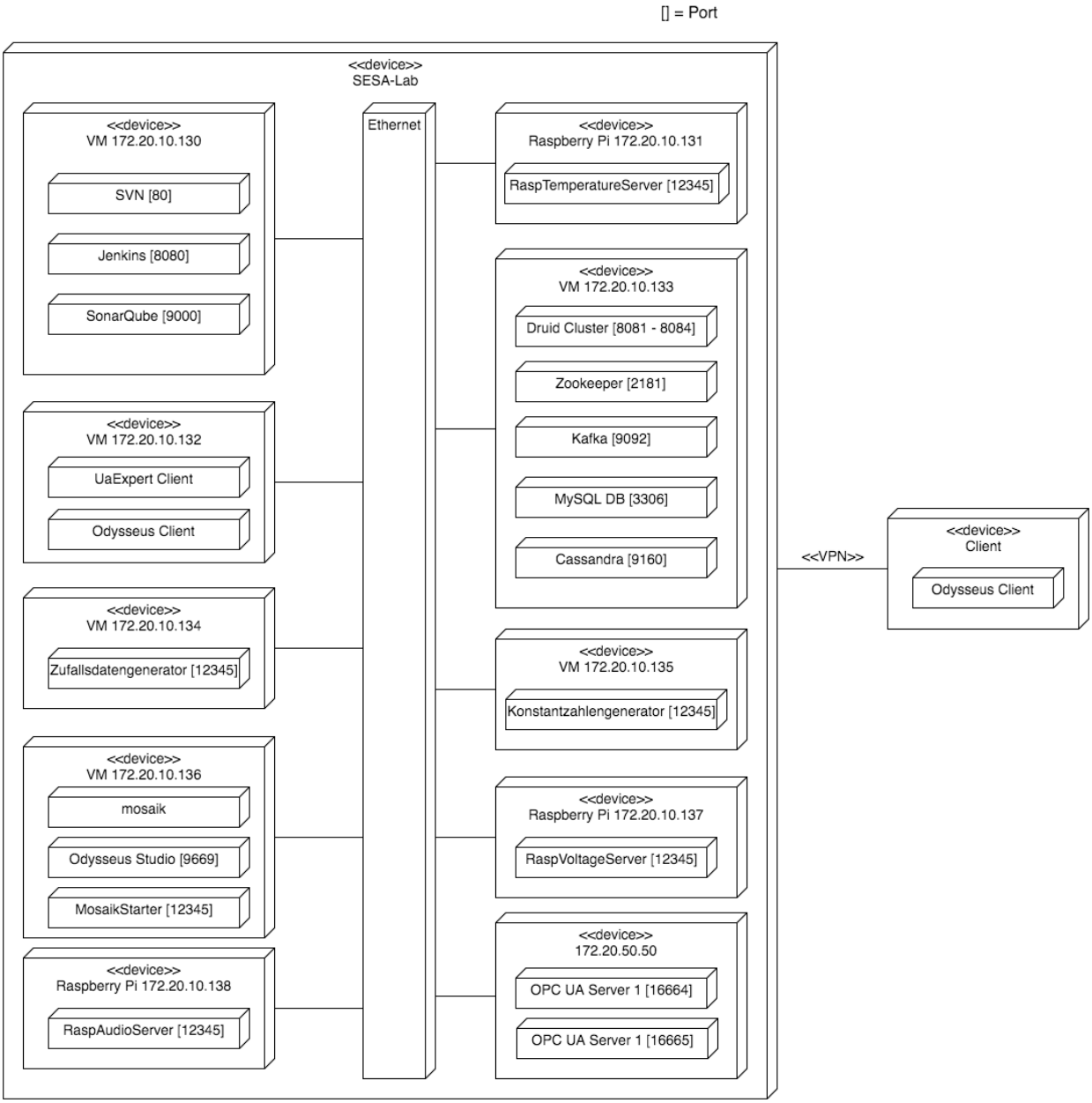


Abbildung 6.2.: Verteilungsdiagramm

6.3.1. Komponentendiagramm

MM, UG, KB

In Abbildung 6.3 ist das Komponentendiagramm zu sehen, welches darstellt, wie LAOTSE implementiert wurde. Dieses weicht in einigen Details von der ursprünglichen Planung aus dem Lastenheft ab. Diese Abweichungen sind durch die Verwendung von Druid als Datenbank und JavaFX als Application-Framework entstanden und werden in 6.3.1 genauer erläutert.
Die Komponenten lassen sich in drei Kernbereiche unterteilen. Zum einen gibt es Komponenten, die sich mit der Datenhaltung beschäftigen. Auf diese Komponenten greifen dann die Komponenten aus dem Bereich der Logik zu, die für die Datenverarbeitung und Datenanalyse zuständig sind. Im letzten Schritt sollen diese Daten für den Benutzer visualisiert werden. Dafür sind die

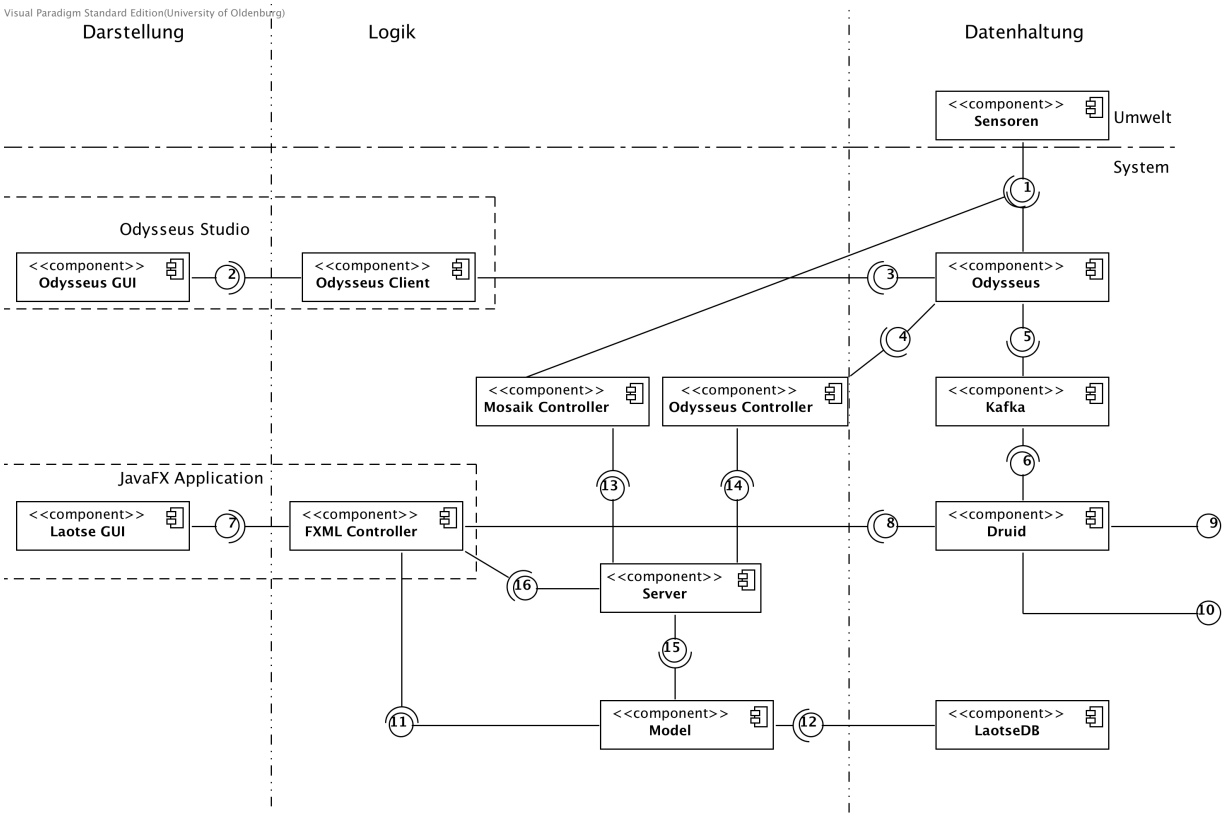


Abbildung 6.3.: Komponentendiagramm LAOTSE

Komponenten aus dem Bereich Darstellung zuständig, welche die Benutzeroberfläche für das System darstellen.
Im Folgenden werden alle Komponenten im einzelnen beschrieben.

Sensoren Nicht zum LAOTSE System gehören die Sensoren, die außerhalb des Systems liegen. Im Komponentendiagramm sind sie allerdings aufgeführt, da sie die Daten für das System liefern.

Odysseus Server Die erste Komponente, die dem Bereich der Datenhaltung zuzuordnen ist, ist der Odysseus Server, der die Daten, welche die Sensoren liefern, in Echtzeit verarbeiten kann. Odysseus ist also für den Datenstrom zuständig und ist somit die Grundlage für die Live-Datenanalyse.

Kafka Kafka ist ein Mitteilungssystem, das als Schnittstelle zwischen dem Datenstrom aus Odysseus und der Datenbank Druid dient. Es puffert ggf. alle Daten und leitet diese aus dem Datenstrom an Druid weiter. Der Datenstrom aus Odysseus ist dabei als Kafka-Producer implementiert und Druid als Kafka-Consumer. Dieser ruft die Daten aus Kafka ab, sobald diese auftauchen. Kafka kann genau wie Druid mehrere verteilte Partitionen haben.

Druid Druid ist eine Datenbank, die alle Messwerte in In-Memory und Langzeittechnologie speichert. Siehe dazu Abschnitt 5.3.2

Zu beachten ist bei Druid, dass die Abbildung nicht die Datenbank an sich darstellt, sondern den Datenbank Handler, welcher die Zugriffe auf der jeweiligen Datenbank ausführt. Die Datenbanken selbst werden durch die Symbole rechts davon dargestellt.

Die 3 Komponenten Odysseus Server, Kafka und Druid organisieren also die Datenhaltung innerhalb des LAOTSE Systems. Damit die gegebenen Daten jedoch auch organisiert und verarbeitet und nicht nur gespeichert werden können, müssen auch Logikkomponenten existieren, die dies umsetzen.

Model Das Model speichert die Konfigurationsdaten für das System in einer Datenbank und stellt sie für die anderen Logikkomponenten zur Verfügung. Diese Komponente enthält z.B. die Metadaten der einzelnen angeschlossenen Sensoren. Somit kann aus diesen Daten ermittelt werden, welche Sensoren aktuell an das System angeschlossen sind und es können über diese Komponenten ebenfalls Sensoren hinzugefügt werden. Das Selbe gilt für die zur Verfügung stehenden Analysen. Aus diesem Grund sind alle anderen Controller des Systems Observer dieser Modelkomponente, um auf Änderungen und neue Sensoren reagieren zu können und so zu gewährleisten, dass eine korrekte Analyse durchgeführt wird. Konkret sind das der FXML-Controller und der Odysseus-Controller.

Odysseus Client Für die Odysseus Komponenten existiert eine eigene Logikkomponente. Diese Komponente ist der Odysseus Client, der auf dem Datenstrom des Odysseus Servers Anfragen laufen lassen und auf diese Art und Weise die Daten analysieren oder aggregieren kann. Gemeinsam mit der dazugehörigen Odysseus GUI ist diese Komponente im Odysseus Studio vereint, das Anfragen auf den Datenstrom aus Odysseus ausführen kann und diese analysierten Daten im nächsten Schritt visualisiert. Das Odysseus Studio sollte vor allem dazu genutzt werden, zu visualisieren, welche Sensoren aktuell Daten liefern und gegebenenfalls, ob Warnzustände vorliegen. Bei der Umsetzung ist aus Performance- und Austauschbarkeitsgründen nach Möglichkeit die Trennung der Clients nach Funktionalitäten vorzunehmen. Eine denkbare Unterteilung wäre:

1. **Sensor-Client (write)**
2. **Analyse-Client (read)**
3. **Visualisierungs-Client (read)**

Odysseus GUI Die Odysseus GUI visualisiert dabei die im Odysseus Client ermittelten Daten und stellt somit die Komponente für die Echtzeitüberwachung dar.

Odysseus Controller Der Odysseus Controller beobachtet das Model und reagiert auf dessen Änderungen mit Anweisungen an Odysseus. Mit diesen Anweisungen können Anpassungen in den Datenströmen in Odysseus vorgenommen werden. Dies könnte z.B. das Einbinden von neuen Datenquellen und -senken oder deren Aggregation sein.

MosaikController Der Mosaik Controller hat eine Verbindung zu einem Server, auf dem Mosaik Szenarios simuliert werden können. Er steuert deren Initialisierung und Simulation. Dazu kommuniziert er mit dem Server, dem OdysseusController und dem Modell um z.B. virtuelle Sensoren aus Mosaik zu erkennen, in LAOTSE einzufügen, in die Datenbank zu schreiben und deren Anfragen an Odysseus zu senden. Dadurch können die simulierten Daten aus Mosaik in LAOTSE aufgezeichnet und verarbeitet werden.

Server Der Server ist die zentrale Kommunikationsstelle des Systems. Er realisiert die Plattform für die Kommunikation des Clients mit den Server-Komponenten. Über diese Plattform kann der Client Nachrichten an den Odysseus Controller, den Mosaik Controller und das Model und demnach auch die LaotseDB senden und Nachrichten bzw. Daten von diesen

Komponenten empfangen. Diese Schnittstelle ist vor allem dazu da, dass der Benutzer nicht auf wichtige Konfigurationen zugreifen kann und somit nur über den Server kommuniziert, der ihm die angeforderten Daten weiterleitet.

FXML Controller FXML Controller sind Bestandteil jeder JavaFX-Application, wenn die eigentliche GUI in FXML Dateien ausgelagert wird. Diese Controller steuern die GUI, indem die Elemente der GUI mit Daten versorgt werden und Interaktionen des Nutzers mit GUI von diesen Controllern verarbeitet werden. Daher besteht eine Schnittstelle zu Druid, über die zum einen Daten an die Controller und zum anderen von dem Controller Anfragen an die Datenbank geleitet werden können. Die FXML Controller sind so wie alle Controller Observer des Models, um auf Änderungen des Models auch in der GUI und in der Datenbank reagieren zu können. Diese Änderungen werden von den Controllern koordiniert.

LAOTSE GUI Die angesprochene LAOTSE GUI ist die Haupt-Benutzerschnittstelle des Systems. Sie ermöglicht dem Benutzer bspw. Analysen auf dem Analyse-Tool auszuführen oder neue Analysen hinzuzufügen, die dann in der Konfigurationsdatenbank gespeichert werden und zu einem späteren Zeitpunkt erneut ausgeführt werden können. Ebenso bietet die LAOTSE GUI dem Benutzer die Möglichkeit neue Sensoren im System zu registrieren oder bestehende Sensoren zu bearbeiten, damit diese für spätere Analysen genutzt werden können. Dabei gibt die LAOTSE GUI die neuen Informationen an die Konfigurationskomponente, die diese Daten dann speichert und die anderen Logikkomponenten bekommen Nachricht über die Änderung und können sich die neuen Daten holen und damit arbeiten. Ebenso ist die LAOTSE GUI aber auch dazu da, die Ergebnisse der durchgeführten Analysen darzustellen und sie dem Benutzer so auf eine geeignete Art und Weise anschaulich zu machen. Diese Komponente besteht aus FXML Dateien, die lediglich die Struktur der GUI-Elemente angeben. Es wird also definiert, welche Elemente wo in der GUI platziert werden. Außerdem kann das Erscheinungsbild mit CSS bestimmt werden.

LAOTSEDB In der LAOTSEDB werden alle Metadaten zu Analysen und aufgenommenen Sensoren gespeichert. Diese können von der GUI ausgelesen und visualisiert werden. Diese Komponente wird im System als MySQL-Datenbank realisiert.

Schnittstellen

MM, KB

In Abbildung 6.3 ist das Komponentendiagramm des Systems mit seinen Schnittstellen dargestellt. Es werden verschiedene Schnittstellen eingesetzt. Dadurch können heterogene Komponenten, wie z.B. verschiedene Sensoren standardisiert eingebunden und genutzt werden. Auch wird dadurch Flexibilität sichergestellt und es können Komponenten ausgetauscht werden, ohne andere Komponenten zu bearbeiten. Die Schnittstellen sind fortlaufend nummeriert und werden in diesem Abschnitt beschrieben:

1. Die Sensoren implementieren eine Schnittstelle, die Metadaten und ggf. Datenformate der Messwerte beschreibt. Über diese Schnittstelle können Daten von den Sensoren an das System übermittelt werden und das System kann Metainformationen über die Sensoren abfragen. Außerdem kann der Mosaik Controller über diese Schnittstelle auf mosaik als Sensor zugreifen und Simulationen starten bzw. die Sensoren der Simulationen ermitteln.

2. Die grafische Benutzeroberfläche des Odysseus-Studio implementiert eine Schnittstelle, die grafische Funktionen bereitstellt. Es werden Benutzereingaben an Odysseus und Visualisierungen von Odysseus verarbeitet.

3. Der Odysseus-Server implementiert eine Schnittstelle, die vom Client des Odysseus-Studio genutzt wird. Über diese Schnittstelle lässt sich der Server steuern und z.B. Abfragen an die Datenströme erzeugen, starten und stoppen.

4. Analog zu Schnittstelle 3. Hier wird die Schnittstelle von dem Odysseus Controller genutzt, der auf Änderungen des Models reagiert. Es können z.B. neue Sensoren eingebunden werden, was durch die LAOTSE GUI ermöglicht wird.

5. Odysseus implementiert einen Kafka Producer. Über diese Schnittstelle können die Daten der Datenströme aus Odysseus an Kafka gesendet werden.

6. Druid implementiert einen Kafka Consumer. Über diese Schnittstelle können die Daten aus Kafka weiter an die Datenbank geleitet werden.

7. Analog zu Schnittstelle 2 implementiert die LAOTSE GUI eine Schnittstelle, die von der Logik zur Visualisierung von Daten genutzt werden kann. Dabei gibt es für jede einzelne Visualisierung, die durch FXML Dateien umgesetzt ist, eine Schnittstelle zu einem eigenen Controller. Über die Schnittstelle werden die Elemente der GUI bereit gestellt, damit sie mit Daten gefüllt werden können oder auf Aktionen von ihnen gelauscht werden kann.

8. Druid implementiert eine Datenbank-Schnittstelle, die von einem Controller genutzt werden kann, um auf die Daten aus der Datenbank zuzugreifen und Anfragen zu stellen.

9. Diese Schnittstelle stellt die interne In-Memory Datenbank von Druid dar.

10. Diese Schnittstelle dient als Verbindung für einen Deep-Storage, den Druid als Langzeitdatenbank nutzt. In der Konfiguration, wie sie bei LAOTSE eingesetzt wird, ist dies Cassandra.

11. Das Model stellt eine Schnittstelle bereit, die von beliebigen Controllern genutzt werden kann. Die Controller werden so über Änderungen in dem Model benachrichtigt. Dies ist in LAOTSE über das Observer-Pattern umgesetzt, indem jedes Model ein Observable ist und die anderen Komponenten sich als Beobachter registrieren.

12. Das Model hat eine weitere Schnittstelle zu einer Datenbank, in der die Konfiguration des Model persistent gespeichert werden kann. In LAOTSE geschieht dies in einer MySQL Datenbank.

13-16. Der Server ist die Instanz, die die Kommunikation der einzelnen Komponenten untereinander regelt. Dafür stellt er Schnittstellen bereit, die von verschiedenen Komponenten implementiert werden. Diese Komponenten sind der Odysseus Controller, der Mosaik Controller, die Datenbank (das Model) und der Client (die GUI). Über den Server können diese Komponenten Nachrichten von den anderen Komponenten empfangen und an die anderen Komponenten senden. Diese Schnittstellen realisieren also die komplette Kommunikation sowohl zwischen dem Client und dem Server, als auch zwischen den einzelnen Serverkomponenten. Die Komponenten registrieren sich dafür beim Server und der Server ordnet die ankommenden Nachrichten dem richtigen Empfänger zu und sendet über die Schnittstelle die Nachricht an diesen Empfänger weiter.

Schnittstelle 5

KB

Diese Schnittstelle ist die Verbindung zwischen Odysseus und Kafka. Sie ist dafür zuständig die Daten für die Speicherung in Druid zur Verfügung zu stellen. In Odysseus ist für eine solche Schnittstelle der ACCESS-Operator zuständig. Dieser benötigt 3 verschiedene Handler für die Übertragung der Daten.
Diese sind der Protocol Handler, der Data Handler und der Transport Handler. Innerhalb des Protocol Handlers wird definiert, wie die eingehenden Daten weiter verarbeitet werden. Für diese spezielle Schnittstelle wird der JSON Protocol Handler genutzt, da für die Übertragung an Kafka beziehungsweise später an Druid eine JSON Nachricht gesendet werden muss. Dieser JSON Protocol Handler wandelt die eingehenden Daten in eine JSON Nachricht um und gibt diese an den Transport Handler weiter. Der dazugehörige Data Handler gibt an welches Datenformat die übergebenen Nachrichten haben. In diesem Fall wird ein Key-Value-Object Data Handler verwendet, da JSON generell im Key Value Format vorliegt.
Die fertigen JSON Nachrichten kommen dann beim Transport Handler an. Für LAOTSE wurde dazu ein neuer Transport Handler erstellt - der Kafka Transport Handler. Dieser erhält die JSON Nachrichten mit dazugehörigen Options. Die Options enthalten hauptsächlich, für die Verbindung mit Kafka relevante, Konfigurationen. Ebenso ist in den Options das Kafka Topic gegeben, in das die Daten geschrieben werden sollen und der Schlüssel, nach welchem die Daten später partitioniert werden sollen.
Die Werte für diesen gegebenen Schlüssel werden aus dem JSON String geparst und können somit als Key an Kafka übergeben werden. Kafka kann mit Hilfe eines Partitionierers die Daten auf die verschiedenen Kafka Instanzen verteilen und die Daten somit verteilt und damit performanter speichern. Um die Performanz weiter zu steigern, werden die einzelnen Werte mit ihrem Key zunächst in Listen gesammelt und ab einer bestimmten Größe an Kafka übermittelt, damit nicht für jeden Datenwert eine neue Verbindung zu Kafka hergestellt und die Daten an Kafka übertragen werden müssen.

Differenz zum Konzept

MM

Im Konzept ist bereits ein Entwurf des Komponentendiagramms abgebildet, welches im Lastenheft vorgestellt wurde. Dieses ist in Abbildung 3.4 zu sehen. Das hier vorgestellte Komponentendiagramm beschreibt die Implementierung, wie sie wirklich umgesetzt ist. Man kann erkennen, dass diese sich an einigen Stellen vom ursprünglichen Konzept unterscheidet. In diesem Abschnitt werden diese Unterschiede beschrieben und erklärt.
Da der Client mit einer JavaFX-Application realisiert wurde, ist die GUI um die Komponente der FXML Controller erweitert. Dieses Prinzip des Model-View-Controller Patterns wurde in dem ganzen System umgesetzt. Das hat zur Umbenennung der Komponenten des Analyse-Tool zum Analyse-Controller und der Konfig zum Model geführt. Außerdem wurden einige Teilaufgaben des Analyse-Tools direkt in die FXML-Controller überführt. Dazu zählt z.B. die Analyse von Daten, deren Ergebnisse nur einmalig abgerufen und visualisiert werden sollen.
Der Bulk-Storer entfällt als einzelne Komponente komplett, da seine Aufgabe der Übertragung

der Aufgaben von der In-Memory- in die Langzeitdatenbank von Druid selbst übernommen wird. Damit werden auch die Komponenten der beiden Datenbankwrapper in Druid zusammengefasst.

Druid benötigt zur Anbindung an Odysseus Kafka. Dieses stellt den Datenstrom für Druid zur Verfügung, übernimmt aber keine weiteren Funktionalitäten innerhalb von LAOTSE .

Die Schnittstelle zur Datenbank des Models wird in eine einzelne Komponente exportiert. Dies ist die LAOTSEDB. Dadurch wird Austauschbarkeit sichergestellt.

Der Analyse Controller wurde aufgrund von Anforderungsmanagement entfernt. Es wurde im Laufe des Projekts deutlich, dass in der LAOTSE-GUI ein Dashboard mit einfachen „Analysen “ wünschenswerter ist. Der `AnalyseController` wurde in den Ausblick verschoben. Zu diesen neuen einfachen Analysen gehören z.B. Aggregationen wie der Mittelwert. Für den `Analyse-Controller` war geplant, dass er dauerhafte Analysen verarbeitet, die auf den Daten ausgeführt werden und ständig Ergebnisse bereit stellen bzw. aktualisieren. Eine solche Analyse kann z.B. in einem festen Zeitintervall wiederholt werden. Diese werden an Druid geleitet und die Daten, die als Ergebnis zurück kommen werden an das Model geleitet.

6.3.2. Projektstruktur

LS

Die Projektstruktur von LAOTSE ist dargestellt in Abbildung 6.4. Dabei sind alle Parent Pom-Projekte für die Vereinfachung des Buildprozesses zuständig.

Es müssen alle Maven-Abhängigkeiten (Tycho etc.) nur in diese eingefügt werden und sind dann automatisch in den untergeordneten Modulen geladen.

Tycho bietet damit die Möglichkeit die Packaging-Methoden *eclipse-plugin*, *eclipse-feature* und *eclipse-repository* in diesen zu nutzen.

Zusätzlich übernimmt das Parent-Projekt die Aufgabe alle Unterprojekte zu bauen und so kann mit einem Maven build z.B. des *ServerParents* das fertige, ausführbare *ServerProduct* erzeugt werden, wofür ansonsten alle Plugins und Features einzeln hätten gebaut werden müssen.

Die jeweiligen Plugins werden auf die Features verteilt.

Jedes Feature bietet eine Gesamtfunktionalität mit allen zugehörigen Abhängigkeiten an. Dadurch lassen sich Plugins, die zusammen eine bestimmte Funktionalität anbieten, praktisch im Paket importieren.

Es gibt in LAOTSE die folgenden Features, gruppiert nach Zugehörigkeit zu entweder Server, Client oder dem geteilten Model:

Model:

DruidFeature Enthält die Plugins für das Anbinden bzw. die Anfragen an Druid: *DruidWrapper*.

ModelFeature Enthält alle zum Model, der Kommunikationsgrundlage zwischen Server und Client, gehörigen Plugins: *Properties*, *LaotseModel*, *ModelProperties*. Dieses enthält Metadaten zu Sensoren, Analysen etc.

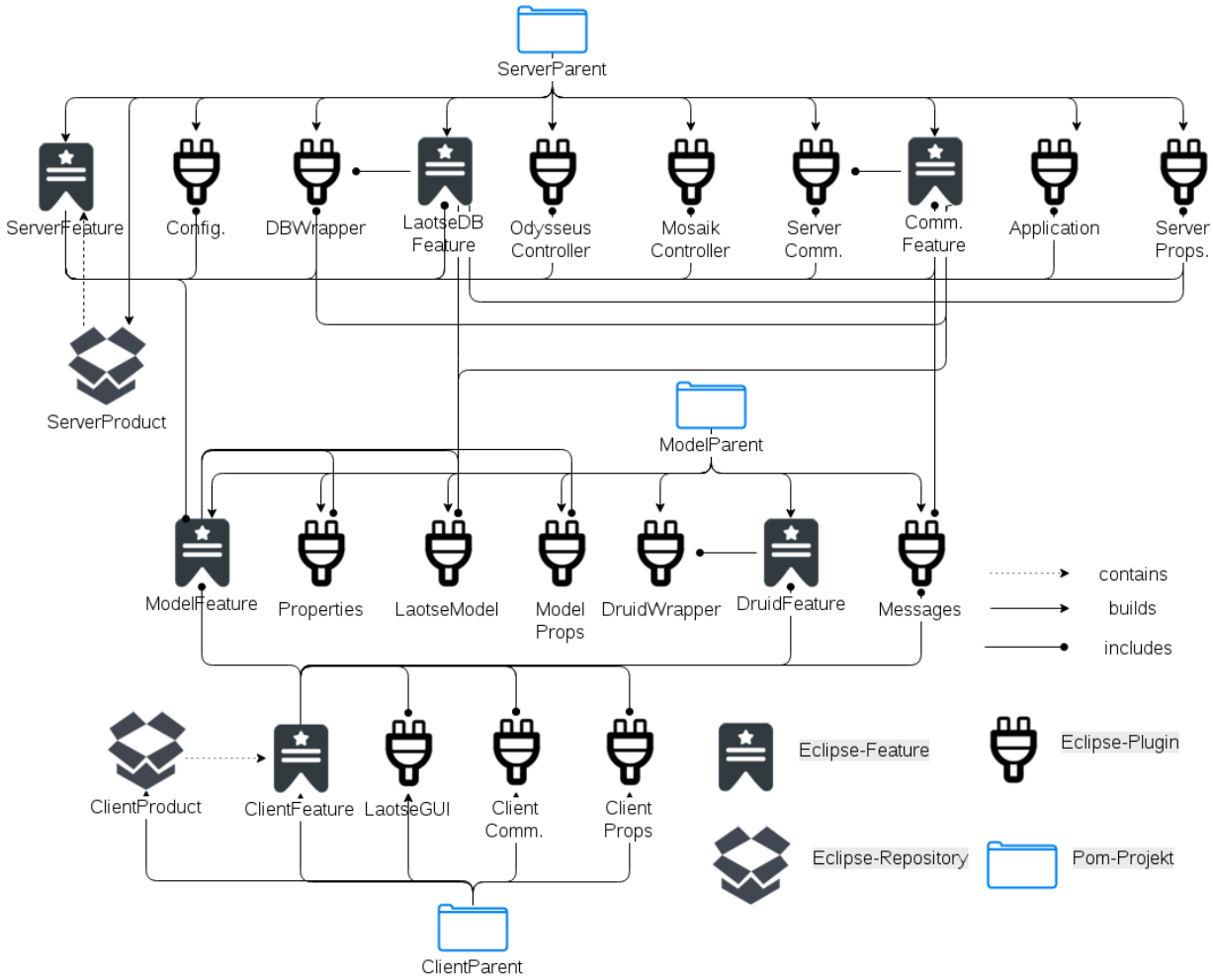


Abbildung 6.4.: LAOTSE Projektstruktur

Client:

ClientFeature Enthält alle Plugins, die für den LAOTSE-Client nötig sind: *Messages*, *LaotseGUI*, *ClientCommunication*, *ClientProperties*. Weitere Plugins sind über die Features: *ModelFeature* und *DruidFeature* eingebunden.

Server:

LaotseDBFeature Enthält alle zum LAOTSEDB Zugriff notwendigen Plugins: *LaotseDBWrapper* (kurz: DBWrapper), das *LaotseModel* und die *ServerProperties*, die die Datenbankverbindungseinstellungen enthalten.

CommunicationFeature Enthält alle Plugins, die für die Server Kommunikation notwendig sind: *ServerCommunication* und *Messages*.

ServerFeature Enthält über den Einzug von Features alle serverseitigen Plugins von LAOTSE, wie in Abbildung 6.4 dargestellt, sowie das geteilte *ModelFeature*.

Features können nun in Eclipse-Products zusammengefasst und auslieferbar gemacht werden. Der zugehörige Tycho Packagingtype ist Eclipse-Repository.

Resultierend gibt es jeweils ein *Client*- und ein *ServerProduct*, die die namentliche Funktionalität komplett enthalten, indem sie das jeweilige Hauptfeature einbinden.

6.3.3. Odysseus Controller

DN

Der Odysseus Controller ist dafür zuständig, dass für alle Sensoren des Systems, die aktiv sind die passenden Odysseus Querys gestartet werden. Diese Komponente ist deshalb von besonderer Bedeutung, weil ohne ein Starten der passenden Querys keine Daten vom Sensor durch Odysseus durchlaufen und somit auch keine Daten in der Datenbank ankommen können.
Beim Odysseus Controller ist zu beachten, dass für jeden Sensor, der Daten liefern soll mindestens ein Odysseus Query gestartet werden muss, um Daten von diesem Sensor abrufen zu können. Ebenso muss auch für eine mosaik Simulation mindestens ein Odysseus Query pro Datenquelle gestartet werden. Dies muss automatisiert geschehen können, damit aus der Laotse GUI heraus mosaik und das dazugehörige Szenario gestartet werden können.
Der Odysseus Controller ist dementsprechend dafür zuständig Odysseus Querys von den einzelnen Sensoren und mosaik Szenarien abzurufen und diese innerhalb von Odysseus zu starten. Er stellt also die Schnittstelle zwischen Laotse und Odysseus dar und verwaltet alle Sensoren, die im System angemeldet sind. Der Odysseus Controller besitzt also sowohl eine Verbindung zur Datenbank, als auch zu Odysseus. Es muss bei der Ausführung der Queries auch betrachtet werden, ob der Query eines Sensors bereits gestartet wurde oder nicht und dass an den Sensoren auch Änderungen vorgenommen werden können. Somit muss der Odysseus Controller immer über den aktuellen Status der einzelnen Queries informiert sein. Außerdem müssen Methoden zur Verfügung gestellt werden, mit denen es möglich ist, bereits gestartete Queries zu pausieren und aus Odysseus zu entfernen, falls z.B. Sensoren aus dem Sensoren entfernt werden oder ausfallen. Außerdem sollte es möglich sein, dass der Query angepasst wird, wenn am Sensor Veränderungen vorgenommen werden. Das würde für den Query bedeuten, dass er gestoppt, dann geändert und abschließend erneut gestartet werden müsste.
Für all diese Fälle stellt der Odysseus Controller Funktionen bereit. In der Datenbank existiert zudem noch eine Tabelle, die die jeweiligen Sensoren mit dazugehörigen Queries enthalten. Auf diese Art und Weise kann der Odysseus Controller alle Queries speichern, die er gestartet hat und wird dann ein Query entfernt, kann dieser einfach aus der Tabelle herausgenommen werden und wird im System nicht weiter betrachtet.
Der Odysseus Controller ist also einer der Hauptbestandteile des Systems, der die Sensorinformationen an Odysseus weitergibt und so den Datenstrom eines jeden Sensors erst ermöglicht. Er ist also dafür verantwortlich, dass die Daten der Sensoren bei der Datenbank ankommen und somit abgespeichert und analysiert werden können.

6.4. Hardwaredesign

UG

Im diesem Abschnitt wird die im LAOTSE-Systems verwendete Hardware im Detail beschrieben. Dabei liegt der Fokus auf diejenigen Komponenten, die im Rahmen des Projektes zur Realisierung angeschafft und verwendet werden. Hierbei handelt es sich um die Komponenten zur

Abbildung der vorgegebenen Datenquellen, um diese in das Zielsystem zu integrieren. Unberücksichtigt bleibt die vom Auftraggeber bereitgestellte IT-Infrastruktur.

6.4.1. Hardwareauswahl

UG

Ziel dieser Hardwareauflistung ist die Auswahl der benötigten Komponenten zur Realisierung des Projektes. Dabei werden zur Bewertung neben dem Preis und der Leistung auch die Erweiterbarkeit sowie die Verbreitung der Komponenten berücksichtigt. In diesem Abschnitt sind aus Übersichtlichkeitsgründen nur die ausgewählten Komponenten aufgelistet. Die vollständige Hardwareauswahl befindet sich im Anhang.
Zunächst werden die Anforderungen an die Komponenten aus dem Projekt dargestellt, auf deren Grundlage die Auswahl getroffen wird. Bisher besteht der Bedarf an Komponenten zur Signalerzeugung und -verarbeitung der Sensoren, welche über Einplatinencomputer wie Raspberry PIs angeschlossen werden können. Diese Sensordaten sollen an das Datenmanagementsystem (DMS) übertragen und dort persistent gespeichert und bei Bedarf analysiert werden. Im Rahmen der Anforderungserhebung wurde die Einbindung der folgenden drei Sensortypen festgelegt.

Temperatur Die Umgebungstemperatur soll mit einer geringen Abtastrate im Herzbereich erfasst und an das DMS weitergeleitet werden. Ein Temperatursensor ist bereits aus einem vorherigen Projekt vorhanden und kann bei Bedarf verwendet werden. Der vorhandene Sensor bietet die Anschlussmöglichkeit über eine I²C-Schnittstelle.

Schallwellen Hochfrequente analoge Signale im 30 kHz-Bereich sollen aufgezeichnet und an das DMS übertragen werden.

Spannungswerte Spannungsgrößen sollen über Spanungswandler nach Möglichkeit im kHz-Bereich abgetastet und an das DMS übertragen werden. Dabei sind genormte analoge Eingangssignale von 0 - 10V und 4 - 20 mA zu berücksichtigen.

Funktionale Anforderungen

Die folgende Tabelle zeigt die funktionalen Anforderungen.

ID	Prio	Anforderung
H-FA1	A	Die Hardwarekomponenten müssen die Messwerte verlustfrei verarbeiten.
H-FA2	A	Das System muss die Messwerte in ihrer Entstehungsgröße speichern.
H-FA3	A	Die Hardwarekomponenten müssen Zeitstempel liefern.
H-FA4	B	Die Hardwarekomponenten sollen um weitere Quellen erweiterbar sein.

Nicht-funktionale Anforderungen

Die folgende Tabelle zeigt die nicht-funktionalen Anforderungen.

ID	Prio	Anforderung
H-NFA1	A	Die Hardwarekomponenten müssen hochfrequente analoge Signale (Schallwellen) bis 30 kHz verarbeiten können.
H-NFA2	A	Die Hardwarekomponenten müssen analoge Eingangssignale von 0 – 10V verarbeiten können.
H-NFA3	A	Die Hardwarekomponenten müssen analoge Eingangssignale von 4 – 20 mA verarbeiten können.
H-NFA4	A	Die Hardwarekomponenten sollen die Spanungswerte im kHz-Bereich abtasten.
H-NFA5	B	Die Hardwarekomponenten müssen zuverlässig und erprobt sein.
H-NFA6	B	Die Hardwarekomponenten sollten nach Möglichkeit standardisierte Schnittstellen bereitstellen.
H-NFA7	B	Die Hardwarekomponenten sollen nach Möglichkeit als fertige Komponenten beschafft werden.
H-NFA8	C	Die Hardwarekomponenten sollen nach Möglichkeit über eine Hutschiene montierbar sein.

Auswahl Einplatinencomputer (Raspberry PI 2 B)

Unter Berücksichtigung der Anforderungen kamen die folgenden drei Geräte in die engere Auswahl. Der Raspberry PI 2 B und der Banana Pi ähneln sich von der Bauform und den Leistungsmerkmalen sehr.

Abbildung 6.5.: Raspberry PI 2 B

Beide sind vollwertige Einplatinencomputer welche über ein Mindestmaß an Schnittstellen verfügen. Der Tinkerforge ist dagegen sehr modular aufgebaut und besitzt in seiner Grundform, dem so genannten Brick, nicht einmal eine Ethernetschnittstelle. Die Auswahl erfolgte über eine Nutzwertanalyse, bei der die wichtigsten Anforderungen an die Hardwarekomponenten in die Bewertung einfließten. Die Datenrate und die Verarbeitung von analogen Signalen waren dabei

das ausschlaggebendsten Kriterien. Das detaillierte Auswahlverfahren kann im Anhang (Hardwareauswahl) nachgeschlagen werden. Als weiteres Ausschlusskriterium für den Einsatz einzelner Komponenten kam die Eingrenzung des Beschaffungsmarktes hinzu. Im Rahmen des Projektes durfte ausschließlich am deutschen Markt bestellt werden, wodurch die Komponenten ausländischer Märkte ausschieden. Unter Berücksichtigung aller Anforderungen und Einschränkungen wurden die folgend aufgelisteten Komponenten zur Realisierung des Zielsystems bestellt:

Dieser vollwertige Einplatinencomputer ist mit den gebräuchlichsten Schnittstellen versehen und kann direkt ohne Erweiterung autark betrieben werden. Im Vergleich ist dieser mit insgesamt vier Cortex-A7-Kernen mit bis zu 900 MHz am leistungsstärksten. Ein weiteres Leistungsplus verschafft der vergrößerte Arbeitsspeicher von 1024 MB.

Technische Spezifikation

CPU:	Quad-Core 900 MHz
RAM:	1 GB LPDDR2 SDRAM
Netzwerk:	10/100 Mbits
Analoge Eingänge:	nein
Digitale Eingänge:	40 Pin GPIO / SPI / UART / I^2C
USB:	4 Anschlüsse
Mikrofonanschluss:	nein
Spannungsversorgung:	5 V
Preis:	38,50 €

ADS1015 12-Bit ADC - 4-Kanal Spannungsmessung

- Misst Spannung 0 – 5 V
- 4 analoge Eingänge
- Schnittstelle I^2C Bus (100 Khz/ 400 Khz/ 3,4 MHz)
- Auflösung 12 Bit
- bis zu 3300 sps
- Preis 9,95 €
- keine Strommessung

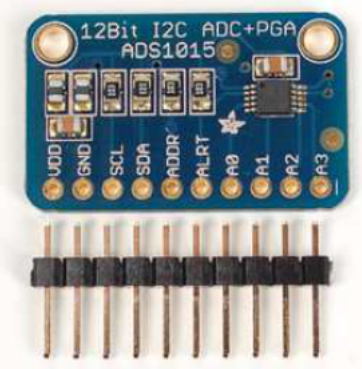


Abbildung 6.6.: Adafruit ADS1015 12-Bit ADC - 4-Kanal [Adab]

Adafruit INA169 Analog DC Stromstärkemessung

- Misst Ströme bis +5A
- 1 analoger Ausgang
- Messung High Side bei bis zu +60VDC
- Ausgangsspannung 1V pro Ampere (max. 5 V)
- Preis 10,45 €

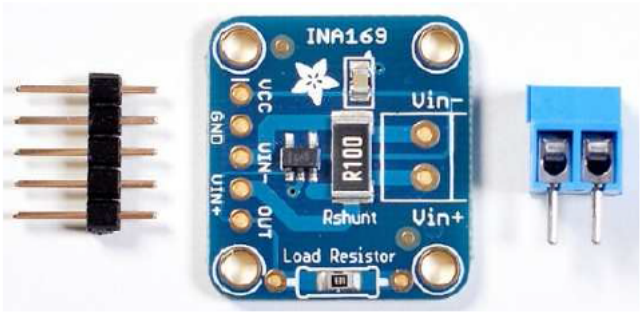


Abbildung 6.7.: Adafruit ADS1015 12-Bit ADC - 4-Kanal [Adac]

Cirrus Logic Audio für Raspberry PI Schallwellen

Diese Erweiterungskarte für Raspberry PIs ermöglicht die Ein- und Ausgabe von Audiodaten, um die geforderten Schallmessungen durchführen zu können.

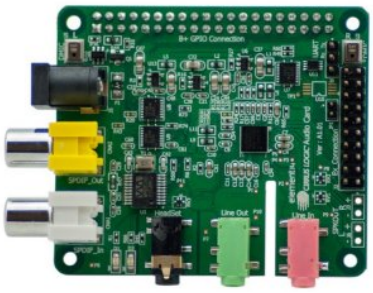


Abbildung 6.8.: Adafruit ADS1015 12-Bit ADC - 4-Kanal [Cira]

Eingänge:

- 3,5 mm Klinke (4-polig) - Mikrofon z.B. für Spiele oder VoIP-Anwendungen
- 2 x On-Board-MEMS-Mikrofon
- 3,5 mm Klinke Stereo-Line-Eingang zum Anschluss von z.B. digitalen Audio-Player
- SPDIF Coaxial Digital (nicht optisch)

Ausgänge:

- 3,5 mm Klinke (4-polig) für Headset bzw. Kopfhörer
- 3,5-mm-Buchse Stereo-Line-Ausgang für Geräte wie z.B. externer Stereo-Verstärker
- SPDIF Coaxial Digital

zusätzliche Funktionen:

- On-Board-Klasse-D-Leistungsverstärker für externe Lautsprecher
- Pin-Header für zusätzliche externe Anbindungen
- Raspberry PI Hat Formfaktor

Creative Soundblaster Play Soundkarte extern Schallwellen

Diese Soundkarte wurde zur Erzeugung von Testsignalen angeschafft.

- Linuxgeeignet
- Signal-Rausch-Verhältnis: 90 dB
- unterstützt 16 Bit/48 kHz
- Preis 16,16 €

Abbildung 6.9.: Creative Soundblaster Play [Sou]

6.4.2. Sensormontage

UG

Das Projekt erweitert den Funktionsumfang des SESALabs und ist für die zukünftige Nutzung und Analyse der Simulationsdaten von großer Bedeutung. Aus diesem Grund sollten alle Komponenten für die Realisierung des Projektes in das SESALab eingebaut werden.

Für den Einbau stand ein Montagefeld in der Größe von 60 cm x 18 cm x 30 cm (B,H,T) zur Verfügung, welches zur Montage der Endgeräte mit einer Hutschiene ausgestattet ist. Für die Spannungsversorgung der Geräte steht wahlweise 230 V Wechselspannung oder eine Gleichspannung von 0V – 24V zur Verfügung. Der Anschluss an das Netzwerk des SESALabs wird über RJ45 Netzwerkdosen ermöglicht, welche zum Patchfeld des Switches führen. Das Feld ist ringsum von einem Kabelkanal umgeben, in dem die Verkabelung untergebracht wird.

Da das SESALab als zukunftsträchtige Laborumgebung immer wieder im öffentlichen Interesse steht und für Führungen und Demonstrationen häufig besucht wird, ist eine ansehnliche und sichere Montage unerlässlich. Die Sicherheit beschränkt sich primär auf den Berührungsschutz und die sicheren Spannungsversorgung. Aus diesem Grund sollen die Raspberry PIs und die Sensoren mittels Hutschienengehäuse sicher verbaut werden. Das Gehäuse muss aber so beschaffen sein, dass die Schnittstellen für zukünftige Erweiterungen erreichbar sind.

Insgesamt ergaben sich für die Montage der Sensorhardware in das SESALab folgende Anforderungen:

- das Gehäuse soll nicht transparent sein
- die Hardware soll vor Berührungen geschützt werden
- die Gehäuse sollen an Hutschienen befestigt werden können
- das Netzteil soll die Hardware mit ausreichend Leistung versorgen und über 230 V Wechselspannung oder 0V - 24V Gleichspannung betrieben werden können
- die Schnittstellen des Raspberry PIs müssen von außen zugänglich sein
- das Gehäuse soll ansehnlich sein

Da die Audiokarte direkt auf dem Raspberry PIs mittels Erweiterungsstecker angeschlossen wird, gestaltete sich die Suche nach einer passenden Lösung sehr schwierig. Keines der verfügbaren Gehäuse ermöglichte die direkte Montage mit gesteckter Audiokarte. Entweder musste das Gehäuse vergrößert oder die Audiokarte mittels Steckverbinder angeschlossen werden. Alternativ war die Erstellung eines Gehäuses mit einem 3D Drucker in der erforderlichen Größe möglich. Das Problem wurde mit den Auftraggebern besprochen und verschiedene Lösungsvorschläge unterbreitet (Siehe Anhang Gehäuseauswahl D).
Nach sorgfältiger Überprüfung sind vom Auftraggeber die gewünschten Gehäuse und Materialien beschafft worden. Das ausschlaggebendste Kriterium für die Auswahl war der integrierte Spannungsregler, welcher bei einer Spannungsquelle von 9 – 35V DC eine konstante Versorgungsspannung von 5V DC / 1A DC ermöglicht. Zusätzlich beinhaltet die Platine ein Entwicklungsboard mit frei belegbaren Ein- und Ausgangsklemmen. Der Bausatz Raspibox Open Plus besteht aus den folgenden Einzelkomponenten und ist unmontiert.

- vorgefrästes Hutschienengehäuse mit 6 TE (Made in Germany)
- passendes Experimentierboard zum Aufstecken der RasPi
- 40 polige Buchsenleiste
- 10 Schraubklemmen im Rastermaß 5 mm
- Lochraster-Bereich für bedrahtete Bauteile
- Footprints für SSOP und SOIC Schaltkreise
- Footprints für SMD-Widerstände und SOT23 Bauteile
- Alle Anschlüsse des Raspberry PIs sind auf eine beschriftete Anschlussleiste geführt
- Layout für 5V/1A Schaltregler auf der Platinenrückseite
- geeignet für die Raspberry PI Modelle A+, B+, 2 B und 3 B

Zuerst erfolgte für beide Bausätze die Montage des Spannungsreglers, der 40 poligen Steckleiste zur Verbindung des Raspberry PIs sowie der Ein- und Ausgangsklemmen.
Anschließend ist die sensorabhängige Montage des Spannungs- und Audiomoduls umgesetzt worden.

Abbildung 6.10.: Übersicht Bausatz

Spannungsmodul

Zu Fertigstellung des Spannungsmoduls musste der AD-Wandler in das Gehäuse montiert und mit dem Raspberry PI sowie den Eingangsklemmen zum Anschluss der analogen Signale verbunden werden. Der Anschluss erfolgte mit Dupont Verbindungskabel, welche mit der einen Seite an die entsprechenden GPIO-Pins des Raspberry PIs und den Eingangsklemmen gelötet wurden. Die Stecker der anderen Seite sind mit den entsprechenden Verbindungspins des AD-Wandlers verbunden. Das Anlöten konnte durch die Vorteile das Experimentierboards erheblich vereinfacht erfolgen, da dieses alle GPIO-Verbindungen des Raspberry PIs und die Eingangsklemmen auf die Lochrasterplatine führt. Nachdem die Lötverbindungen hergestellt und der AD-Wandler ordnungsgemäß angeschlossen war, erfolgte die Befestigung auf dem Experimentierboard. Zum

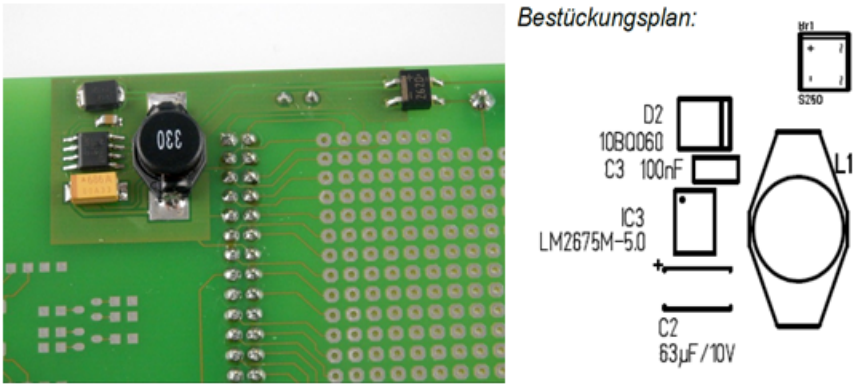


Abbildung 6.11.: Spannungsregler

Schluss erfolgte die Funktionsprüfung mit einem Spannungsgenerator und einem Digitalmultimeter.

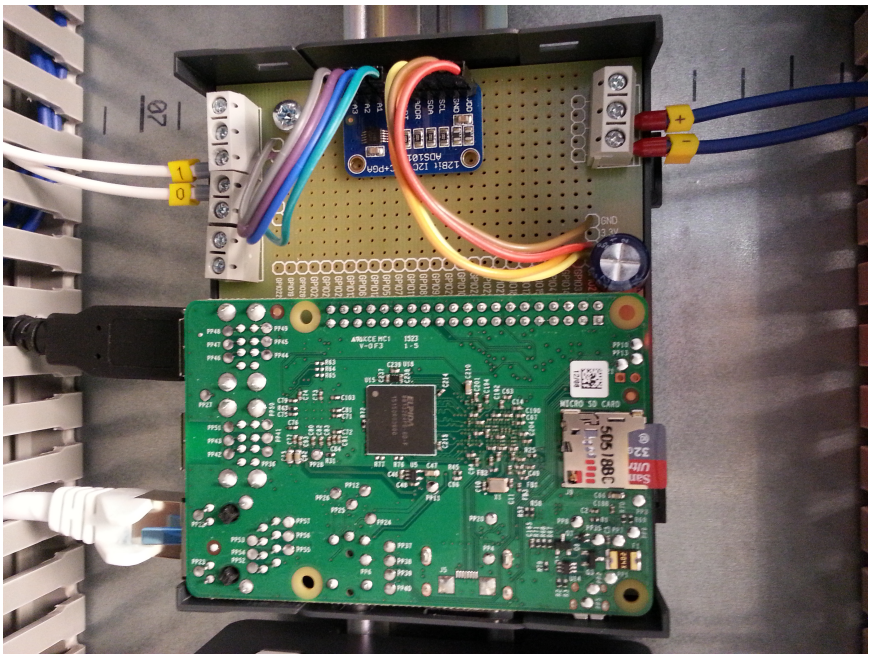


Abbildung 6.12.: Spannungssensor

Montage Audiomodul

Die Fertigstellung des Audiomoduls war etwas umfangreicher und erforderte noch einige Umbaumaßnahmen. Das größte Problem bestand in der Unterbringung und dem Anschluss der Audiokarte am Raspberry PI. Zunächst musste das Platzproblem so gelöst werden, dass das Audiomodul, unter Beachtung der Anschlussmöglichkeit von außen, im Deckel montiert wurde. Für den Anschluss der Audio Ein- und Ausgänge erfolgte die Einfräsung entsprechender Öffnungen. Zur sicheren Befestigung des Moduls wurden Gewindehülsen aus Kunststoff befestigt, über welche die Audiokarte anschraubbar ist.

Damit das Gehäuse auch problemlos wieder geöffnet werden kann, ist die Verbindung zum Audiomodul über eine Steckverbindung realisiert. Dazu wurde zunächst eine 40 polige Steckleiste auf das Experimentierboard gelötet und mit Verbindungskabel an die entsprechenden GPIO-Pins des Raspberry PIs verbunden. Aufgrund des gedrehten Anschlusskabels erfolgte eine gerade Verbindung der Steckleiste. Das sehr begrenzte Platzangebote auf der Unterseite erforderte höchster Präzision bei der Verbindung.
Ein weiteres Problem bestand bei der Realisierung der Steckverbindung zum Audiomodul. Der Abstand zwischen dem Stecker und der Platine des Raspberry PIs ist so gering, dass ein Winkelstecker erforderlich war. Leider gab es, aufgrund der zu großen Abstände der Stiftreihen, keine passenden Winkelstiftleisten zu kaufen. Zur Lösung des Problems konnte entweder die Steckleiste des Audiomoduls gekürzt oder die Winkelleiste selber kompakter gefertigt werden. Um die Audiokarte nicht zu beschädigen und zukünftig auch für weitere Zwecke einsetzten zu können, wurde die Winkelleiste aus einer geraden Stiftleiste geformt. Dazu erfolgte die Formung der Winkel einzeln, in einem möglichst engen Abstand auf ca. 120°. Damit sich die Pins nicht berühren

Abbildung 6.13.: Audiosensor

können, erfolgte eine Isolierung zueinander.

Nachdem die Steckverbinder hergestellt waren, wurde das Flachbandkabel so dimensioniert (20 cm), dass dieses aufgrund des sehr geringen Platzangebotes in das Gehäuse passt und trotzdem noch ein Öffnen des Gehäuses ermöglicht.
Nach dem Schließen des Gehäuses erfolgte ein Funktionstest mit einem Tongenerator.

6.4.3. Hardwarekonzept Serverumgebung

UG

Für die Realisierung des Zielsystems ist die richtige Auswahl und Dimensionierung der eingesetzten Hardwarekomponenten unerlässlich. Falsch ausgewählte oder unterdimensionierte Komponenten führen zu Nichterfüllung der Anforderungen und können im schlimmsten Fall zum Scheitern des Projektes führen. Aus diesem Grund ist die Auswahl und der Einsatz der Hardwarekomponenten im Rahmen des Projektes ausgiebig geplant worden. Neben den Anforderungen (siehe Abschnitt 3.2 und 3.3) der Auftraggeber, bilden die zur Realisierung des Zielsystems verwendeten Softwarekomponenten (siehe Abschnitt Softwaredesign) die wichtigste Grundlage für das Konzept. Deshalb werden zunächst die, für die Auswahl, relevanten Anforderungen und ein grober Entwurf des Hardwarekonzeptes dargestellt. Darauf aufbauend erfolgt die detaillierte Erläuterung der eingesetzten Komponenten mit Angabe der wesentlichen Merkmale und des Verwendungsgrundes. Die richtige Dimensionierung des Backends ist für die Leistungsfähigkeit und Funktionalität des Zielsystems von größter Bedeutung und bedarf daher einer gesonderten Betrachtung. Diese Überlegungen zur Ermittlung des Bedarfs an Hardwarekomponenten für die Verteilung der Systemkomponenten, bilden die Grundlage der Testumgebung und werden in einem Serverkonzept festgehalten. Zum Abschluss wird eine Hardwareübersicht aller verwendeten Komponenten gegeben, welche den Aufbau der Testumgebung darstellt.

Anforderungen an die Hardware

Um die Hardwareinfrastruktur anforderungskonform und zur Zufriedenheit der Auftraggeber erstellen zu können, werden zunächst die dafür relevanten Anforderungen aus der Spezifikation zusammengefasst.

- Das System soll die eingehenden Daten verlustfrei verarbeiten können.
- Das System soll die Möglichkeit bieten, Systemerweiterungen (Hardware, Software) im laufenden Betrieb vornehmen zu können.
- Die Technologie kann grundsätzlich frei gewählt werden, außer Vorgaben oder vorhandene Systeme widersprechen dem.
- Die Technologie soll unter Berücksichtigung der Ausbaufähigkeit und des langfristigen Einsatzes ausgewählt werden.
- Die Speicherkapazität für die Langzeitspeicherung steht aufgrund der immer weiter sinkenden Kosten ausreichend zur Verfügung.
- Das Projektteam muss den Ausfall einzelner Systemkomponenten wie DB-Ausfälle oder sonstige Katastrophen nicht berücksichtigen.
- Das System soll nach Möglichkeit modular aufgebaut werden, um System- und Kapazitätserweiterungen zur Laufzeit zu ermöglichen.
- Das System soll die Möglichkeit bieten Daten, von bis zu 1000 Sensoren, zu verarbeiten.
- Das System soll fähig sein, hochfrequente Daten bis zu 30 kHz, zu verarbeiten.
- Kapazität- und Systemerweiterungen sollen im laufenden Betrieb möglich sein.
- Das System soll auf Anfragen, die einen Zeitraum von maximal zwei Jahren umfassen, innerhalb einer Stunde das Ergebnis liefern können.

Neben diesen Anforderungen liefern die bisherigen Erfahrungen aus den Testreihen, bei der Auswahl und Überprüfung der Zweckeignung der einzelnen Komponenten, wichtige Erkenntnisse für die Konzeption der Hardwareumgebung.

Odysseus liefert alle eingehenden Daten an das Zielsystem und sollte auf einem einzelnen Server betrieben werden. Mit Zunahme der Datenquellen ist eine Verteilung von Odysseus nötig. Der verteilte Betrieb ist derzeit nicht umsetzbar, da die Implementierung noch nicht abgeschlossen ist. Für die Testumgebung, in der nur wenige Datenquellen genutzt werden, reicht eine Odysseusinstanz aus.

Empfehlung: mit Zunahme der Anzahl an Datenquellen und der Übertragungsfrequenz sollte die Performanz der Ressourcen angepasst werden.

Kafka schreibt die von Odysseus übermittelten Daten sequentiell auf die Festplatte und geht dabei nicht platzsparend vor. Die Daten sind vorläufig gesichert, bis die Speichermedien voll sind. Um Datenverlust zu vermeiden, kann die Größe des Speichers begrenzt werden, wodurch ältere Aufzeichnungen entfernt werden. Das Einlesen für die weitere Übermittlung an Druid erfolgt ebenfalls sequenziell, wodurch die Lesegeschwindigkeit ebenfalls wichtig ist. Die Arbeitsspeicher und CPU-Anforderungen sind nicht so hoch.

Empfehlung: schnelle Festplatten optimal gleichzeitig Lese- und Schreibvorgänge.

Druid-Realtime Dieser Knoten ist für die segmentweise Übertragung der Eingangsdaten zuständig. Die Herstellerempfehlungen für die Segmentgröße liegen bei 5 MB. Mit Zunahme der Segmentgröße und der Übertragungsfrequenz steigt der Bedarf an Arbeitsspeicher. In der Testumgebung führte eine Datenübertragungsrate von 3 kHz in kürzerer Zeit zu einer `OutOfMemoryException`. Die Erhöhung des Arbeitsspeichers hat das Problem gelöst. Die geeignete Menge des Arbeitsspeichers ist immer von der Eingangsdatenmenge abhängig und im Einzelfall zu verifizieren. Aufgrund der Zwischenspeicherung zur Segmentbildung (Indexierung, Komprimierung) auf der Festplatte, sind schnelle Schreib- und Lesezugriffe nötig.

Empfehlungen: Bei höherer Eingangsfrequenz und großen Segmentgrößen kommt es bei 25 GB RAM schnell zum Systemausfall → ausreichend RAM oder kleine Segmentgrößen. Festplatte → schnelle Schreib- u. Lesezeiten wodurch eine SSD empfehlenswert geeignet ist.

Druid-Broker ist für die Anfragenbearbeitung und Ergebniszusammenführung zuständig. Dieser ist für die Langzeitdatenanalysen wichtig, damit die Ergebnisse zeitnahe bereitgestellt werden. Für die reine Datenspeicherung kann diese Komponente vernachlässigt werden. Aber die Analyse und Zusammenführung ist sehr arbeitsspeicherintensiv.

Empfehlungen: mit Zunahme der Anfragezeiträume und Komplexität der Aggregationen wird zunehmend mehr RAM benötigt.

Druid-Historical lädt auf Anweisung des Coordinators Segmente aus dem Deep Storage und reagiert auf Anfragen. Grundsätzlich werden die benötigten Datensegmente auf die Festplatte geschrieben, um für die weitere Anfrageverarbeitung zur Verfügung zu stehen.

Empfehlungen: braucht ausreichend und schnellen Festplattenspeicher um Anfragen effizient verarbeiten zu können (empfehlenswert SSD). Der Bedarf an Arbeitsspeicher verhält sich ähnlich. Viele und schnelle Anfragen → mehr Festplattengröße und RAM.

Druid-Coordinator regelt die Verteilung der Langzeitdaten aus Cassandra auf die Historical Knoten, damit diese für Anfragen zur Verfügung stehen. Prüft dazu welche Segmente vorhanden sind und noch verteilt werden müssen. Dazu wird in Zookeeper eine Aufgabenliste erstellt, welche von den Historical Knoten abgearbeitet werden. Diese Aufgabe erfordert weniger Performanz.

Empfehlung: der Einsatz von SSD und übermäßige Ressourcen sind nicht erforderlich.

Cassandra dient der Langzeitspeicherung und erfordert große Speicherkapazität, sowie schnelle Speichermedien wie SSD. Die Verwendung von ZFS zur erhöhten Ausfallsicherheit ist empfehlenswert. Der Hersteller empfiehlt bei der Verwendung von Festplattenlaufwerken die Trennung der Commitlog- und Datenverzeichnisse auf separate Speichermedien.

Empfehlung: große SSD Speicher mit ZFS.

Zookeeper Verzeichnisdienst zur Organisation der Zusammenarbeit der einzelnen Druid- und Kafkaknoten. Im Testumfeld wurden die Ressourcen nicht ansatzweise ausgelastet.

Empfehlung: Die Leistungsfähigkeit der Ressourcen kann eher gering gehalten werden.

MySQL speichert die LAOTSE-Konfiguration (Modell) und den Druid Storage (Liste der existierenden Datensegmente). Im Testumfeld wurden die Ressourcen nicht ansatzweise ausgelastet.

Empfehlung: Die Leistungsfähigkeit der Ressourcen kann eher gering gehalten werden.

Überblick Hardwarekonzept

Unter Berücksichtigung der für die Realisierung des Zielsystems erforderlichen Komponenten ergibt sich eine erste grobe Übersicht des Hardwarekonzeptes.

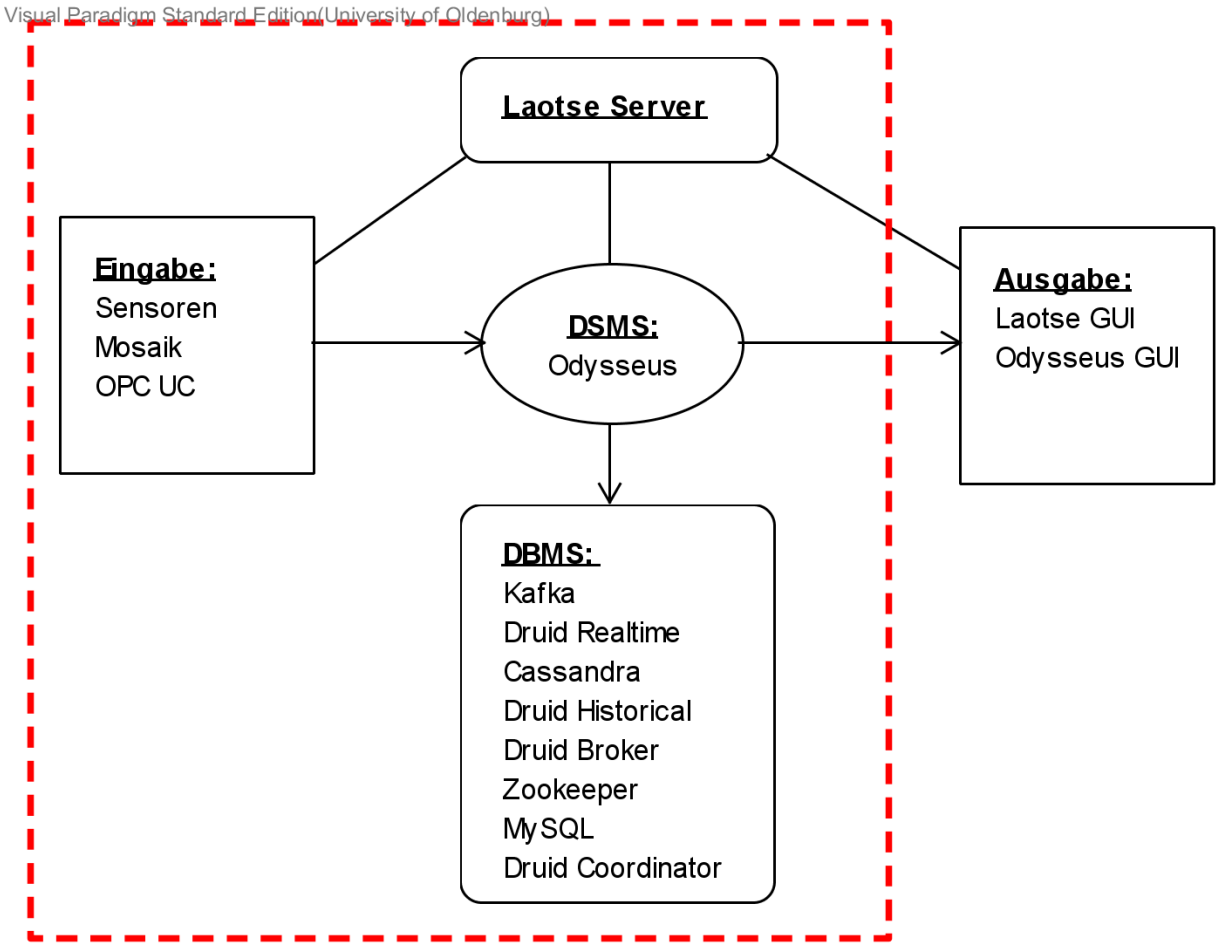


Abbildung 6.14.: Übersicht Hardwarekonzept

Dieses ist in vier Hauptbereiche unterteilt. Die Eingabe umfasst neben den bereits ausgewählten Sensoren (siehe Anhang Sensorauswahl und Einplatinencomputer), mosaik und OPC UA, welche

mit entsprechender Serverkapazität zu berücksichtigen sind. Das DSMS Odysseus wird nur in einer Instanz betrieben. Der dritte Bereich umfasst den komplexesten Teil, welcher mit der Vielzahl an Einzelkomponenten für die verlustfreie persistente Speicherung der Daten zuständig ist. Das DBMS wird die meiste Hardwarekapazität beanspruchen und erfordert eine durchdachte Planung der eingesetzten Hardware, Kapazitäten und Verteilung der Komponenten. Im letzten Bereich wird der Laotse Server realisiert, welcher für die Steuerung der Mosaiksimulationen, Odysseus und des Laotse Clients zuständig ist. Darüber hinaus verwaltet dieser das Modell des Zielsystems. Die Betrachtung und Beschaffung der Ausgabehardware ist nicht erforderlich, da das Zielsystem endgeräteunabhängig für verschiedene Plattformen (Linux und Windows) bereitgestellt wird. Die Ausgabe kann auf jedem Endgerät erfolgen, welches über Java 8, des Laotse Clients sowie einer Internetverbindung verfügt.

Auswahl Hardwarekomponenten

Für die richtige, zweckdienliche und kostenoptimale Auswahl der Hardware (Technologie und Kapazität) sind zunächst die wesentlichen Aufgaben und Merkmale der zu betreibenden Systemkomponenten zu ermitteln. Diese Erkenntnisse ermöglichen Empfehlungen für die Verteilung der Systemkomponenten und bilden eine gute Grundlage zur Auswahl geeigneter Technologien und Kapazitäten. Ergänzend zu den im vorherigen Abschnitt 6.4.3 beschriebenen DBMS-Komponenten und dem DSMS Odysseus, werden folgend die übrigen im Hardwarekonzept zu berücksichtigenden Komponenten betrachtet.

Sensoren liefern die Messwerte mittels Datenströmen an Odysseus. Gemäß der Anforderungen werden Temperaturen (Abtastrate 1 Hz), Spannungswerte (Abtastrate bis 1 kHz) und Schallwellen bis (30 kHz) verarbeitet und persistent gespeichert. Die Auswahl der Komponenten ist bereits erfolgt (siehe Abschnitt 6.4.1) und sind daher im Hardwarekonzept gegeben.

Mosaik diese Komponente wird zur Simulation der mosaik Szenarien und der Datenübermittlung an Odysseus benötigt. Die Leistungsfähigkeit der verwendeten Ressourcen hängt maßgeblich vom Szenario ab. Das bereitgestellte Testszenario ist aufgrund der geringeren Größe, der Anzahl an Messwerten, sowie der Berechnungsschritte nicht besonders Ressourcenintensiv.

OPC UA für die Datenübertragung dient ein bestehender Testserver, von dem die Testdaten abgerufen wurden. Dieser ist vorgegeben und braucht daher nicht mehr eingerichtet werden. Die Leistungsanforderungen an den Testserver sind eher gering, da dieser lediglich vorher generierte Daten bereitstellt.

Laotse Server ist die Kernkomponente des Zielsystems und mittels der Entgegennahme von Anfragen und das Senden von Statusmeldungen für deren Steuerung zuständig. Er empfängt die Nachrichten des Clients und Benachrichtigt die entsprechenden Komponenten mit Anfragen oder Nachrichten bzgl. mosaik, Odysseus und der Konfiguration. Darüber hinaus wird das Starten einer mosaik Simulation und die Übermittlung der Statusmeldungen ermöglicht. Zur Wahrung der Hauptfunktionalität als `WebSocketServer` halten sich die Leistungsanforderungen in Grenzen.

Mindestanforderungen, welche die Lauffähigkeit des Systems garantieren, sind für komplexe Systeme dieser Art nicht nur schwer ermittelbar, sondern i.d.R. auch nicht allgemein definierbar.

Anforderungen hängen immer vom Einsatzzweck und Systemumfeld ab. Welche im Einzelfall zu betrachten und durch ausgiebige Tests zu bestimmen sind. Die Auswahl und Dimensionierung sollte immer unter Berücksichtigung der Aufgaben und Merkmale der Systemkomponenten erfolgen.
Nachdem die für die sach- und fachgerechte Bewertung benötigten Informationen ermittelt sind, kann der Auswahlprozess beginnen. Eine Evaluierung der Server- und Netzwerkinfrastruktur gehörte nicht zum Teil des Projektes, da die vorhandenen Komponenten des SESA-Labs aus wirtschaftlichen Gründen genutzt werden sollten. Aus diesem Grund entfiel die Evaluierung und Auswahl dieser Komponenten.
Stattdessen ist die Nutzung der bereitgestellten Ressourcen unter Berücksichtigung der Aufgaben und Merkmale der Systemkomponenten optimal geplant und in einem Serverkonzept übersichtlich dargestellt worden. Zunächst erfolgt die Auflistung der bereitgestellten Ressourcen mit der Angabe der wesentlichen Leistungsmerkmale.
Der Projektgruppe standen folgende Server und Netzwerkinfrastruktur zur Verfügung, die nach Absprache und Verfügbarkeit frei eingerichtet und verwendet werden konnte.

Physikalische Server Die Server bestanden neben zwei HP ProLiant SE316M1 aus acht ProLiant BL460C G6 Einheiten, welche in einem BladeCenter zu jeweils 2 Cluster verbunden waren. Die folgende Übersicht stellt die wesentlichen Leistungsmerkmale der bereitgestellten Umgebung zusammen.

Name	Typ	CPU1 und CPU 2	RAM	HDD
SRVELAB-P1	ProLiant SE316M1	Intel Xeon E5520	16384 MB	2 x 146 GB SAS Systempartition (RAID 1) + 2 x 120 GB SSD ZFS Mirror für Daten
SRVELAB-P2	ProLiant SE316M1	Intel Xeon E5520	16384 MB	2 x 120 GB SSD Systempartition (RAID 1) + 2 x 146 GB SAS ZFS Mirror für Daten
BLSDATA-50-90	ProLiant BL460c G6	Intel Xeon E5530 @ 2.40GHz (4 Cores)	32768 MB	2 x 146 GB SAS 6G DP 10K (RAID 1)
BLSDATA-50-91	ProLiant BL460c G6	Intel Xeon E5530 @ 2.40GHz (4 Cores)	20480 MB	2 x 146 GB SAS 6G DP 10K (RAID 1)
BLSDATA-50-92	ProLiant BL460c G6	Intel Xeon E5530 @ 2.40GHz (4 Cores)	30720 MB	2 x 146 GB SAS 6G DP 10K (RAID 1)
BLSDATA-50-93	ProLiant BL460c G6	Intel Xeon E5530 @ 2.40GHz (4 Cores)	20480 MB	2 x 146 GB SAS 6G DP 10K (RAID 1)
BLSDATA-50-94	ProLiant BL460c G6	Intel Xeon E5530 @ 2.40GHz (4 Cores)	18432 MB	2 x 146 GB SAS 6G DP 10K (RAID 1)
BLSDATA-50-95	ProLiant BL460c G6	Intel Xeon E5530 @ 2.40GHz (4 Cores)	16384 MB	2 x 146 GB SAS 6G DP 10K (RAID 1)
BLSDATA-50-96	ProLiant BL460c G6	Intel Xeon E5530 @ 2.40GHz (4 Cores)	131072 MB	2 x 146 GB SAS 6G DP 10K (RAID 1)
BLSDATA-50-97	ProLiant BL460c G6	Intel Xeon E5530 @ 2.40GHz (4 Cores)	163840 MB	2 x 120 GB SATA SSD (RAID 1)

Abbildung 6.15.: Übersicht physikalische Server

Virtuelle Server Für den parallelen Betrieb verschiedener Systeme, der einfachen Verwaltung und Portierbarkeit wurden virtuelle Maschinen bereitgestellt. Die Bereitstellung der folgend aufgelisteten virtuellen Server erfolgte mittels VMware auf dem VM-Host-10 im SESALab.

- VMSDATA-10-130
- VMSDATA-10-132
- VMSDATA-10-133
- VMSDATA-10-134
- VMSDATA-10-135
- VMSDATA-10-136
- VMSDATA-10-139

Netzwerkinfrastruktur Für die Anbindung der Geräte stand ein administrierbarer Netzwerkswitch von Enterasys zur Verfügung, an dem die Geräte über Gigabit Ethernet Ports an das Netzwerk des SESA-Labs anschließbar sind. Die Verbindung zum Netzwerk des SESA-Labs gestattet eine Kommunikation untereinander und das Erreichen von Außerhalb über eine sichere VPN-Verbindung.

- 48 x 10/100/1000 BaseTX RJ45 Ports
- administrierbar über Konsole oder Webinterface
- Netzwerkanalysefunktion

Internetanbindung Der Zugriff auf die Testumgebung von außerhalb erfolgt über eine gesicherte VPN-Verbindung. Diese baut nach erfolgreicher Authentifizierung eine gesicherte Verbindung über den Server: *alabrt.offis.uni-oldenburg.de* zum internen Netzwerk auf. Dadurch können die Administration und der Zugriff der Clients über das Internet erfolgen.

Die Übrigen Komponenten der Testumgebung wurden im Rahmen des Projektes ausgiebig evaluiert und ausgewählt oder können wie die Clients beliebig verwendet werden.

Clients Das Zielsystem wurde für eine endgeräteunabhängige Nutzung entwickelt und wird für die folgenden Plattformen angeboten:

- Linux 32 bit / 64 bit
- Windows 32 bit / 64 bit

Für die Nutzung des Zielsystems ist lediglich JAVA 8 und eine Internetverbindung erforderlich.

Einplatinencomputer/Sensoren Um die Messwerte der verschiedenen Sensoren an das Zielsystem übertragen zu können, ist der Einsatz von Einplatinencomputern erforderlich. Gemäß Hardwareauswahl (siehe Anhang Hardwareauswahl) werden dazu Raspberry PIs verwendet, an denen die Sensoren angebunden sind. Entsprechend der Anforderungen ist die Anbindung eines Temperatursensors, eines AD-Wandlers (Spannungsmessung) und einer Audiokarte (Schallwellen) vorgenommen zu realisieren (siehe Abschnitt 6.2.1 Hardwareauswahl). Dafür ist die Verwendung der folgenden Geräte erforderlich:

- 2 x Raspberry PI 2 B Leistungen (Audio und Spannung)

CPU:	Quad-Core 900 MHz
RAM:	1 GB LPDDR2 SDRAM
Netzwerk:	10/100 Mbits

- 1 x Raspberry PI B Leistungen (Temperatur)

CPU:	Dual-Core 700 MHz
RAM:	512 MB SDRAM
Netzwerk:	10/100 Mbits

- 1 x Temperatursensor: DS18B20
- 1 x Spannungsmessung Adafruit ADS1015 12-Bit ADC
- 1 x Schallwellen Cirrus Logic Audio für Raspberry PI

Alle Komponenten sind in das SESALab verbaut und sind mittels VPN über das Internet erreichbar. Die folgende Abbildung gibt einen Überblick über den Aufbau der verwendeten Hardwarekomponenten.

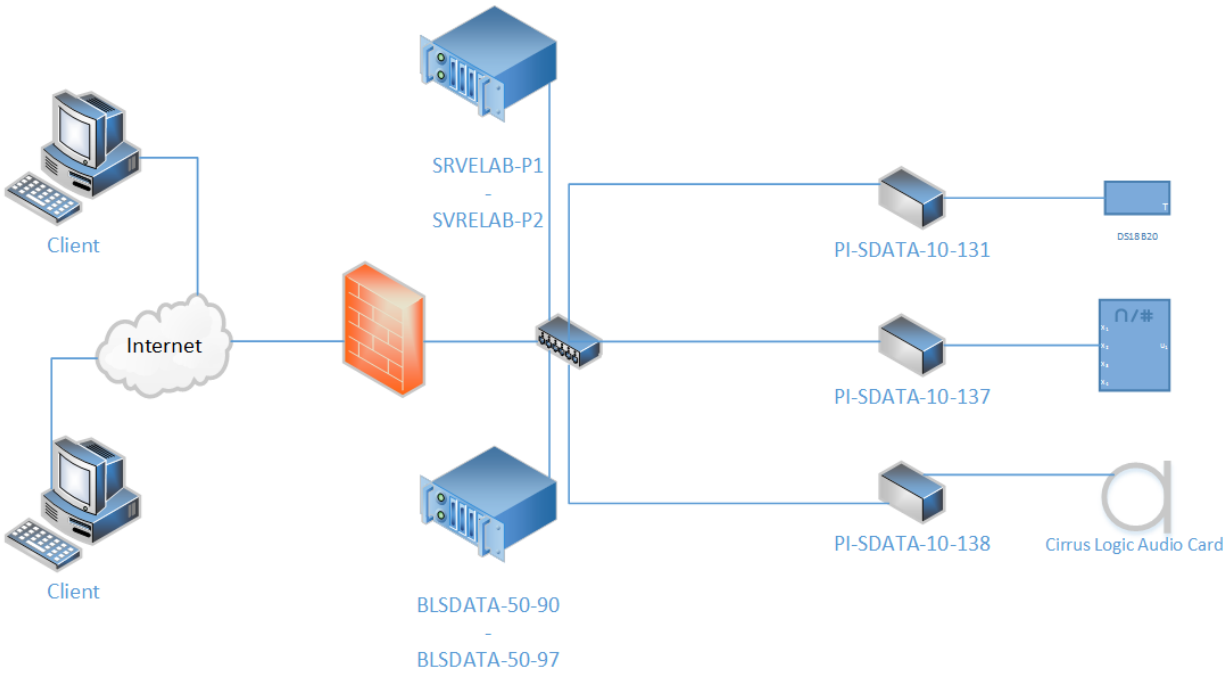


Abbildung 6.16.: Übersicht Testumgebung

Abschließend erfolgt die Darstellung des Serverkonzeptes, welches die Nutzung der bereitgestellten Ressourcen für die Realisierung der Systemkomponenten beschreibt. Die Komponenten sind unter Berücksichtigung ihrer Aufgaben und Leistungsmerkmale auf die Ressourcen verteilt worden und bilden die Ausgangssituation der Testumgebung.

6.5. LAOTSEDB Aufbau

LS

In der LAOTSEDB werden alle vorhandenen Metadaten zentral gespeichert. Da das Ausmaß der Daten in diesem Fall eher überschaubar ist, ist die Geschwindigkeit der genutzten Datenbank nicht das ausschlaggebende Auswahlkriterium für diese Datenbank. Aus diesem Grund wurde für die Implementierung eine einfache MySQL-Datenbank ausgewählt. Zur Aufstellung eines geeigneten relationalen Datenbankschemas wurde mit der Konstruktion eines ERD (Entity Relationship Diagram) begonnen, welches in Abbildung 6.18 zu sehen ist.

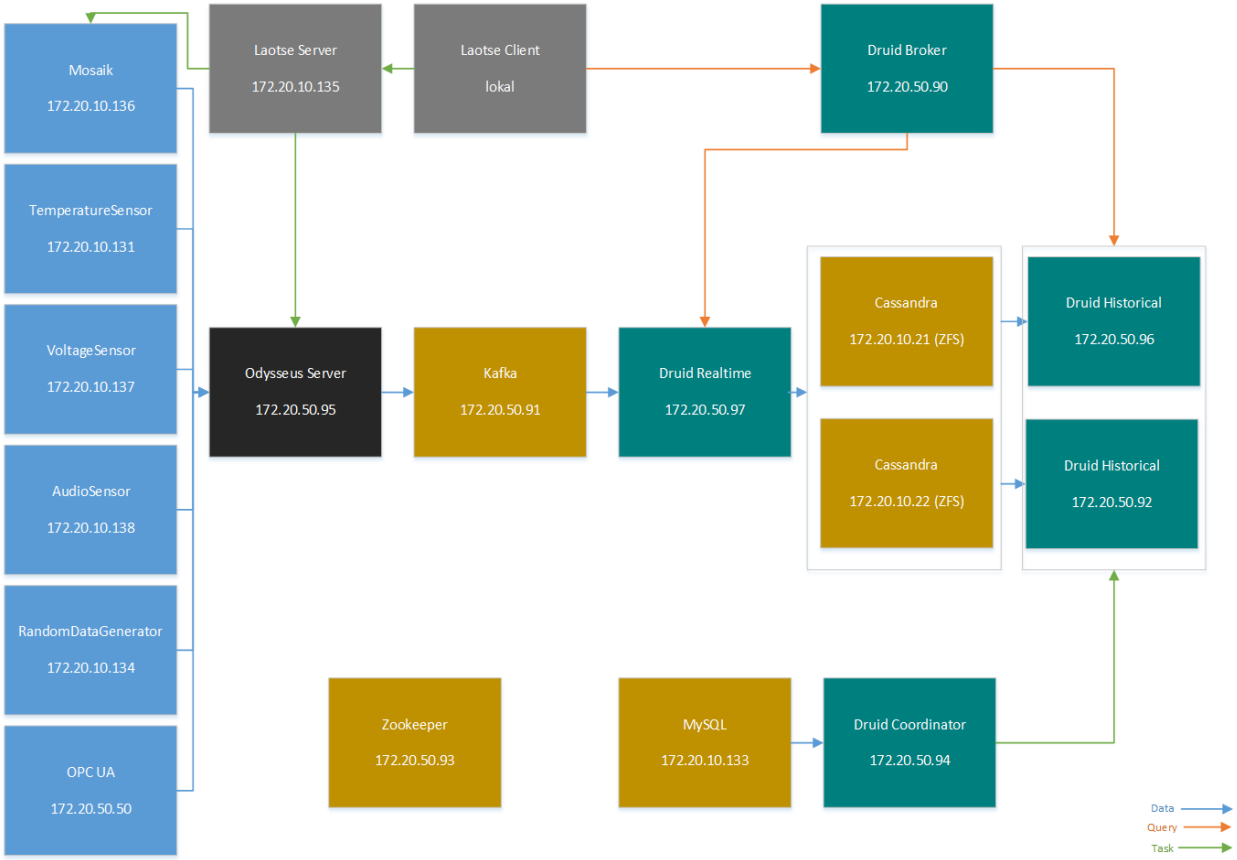


Abbildung 6.17.: Übersicht Serverkonzept

6.5.1. Entitäten

In diesem Abschnitt werden die Entitäten des ERD mit ihren Attributen genauer beschrieben. Allgemein wird jede Entität durch ihre eindeutig vergebene ID identifiziert.

Sensor Ein Sensor, der dem System von einem Nutzer hinzugefügt wurde. Er hat eine Bezeichnung (name) sowie eine geografische Position, die mittels Längen- und Breitengrad eindeutig festgelegt ist. Zusätzlich wird eine Modellnummer für die Identifikation der physikalischen Eigenschaften des Sensors angegeben und das Datum der ersten Inbetriebnahme (start) gespeichert. Optional kann zu jedem Sensor ein spezifischer Kommentar verfasst werden.

SensorType Jeder Sensor ist einem bestimmten Sensortyp zugeordnet, z.B. Temperatursensor, Mikrofon, Windstärkesensor etc.; dieser gibt Angaben über das Format und die Struktur der Sensormesswerte, indem er jedem seiner Felder direkt mit einem eindeutigen Format beschreibt: Z.B. SensorField: Temperatur, Format: °C, SensorField: Frequenz, Format: Hz. Ein Sensor würde diese Felder mit einem konkreten Wert instanziieren (Bsp. value: 30).

Analysis Auf den Sensoren können Anfragen ausgewertet werden. Diese können in einem bestimmten Intervall von einem gewissen Startzeitpunkt (start) wiederholt werden (Langzeitanalysen).

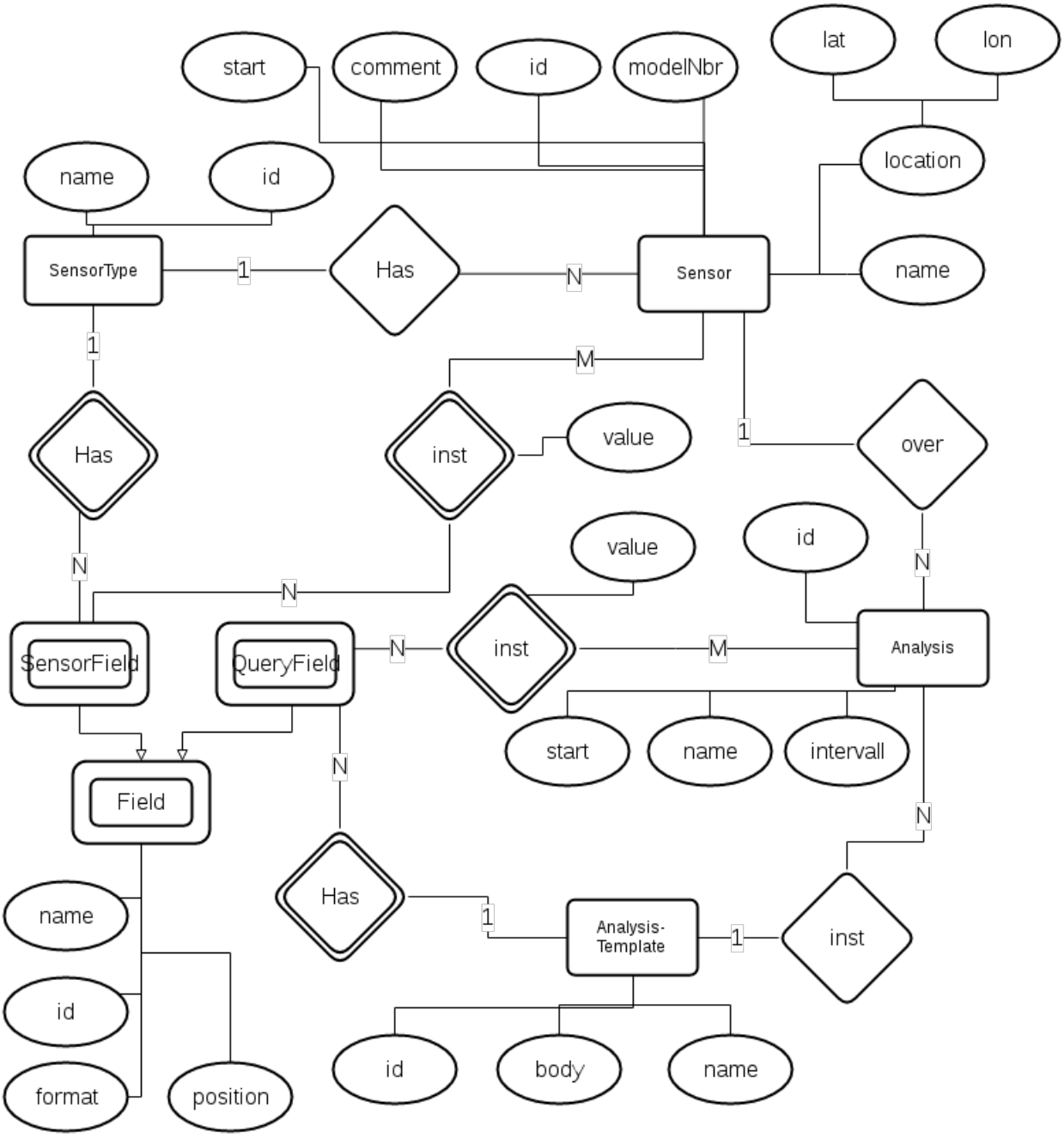


Abbildung 6.18.: LAOTSEDB ursprüngliches Entity Relationship Diagramm

AnalysisTemplate Im LAOTSE-System gibt es einige vordefinierte Anfragerümpfe, deren Felder nach Belieben ausgefüllt werden können. Jede Anfrage ist eine Belegung dieser Rümpfe.

6.5.2. Schema

Aus dem ER-Diagramm wird nun in einigen Schritten nach 6.18 ein Datenbankschema erstellt, wobei nur die für diesen Anwendungsfall relevanten Schritte aufgeführt werden.

1. **Behandlung starker Entitäten** Erzeuge für jede starke Entität eine Tabelle mit allen Attributen und dem Primärschlüssel.

Sensors:

ID	Name	Lat	Lon	ModelNbr	Start	Comment

SensorTypes:

ID	Name

Analyses:

ID	Name	Start	Intervall

AnalysisTemplates:

ID	Name	Body

2. **Behandlung schwacher Entitäten** Erzeuge für jede schwache Entität eine Tabelle, wobei der Primärschlüssel der schwachen Entität aus den Primärschlüsseln der zugeordneten starken Entitäten zusammengesetzt ist.

Sensors:

ID	Name	Lat	Lon	ModelNbr	Start	Comment

SensorTypes:

ID	Name

Analyses:

ID	Name	Start	Intervall

AnalysisTemplates:

ID	Name	Body

SensorFields:

TypeID	ID	Name	Format

AnalysisFields

TemplateID	ID	Name	Format

3. **Behandle 1:N Beziehungen** Erweitere die N-seitige Tabelle um einen Fremdschlüssel zur 1-seitigen. Dabei ist für die Fields Tabellen nichts zu tun, da sie als schwache Entitäten bereits auf einen Fremdschlüssel verweisen.

Sensors:

ID	Name	Lat	Lon	ModelNbr	Start	Comment	*TypeID*

SensorTypes:

ID	Name

Analyses:

ID	Name	Start	Intervall	*TemplateID*

AnalysisTemplates:

ID	Name	Body

SensorFields:

TypeID	ID	Name	Format

AnalysisFields

TemplateID	ID	Name	Format

4. **Behandle N:M Beziehungen** Erzeuge eine neue Tabelle mit einem zusammengesetzten Schlüssel aus den betroffenen Primärschlüsseln.

Sensors:

ID	Name	Lat	Lon	ModelNbr	Start	Comment	*TypeID*

SensorTypes:

ID	Name

Analyses:

ID	Name	Start	Intervall	*TemplateID*

AnalysisTemplates:

ID	Name	Body

SensorFields:

TypeID	ID	Name	Format

AnalysisFields

TemplateID	ID	Name	Format

AnalysisOverSensors:

SensorID	QueryID

AnalysisFieldInstances:

AnalysisID	TemplateID	ID	value

SensorFieldInstances:

SensorID	TypeID	ID	value

6.5.3. Einschränkungen

Es gibt einige *Constraints*, die sich nicht explizit durch das ERD in Abbildung 6.18 darstellen lassen. Diese werden im Folgenden definiert.

1. Ein Sensor soll genau die Felder initialisieren, die von seinem Sensortypen vorgegeben werden. $\forall i \in SensorFields : Sensors(i.SensorID).TypeID == i.TypeID$

2. Gleiches gilt für Analysis und AnalysisTemplate.
   $\forall i \in AnalysisFields : Analyses(i.AnalysisID).TemplateID == i.TemplateID$

3. Ein SensorType-Feld soll pro Sensor genau einmal mit einem Wert belegt sein.
   $\forall i \in Sensors, \forall j \in Fields, j.ContextID == i.TypeID : \exists! t \in SensorFields : t.TypeID == j.ContextID, t.ID == j.ID, t.SensorID == i.ID$

4. Analog für Analysen.
   $\forall i \in Analyses, \forall j \in Fields, j.ContextID == i.TemplateID : \exists! t \in AnalysisFields : t.TemplateID == j.ContextID, t.ID == j.ID, t.AnalysisID == i.ID$

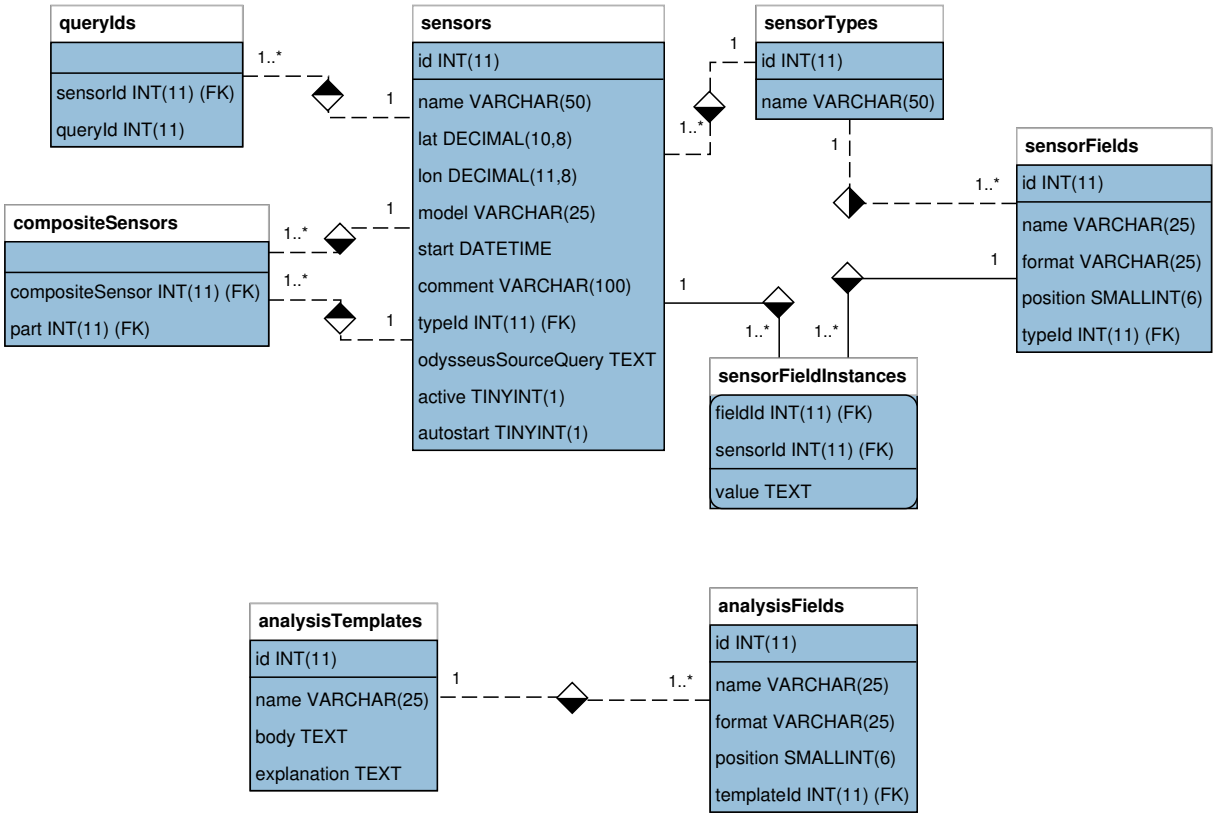


Abbildung 6.19.: Endgültiges Schema der LAOTSEDB

6.5.4. Anpassungen

Im Laufe des Projektes gab es einige Anpassungen am LAOTSEDB Schema, vor allem bedingt durch den Wegfall des AnalyseTools und damit der Speicherung der Queryinstanzen in der LAOTSEDB, die in dem endgültigen Schema zu sehen in Abbildung 6.19 resultieren.

- Wegfall Tabellen: Analyses, AnalysisFieldInstances, AnalysisOverSensors.
- Hinzugefügt:

  CompositeSensors Tabelle für die Erzeugung einer baumartigen Sensorstruktur, in der einige Sensoren (z.B. MosaikSensor) mehrere Untersensoren besitzen können.

  QueryIds Tabelle für die Zuordnung von SensorIds zu zugehörigen Odysseus-QueryIds (s. Abschnitt 7.3.3).

Sensor Felder Query zum Starten in Odysseus (s. Abschnitt 7.3.3) und zwei Flags, die anzeigen, ob die Query direkt beim Start des Servers in Odysseus gestartet werden soll (autostart) und ob der Sensor aktiviert ist (active).

6.6. Datengeneratoren

DN

Aus dem Odysseus-Projekt steht ein Framework zum Bauen von Datengeneratoren zur Verfügung [Ody15]. Der Datengenerator sendet push-basiert Daten über einen TCP-Stream. Die Konsumenten können sich bei dem Datengenerator registrieren, damit sie in die Liste der Adressaten aufgenommen werden. Jeder in diesem Projekt entwickelte Datengenerator liefert Tupel aus, die in Odysseus eingelesen werden können. Ein Tupel besteht dabei aus einem Zeitstempel und einem erzeugten Wert.

6.6.1. Konstantzahlengenerator

Dieser Generator wurde genutzt, um die Performance von Odysseus zu testen. Er erzeugt Datensätze mit konstanten Fließkommazahlen. Werden mehrere dieser Datengeneratoren eingesetzt, so lässt sich anhand des erzeugten Wertes zurückverfolgen, welcher Generator den Datensatz erzeugt hat.

6.6.2. Zufallszahlengenerator

Der Zufallszahlengenerator wird genutzt, um schnell zufällige Fließkommazahlen im Intervall [0,1) zu erzeugen. Dieser Generator erzeugt Zahlen, die nicht so komprimiert werden können, wie es bei konstanten Zahlen der Fall ist.

6.6.3. Temperatursensor

Der Temperatursensordatengenerator erzeugt keine Daten selbst, sondern liest die Temperatur in der Umgebung des Raspberry PIs. Dazu wird der Wert von einem Sensor DS18B20 über eine 1-Wire-Verbindung ausgelesen. Der Datengenerator ist für den Betrieb auf einem Raspberry PI B ausgelegt.

6.6.4. Audiosensor

Der Audiosensor überträgt Schallpegelmessungen. Er liest den Schallpegel über das Cirrus Logic Audio Board ein, das auf einem Raspberry PI 2 Model B gesetzt wurde. Der Datengenerator ist so ausgeführt, dass er 2 Kanäle in einem Kanal zusammenführt. Ein Sample wird mit 16 Bit Auflösung dargestellt.

6.6.5. Spannungssensor

Der Spannungssensor überträgt Messwerte, die von einem Analog-zu-Digital-Wandler ADS1015 12-Bit ADC - 4 Channel with Programmable Gain Amplifier gemessen werden. Der Datengene-

rator ist so aufgebaut, dass er eine Differenzialmessung zwischen zwei Kanälen durchführt. Der Datengenerator wurde für den Betrieb auf einem Raspberry PI 2 Model B entwickelt und getestet.

6.7. Datenbanken

MM

In diesem Abschnitt wird die Auswahl der Datenbanken dokumentiert. Dazu werden mögliche Datenbanken vorgestellt und verglichen. Eine ausführlichere Dokumentation der Tests und der Testergebnisse der Datenbankauswahl ist in Anhang E zu finden.

6.7.1. Test für die Datenbankauswahl

KB

Für die Auswahl einer geeigneten Datenbank wird ein Test durchgeführt, bei dem die Performanz verschiedener Datenbanken getestet wird. Welche Datenbanken dabei getestet werden sollen, muss zunächst ausgewählt werden. Für den Test wird eine Textdatei erstellt, in der zeilenweise Datensätze mit Timestamp und einem Double-Wert hinterlegt sind. Diese Daten sollen die Daten der Sensoren simulieren. In der Textdatei sind so viele Daten hinterlegt, wie ein Sensor mit einer Abtastrate von 30 kHz in einer Stunde erzeugen würde. Die gegebenen Daten entsprechen also ansatzweise den später zu verwendenden Daten. Für die Datenbanktests sollen diese Daten eingelesen und anschließend in die jeweilige Datenbank eingefügt werden. Während dieses Einfügevorgangs wird die Zeit gemessen, die die Datenbank dafür benötigt. Auf Basis dieser gemessenen Zeiten können die Datenbanken später verglichen werden. Da allerdings nicht nur die Schreib-, sondern auch die Lesegeschwindigkeit von Bedeutung ist, werden verschiedenste Anfragen, wie zum Beispiel das Herausfiltern des Maximalwerts, entwickelt um mit Hilfe dieser die Lesegeschwindigkeit der einzelnen Datenbanken zu ermitteln. Dabei wird wie beim Schreibvorgang auch die Zeit, die für den Lesevorgang benötigt wird, gemessen um einen Vergleich zwischen den einzelnen Datenbanken ziehen zu können. Diese Tests werden dabei sowohl für In-Memory Datenbanken, als auch Langzeitspeichersysteme durchgeführt, da für beide Anwendungsfälle die richtige Datenbank gewählt werden muss.

6.7.2. Mögliche Datenbanken

MM

Wie im vorherigen Abschnitt beschrieben, müssen die ausgewählten Datenbanken sehr schnell arbeiten können um den entwickelten Test zu bestehen. Wenn zuvor einige Recherchen und Überlegungen angestellt wurden, wird eine Auswahl an Datenbanken getroffen, die getestet werden sollen. Es werden einige Datenbanken vorgestellt, die Techniken verwenden, die in den Grundlagen eingeführt wurden. Diese Datenbanken sollen für den Einsatz in LAOTSE getestet werden.

In-Memory Datenbanken

- **VoltDB** ist eine NewSQL Datenbank. Sie wirbt mit, der möglichen Nutzung von SQL und gleichzeitiger Analyse von Echtzeitdaten in Millisekunden. Es sollen schnelle Datenströme verarbeitet werden können [Vol15].

- **Aerospike** ist eine NoSQL Datenbank, die einen Key-Value Store nutzt. Dabei ist Aerospike keine reine In-Memory Datenbank. Es werden nur die Indizes im Hauptspeicher gehalten. Die eigentlichen Daten werden auf einer SSD gesichert [Aer15].

Langzeitdatenbanken

- **InfluxDB** ist eine Zeitreihenbasierte Datenbank, die eigenständig lauffähig ist. Sie ist spezialisiert auf Sensordaten und soll Analysen in Echtzeit unterstützen. Dabei wird eine SQL-ähnliche Anfragesprache bereitgestellt. Aktuell befindet sich InfluxDB noch in der Alphaphase. Außerdem gibt es bereits Internetapplikationen, die z.B. eine Visualisierung der Daten ermöglichen [Inf15].

- **Prometheus** ist eine Zeitreihenbasierte Datenbank, die auf das Monitoring von Datenströmen ausgelegt ist. Das Datenmodell nutzt metrische Namen und Key-Value Paare. Es können unter bestimmten Umständen Alarme ausgelöst werden und die Daten und Alarme in Dashboards visualisiert werden. Eingesetzt wird Prometheus bei soundcloud.com [Sou15]. Insgesamt bietet es also einige erweiterte Funktionen.

- **Cassandra** ist eine NoSQL Datenbank, die hauptsächlich auf Skalierbarkeit und hohe Verfügbarkeit ausgelegt ist. Die Performance soll ebenfalls sehr hoch sein. Allerdings werden Aggregationen nur teilweise und Cursor noch nicht unterstützt [Apa15].

- **BlueFlood** ist eine Zeitreihenbasierte Datenbank, die sehr stark auf Skalierbarkeit ausgelegt ist. Es kann ähnlich wie Cassandra in einer Cloud eingesetzt werden und Daten für ein Monitoring-System bereitstellen und verarbeiten, wie es z.B. in Prometheus bereits integriert ist [Rac15].

- **ScaleDB** ist eine auf MySQL und MariaDB basierende Datenbank. Sie ist ein RDBMS, dass auf das schnelle Einfügen von Daten spezialisiert ist. Sogenannte Streaming Tables erlauben einen hohen Durchsatz beim Einfügen. Weitere Schreiboperationen sind aber begrenzt [Sca]

Hybridlösung

- **Druid** ist eine Zeitreihenbasierte Datenbank. Die Besonderheit bei Druid ist, dass sie eingehende Daten direkt aus einem Datenstom akzeptiert und in einer In-Memory Datenbank speichert. Es gibt spezielle Broker-Knoten, die Daten aus der In-Memory Datenbank in einen angeschlossenen historischen Speicher schreiben können. Für diesen kann das Dateisystem oder auch eine andere Langzeitdatenbank genutzt werden. Ein Client kann Anfragen an die Broker-Knoten stellen, die zur Beantwortung auf die In-Memory Datenbank und die Langzeitdatenbank zurückgreifen können. Damit sind bereits einige Funktionalitäten des Bulk-Storers und des Analysetools aus LAOTSE von der Datenbanklösung, die In-Memory und Langzeitdatenbank vereint gegeben. Allerdings werden einige externe Komponenten

benötigt um z.B. Metadaten über die Knoten zu speichern. Insgesamt ist Druid als sehr schnell und skalierbar beworben, da alle Funktionalitäten verteilt implementiert sind. Die Anfragesprache ist für Analysen ausgelegt und bietet zahlreiche Filter und Berechnungen. Druid befindet sich momentan in der Version 0.8.0 wird aber trotzdem bereits von zahlreichen großen Unternehmen, wie z.B. Netflix, Cisco, Yahoo, Paypal und eBay produktiv eingesetzt [Dru15].

6.7.3. Ungeeignete Datenbanken

KB

Einige der Datenbanken, die im vorherigen Abschnitt beschrieben wurden, erwiesen sich bereits vor der Durchführung des Tests als ungeeignet für den gegebenen Anwendungsfall. Auf diese Datenbanken soll in diesem Abschnitt noch einmal genauer eingegangen und dabei vor allem beschrieben werden, aus welchen Gründen die Datenbanken ungeeignet für den gegebenen Anwendungsfall sind.

BlueFlood

BlueFlood ist eine zeitstempelbasierte Datenbank. Das bedeutet, dass sie dafür geschaffen wurde um Werte, die jeweils zu einem Zeitpunkt anfallen auf gute Art und Weise zu speichern. Ebenso bedeutet das, dass zu jedem Wert, der in die Datenbank geschrieben wird ein Zeitstempel existiert. Dabei können in BlueFlood die Zeitstempel auf Mikrosekunden genau gespeichert werden. Das heißt, dass für jede Mikrosekunde, die abgelaufen ist ein oder mehrere Zeitstempel gespeichert werden können. Da der Datendurchsatz für das zu entwickelnde Produkt allerdings sehr hoch ist, werden für eine Mikrosekunde mehrere Werte anfallen. Ideal wäre in diesem Falle, dass das Datenbanksystem auf Nanosekunden genau Werte speichern könnte. Bei BlueFlood ist das allerdings nicht der Fall. Aus diesem Grund müsste dementsprechend für einen Zeitstempel in der Datenbank ein Sample von Werten gespeichert werden, die diesem Zeitstempel zugeordnet sind. Im Grunde wäre das auch noch kein Ausschlusskriterium für eine Datenbank, denn aus mehreren Werten ein Sample von Werten zu erstellen, stellt keine all zu große Hürde dar. In BlueFlood können diese Samples auch problemlos für einen Zeitstempel gespeichert werden. Allerdings können die einzelnen Werte des Samples in einer späteren Abfrage nicht einzeln ausgelesen werden, sondern BlueFlood ist nur in der Lage, zum Beispiel den Durchschnittswert, den Maximalwert oder den Minimalwert eines Samples auszulesen und zurückzugeben. Die Projektanforderungen verlangen allerdings, dass alle Werte, die von den Sensoren ankommen, auch in einer späteren Abfrage wieder ausgelesen werden können. Deshalb wird BlueFlood für das spätere Produkt nicht eingesetzt, da die Projektanforderungen nicht erfüllt werden können.

Prometheus

Prometheus ist wie im vorherigen Abschnitt aufgeführt, besonders auf das Monitoring von Datenströmen spezialisiert. Aus diesem Umstand folgt leider auch, dass sie für die persistente Speicherung und nachträgliche Auswertung von Datenwerten eher ungeeignet ist bzw. für diese Arte der Nutzung keine unterstützenden Funktionen anbietet. So gibt es z.B. keine Möglichkeit Daten aktiv in die Prometheus-Datenbank einzufügen, sofern sie nicht aus einem vorher definierten Datenstrom stammen. Auch das Hinzufügen von weiteren Datenströmen zur Laufzeit ist sehr

umständlich. Im Projekt würde die Datenbank also quasi „entgegen ihrer Natur“eingesetzt. Des Weiteren ist die Dokumentation von Prometheus, wie leider bei einer sich noch in ständiger Entwicklung befindlichen Software nicht unüblich, insgesamt lückenhaft und unstimmig, was das Aufsetzen einer funktionierenden Instanz durchaus verkompliziert.
Aus diesen Gründen wurde mehrheitlich beschlossen die Prometheus-Datenbank nicht im Projekt einzusetzen und sie folglich aus der Liste, der zu testenden Datenbanken, zu entfernen.

ScaleDB

ScaleDB bietet die Möglichkeit, Daten mit einem hohen Durchsatz einzufügen. Die Daten werden dabei einem Zeitstempel zugeordnet. Der Zeittempel kann nicht von außen vorgegeben werden. Für jeden Datensatz wird die aktuelle Systemzeit von ScaleDB verwendet, wenn der Datensatz die Datenbank erreicht. Es ist nicht möglich den ursprünglichen Zeitstempel der Erfassung der Messdaten zu verwenden. Dieser Zeitstempel könnte zwar als Attribut mitgeliefert werden, allerdings ist die Abfrage über dieses Attribut nicht performant. Der Grund dafür ist, dass in Streaming Tables nur Hash-Indizes gebildet werden können. Es ist wenig sinnvoll die Zeistempel zu hashen, da aufeinanderfolgende Daten nicht zusammen in eine Partition gelangen. Über einen solchen Index wird die Zeitreihe auseinander genommen.

6.7.4. Durchführen von Datenbanktests

KB, DN

Für die Durchführung der Datenbanktests werden zunächst alle Datenbanken, die in diesem Test geprüft werden sollen auf einer virtuellen Linuxmaschine aufgesetzt und so konfiguriert, dass die vorgegebenen Zeitstempel-Wert-Paare in ein korrektes Schema eingefügt werden können. Da die Tests allerdings nicht verfälscht werden dürfen, gilt dass während der Durchführung immer nur eine Datenbank läuft. Währenddessen müssen alle anderen Datenbanken für diesen Zeitraum heruntergefahren sein. So sind für alle Kandidaten die gleichen Grundvoraussetzungen gegeben.

Während der Tests wird dementsprechend auch immer die gleiche Menge an Daten in die einzelnen Datenbanken eingefügt, um so einen Vergleichswert zwischen den einzelnen Datenbanken zu haben. Die einzufügenden Daten werden dafür von einem Datengenerator geschrieben. Dieser Datengenerator generiert dabei genau so viele Zeitstempel-Wert-Paare wie in der Testkonfiguration angegeben wurde. Der Wertebereich für die einzelnen Werte kann dabei ebenfalls in der Testkonfiguration angegeben werden. Für den Test werden verschiedene Anfragen an die einzelnen Datenbanken gesendet.
Neben einem einfachen Insert der Zeitstempel-Wert-Paare werden dabei auch Anfragen gestartet, die die Daten wieder auslesen. Dafür gibt es vier verschiedene Anfragetypen für jede Datenbank. So ist es zunächst einmal möglich, alle Werte in einem Bereich zwischen zwei Zeitstempeln auszulesen. Da allerdings nicht nur die einzelnen Werte innerhalb eines Bereiches interessant sind, sondern auch Extremwerte, gibt es noch die Möglichkeit, den Maximal- und den Minimalwert innerhalb eines solchen Bereiches auszulesen. Außerdem kann innerhalb eines Bereiches zwischen zwei Zeitstempeln auch der Mittelwert gebildet werden, um so Vergleichswerte zwischen verschiedenen Bereichen zu schaffen.
Während des Tests wird jeder dieser Anfragetypen auf den einzelnen Datenbanken angewendet. Dazu existiert zu jeder Datenbank ein Datenbankwrapper, in dem die verschiedenen Methoden

für diese einzelnen Datenbanken implementiert sind, da auf jede der Datenbanken auf eine unterschiedliche Art und Weise zugegriffen werden kann. Das Testprogramm liest im Endeffekt also nur die einzelnen Testkonfigurationen aus und führt diese Tests der Reihe nach durch. Soll in dem Test ein Insert gemacht werden, muss zunächst der Datengenerator gestartet werden, der daraufhin die Daten für den Test generiert. Beim Auslesen wird der Datengenerator dementsprechend nicht benötigt.
In der Testkonfiguration ist ebenfalls festgelegt, auf welche Datenbank dieser Test angewendet werden soll. Je nachdem welche Datenbank in der Konfiguration angegeben ist, muss das Testprogramm den richtigen Datenbankwrapper nutzen, um die Anfrage an die richtige Datenbank zu senden. Wird die Anfrage an die Datenbank gesendet startet ein Timer, der die Zeit misst, die für die Anfrage benötigt wird. Es wird die Zeit zwischen dem Start der Anfrage und dem bekommen der Antwort von der Datenbank gemessen. Diese gemessene Zeit dient später als Vergleichswert zwischen den einzelnen Datenbanken und anhand dieses Wertes wird dann entschieden, welche Datenbank für das zu entwickelnde Produkt am Besten nutzbar ist.

6.7.5. Genutzte Datenbankschemata

MM

In diesem Abschnitt wird beschrieben, welche Schemata angelegt wurden, um diese in den jeweiligen Datenbanken für den Datenbanktest zu nutzen. Dabei werden die im vorherigen Kapitel durch Recherche und erste Tests bereits ausgeschlossenen Datenbanken außer Acht gelassen. Jedes Schema muss dabei die Daten so speichern, dass schnelle Speicher- und Abrufung möglich sind. Man möchte die Messwerte einem Sensor und einem Zeitstempel zuordnen. Außerdem sollten Berechnungen auf den Messwerten, wie z.B. die Bestimmung eines Maximalwertes oder Mittelwertes einer bestimmten Zeitreihe durchgeführt werden können. Es gibt jeweils noch die Möglichkeit pro Zeitstempel nur einen Messwert zu speichern oder ein Sample an Daten aus einer Zeitreihe mit einer fest definierten Länge.

VoltDB: Da VoltDB eine NewSQL-Datenbank ist, nutzt sie ein relationales Schema. So kann für jeden Sensor eine Tabelle angelegt werden und für jeden Messwert ein Tupel mit jeweils einem Zeitstempel und dem zugehörigen Wert.

Aerospike: Aerospike nutzt ein Key-Value Datenmodell, das zeilenbasiert ist und in einigen Punkten mit einem relationalen Modell verglichen werden kann. Es wird für jeden Sensor ein Set angelegt, das mit einer Tabelle vergleichbar ist. Ein Record entspricht einem Tupel und wird für jeden Messwert angelegt. Diese enthalten einen `Key`, der den Timestamp beinhaltet und ein `Bin`, in dem der eigentliche Wert gespeichert wird.

InfluxDB: In InfluxDB kann eine SQL-ähnliche Anfragesprache genutzt werden, obwohl diese kein klassisches relationales Datenschema besitzt. Es werden unter einem `Title` beliebig viele Tupel automatisch gruppiert. Diese können Werte unter verschiedenen `Tags` speichern, die Attributen ähneln und immer als Strings gespeichert werden. Im umgesetzten Schema wurden die Messwerte als Wert mit Timestamp unter dem Sensor gruppiert.

Die Metadaten wurden als Tags hinzugefügt. Das hat den Vorteil, dass nach den Messwerten, Zeitstempeln und Metadaten gesucht werden kann, die Metadaten und die Sensor-ID

pro Sensor nur einmal gespeichert werden, ein schneller Zugriff auf alle Daten eines Sensors möglich wird und im Falle der Veränderung der Metadaten eines Sensors, z.B. einer Positionsänderung, automatisch eine neue Gruppierung angelegt wird.

Cassandra: In Cassandra können Tabellen ähnlich einem relationalen Datenbankschema angelegt werden. Dabei ist der Unterschied, dass in Cassandra der Speicherplatz bei nicht belegten Attributen in einem Tupel nicht durch nulls gefüllt werden muss, wie es in relationalen Datenbankschemata der Fall ist, sondern nur dort Speicherzellen belegt werden, wo wirklich Daten vorhanden sind. Es wurde für jeden Sensor eine Tabelle angelegt und für jeden Messwert ein Tupel.

Druid: In Druid wird das konkrete Datenschema für die Speicherung in den Echtzeit- bzw. historischen Knoten durch eine Konfigurationsdatei festgelegt. Diese gibt ebenfalls an, in welchem Datenschema die Daten aus dem Datenstrom erwartet werden. In LAOTSE wird dazu pro Messwert, also pro Element aus dem Datenstrom ein JSON-Objekt genutzt. Dieses definiert den Zeitstempel als `Timestamp`, den Sensornamen bzw. die Sensoridentifikation als `sensor` und den Messwert als `value`.

6.8. Testergebnisse

KB, MM

In diesem Abschnitt erfolgt die Begründung der Datenbankauswahl, die im Rahmen der Projektgruppe SESAdata getroffen wurde. Zuerst werden die getesteten Datenbanken aufgezählt und die Testergebnisse dargestellt. Die anschließende Auswertung begründet, warum die Entscheidung auf Druid gefallen ist.

Insert

Es wurden 50 Millionen Tupel in die Datenbanken eingefügt und die benötigte Zeit in Millisekunden gemessen. In Abbildung 6.20 sind die Testergebnisse dargestellt.
Während des Testlaufs ist in Aerospike ein reproduzierbarer Speicherüberlauf aufgetreten, obwohl über 20 GB Hauptspeicher zur Verfügung standen. Aerospike ist für LAOTSE ungeeignet, da es nicht so große Datenmengen verarbeiten kann. Weiter fällt auf, dass die reine In-Memory Datenbank VoltDB beim Einfügen der Daten langsamer ist, als die reine Langzeitdatenbank Cassandra. Der Hybrid Druid liegt nur knapp hinter Cassandra und ist damit die schnellste Datenbank, die In-Memory Fähigkeiten besitzt.

Select

Hier wurde ein Durchschnittswert über 90 000 Tupel gebildet. In dieser Abfrage und allen folgenden wird Cassandra nicht weiter betrachtet, da die Lesegeschwindigkeit bei Cassandra ca. 100 mal langsamer ist als bei allen anderen Datenbanken. In den folgenden Abbildungen sind die Testergebnisse der Abfragen zu sehen.
In den Abbildungen 6.21, 6.22 und 6.23 ist zu sehen, dass die Lesegeschwindigkeit bei VoltDB am schnellsten ist. InfluxDB belegt den zweiten Platz und Druid den dritten Platz. Aerospike kann außer acht gelassen werden, das diese Datenbank bereits beim Einfügen der vielen Daten

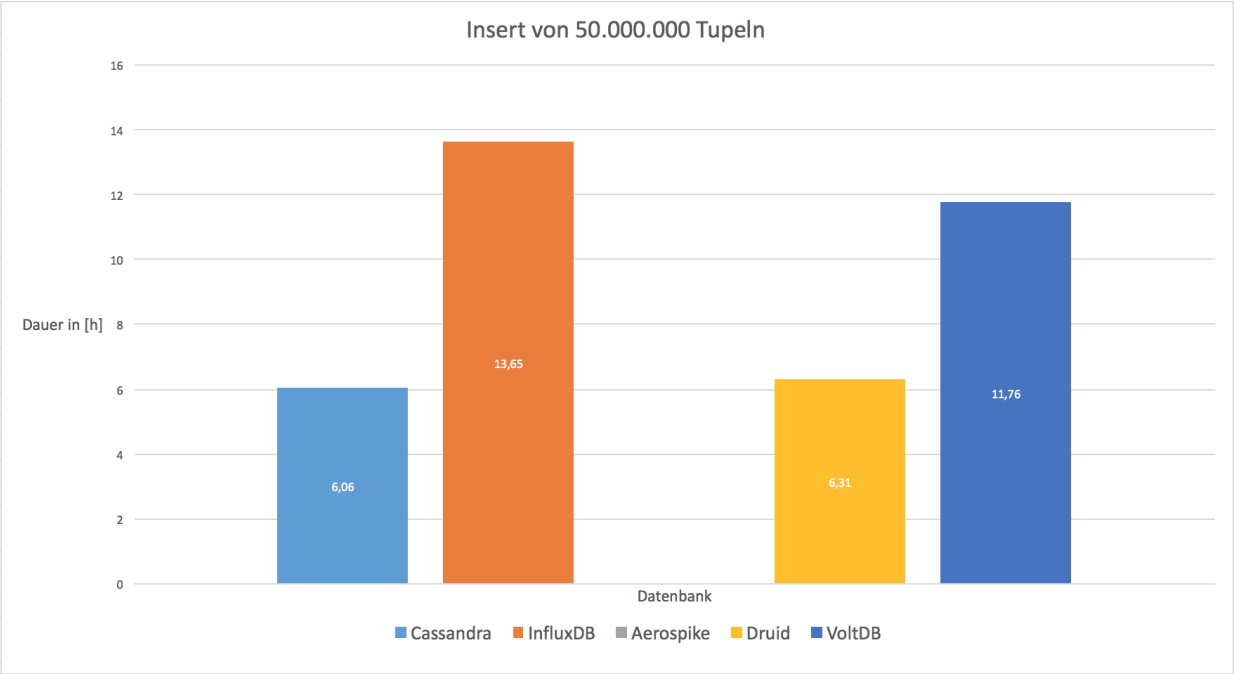


Abbildung 6.20.: Benötigte Zeit in Stunden beim Einfügen von 50 Millionen Tupeln

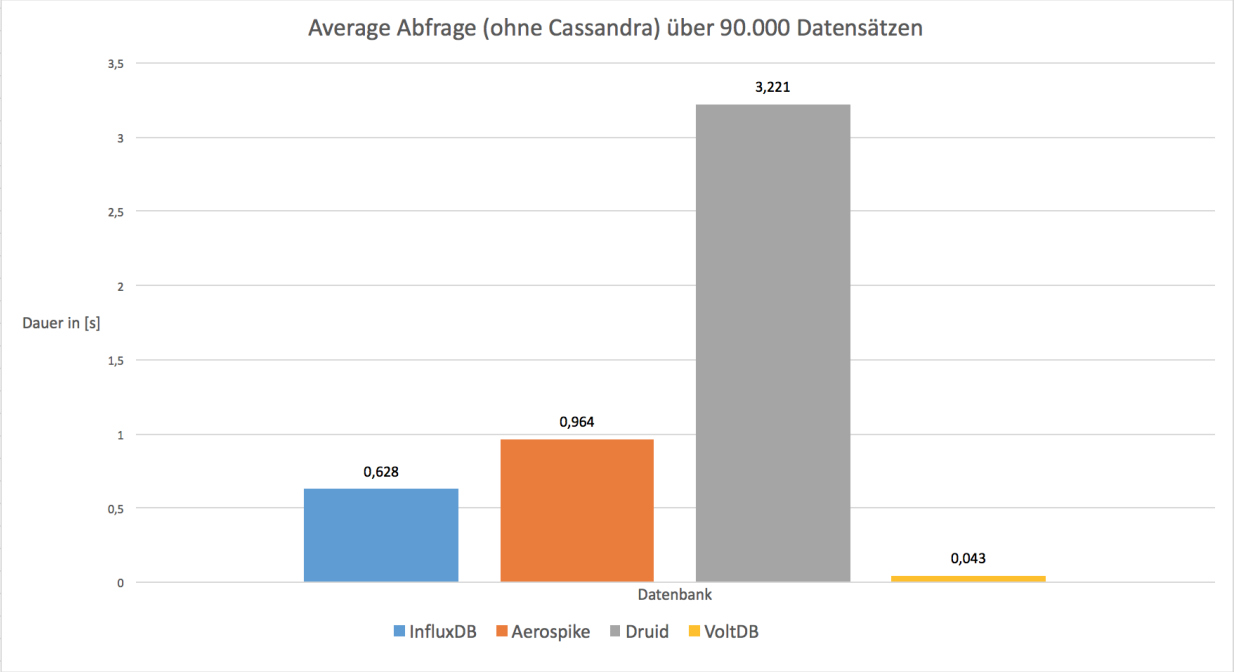


Abbildung 6.21.: Benötigte Zeit in Sekunden zur Bildung eines Durchschnitts über 90 000 Tupel

versagte. In Abbildung 6.24 ist zu sehen, dass InfluxDB hier bedeutend länger benötigte. Das liegt daran, dass InfluxDB keine Rangeabfrage nativ unterstützt und dies über zahlreiche einzelne Anfragen imitiert werden musste.

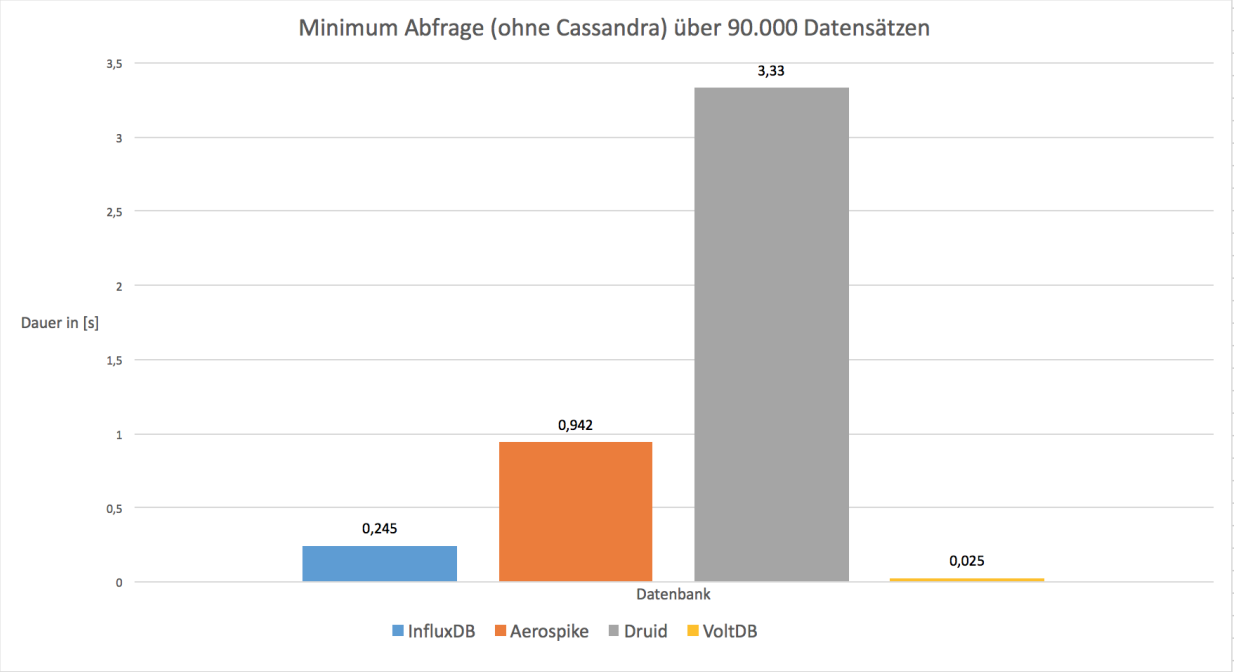


Abbildung 6.22.: Benötigte Zeit in Sekunden zur Bildung eines Minimums über 90 000 Tupel

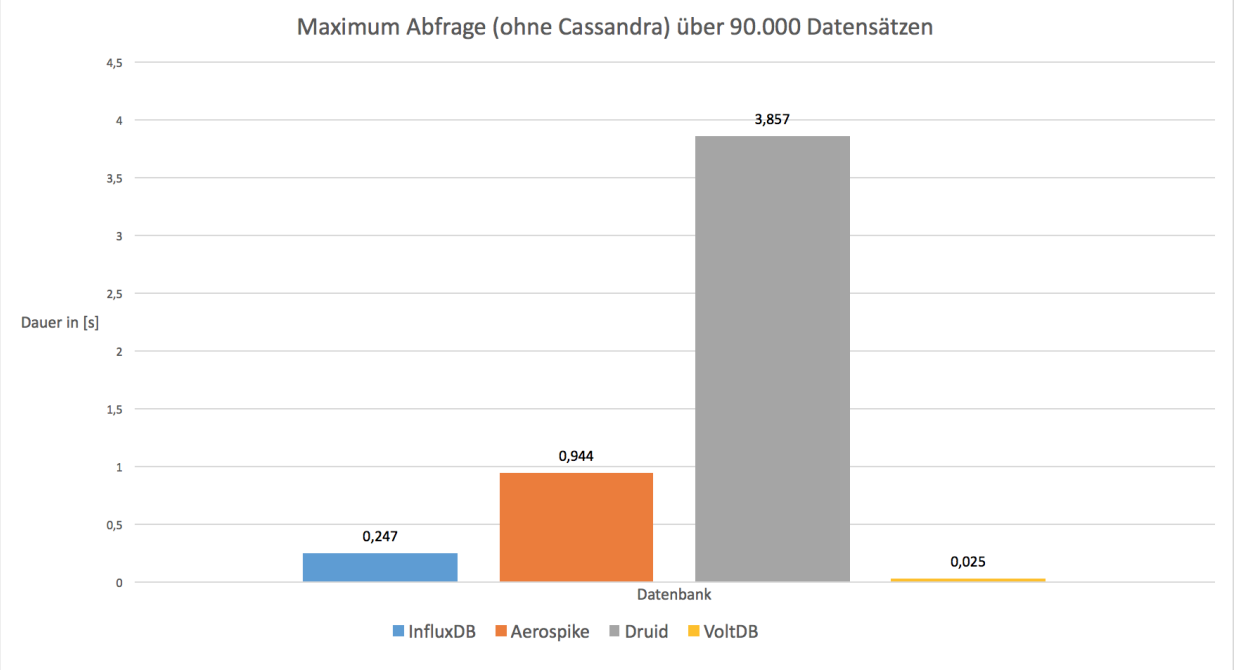


Abbildung 6.23.: Benötigte Zeit in Sekunden zur Bildung eines Maximums über 90 000 Tupel

Auswertung der Testergebnisse

Wenn die Testergebnisse mit den entsprechenden Erläuterungen betrachtet werden bleiben final folgende drei Kandidaten zur Verwendung in LAOTSE:

- InfluxDB (Langzeitdatenbank)
- Druid (Langzeit- und In-Memory Datenbank)

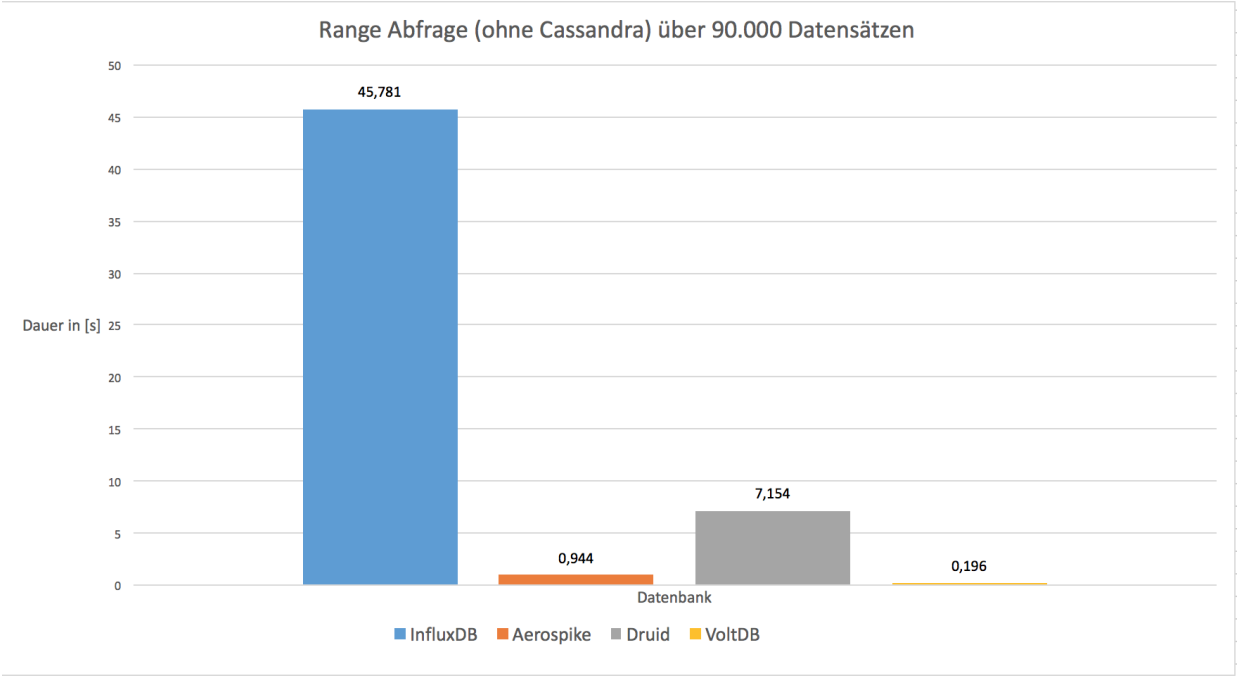


Abbildung 6.24.: Benötigte Zeit in Sekunden zur Bildung eines Datenumfangs über 90 000 Tupel

- VoltDB (In-Memory Datenbank)

Die Anforderungen von LAOTSE sehen vor, dass die Geschwindigkeit beim Einfügen von Tupeln oberste Priorität hat. Daher scheint Druid der geeignetste Kandidat zur Verwendung in LAOTSE . Weiter noch sieht die Architektur von LAOTSE die Verwendung einer In-Memory und einer Langzeitdatenbank vor. Die Alternative zur Verwendung von Druid wäre also die Verwendung von VoltDB und InfluxDB. Damit wäre die In-Memory Datenbank VoltDB, die direkt an den Datenstrom aus Odysseus angebunden werden soll, allerdings bereits ein Flaschenhals beim Einfügen der Daten. Die Daten könnten zwar u.U. schneller ausgelesen, aber gar nicht so schnell neue Daten eingefügt werden. In LAOTSE werden allerdings mehr Daten eingefügt als ausgelesen. Zudem würde die Verwendung eines selbst implementierten Bulk-Storers, der die Daten von VoltDB nach InfluxDB überträgt den Datenfluss noch weiter ausbremsen.

Druid bringt diese Techniken der Übertragung der Daten aus einer eigenen In-Memory Datenbank in eine eigene Langzeitdatenbank bereits mit. Zudem kann Druid einfach skaliert werden, da es als Cluster aufgebaut ist. Das Lesen von Daten kann aus speziellen Knoten erfolgen, die die Verteilung der Daten auf In-Memory Knoten und Langzeitknoten kennen. Somit ist kein Management über die Datenverteilung über die verschiedenen Datenbanken mehr nötig.

Somit fällt die Wahl auf Druid.

Es fällt allerdings auf, dass die Insert-Geschwindigkeit für die Anforderungen von LAOTSE trotzdem nicht genügen. Dies wird in folgenden Arbeitsschritten durch eine Optimierung der Datenbank behoben. Es könnten z.B. Samples gebildet und die Datenbank verteilt eingesetzt und im Folgenden die Möglichkeiten geprüft und dokumentiert werden.

6.9. Systemoptimierung

KB

Dieser Abschnitt beschäftigt sich mit dem Optimierungsprozess von Druid.

6.9.1. Druid Insert-Möglichkeiten

In diesem Abschnitt soll beschrieben werden, welche Möglichkeiten es gibt, Daten in Druid einzufügen. Anschließend wird begründet, welche dieser Möglichkeiten genutzt werden soll und warum sie gerade im Vergleich zu den anderen vorzuziehen ist.
Beim Einfügen von Daten in Druid ist zunächst einmal zu beachten, dass Druid sowohl eine In-Memory, als auch eine Langzeitdatenbank anbietet. Demzufolge ist es möglich, Echtzeitdaten von einem Datenstrom oder historische Daten als Batch-Daten direkt in die Langzeitdatenbank einzufügen.
Für das Einfügen von Echtzeitdaten gibt es zwei Möglichkeiten:

1. Daten mittels eines so genannten Stream Prozessors einfügen. Dafür wäre Tranquility als Lösung denkbar. Allerdings ist es aktuell nicht möglich Tranquility für die Neueste Version von Druid zu nutzen.

2. Apache Kafka als „Nachrichten-Knoten“ nutzen.

Somit bleibt noch die Möglichkeit, Daten über Kafka in Druid einzufügen. Bei dieser Methode wird der Datenstrom als Kafka Producer an Kafka angebunden und schickt über diesen Weg die Daten an Kafka. Bei Kafka können nun verschiedene Consumer - einer davon sollte Druid sein - registriert werden, welche die Daten von Kafka erhalten, sobald neue zur Verfügung stehen. Auf diesem Wege erreichen die Daten im JSON Format Druid und können dann in der In-Memory Datenbank gespeichert werden.
Stehen allerdings ältere, statische Daten zur Verfügung, die nicht in Echtzeitknoten von Druid, sondern direkt im historischen Knoten gespeichert werden sollen, existiert eine weitere Möglichkeit, diese Daten, durch Laden von Batch Dateien, in Druid einzufügen. Dazu wird ein Indexer benötigt, über den die Daten gespeichert werden. Derart können historische Daten eingefügt und indiziert werden, um diese in die Langzeitdatenbank, den historischen Knoten von Druid zu speichern.
Die Varianten des Einfügens mittels Kafka bzw. Ladens von Batch-Daten sind dabei sehr unterschiedlich und können je nach Anwendungsfall besser geeignet sein, als die jeweils andere Variante. So dass man zwischen diesen Beiden nicht direkt eine ausschließen kann, die für das Einfügen von Daten schlechter geeignet ist.

6.10. GUI

MM

Dieser Abschnitt stellt die LAOTSE-GUI vor. Dazu wird zuerst beschrieben, welche Aufgaben diese übernimmt und wie sie anhand eines Mockups konzeptioniert wurde. Anschließend wird die Umsetzung mit JavaFX dokumentiert, die mit der Darstellung von Screenshots abschließt.

6.10.1. Aufgaben der LAOTSE-GUI

MM

Die Hauptaufgabe der LAOTSE-GUI ist die Steuerung des Gesamtsystems LAOTSE. Es werden Eingaben vom Benutzer getätigt, die das Modell ändern können und damit verschiedene Komponenten von LAOTSE beeinflussen. So kann eine Analyse gestartet werden, durch die Daten aus Druid gelesen oder in Druid geschrieben werden, wenn die Ergebnisse gespeichert werden sollen. Es kann aber auch eine Beeinflussung von Odysseus stattfinden, wenn z.B. neue Sensoren angebunden werden, müssen diese auch dort hinzugefügt werden. Ansonsten dient die LAOTSE-GUI zur Analyse und Visualisierung der Sensoren und der historischen Daten. Der Datenstrom der Echtzeitdaten und eine Analyse dieses Datenstroms kann mit der Odysseus-GUI bewerkstelligt werden.
Die Aufgaben der LAOTSE-GUI sind:

- Hinzufügen von Sensoren
- Anzeigen der Metadaten eines Sensors
- Bearbeiten der Metadaten eines Sensors
- Hinzufügen von Analysen nach bestimmten Mustern
- Starten von Analysen unter Angabe von Parametern
- Anzeige der Ergebnisse einer Analyse

6.10.2. Mockup

MM, KB

Die LAOTSE-GUI wurde anhand eines Mockups konzeptioniert. In Abbildung 6.25 und Abbildung 6.26 sind Mockups der LAOTSE-GUI dargestellt. Auf der rechten Seite ist je nach Auswahl eine Liste der Verfügbaren Analysen oder Sensoren zu sehen. Unten findet sich eine Konsole und oben rechts die Detailansicht der jeweils aktuell gewählten Analyse oder des jeweils aktuell gewählten Sensors.
Abbildung 6.25 zeigt beispielhaft die Detailansicht einer Range-Analyse. Es ist die Möglichkeit gegeben auszuwählen, über welche Sensoren und welchen Zeitraum die Analyse erfolgen soll. Die Analyseergebnisse werden in einem Diagramm dargestellt. Diese Ansicht soll dazu dienen eine Analyse zu starten und zu konfigurieren.
Abbildung 6.26 zeigt beispielhaft die Detailansicht eines Sensors. Neben dem Namen ist die Position auf einer Karte zu sehen. Im unteren Bereich sind die weiteren Metainformationen, wie der Sensortyp und die Einheit, in der gemessen wird dargestellt. Diese Ansicht soll dazu dienen die Metadaten eines Sensors zu betrachten und zu bearbeiten.

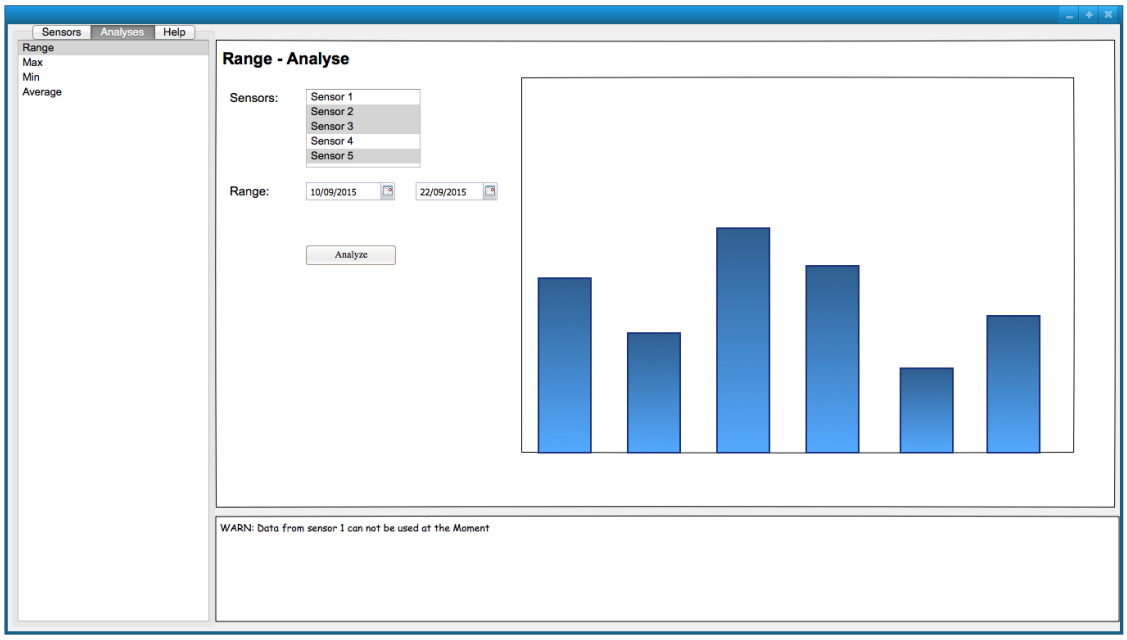


Abbildung 6.25.: Mockup der Detailansicht einer Analyse

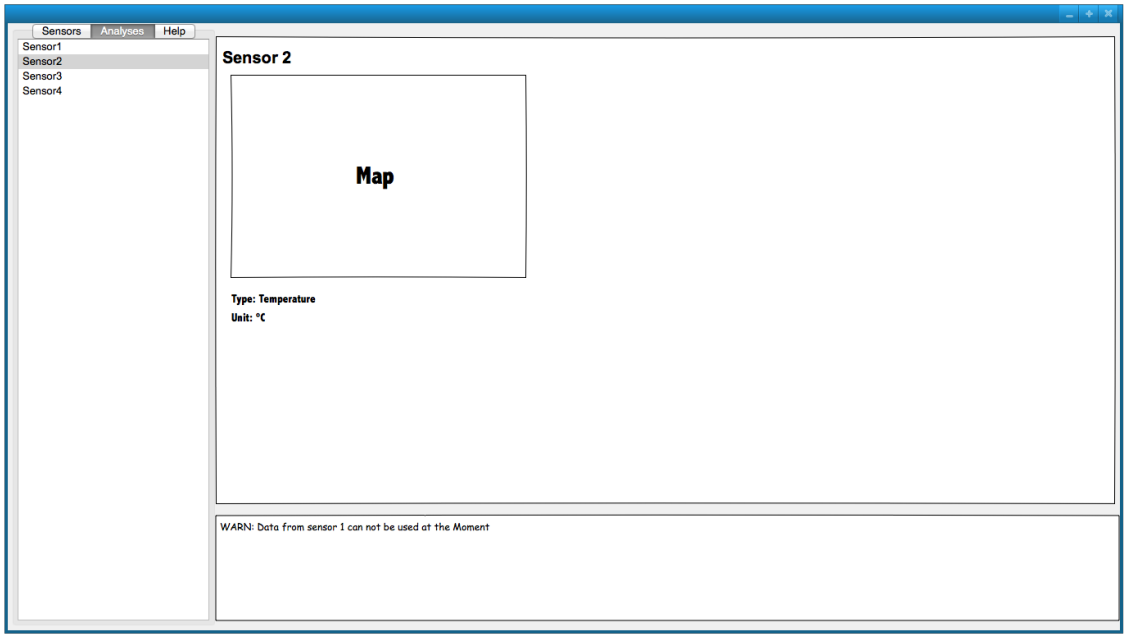


Abbildung 6.26.: Mockup der Detailansicht eines Sensors

6.10.3. Umsetzung in JavaFX

MM

Nach dem Model-View-Controller Pattern wurde eine JavaFX-Applikation implementiert. Dabei stellen die FXML-Dateien, die View dar. Diese definieren die Struktur der GUI so, wie sie in den Mockups dargestellt ist. Die Controller fügen die Funktionalitäten hinzu. Jeder Controller kennt ein eigenes Model, welches die Daten enthält, die in der jeweiligen View dargestellt werden. So besitzt jede View einen Controller und falls Daten zu visualisieren sind, auch ein Model.
Es wurden folgende Views mit entsprechenden Controllern implementiert:

Root Die `RootView` ist das eigentliche Fenster und ein Container für alle weiteren Views. Sie definiert, wo und in welcher Form die weiteren Views innerhalb des Fensters dargestellt werden, indem der `RootController` Views anordnet. Über diesen können auch alle anderen Controller miteinander kommunizieren, wenn ein Controller Daten bereit stellen möchte, die in einer anderen View visualisiert werden sollen. Die Verteilung erfolgt dann über den `RootController`.

List Die `ListView` stellt Listen der verfügbaren Sensoren, Analysen und Hilfethemen dar. Der Controller steuert das Hinzufügen und Entfernen neuer Elemente. Wenn ein Element ausgewählt wird, leitet der Controller diese Information an den `RootController`, damit dieser eine neue View laden kann, in der die Details und Metadaten z.B. eines Sensors oder einer Analyse, dargestellt werden.

Sensor Die `SensorView` visualisiert die Metadaten eines Sensors und stellt seine Detailansicht dar. Der `SensorController` kann die Bearbeitung dieser Daten, das Löschen und das Hinzufügen eines Datensatzes handhaben, indem das Model bearbeitet wird.

Analyse Die `AnalyseView` ist analog zu der `SensorView` für die Analysen aufgebaut. Darüber hinaus kann sie noch Analysen in Druid starten und die Ergebnisse visualisieren.

Help Die `HelpView` zeigt einen einfachen Hilfetext an.

Center Die `CenterView` ist ein Container, in dem beliebig viele Sensor-, Analyse- und Hilfe-Views angezeigt werden können. Es wird je nach Art der anzuzeigenden View im `CenterController` ein Duplikat erlaubt oder auch nicht.

Console Die `ConsoleView` zeigt eine einfache Konsole. Der `ConsoleController` sorgt dafür, dass Fehlermeldungen und andere Statusmeldungen dargestellt werden können.

Die Controller beobachten dabei ihr Model und ggf. andere relevante Models nach dem Observer-Pattern, um über Änderungen informiert zu werden. Dadurch besteht die Möglichkeit, die View ständig zu aktualisieren.
Bei dem Prototypen, der zum Zwischenbericht fertiggestellt wurde, handelt es sich bei der GUI um einen sehr einfachen Prototypen. Es wurden die Hauptfunktionen der einzelnen Views und Controller so implementiert, wie sie oben aufgeführt sind. In weiteren Iterationen des Prototypen folgt u.a. eine Gestaltung der GUI. Die momentane Ansicht der Details eines Sensors ist in Abbildung 6.27 zu sehen. Es können bereits die Daten bearbeitet werden. Hierzu öffnet sich eine Eingabemaske mit Textfeldern. In dem weißen Bereich wird der Sensor später auf einer Karte dargestellt werden. Über das Plus unten links ist es möglich einen weiteren Sensor hinzuzufügen.

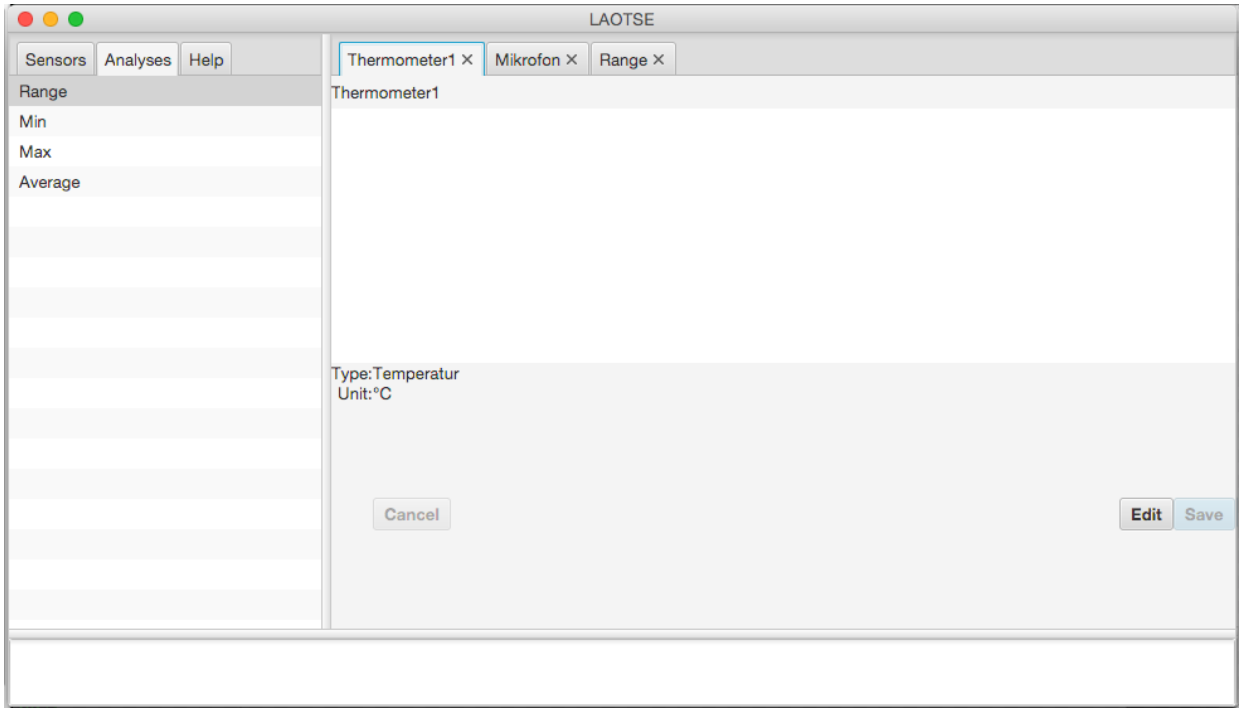


Abbildung 6.27.: Screenshot der Detailansicht eines Sensors im ersten Prototypen

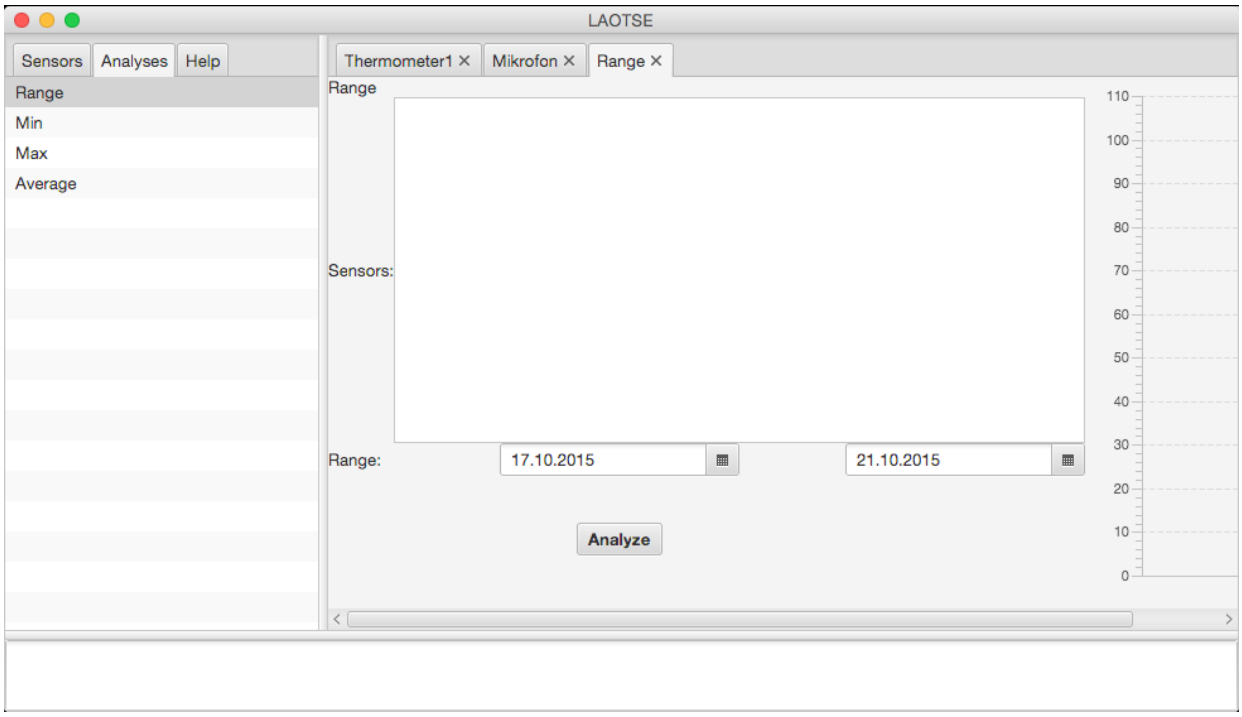


Abbildung 6.28.: Screenshot der Detailansicht einer Analyse im ersten Prototypen

In Abbildung 6.28 ist der Screenshot der Detailansicht einer Analyse zu sehen. In dem weißen Feld wird später auswählbar sein, über welche Sensoren die Analyse durchgeführt werden soll. In diesem Prototypen ist dieser Parameter noch hartkodiert. Es kann aber bereits ein Zeitraum gewählt werden, über den die Analyse gestartet wird. Dieser soll später feingranularer spezifiziert

werden können. Die Ergebnisse, die im Prototypen noch in der Konsole angezeigt werden, können später rechts in einem Diagramm dargestellt werden.

6.10.4. Änderungen an der GUI

KB

Dieser Abschnitt behandelt Änderungen an der GUI, die im Verlauf des Projekts beschlossen wurden und somit den Wandel vom ersten Prototypen, bis hin zur fertigen GUI beschreiben. Dementsprechend wird in diesem Abschnitt vor allem dargestellt, welche Änderungen am ersten Prototypen gemacht wurden und aus welchem Grund diese Änderungen beschlossen wurden und was diese beschlossenen Änderungen für das Projekt bedeuten.
Der entwickelte Prototyp, der zur Hälfte der Projektlaufzeit bereitgestellt wurde hat die ersten groben Funktionalitäten bereitgestellt um Sensordaten aus Druid auszulesen. Allerdings war zu diesem Zeitpunkt noch keine geeignete Visualisierung dieser Daten vorhanden. Außerdem waren Testsensoren in der GUI aufgeführt, die mit den realen Sensoren verlinkt wurden. Die Originalsensoren konnten im Prototypen noch nicht angezeigt werden. Somit musste eine Anbindung an die Datenbank des Systems geschaffen werden. Mit Hilfe dieser Datenbank können die Daten aller für das System vorhandenen Sensoren ausgelesen werden und diese Sensoren dementsprechend angezeigt und die dazugehörigen Daten visualisiert werden.
Des Weiteren konnten im Prototypen die Daten in Form von verschiedenen Analysen ausgelesen werden. Die vier Analysetypen, um Analysen auf den Daten auszuführen, sind Maximum, Minimum, Durchschnitt und Range. Da diese Formen allerdings nicht Analysen im eigentlichen Sinne entsprechen und eine korrekte Analyse von Daten sehr viel mehr Aufwand bedeuten würde, soll für die Analyse dieser Daten die Programmiersprache R genutzt werden, da diese Programmiersprache viel bessere Methoden für Analysen auf einem Datensatz bietet.
Dementsprechend wurden die Analysen aus der GUI entfernt. Die bereits vorhandenen Analysetypen werden allerdings weiterhin genutzt und zwar in Form eines Dashboards, dass dem Benutzer die Daten visualisieren soll, um zu zeigen wo Maxima oder Minima im Datensatz sind oder einfache Schlüsse über den gegebenen Datensatz zu ziehen. Dabei stehen zwei Dashboards zur Verfügung, eines bei dem eine Granularität angegeben werden kann, um die Daten entsprechend zu aggregieren und eines um einen Wert anzugeben, wieviele Daten des Datensatzes angezeigt werden sollen.
Außerdem existierte im Prototypen noch keine Anbindung an mosaik. Diese wurde im weiteren Projektverlauf entwickelt, sodass nun auch ein mosaik-Szenario gestartet und die Daten der einzelnen Sensoren des Szenarios aus Druid abgerufen werden können. Dazu wurde eine mosaik-View in der GUI geschaffen, mit Hilfe derer es möglich ist ein ausgewähltes Szenario zu starten und die entsprechenden Sensoren im System anzulegen.

6.10.5. Aufbau und Funktion der GUI

KB

Die GUI ist die grafische Schnittstelle für den Benutzer, mit Hilfe derer der Benutzer die Kontrolle über alle Funktionen hat, die das fertige Produkt bereitstellt. Sie ist das Steuerungswerkzeug, um dem System bspw. neue Sensoren hinzuzufügen oder bereits vorhandene Sensoren zu ändern und sich zu diesen Sensoren gemessene Werte anzuschauen. Aufgrund dieser Anforderungen muss die

GUI dadurch übersichtlich gestaltet werden, sodass sich auch ein neuer Benutzer sofort zurecht findet, aber sie muss dennoch alle geforderten Funktionen bereitstellen und dies auf eine Art und Weise, dass der Benutzer intuitiv weiß wo er Daten eingeben oder klicken muss, um eine bestimmte Aktion auf dem System auszuführen. Daher lag bei der Entwicklung der GUI das Hauptaugenmerk auf der intuitiven Bedienbarkeit und nicht auf der Komplexität der Funktionen, sodass die Funktionen nur auf einfachste Art und Weise abgebildet wurden.

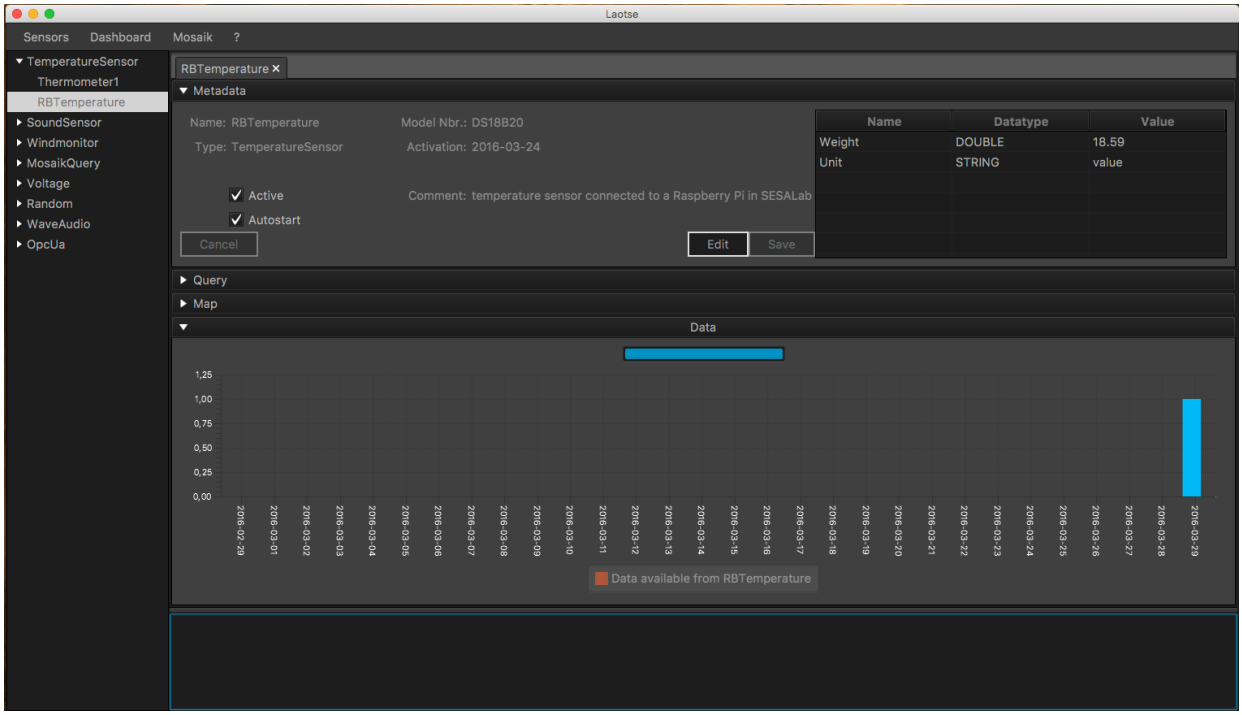


Abbildung 6.29.: Screenshot der Detailansicht eines Sensors

Abbildung 6.29 zeigt die Sensoransicht, wie sie in der fertigen GUI ist. Im Vergleich zum ersten Prototypen fällt zunächst einmal der Wechsel vom grellen Weiß zum etwas milderen Schwarz auf. Von den grundlegenden Funktionen hat sich in dieser Ansicht, im Vergleich zum ersten Prototypen, nicht ganz viel getan. Die Ansicht ist nun etwas variabler gehalten. So können einzelne Teile der Ansicht per Klick auf die Überschrift ausgeblendet werden. Die Übersicht ist nun in 2 Teile aufgeteilt. Einmal existieren, wie bereits im Prototypen, die Metadaten, die beispielsweise anzeigen, wo der Sensor liegt, in welcher Einheit der Sensor Daten liefert und Ähnliches. Der zweite Teil der Übersicht ist ein neues Diagramm, das so genannte Data Availability Chart (DAC) ist neu hinzugekommen. Dieses Diagramm zeigt an, an welchen Tagen der angezeigte Sensor Daten geliefert hat. Dabei werden immer die letzten 7 Tage angezeigt. Wenn man auf einen der Balken klickt bekommt man eine Übersicht über den kompletten Tag, auf den geklickt wurde. In dieser Übersicht wird im Stundentakt über den Tag verteilt angezeigt in welchen Intervallen der Sensor Daten geliefert hat. Auf einen dieser Balken kann erneut geklickt werden und mit diesem Klick wird dann ein neues Dashboard geöffnet. Das Dashboard hat, wie im vorherigen Abschnitt bereits beschrieben, die einzelnen Analysen aus dem ersten Prototypen abgelöst und zeigt nun alle Analysetypen auf einen Blick an.
In der Abbildung 6.30 wird die Ansicht eines Dashboards gezeigt wie es nach dem Öffnen aussieht. Im Bereich Options können die Daten für die Analyse eingetragen werden. So muss hier ein Start- und ein Enddatum für die Analyse ausgewählt werden. Außerdem ist es im Gegensatz

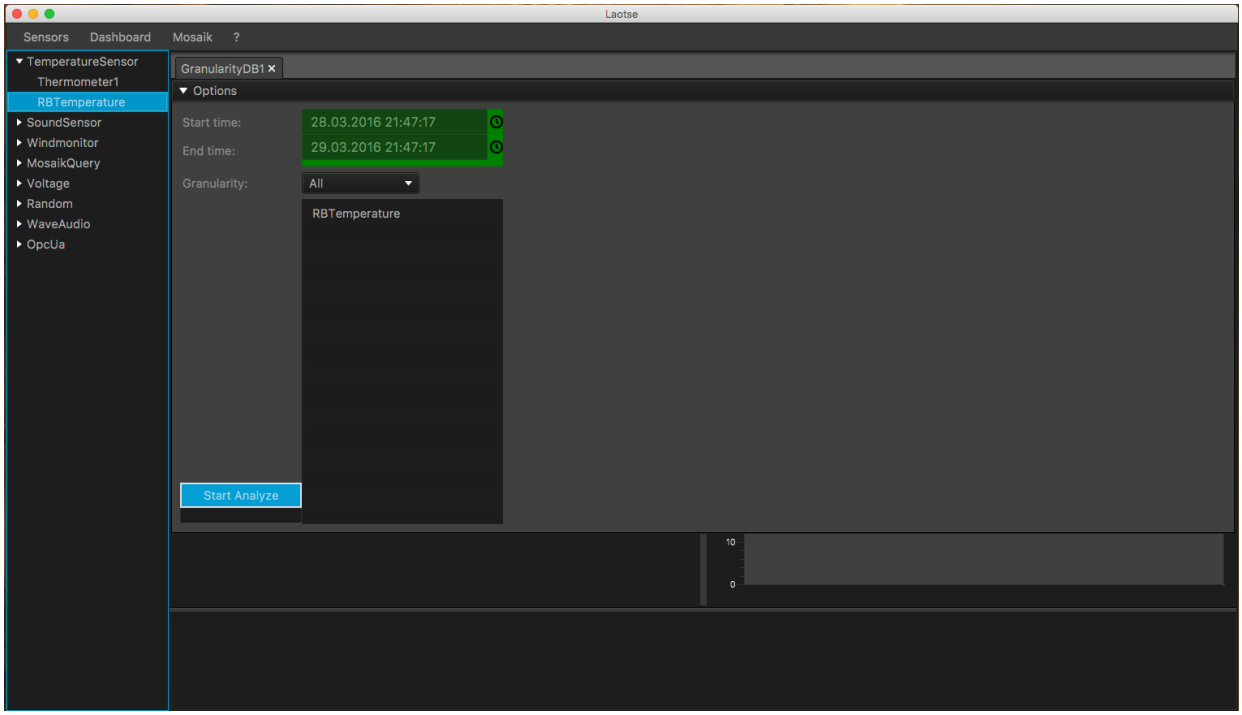


Abbildung 6.30.: Screenshot eines Dashboards mit ausgeklappten Options

zum Prototypen nun möglich eine Zeit mit anzugeben. Für die Angabe der Zeit sind 4 Spinner vorgesehen, mit Hilfe derer die Zeit für das Intervall angegeben werden kann. Auf diese Art und Weise kann die Zeit auf die Nanosekunde genau angegeben werden und somit ist eine deutlich genauere Analyse der Daten möglich. Unter diesen Auswahlfeldern für das Intervall befindet sich die Sensorliste, die bereits aus dem Prototypen bekannt ist. Also auch in einem Dashboard ist es möglich eine Analyse über mehrere Sensoren durchzuführen. Allerdings müssen die Sensoren vom gleichen Sensortyp sein, da sie für eine vernünftige Visualisierung ähnliche Daten liefern müssen. Sind alle Daten für die Analyse angegeben kann mit einem Klick auf den Button „Start Analyze“ die Analyse gestartet.

Wurde die Analyse gestartet, wird automatisch der Bereich für die Options ausgeblendet, damit mehr Platz für die Visualisierung der Ergebnisse vorhanden ist (siehe Abbildung 6.31). Im Prototypen war dies noch nicht der Fall und das Diagramm für die Range Analyse konnte dementsprechend kaum vernünftig angezeigt werden. Im Dashboard ist es nun so, dass das Diagramm nahezu den kompletten Bildschirmbereich zur Verfügung hat und damit gut angezeigt werden kann. Für die anderen Analysen stehen Labels zur Verfügung. Die Ergebnisse dieser Analysen werden neben dem Diagramm angezeigt. Somit erhält der Benutzer alle Ergebnisse auf einen Blick. Auch der Options Bereich ist nicht dauerhaft ausgeblendet, sondern kann mit einem Klick auf die Überschrift wieder eingeblendet werden, um Änderungen an den Daten oder der Sensorliste vorzunehmen. Dementsprechend kann das Dashboard immer so verändert werden, dass man sich andere Daten anzeigen lassen kann und für solche Fälle nicht extra ein neues Dashboard geöffnet werden muss.

In der Menüleiste existiert ein Menüpunkt „Mosaik“. Über diesen Menüpunkt kann eine neue mosaik-View geöffnet werden.

Wie eine solche mosaik-View aussieht zeigt Abbildung 6.32. Diese zeigt auf der linken Seite den

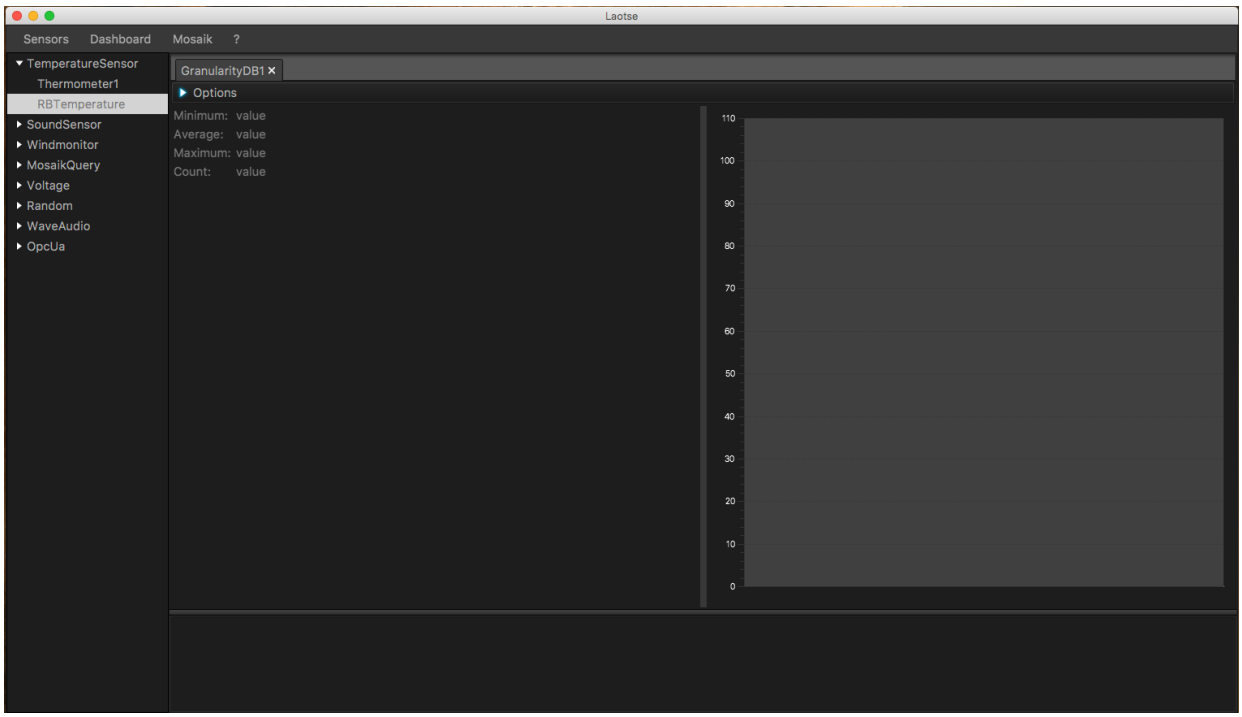


Abbildung 6.31.: Screenshot eines Dashboards

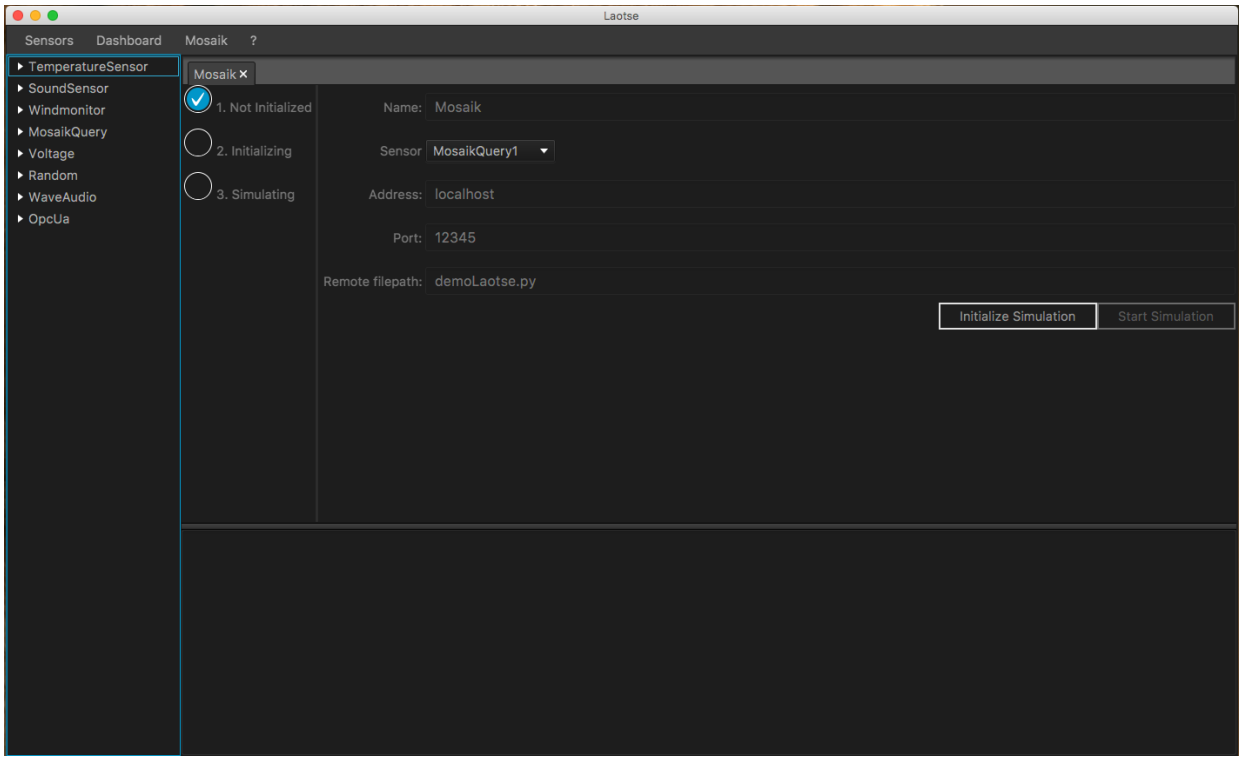


Abbildung 6.32.: Screenshot einer mosaik-View

aktuellen Status der mosaik-Simulation. Auf der rechten Seite der View können die Parameter für die Simulation angegeben werden. Hier existieren Parameter für den Namen der View, den Mosaik Sensor für das Szenario, unter dem die Sensoren des Szenarios gespeichert werden sollen,

die mosaik-Adresse und den Port, unter dem mosaik zu erreichen ist. Außerdem muss noch der Pfad zum mosaik-Szenario angegeben werden. Mit einem Klick über die Buttons kann dann die Initialisierung oder die Simulation des Szenarios gespeichert werden.
Von den grundlegenden Funktionen hat sich im Gegensatz zum Prototypen nicht all zu viel getan. Allerdings wurde an der Benutzerfreundlichkeit und der Übersichtlichkeit der GUI gearbeitet. Außerdem wurden weitere Details visualisiert und ermöglichen es zum Beispiel für einen Sensor stundenweise Daten anzuzeigen. Außerdem wurde die Datenbank an die GUI angebunden. Das bedeutet, dass nun auch die Daten der angeschlossenen Sensoren geladen werden und nicht irgendwelche Beispieldaten.

7. Implementierung

LS

Dieses Kapitel beschreibt die grundsätzliche Implementierung von LAOTSE und von einigen weiteren Programmen, die während der Entwicklung erstellt wurden.
Da LAOTSE auf OSGi basiert ist es sehr modular aufgebaut. Es gibt drei Hauptkomponenten, denen alle LAOTSE-Javaprojekte untergeordnet sind. Im Folgenden wird nach diesen drei Projekten, Server, Client, Common aufgeteilt und jeweils alle untergeordneten OSGi-Bundles in ihrer Funktion und ihrem Aufbau beschrieben.
Es folgt eine Beschreibung der Beziehungen zwischen den Bundles und eine Erläuterung der Anpassungen von Drittprogrammen, sowie die genutzten Hilfsprogramme.

7.1. LAOTSE Client

Dieser Abschnitt beschreibt alle Bundles, die zum Laotse-Client gehören.

7.1.1. Client Communication

LS

Das OSGi-Bundle Client Communication erfüllt die Aufgabe der Kommunikation zwischen der LAOTSE-GUI, beschrieben in Abschnitt 7.1.2 und dem Laotse Server siehe Abschnitt 7.3.2 wozu im Speziellen die Klasse `LaotseWebSocketClient` verwendet wird.
Diese erbt vom Client der WebSocket-Bibliothek `org.java-websocket` und implementiert die Interfaces `ConfigClient` sowie `MosaikClient`, welche befähigen Nachrichten bezüglich Konfigurationsänderungen bzw. Mosaikanfragen zu versenden, wie zu sehen im Klassendiagramm 7.1.

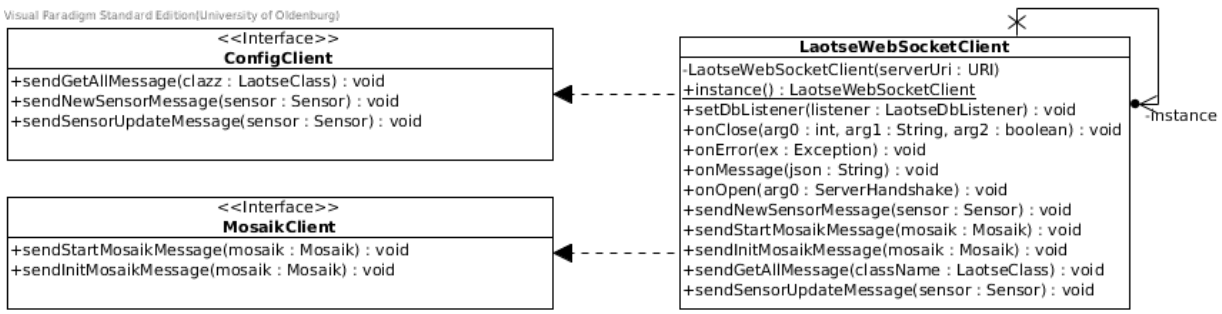


Abbildung 7.1.: WebSocketClient Klassendiagramm

Des Weiteren bietet der `LaotseWebSocketClient` die Möglichkeit, Listener für Mosaik-/Konfigurationsänderungen zu registrieren, die, im Falle des Empfangs einer Nachricht vom Server über Änderungen dieser Modelle, aufgerufen werden.
Zur Serialisierbarkeit von Nachrichten zwischen Server und Client wird Json genutzt. In der `onMessage(json)` Methode wird solch eine Nachricht deserialisiert und vom Client entsprechend verwertet, indem die Listener benachrichtigt werden.
Der `LaotseWebSocketClient` ist als Singleton-Pattern realisiert, da es pro LAOTSE-GUI aus Synchronisierungsgründen nur jeweils einen solchen Client geben sollte. In der `instance()` Methode des Client wird die aktuelle Verbindung überprüft und bei Verbindungsausfall ggf. neu erstellt, um solcherlei Fehler tolerieren zu können. Ein detaillierter Ablauf der Kommunikation zwischen Server und Client ist im Abschnitt 7.4.3 mit Hilfe von Abbildung 7.13 beschrieben.

7.1.2. Laotse GUI

MM, KB

Dieser Abschnitt beschreibt die Implementierung der LAOTSE-GUI grob. Aufgrund des Umfangs der Implementierung dieser wird dabei der Fokus auf die Architektur und einige Konzepte gelegt.

FXML - Controller

MM

In JavaFX wird jede View, die durch eine *.fxml*-Datei implementiert wird, von einem FxmlController kontrolliert. Dieser ist als Java-Klasse implementiert. Kontrollieren bedeutet dabei, dass die Daten in den Elementen der View gesetzt sowie bearbeitet werden und Nutzereingaben und andere Ereignisse in der GUI verarbeitet werden können. Der FxmlController implementiert die Logik der GUI, die damit deutlich von der View getrennt ist. Das Modell, das die Daten enthält, ist ebenfalls separat implementiert.
In LAOTSE gibt es zwei abstrakte Controller: Der `Controller` und der `ControllerWithModel`. Dabei bietet der `ControllerWithModel` verschiedene Möglichkeiten ein vorhandenes Modell anzuzeigen, zu bearbeiten bzw. ein noch nicht vorhandenes Modell neu anzulegen. In letzterem Fall wird die entsprechende View, in der die Daten des neuen Modells eingegeben werden sollen, in einem neuen Fenster geöffnet. In 7.1.2 wird beschrieben, welche Elemente der GUI von welcher abstrakten Klasse erben.
Die Klasse `FxmlControllerLoader` bietet die Möglichkeit den `Controller` und die View eines beliebigen Modells zu laden. Dabei wird in Konfigurationsdateien angegeben, zu welchem Modell welche View gehört. Es ist auch möglich, eine View und den `Controller` ohne ein Modell zu laden. Es werden beim Ladevorgang automatisch die View und ggf. das Modell beim Controller registriert.

FxThreadUtil

KB

Die Klasse `FxThreadUtil` beinhaltet eine Methode, mit der sicher gegangen werden kann, dass die GUI gezeichnet werden kann. Dazu wird der UI-Thread genutzt, da in diesem die GUI gezeichnet wird. Innerhalb dieses Threads können dann alle Operationen für das Zeichnen der GUI aufgerufen werden. Diese Methode wird benötigt, weil die JavaFX-GUI nicht verändert werden darf, falls man sich in einem anderen Thread als dem UI-Thread befindet. Das bedeutet, dass sicher gestellt werden muss, Veränderungen an der GUI immer aus dem UI Thread heraus zu starten. Genau in solch einem Fall wird diese Methode benötigt, denn in diesem System werden an einigen Stellen Operationen in anderen Threads ausgeführt, um so eine höhere Performanz gewährleisten zu können. Anfragen, die an Druid gestellt werden, können zum Beispiel sehr rechenaufwendig werden. Das würde dann für die Progress-Bar in der GUI bedeuten, dass diese erst zu einem späteren Zeitpunkt neu gezeichnet werden würde. Aus diesem Grund wird diese Operation im UI-Thread ausgeführt, damit nicht auf Druid gewartet werden muss.

Architektur der GUI

MM

In diesem Abschnitt wird die Architektur der GUI beschrieben.

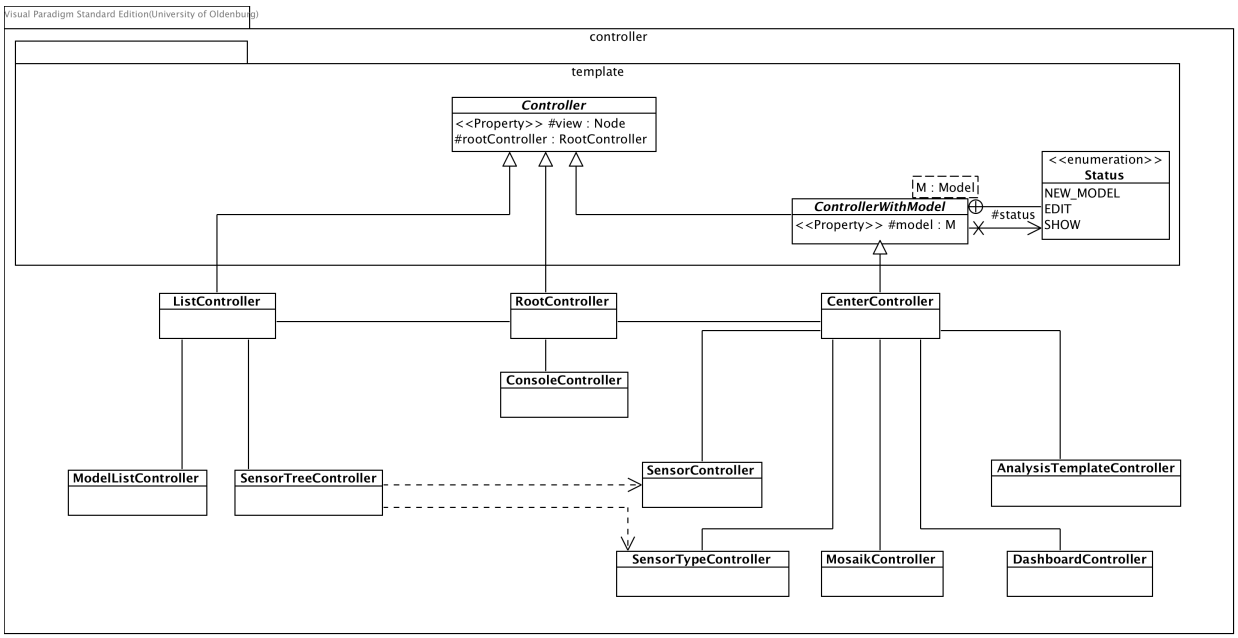


Abbildung 7.2.: Vereinfachte Architektur der Controller der GUI

In Abbildung 7.2 ist die vereinfachte Architektur der FXML-Controller der GUI dargestellt, die im Abschnitt 7.1.2 beschrieben wurden. Hier folgt eine Erläuterung der einzelnen Controller und ihrer Beziehungen.
Die Aufteilung des Fensters wird in der `RootView` implementiert, die vom `RootController` gesteuert wird. Dieser ordnet in seiner View die `CenterView`, die `ConsoleView` und

die `ListView` an. Die `RootView` hat also kein eigenes Modell, sondern ist lediglich ein Container, der andere Views beinhaltet. Der `RootController` hält Referenzen zu den Controllern der beinhalteten Views. Das Laden dieser Views und Controller erfolgt durch den in 7.1.2 beschriebenen `FxmlControllerLoader` und wird durch die Klasse `MainApp` initialisiert und gesteuert. Diese Klasse handhabt ebenfalls den Lebenszyklus der `Scene`, die das auf dem Bildschirm angezeigte Fenster implementiert und die `RootView` beinhaltet. An die `RootView` wird eine *.css*-Datei gebunden, die das Layout der LAOTSE-GUI definiert. Dadurch kann auch die optische Darstellung der GUI einfach durch einen css-Designer angepasst werden. Auch zur Laufzeit werden einige css-Klassen für verschiedene GUI-Elemente gesetzt und entfernt um die Darstellung zu verändern. In der *.css*-Datei, ist ein einheitliches Farbschema definiert. Die verschiedenen Effekte werden durch relative Anpassung z.B. einiger Grundfarben erreicht.
Der `ListController` stellt ebenfalls kein eigenes Modell dar, sondern dient als Container für verschiedene Listen von Modellen. Eine solche Liste besitzt allerdings eine View, die in der GUI dargestellt wird. Dabei hält auch der `ListController` eine Referenz auf die View und den Controller der aktuell angezeigten Liste, so dass direkt auf diese zugegriffen werden kann. Diese aktuell angezeigte View wird auch „aktuelle List-View “ genannt.
Der Controller der aktuellen List-View kann eine Instanz des `ModelListController` oder `SensorTreeController` sein. Dabei kann die View eines `ModelListController` eine Liste an beliebigen Modellen darstellen. Der `SensorTreeController` ist spezialisiert auf die Darstellung eines Baums an Sensoren. Für die konkreten Controller der Listen, wie z.B. den Controller der Liste der Analysetypen, den Controller der Sensortypen oder dem Controller des Baums der Sensoren, wird innerhalb des `ListController` jeweils ein Singleton-Pattern [Sin] genutzt. Es wird eine Methode zur Verfügung gestellt, die den jeweiligen Controller liefert und gleichzeitig die entsprechende View in der `ListView` anzeigt. Dadurch wird sichergestellt, dass immer die aktuelle Version des Controllers und der View zur Verfügung steht und gleichzeitig nur eine Liste dargestellt werden kann.
Die Modelle der einzelnen Listen können über das gemeinsame Modell `Lists` erreicht werden, welches nur im Client genutzt wird, um Zugriff z.B. auf die Liste aller Sensoren zu gewährleisten. `Lists` wurde nach dem Instance-Pattern implementiert und nutzt dieses Pattern auch für alle Listen, die es beinhaltet. Bei der Initialisierung der Instanzen werden entsprechende Anfragen an den Server übersandt, die die Listen aus der LAOTSE-Datenbank lädt.
Der `CenterController` ist ebenso ein Container für verschiedene Controller und Views. In diesem können jedoch mehrere verschiedene Views gleichzeitig angezeigt werden. Außerdem wird hier auch ein eigenes Modell genutzt. Er nutzt eine `ModelList`, die eine Liste an Modellen implementiert und zur Laufzeit von LAOTSE die Modelle enthält, deren Views jeweils aktuell in der `CenterView` dargestellt werden. Die `ModelList` bietet Funktionen zur Verwaltung der Observer der Modelle der Liste und implementiert selbst ein Modell (vgl. dazu Abschnitt 7.2.2).
Der `CenterController` nutzt `Tabs`, die Views in Tabs in einem `TabPane` anordnet und jeweils die ID eines Modells zuordnet. So wird für die View jedes Modells ein Tab genutzt, dessen Inhalt von dem jeweiligen Controller gesteuert wird.
Diese Controller sind:

- `SensorController`: Steuert die `SensorView` und damit die Anzeige der Metadaten und die Datenübersicht eines Sensors. Es können Metadaten bearbeitet und neue Sensoren angelegt werden, je nach aktuellem Status.

- `SensorTypeController`: Steuert die `SensorTypeView` und damit die Anzeige der Metadaten eines Sensortyps.

- `MosaikController`: Steuert die `MosaikView` und bietet die Möglichkeit, eine Simulationsdatei für Mosaik zu wählen, die entsprechende Simulation zu initialisieren und zu starten. Der Status der Initialisierung und der Simulation wird graphisch in einem Ablauf dargestellt, so dass der Nutzer den aktuellen Zustand erkennen kann. Dazu werden Updates aus dem Server verarbeitet, die den aktuellen Status übermitteln.

- `AnalysisTemplateController`: Steuert die `AnalysisTemplateView` und damit die Anzeige der Metadaten einer Analyse, die im Dashboard genutzt werden kann.

- `DashboardController`: Steuert die `DashboardView`. Es können die Parameter für die Analysen eingegeben und die Analyseergebnisse visualisiert werden. Die Analysen können beliebig oft durchgeführt werden. Nach jeder Analyse wird die Visualisierung im Dashboard aktualisiert. Dieser Controller steuert außerdem die Anbindung an die Druid-Komponente von LAOTSE. Es werden also die eingegebenen Parameter für die Analysen verarbeitet und die fertigen Analysen an die Druid-Komponente übermittelt und die Ergebnisse aus der Druid-Komponente empfangen und visualisiert.

Jeder `ControllerWithModel` befindet sich zur Laufzeit in einem bestimmten Status. Dieser Status wird durch ein Enum implementiert. Der Status kann folgende Ausprägungen annehmen:

`NEW_MODEL` Es gibt noch kein Modell und die GUI wird dazu genutzt, ein neues Modell zu erstellen.

`EDIT` Die GUI kann dem Status `NEW_MODEL` ähneln, allerdings wird hier ein bereits vorhandenes Modell bearbeitet.

`SHOW` Die GUI zeigt die Daten des Modells an. Das Modell kann so nicht bearbeitet werden.

Der Controller passt bei einer Änderung des Modells die View entsprechend an. In Abbildung 7.3 ist in einem Zustandsdiagramm dargestellt, welche Interaktionen zwischen des Stati bestehen.

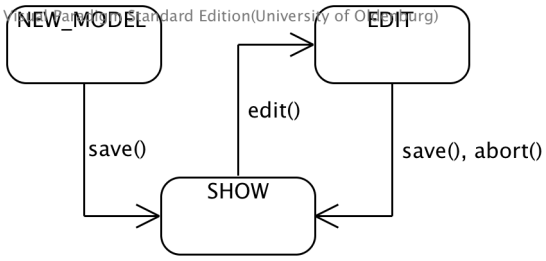


Abbildung 7.3.: Zustandsdiagramm der `ControllerWithModel`

Es stehen einige Controller in Interaktion zueinander. Diese Interaktion kann z.B. die Form annehmen, dass durch eine Benutzereingabe auf der `SensorTreeView` eine bestimmte `SensorView` in der `CenterView` geöffnet wird. Jede Interaktion, zwischen verschiedenen Controllern wird von dem `RootController` derart zentral gesteuert, dass beliebige Controller dem `RootController` Anfragen und Aufträge übermitteln können. Dieser übermittelt sie entsprechend der Art der Aufgabe an den `ListController`, `CenterController` oder `ConsoleController` weiter, die wiederum an entsprechende untere Controller weitervermitteln. Eventuelle

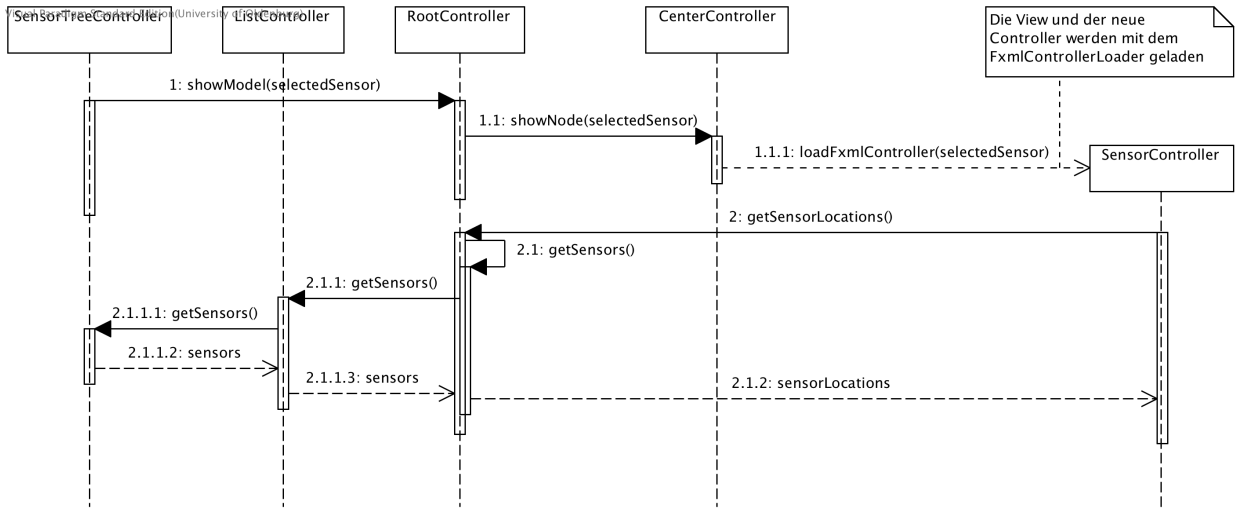


Abbildung 7.4.: Kommunikation zwischen verschiedenen Controllern, hier beispielhaft: Öffnen und Initialisieren einer SensorView

Ergebnisse, wie die Liste von bestimmten Sensoren wird entsprechend dieser Hierarchie wieder zurückgegeben.
In Abbildung 7.4 ist die Kommunikation zwischen verschiedenen Controllern am Beispiel des Öffnens und Initialisierens einer neuen SensorView dargestellt. Diese Interaktion wird vom `SensorTreeController` gestartet, indem der Benutzer in der View die Metadaten eines Sensors öffnet. Das Modell des Sensors wird an den `RootController` übergeben, der das Anzeigen der entsprechenden View an den `CenterController` delegiert. Wenn die View dieses Modells noch nicht vorhanden ist, wird sie geladen. Im Anschluss wird sie in der `CenterView` als Tab angezeigt. Beim Laden eines `SensorController` werden die Standorte aller Sensoren ausgelesen, da diese in einer Karte dargestellt werden. Die entsprechende Anfrage wird wieder über den `RootController` und den `ListController` an den `SensorTreeController` übergeben, der die Sensoren, bzw. ab dem `RootController` die Standorte dieser, zurückgibt. Andere Anfragen verlaufen analog.

Dashboard

KB, DN

Das Dashboard ist für die Visualisierung der Daten aus Druid verantwortlich. Das Senden der Anfragen an Druid und die Visualisierung der zurückgegebenen Daten wird dabei vom Controller übernommen. Da im System allerdings zwei verschiedene Dashboards existieren, existieren auch verschiedene Controller.

Im System existiert ein `ThresholdDashboard` und ein `GranularityDashboard`. Im `ThresholdDashboard` kann ein Wert übergeben werden, wieviele Daten maximal angezeigt werden sollen. Gibt man also beispielsweise den Wert 1000 an, dann werden maximal die ersten 1000 Werte, die von Druid kommen visualisiert. Liefert Druid weniger als 1000 Daten werden entsprechend weniger Daten angezeigt. Dabei wird ausschließlich die Range Analyse durchgeführt, bei dem die Daten im Diagramm angezeigt werden. Die Daten werden in Druid als Strings gespeichert. Dabei werden die Zeilen zuerst nach ihrem Zeitstempel sortiert. Danach ist die Orn-

dung nicht mehr wohl definiert. Wenn Zeilen mit dem gleichen Zeitstempel, also innerhalb der selben Millisekunde liegen, ist die Ordnung nicht sichergestellt. Die Dimension 'ns' wird hier zur feineren Sortierung genutzt. Damit die Daten vollständig sortiert werden können, müssen alle Daten eines Zeitstempels zur Verfügung stehen, da sonst Lücken in der Datenreihe entstehen können. Das Dashboard schneidet deshalb auch Daten ab, die nicht sicher vollständig nebeneinander stehen. Das kann höchstens in der letzten Millisekunde des abgefragten Intervalls auftreten.

Dem `GranularityDashboard` kann anstatt eines Thresholds eine Granularität übergeben werden. Das bedeutet für die Daten, dass generell alle Daten angezeigt werden, diese jedoch anhand der gegebenen Granularität aggregiert werden. Wenn Beispielsweise `Hour` als Granularität ausgewählt wird, dann werden die Daten stundenweise aggregiert und im Diagramm pro Stunde ein Wert angezeigt. Für die anderen Analysen gilt diese Granularität jedoch nicht. Hier werden alle Werte betrachtet und zum Beispiel das Minimum oder Maximum ermittelt.

Dementsprechend gibt es für beide Dashboards verschiedene Controller. Da sich diese allerdings sehr ähneln, existiert der abstrakte `DashboardController`, der die grundsätzlichen Methoden der Analyse beinhaltet. Die einzelnen Controller erben von diesem Controller und implementieren selbst nur die Methoden, die von dem `DashboardController` abweichen, also diejenigen, die entweder die Granularität und den Threshold betreffen. Beide Controller nutzen allerdings die gleiche View. Im `GridPane` existieren dabei an der gleichen Stelle Labels und Textfelder, sowohl für den einen, als auch für den anderen Controller. Diese werden je nachdem in welchem Dashboard man sich befindet ein- und ausgeblendet.

Checker

KB

Innerhalb des Checker Paketes existieren 2 Klassen. Zum einen der `Checker` und zum anderen der `TextInputChecker`. Beide sind in ihrer Gesamtheit dafür zuständig, zu überprüfen, ob ein vom Benutzer eingegebener Wert valide ist oder nicht. In diesem System können dabei jedoch nicht alle von Java unterstützten Datentypen auf ihre Richtigkeit überprüft werden, sondern nur diejenigen, die im System wirklich genutzt werden.
Diese sind Boolean, Double, Integer, String und Long. Vom Benutzer kommt allerdings immer nur ein String im `Checker` an. Um zu überprüfen, ob der Wert einem gegebenen Datentypen entspricht, wird dabei einfach überprüft, ob sich der Wert in diesen Datentypen umwandeln lässt. Falls sich der Wert in diesen Datentypen umwandeln lässt , wird true zurückgegeben, denn der Wert ist in diesem Fall valide. Lässt sich der Wert nicht in diesen Datentypen umwandeln, wird false zurückgegeben.
Allerdings muss auch für den Benutzer visualisiert werden, ob ein Wert valide ist oder nicht, damit der Benutzer den Wert gegebenenfalls anpassen kann. Für diese Visualisierung ist der `TextInputChecker` zuständig. In diesem sind Klassen definiert, die ein Textfeld, je nachdem ob der Wert valide oder nicht valide ist, stylen. Auf diese Art und Weise können also Textfelder zum Beispiel rot oder grün eingefärbt werden, um dem Benutzer zu zeigen, dass sein eingegebener Wert korrekt oder nicht korrekt ist.
Dem `TextInputChecker` können mehrere Textfelder übergeben werden, die von diesem zu überprüfen sind. Aus diesem Grund existieren im `TextInputChecker` sowohl Methoden, um zu überprüfen, ob alle Textfelder, die im `Checker` registriert wurden, auf Validität zu überprüfen,

als auch Methoden um ein einzelnes Textfeld auf Validität zu überprüfen. Außerdem existieren Methoden um den Status eines Textfelds zu setzen. Das bedeutet, dass mit diesen Methoden gesetzt werden kann, dass das Textfeld valide, nicht valide oder normal ist. Das bedeutet für das Textfeld selber, dass es dementsprechend grün, rot oder ohne spezielle Farbe für den Client erscheint. eingegebener Wert valide ist oder nicht.

7.1.3. Client Properties

Der Client bietet verschiedene *.properties*-Dateien, deren Einstellungen durch jeweils eine Implementierung in Java repräsentiert werden. Es können z.B. Verbindungsdaten des Websockets, Einstellungen für die GUI und Standardwerte für mosaik konfiguriert werden. Dabei erbt jede Klasse, die genau eine *.properties*-Datei verwaltet, von der Klasse `ClientProperties`. Diese Architektur ist in Abschnitt 7.2.5 genauer beschrieben.

7.2. LAOTSE Common

Dieser Abschnitt beschreibt alle Bundles in der vom Server und Client geteilten Common Komponente.

7.2.1. Communication Messages

LS

Das `Messages` Bundle dient als gemeinsame Kommunikationsgrundlage zwischen `LaotseWebSocketClient` (vgl. Abschnitt 7.1.1) und `LaotseWebSocketServer` (vgl. Abschnitt 7.3.2).
Es besteht aus verschiedenen Java Klassen, die jeweils eine Nachricht bzgl. einer Anfrage oder eines Ereignisses repräsentieren.
Handelt es sich hierbei um die Veränderung oder Erstellung eines konkreten Modells wird dieses in der Nachricht mitgeliefert, z.B. wird ein `Sensor`-Objekt in der `NewSensorMessage` mitgeliefert.
Um zwischen Server und Client versendet werden zu können, müssen die Nachrichten allerdings in der aktuellen Implementierung in textueller Form serialisiert werden, wozu das Json Format und `Gson` als Parser gewählt wurden.
Die Klasse `JsonUtils` enthält die Methode `getJsonString(message)`, die aus einer beliebigen `Message` ein Json-serialisiertes String Array mit 2 Feldern erzeugt: Das erste Feld enthält den einfachen Namen der jeweiligen Nachrichtenklasse, um eine schnelle Identifizierung zu erleichtern und das zweite Feld enthält die json-serialisierte Form des Nachrichtenobjektes, welche mit Kenntnis der Klasse deserialisiert werden kann.

7.2.2. Laotse Model

KB

Das Bundle Laotse Model beinhaltet alle Models, die im System benötigt werden. So existiert in diesem Bundle für jede View des Systems ein Model, da in diesem Projekt nach dem Model-View-Controller Prinzip gearbeitet wird. Außerdem existieren hier noch zusätzliche Models für den Datenbankzugriff.
Da alle Models gleich aufgebaut sein müssen, existiert in diesem Bundle auch eine abstrakte Model Klasse, die vorgibt wie die anderen Model Klassen aufgebaut sein müssen. Außerdem beinhaltet dieses Bundle einige Exceptions, die geworfen werden, falls mit den verschiedenen Models etwas falsch läuft. Die meisten Fehler können beim Datenbankzugriff auftreten, da in diesem Fall immer wieder eine Verbindung zur Datenbank aufgebaut werden muss. Aus diesem Grund sind die meisten Exceptions auch für den Datenbankzugriff vorhanden.
Des weiteren existieren verschiedene Interfaces, die beschreiben welche Methoden bei einer Verbindung zur Datenbank oder zu Mosaik implementiert werden müssen. So existiert das Interface DatabaseAccess, dass Methoden für die Herstellung und die Schließung einer Verbindung zur Datenbank vorsieht. Ist die Verbindung hergestellt, können Anfragen an die Datenbank gesendet werden. Für die Models steht dafür das Interface ModelDatabaseAccess zur Verfüfung. Dieses Interface sieht Methoden für das einfügen, abrufen und updaten von Models vor.
Ein weiteres Interface ist für die Kommunikation mit Mosaik zuständig. Hier werden Methoden für einen Listener definiert, wie mit einem Update, das von Mosaik kommt, umgegangen wird.
Dieses Bundle definiert also alle für das System relevanten Models. Hinzu kommen die Interfaces, die für die Kommunikation mit Mosaik und der Datenbank definiert wurden. Außerdem sind in diesem Bundle zahlreiche Exceptions definiert, die im Fehlerfall geworfen werden können.

7.2.3. Druid Wrapper

KB

Mit Hilfe des `DruidDbWrappers` wird die Anbindung des Clients an den Brokerknoten von Druid realisiert. Das bedeutet, dass dieser Wrapper dafür zuständig ist, die Anfragen, die der Client an Druid stellen möchte, an Druid weiterzugeben und das daraus resultierende Ergebnis wieder zurück an den Client zu geben. Er organisiert die Kommunikation zwischen dem Client und Druid. Diese Kommunikation wird realisiert durch den `DruidDbWrapperService` und den `DruidDbWrapper`.
Der `DruidDbWrapperService` ist die Schnittstelle zwischen dem Client und dem `DruidDb-Wrapper`, damit der Client nicht direkt auf Druid zugreifen kann. Der Service ist also im Grunde die Schnittstelle, an die der Client seine Anfragen sendet und der das von Druid zurückgegebene Ergebnis für den Client so umwandelt, dass dieser das Ergebnis weiter verwerten kann. Der Service empfängt die Anfrage vom Client, gibt diese an den Wrapper weiter und bekommt nach der Anfrage das Ergebnis vom Wrapper, um es für den Client aufzubereiten. Die Query für Druid liegt dabei als JSON String vor.
Dieser JSON String wird vom Druid Wrapper über eine HTTP-Verbindung an den Druid-Brokerknoten gesendet. Für Druid existiert eine Java API. Diese konnte allerdings in diesem Projekt nicht genutzt werden, da sie veraltet ist und mit neueren Versionen von Druid nicht mehr nutzbar. Da in diesem Projekt die neueste Version von Druid genutzt wird (Version 0.9.0), musste auf eine

Java API verzichtet werden und stattdessen der komplexere Weg über eine HTTP-Verbindung genutzt werden.
Alle Anfragen, die dazu dienen konkrete Daten, die aufgezeichnet wurden, anzuzeigen basieren auf Timeseries-Queries von Druid. Diese sind speziell darauf ausgelegt, Daten über einen angegebenen Zeitraum abzurufen und basierend auf einer gegebenen Granularität zu aggregieren. Als Granularität kann dabei alles erdenkliche definiert werden. Zum Beispiel können so die Daten über eine Stunde, einen Tag oder eine Woche aggregiert werden. Dadurch können auch hochfrequente Daten auf vernünftige Art und Weise visualisiert werden.
Die Ergebnisse, die Druid liefert werden im `DruidDbWrapperService` von einem JSON Parser analysiert. Dabei ist wichtig, dass auf alle im System vorhandenen Analysen anders reagiert werden muss. Da Druid keine Durchschnittsberechnung zur Verfügung stellt, ist die `Average-Analyse` die komplizierteste des Systems. Das bedeutet, dass bei dieser Analyse die Summe und Anzahl berechnet werden muss und auf diesem Wege die Durchschnittsberechnung stattfindet.
Zusätzlich zu den Analysen, die im Dashboard angezeigt werden können, stellt das System zwei weitere Funktionen zur Verfügung, bei denen auf die Daten aus Druid zugegriffen werden muss. Das System kann für einen bestimmten Sensor anzeigen, an welchen Tagen und in welchen Stunden Daten für den Sensor vorhanden sind. Für diese Funktionen stehen zwei Anfragen zur Verfügung, mit Hilfe derer diese Daten abgefragt werden können.
Der Druid Wrapper ist also dafür zuständig, dem Client die benötigten Daten, die in Druid liegen, zu liefern und dem Client somit die Möglichkeit zu geben, die Daten, die von den Sensoren ermittelt wurden, zu visualisieren.

7.2.4. Model Properties

MM

Das Model bietet verschiedene *.properties*-Dateien, deren Einstellungen durch jeweils eine Implementierung in Java repräsentiert werden. Es können z.B. Metadaten der Analysen und deren Parameter, Verbindungsdaten zu Druid und Namen spezieller Sensortypen konfiguriert werden. Diese Sensortypen erfüllen in LAOTSE z.B. bei mosaik spezielle Aufgaben. Dabei erbt jede Klasse, die genau eine *.properties*-Datei verwaltet, von der Klasse `ClientProperties`. Diese Architektur ist in Abschnitt 7.2.5 genauer beschrieben.

7.2.5. Properties

MM

In diesem Abschnitt wird beschrieben, wie und warum die abstrakte Klasse `Properties` in LAOTSE genutzt wird.
Jedes Bundle, das konfigurierbare Einstellungen bietet, die in einer *.properties*-Datei angegeben werden sollen, enthält eine Klasse, die von der abstrakten Klasse `Properties` erbt. Zur Verarbeitung der *.properties*-Dateien wird die Bibliothek Apache Commons Configuration [Apaa] genutzt. Diese Bibliothek bietet Möglichkeiten, eine *.properties*-Datei zu laden und Daten in verschiedenen Formaten anhand von Schlüsseln aus diesen Dateien zu erkennen und zu laden. Dabei wird die Datei unter dem parametrisierten Pfad im Benutzerordner des aktuellen Benutzers gesucht. Der Pfad zu einer *.properties*-Datei in dem home-Verzeichnis wird durch einen Oberordner,

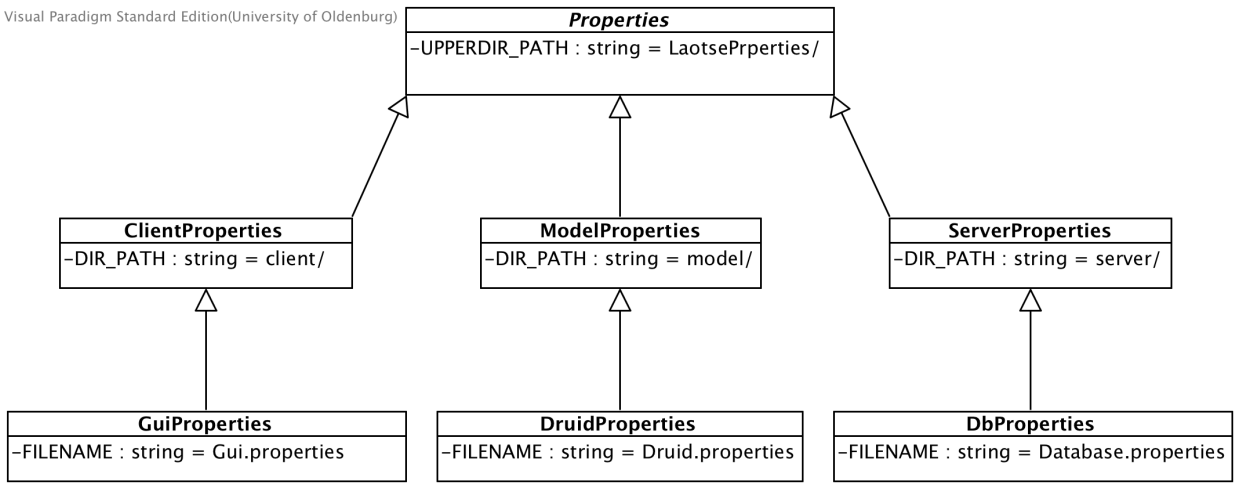


Abbildung 7.5.: Beispielhafte Architektur der Properties

den Ordner und den Dateinamen spezifiziert. Die Architektur ist in 7.5 für einige *.properties*-Dateien beispielhaft dargestellt. Für jede *.properties*-Datei gibt es dabei in der untersten Ebene eine Java-Klasse, die neben dem Dateinamen auch die Namen der Schlüssel in Konstanten hält, die in der jeweiligen Datei genutzt werden können. So wird der Zugriff auf die Werte vereinfacht, der von der abstrakten Klasse `Properties` durchgeführt wird.
Beim ersten Start von LAOTSE werden alle notwendigen *.properties*-Dateien mit Standardwerten im home-Verzeichnis erstellt. So wird sichergestellt, dass beim Client, bzw. beim Server nur die notwendigen *.properties*-Dateien vorhanden sind. Dazu vergleicht jedes Produkt, das *.properties*-Dateien benötigt, also z.B. der Server oder der Client beim Start alle im home-Verzeichnis vorhanden *.properties*-Dateien mit den jeweils notwendigen Standard *.properties*-Dateien. Wenn eine oder mehrere Dateien fehlen, werden diese mit Standardwerten erstellt und das Programm startet.
Wenn eine Instanz einer Properties-Klasse erstellt wird und die geforderte *.properties*-Datei immer noch nicht vorhanden sein sollte, z.B. weil sie in der Zwischenzeit gelöscht wurde, wird diese explizite Datei ebenfalls neu erstellt.

7.3. LAOTSE Server

DN

Dieser Abschnitt beschreibt alle Bundles, die zum LAOTSE-Server gehören.

7.3.1. Server Application

DN

Das Bundle bietet den Startpunkt des LAOTSE-Servers. Die Klasse `LaotseServerApplication` implementiert das Interface `IApplication` und kann damit in einem OSGi-Product verwendet werden. Die Manifest-Datei des Bundles definiert Extensions für die Extension Points `org.eclipse.core.runtime.applications` und `org.eclipse.core.runtime.`

products. Diese werden von dem Laotse Server-Product eingebunden. Im Activator werden Log4j-Logger für den LAOTSE-Server initialisiert. Log4j leitet Log-Nachrichten an sogenannte Appender. Diese Appender zeigen oder sichern die Log-Nachrichten. Im LAOTSE-Server wird zum einen ein ConsoleAppender aus der Basiskonfiguration von Log4j verwendet, zum anderen wird ein eigener Appender WebSocketAppender initialisiert, der Meldungen ab Level ERROR an Laotse Clients über den, im Folgenden beschriebenen, WebSocketServer versendet.

7.3.2. Server Communication

LS

Das Bundle Communication im Server enthält den LaotseWebSocketServer und damit den zentralen Nachrichtenflusspunkt für die Entgegennahme von Anfragen und das Senden von Statusmeldungen. Er erbt vom WebSocketServer der Bibliothek org.java-websocket und empfängt Nachrichten vom Client (siehe Abschnitt 7.1.1) auf die er, mit der Benachrichtigung einer betroffenen Komponente, entsprechend reagiert.
So können sich verschiedene Controller beim LaotseWebSocketServer als Listener registrieren lassen, um Anfragen oder Nachrichten bzgl. mosaik, Odysseus und der Konfiguration zu erhalten.
Des Weiteren übernimmt der LaotseWebSocketServer die Aufgabe nach dem Start einer mosaik-Simulation auf Aktualisierungsmeldungen von dieser zu warten. Als Reaktion auf solch eine Meldung übernimmt er die Aufgabe, den Client, von dem die mosaik-Anfrage ausging, auf den aktuellen Stand zu bringen, indem er das MosaikUpdateListener Interface realisiert (vgl. Abbildung 7.6) und sich selber der MosaikCtrl als Listener übergibt.
Der LaotseWebSocketServer ist als Singleton-Pattern realisiert, damit sichergestellt wird, dass alle Anfragen an einer zentralen Stelle registriert und verarbeitet werden können.

7.3.3. Odysseus Controller

DN

Der Odysseus Controller verwaltet die Datenströme, die auf der Odysseus-Instanz laufen sollen. Beim Start von Odysseus startet er automatisch Queries während er andere bei Bedarf starten

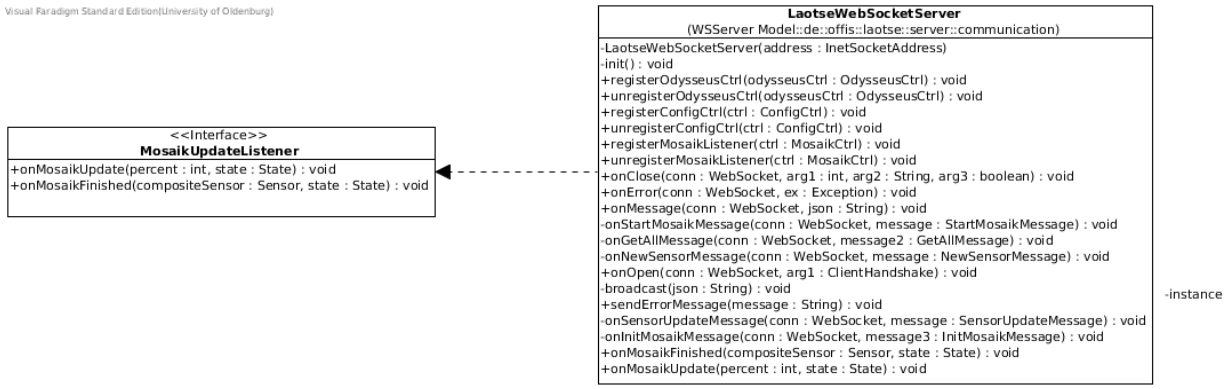


Abbildung 7.6.: Klassendiagramm LaotseWebSocketServer

kann. Der `OdysseusController` hält außerdem eine Liste mit allen gestarteten Queries pro zugeordnetem Sensor.
Der OdysseusController verwendet den, von Odysseus bereitgestellten, Webservice, um Anfragen zu verwalten. Der Webservice ist im Odysseus-Kern enthalten und steht beim Start der Server-Komponente bereit. Der `QueryManager` im `OdysseusController` nutzt die Klasse `WsClient` aus dem Webservice Client Bundle von Odysseus. Für den Zugriff wird eine Session aufgebaut und zur Authentifizierung der Standardnutzer „System“ verwendet.
Jedem Sensor wird in der Konfiguration ein Query-Template mitgegeben, das den Datenstrom für den jeweiligen Sensor definiert. In der Regel ist dies die Definition von Operatoren, die die Datenquelle einbinden und die Daten in ein Format bringen, das in Druid gespeichert werden kann. Abschließend wird eine Datensenke definiert, die die Daten an Druid abgeben. Beispielhafte Queries werden im Benutzerhandbuch vorgestellt.
Damit die Daten in Druid abgelegt werden können, werden diese dem neu entwickelten `KafkaTransportHandler` übergeben.
Die Query-Templates sind nicht vollständig ausgefüllt, da es Argumente gibt, die erst zur Laufzeit in die Query eingesetzt werden können. Beispielsweise werden Datenquellen und -senken anhand der Sensor-Id durchnummeriert. Diese wird erst vergeben, wenn ein Sensor in der Laotse Configuration angelegt wurde. Damit die Query-Templates gefüllt werden, ersetzt die Klasse `QueryEnricher` vordefinierte Platzhalter mit konkreten Werten.
Zur Zeit werden folgende Platzhalter ersetzt:

- %sensorid%: Die Id des Sensors
- %sensorName%: Der Name des Sensors
- %parentSensorid%: Die Id des Parent-Sensors, falls einer eingetragen ist, sonst 0.
- %address%: Die Adresse, unter der das Kafka-Cluster erreicht werden kann.

Nach jedem Start eines Query-Templates werden die Ids der erzeugten Queries in Odysseus in der `QueryRegistry` gesichert. Diese ordnet Sensoren eine Liste von Query-Ids zu. In der Regel wird von einem Query-Template eine Query-Id erzeugt. Damit die Informationen auch nach einem Neustart von Odysseus zur Verfügung stehen, wird die Zuweisung zusätzlich in der LAOTSE-Datenbank gesichert. Für den Zugriff auf die Tabelle wurde im `LaotseDbWrapper` ein `QueryIdsDao` implementiert.

Es kann notwendig sein, die Daten eines Datenstroms in LAOTSE mitzulesen. Der OdysseusController ermöglicht es, einen `ElementStreamListener` an vorhandenen Datenströmen zu registrieren und so die Daten zu bekommen.

Durch die Methode `manageQueries()` werden die Queries der einzelnen Sensoren verwaltet. Für jeden Sensor in der Konfiguration wird überprüft, ob Queries laufen sollten und ob Hinweise auf laufende Queries in der `QueryRegistry` zu finden sind. Queries werden automatisch gestartet, wenn die Attribute `active` und `autostart` gesetzt sind. Starten von Queries bedeutet, dass die Query initialisiert und sofort gestartet wird. Wenn eine Query gestoppt wird, so wird sie aus Odysseus gelöscht. Beim Neustart werden so auch neue Operatoren in Odysseus erzeugt, wodurch Fehler durch Altlasten in Operatoren vermindert werden. Außerdem wird überprüft, ob die in der `QueryRegistry` aufgeführten Queries noch in Odysseus existieren. Ist dieses nicht der

Fall, werden die entsprechenden Einträge gelöscht. Einem Sensor können mehrere Query-Ids zugeordnet sein. Sobald eine Query-Id nicht mehr in Odysseus gefunden wird, werden alle Queries des entsprechenden Sensors gelöscht, sich ein inkonsistenter Zustand eingestellt hat.

7.3.4. Mosaik Controller

MM

In diesem Abschnitt wird das Bundle `MosaikController` beschrieben. Es bietet Klassen, die dazu dienen mosaik zu initialisieren und anschließend eine Simulation zu starten. Dabei bildet es im LAOTSE-Server die Schnittstelle zwischen dem `MosaikStarter` (s. Abschnitt 7.7.2), dem `OdysseusController` (s. Abschnitt 7.3.3) und dem LAOTSE-Client (s. Abschnitt 7.1) und koordiniert die Arbeitsschritte. Durch diese Arbeitsschritte durchläuft eine Instanz eines Mosaik-Modells in LAOTSE mehrere Stati, die in Abbildung 7.7 dargestellt sind. Durch die Stati wird u.a. sichergestellt, dass vor jeder Simulation eine Initialisierung erfolgreich durchlaufen werden muss. Dadurch gibt es keine Konflikte bei Änderungen der Simulation.

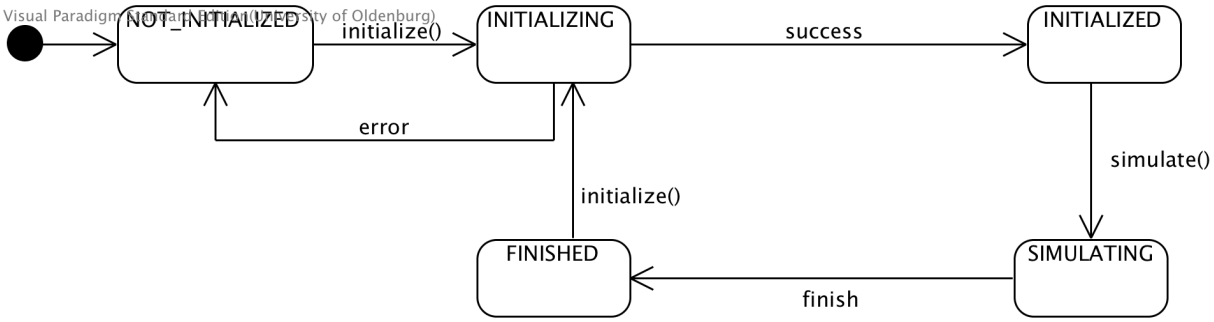


Abbildung 7.7.: Zustandsdiagramm der Stati von Mosaik

Die Stati haben folgende Bedeutungen:

- **NOT_INITIALIZED**: Die Simulation ist noch nicht initialisiert. Dies muss als erstes erfolgen.
- **INITIALIZING**: Es läuft gerade die Initialisierung der Simulation. Es werden durch einen kurzen Probelauf der Simulation die simulierten Sensoren aus Mosaik in LAOTSEerkannt und in LAOTSEals neue Sensoren hinzugefügt. Dabei werden sie alle dem für diese Simulation ausgewählten Mosaik-Sensor als Untersensoren zugeordnet.
- **INITIALIZED**: Die Initialisierung der Simulation ist erfolgreich abgeschlossen, die Sensoren wurden angelegt und die Simulation wieder gestoppt. Sie kann nun für den Produktiveinsatz, also die Aufzeichnung der Messwerte gestartet werden.
- **SIMULATING**: Die Simulation wird gerade durchgeführt.
- **FINISHED**: Die Simulation wurde komplett durchgeführt und die aufgezeichneten Messwerte können eingesehen werden.

In Abbildung 7.8 ist ein vereinfachtes Sequenzdiagramm der Initialisierung von Mosaik zu sehen. Dabei wurden nur die Klassen aus dem `MosaikController` betrachtet und die Klassen

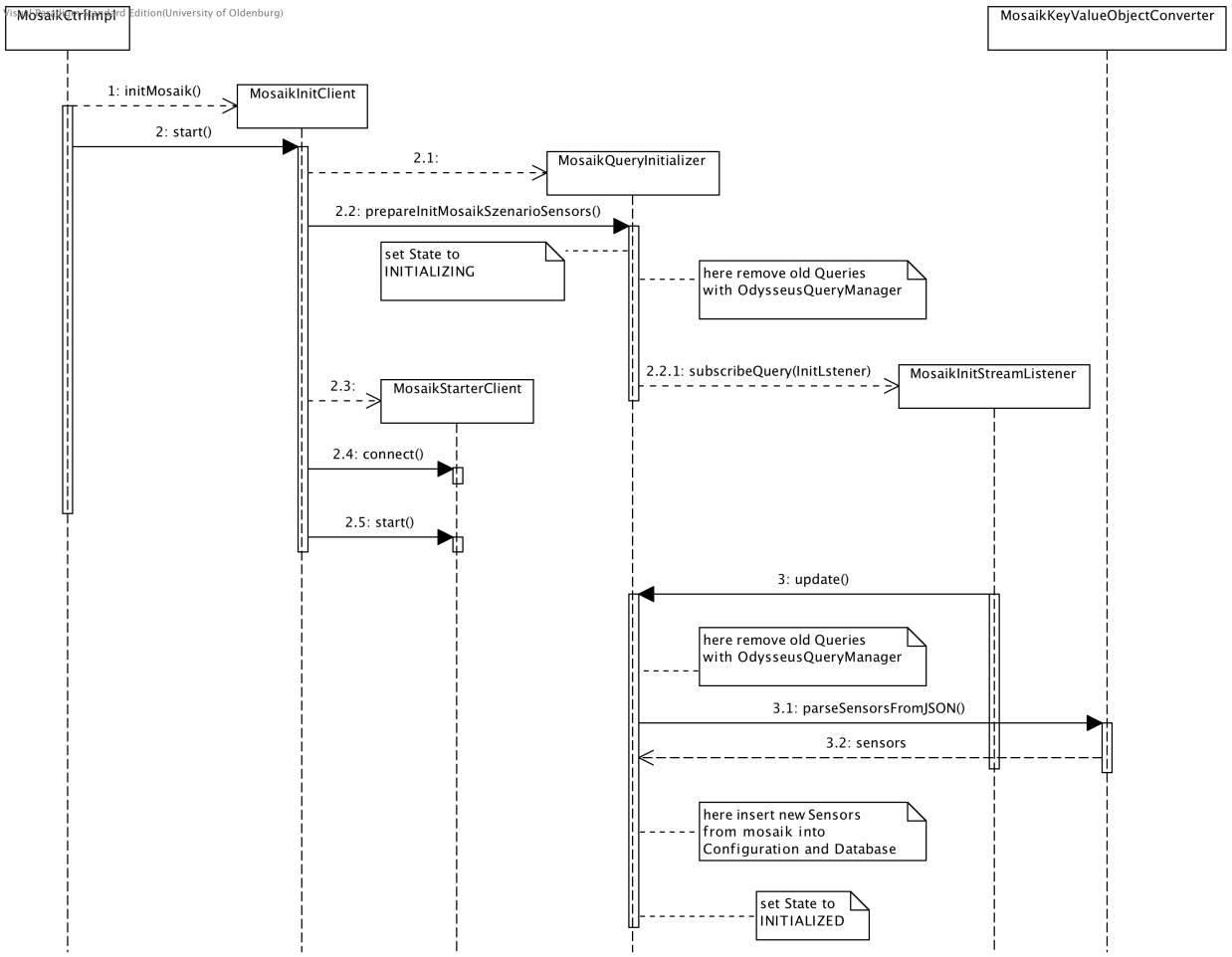


Abbildung 7.8.: Vereinfachtes Sequenzdiagramm der Initialisierung von Mosaik

aus anderen Bundles durch Kommentare ergänzt. Zusammenfassend lässt sich sagen, dass zuerst LAOTSEüber Odysseus an Mosaik angebunden wird. Anschließend wird eine Mosaik-Simulation gestartet. Wenn die ersten Werte erhalten werden, wird die Verbindung über Odysseus wieder unterbrochen und die neuen Sensoren werden aus dem erhaltenen JSON-Objekt gelesen. Diese werden dann in LAOTSE eingefügt und die Initialisierung ist abgeschlossen.

7.3.5. Configuration

LS

Das Configuration Bundle enthält neben seinem `Activator` nur die Klasse `ConfigCtrlImpl`. Diese realisiert die Schnittstelle `ConfigCtrl`, wie zu sehen im Klassendiagramm in Abbildung 7.9, und ist als Listener für Anfragen, die die Konfiguration von LAOTSE betreffen, beim `LaotseWebSocketServer` (siehe 7.3.2) registriert.
`ConfigCtrlImpl` hat Zugriff auf die Klassen des LAOTSE Datenbank Wrappers (siehe 7.3.6), um alle erforderlichen Daten direkt aus der Datenbank abrufen zu können. Außerdem speichert diese Klasse alle aktuellen Listen von Modellen zusätzlich selber ab, um diese bei wiederholten Anfragen nicht zeitintensiv aus der Datenbank neu laden zu müssen, sondern performant abrufen zu können.

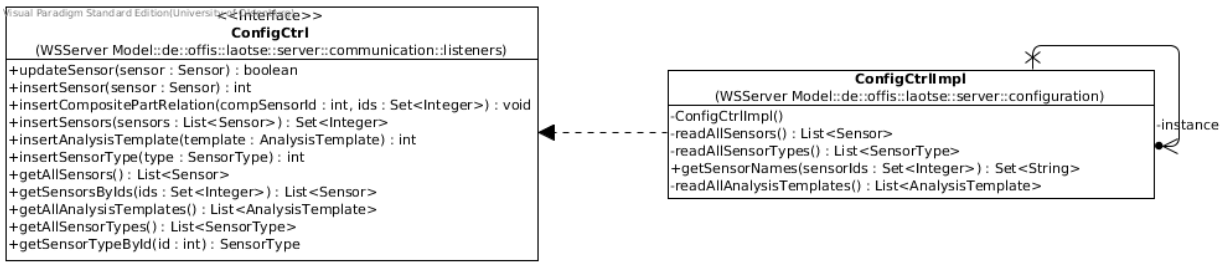


Abbildung 7.9.: Klassendiagramm `Configuration`

Da alle Anfragen an das DBMS über `ConfigCtrlImpl` laufen können die internen Listen bei jeder Veränderung der Modelle zuverlässig angepasst werden.
Die `ConfigCtrlImpl` ist als Singleton realisiert, um sicherzustellen, dass alle Anfragen an die Datenbank an zentraler Stelle registriert werden können und die zurückgesendeten Informationen immer dem aktuellen Stand der Datenbank entsprechen.

7.3.6. Laotse DB Wrapper

LS

Das Bundle LaotseDbWrapper ist für den Zugriff auf die Modelldatenbank von LAOTSE zuständig. Es ist an die aktuelle MySQL-Datenbank angepasst und müsste im Falle einer anderen Datenbanklösung geändert bzw. ausgetauscht werden.
Die Implementierung des Wrapper folgt dem DAO-Pattern, näher erläutert auf der Oracle Website [Dao]. DAO steht für Data Access Object und bedeutet, dass jeder Zugriff auf eine Datenbanktabelle einer bestimmten Entität nur über eine einzige Klasse zu erfolgen hat.
So wird z.B. die Tabelle der Sensoren allein vom `SensorDao` verwaltet, der nach außen alle notwendigen Operationen wie Hinzufügen, Aktualisieren oder Entfernen anbietet.
Da die verschiedenen Modellklassen von LaotseModel (siehe 7.2.2) einiges gemeinsam haben, da sie alle von der Klasse `Model` erben, gibt es auch in der konkreten Realisierung der verschiedenen DAOs mehrere Generalisierungen, vgl. Abbildung 7.10, die im Folgenden näher erläutert werden:

AbstractDbAccess Die abstrakte Klasse `AbstractDbAccess` bietet die Möglichkeit, zu einer beliebigen MySQL-Datenbank zu verbinden oder eine bestehende Verbindung wieder zu schließen. Dazu nutzt sie eine weitere Klasse, den `DbConnectionPool`, die das Verwalten der Verbindungen zur Datenbank übernimmt und bereits bestehende Verbindungen aufgebaut lässt, um für die nächste Anfrage den erneuten, zeitaufwändigen Verbindungsaufbau zu vermeiden. Erst wenn alle bereits aufgebauten Verbindungen in Benutzung sind, wird vom `DbconnectionPool` bei neuer Anfrage auch eine neue Verbindung angelegt.

AbstractDao Die abstrakte Klasse `AbstractDao` erbt von `AbstractDbAccess` und ist für den Zugriff auf eine zugeordnete Modelltabelle implementiert und hat dafür ein generisches Modell `T` zugeordnet. Sie regelt die Tabellenspalten, die alle Hauptentitäten gemeinsam haben: Name und ID. Mit diesen Informationen kann der `AbstractDao` SQL statements, wie z.B. `getById(query)` generisch aufbauen. Für andere Statements, wie z.b. `insert(query)` müssen allerdings die konkreten Spalten der Tabelle bekannt sein. Um auf diese Werte zugreifen zu können, gibt `AbstractDao` die abstrakte Methode `getColumnHeads()` zur Realisierung vor, die ein Array von `Column` Objekten zurückgibt,

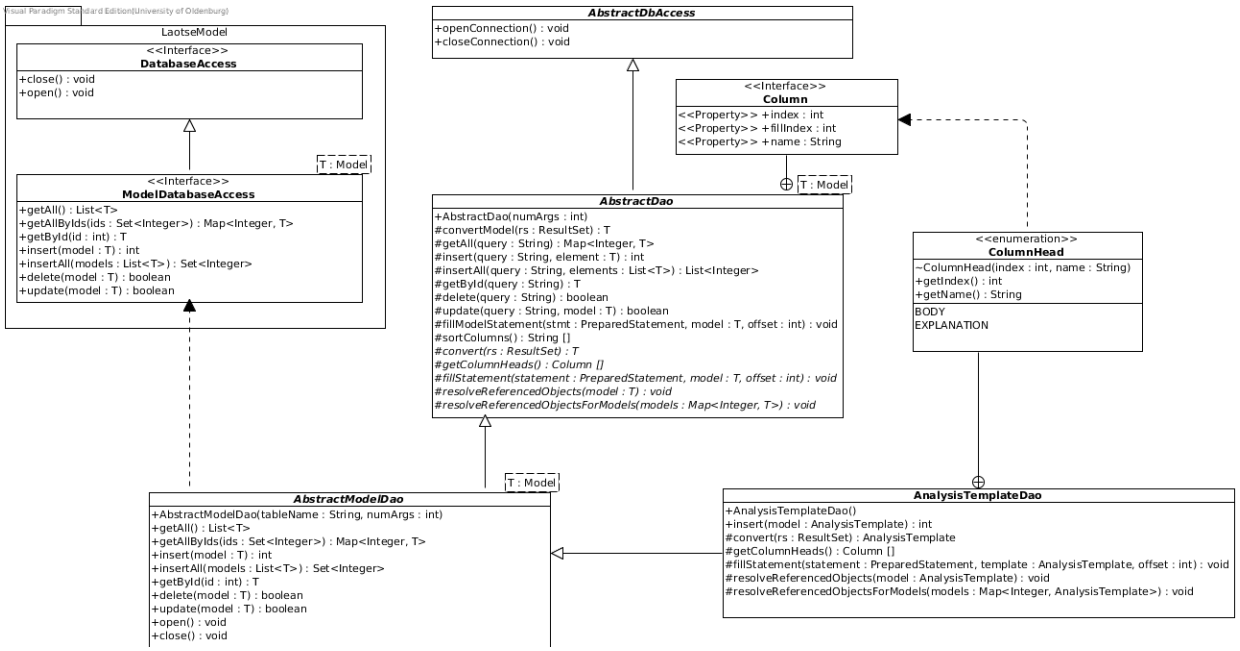


Abbildung 7.10.: Realisierung des Datenbankzugriffs

das genau die Tabellenspalten repräsentiert. `Column` ist ein, in `AbstractDao` enthaltenes, Interface, das eine Tabellenspalte über ihre Position innerhalb der Tabelle, den Index, und ihren konkreten Namen identifiziert. Damit ist es dem `AbstractDao` möglich, generisch PreparedStatements für MySQL zu erzeugen, die nur noch von einer Realisierung mit konkreten Werten gefüllt, bzw. deren Ergebnisse geparst werden müssen. Dazu muss eine von `AbstractDao` erbende Klasse die abstrakten Methoden `fillStatement(statement, model, offset)` und `convert(resultSet)` mit spezifischer Logik überschreiben.

Des Weiteren muss das Auslesen von, vom Model referenzierten, anderen Tabellen durchgeführt werden. Dazu sind die Methoden `resolveReferencedObjects(model)` und `resolveReferencedObjectsForModels(models)` abstrakt definiert, die für alle Fremdschlüssel, in einem gerade durch `convert(rset)` erzeugtem Model, die entsprechenden anderen DAOs aufrufen und die Objekte setzen. Die letztere der beiden Methoden schafft zudem die Möglichkeit für eine Liste von Modellen performant mit einer einzigen Abfrage alle referenzierten Modelle einer Art auf einmal abzufragen.

AbstractModelDao Der `AbstractModelDao` implementiert das `ModelDatabaseAccess` Interface aus dem LaotseModel (vgl. 7.2.2) und übernimmt das Mapping dieser Methoden auf die entsprechenden Aufrufe seiner Oberklasse `AbstractDao`.

ConcreteModelDao Die Implementierung eines konkreten ModelDaos wird im Diagramm 7.10 exemplarisch durch `AnalysisTemplateDao` dargestellt und ist bei `SensorDao` und `SensorTypeDao` analog. Beim `FieldDao` gibt es eine kleine Anpassung, da es sowohl Felder in `SensorType`, wie auch in `AnalysisTemplate` gibt, wird ihm im Konstruktor der jeweilige Kontext übergeben.

Jeder konkrete ModelDao implementiert die abstrakten Methoden von `AbstractDao` und überschreibt ggf. weitere Methoden, sollten Zugriffe auf andere ModelDaos notwendig

sein. So überschreibt z.B. der `AnalysisTemplateDao` die `insert(model)` Methode, da er nach dem Aufruf `super.insert(model)` noch dafür sorgen muss, dass auch seine `Fields` in die Datenbank eingefügt werden.

Für die Realisierung der `getColumnHeads()` Methode hat jeder konkrete ModelDao ein internes Enumn `ColumnHead` definiert, das `Colum` implementiert und die jeweiligen Spalten beschreibt. Leider ist es nicht möglich mit Java dieses Enum abstrakt vorzuschreiben, da es aber in allen konkreten Daos so genutzt wird, sollte dies auch in eventuell neu hinzugefügten Daos gleichartig umgesetzt werden.

Im LaotseDbWrapper gibt es neben den ModelDaos noch den `ConnectionDao`, der für Tabellen, die keine eigenen Entitäten sondern Beziehungen zwischen Entitäten darstellen, zuständig ist und den `QueryIdsDao`, der die Speicherung der Relationen von Odysseus Query IDs zu Sensor IDs übernimmt.
Für alle SQL Query Strings wird die Klasse `DbUtils` verwendet, die über statische Methoden für alle in LAOTSE verwendeten SQL Anfragen Query Strings erzeugen kann.

Test Setup

Die Klasse `TestSetup` kann durch Ausführen der Methode `makeTestSetup()` ein Zurücksetzten einer Datenbank, die dem LAOTSE-Schema entspricht durchführen, also diese leeren und ausschließlich mit erprobten Standardwerten füllen. Dies kann z.B. auch für das Durchführen von Tests auf einer isomorphen Testdatenbank genutzt werden. Zusätzlich bietet `TestSetup` Methoden um schnell Dummy-Modelle mit Defaultwerten zu erzeugen, die z.B. für einen Auslastungstest in Großzahl in die Datenbank eingespeist werden können.

7.3.7. Server Properties

MM

Der Server bietet verschiedene *.properties*-Dateien, deren Einstellungen durch jeweils eine Implementierung in Java repräsentiert werden. Es können z.B. Verbindungs- und Metadaten für die LAOTSE-Datenbank, Verbindungsdaten des Websockets und Verbindungsdaten für Odysseus konfiguriert werden. Dabei erbt jede Klasse, die genau eine *.properties*-Datei verwaltet, von der Klasse `ClientProperties`. Diese Architektur ist in Abschnitt 7.2.5 genauer beschrieben.

7.4. LAOTSE Relationen

Alle

Dieser Abschnitt beschreibt die relevanten Beziehungen zwischen verschiedenen Komponenten.

7.4.1. Model, View, Controller

MM

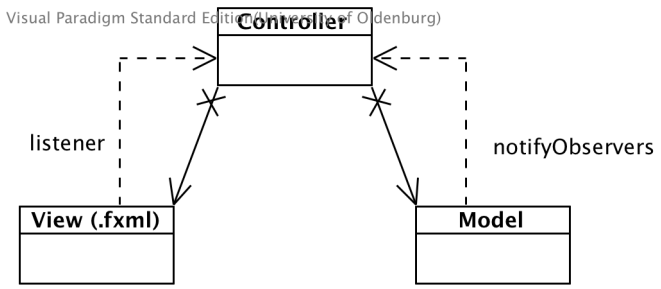


Abbildung 7.11.: Model-View-Controller Pattern in LAOTSE

In LAOTSE wird ein Model-View-Controller Pattern (vgl. [LR09]) genutzt, das an Java-FX angepasst wurde. Dieses ist in Abbildung 7.11 dargestellt.
Die View wird dabei durch *.fxml*-Dateien implementiert, welche die Elemente der GUI anordnen und bereitstellen. Da in der View nur XML als Sprache genutzt werden kann, wird sichergestellt, dass die Logik im Controller bleibt.
Der Controller ist der fxml-Controller (s. 7.1.2), der die GUI steuert. Dafür werden Variablen an Elemente der GUI gebunden und Methoden direkt als Listener für Nutzereingaben registriert. Der Controller setzt Daten in die Elemente der GUI, handhabt den Lebenszyklus einzelner Elemente und verändert die Darstellung. Die definierten Methoden reagieren auf Nutzereingaben und es wird ggf. über den `RootController` mit anderen Controllern kommuniziert, um andere Daten zu erhalten oder die Änderung einer anderen View anzustoßen. Der Controller hält auch eine Referenz zu einem spezifischen Modell. Er kann Daten in diesem lesen und schreiben und wird durch das ObserverPattern (s. 7.4.2) über Änderungen des Modells benachrichtigt. Dadurch kann er auf Änderungen des Modells reagieren und z.B. die View veranlassen geänderte Daten darzustellen.
Das Model hält nur die Daten und benachrichtigt bei jeder Änderung alle Observer.
Insgesamt wird also die gesamte Logik und auch nichts anderes in dem Controller implementiert, wodurch eine logische und übersichtliche Strukturierung des Codes sichergestellt werden kann.

7.4.2. Observer Pattern

KB

In LAOTSE wird das klassische `Observer Pattern` (vgl. [LR09]) genutzt, um der GUI und damit dem Benutzer deutlich zu machen, dass sich an der Datenbasis etwas geändert hat. Abbildung 7.12 zeigt wie das `Observer Pattern` innerhalb von LAOTSE genutzt wird.
Grundlegend gilt für das Observer Pattern, dass ein `Observable` existiert, bei dem sich mehrere `Observer` registrieren können. Ändert sich etwas am Observable, informiert dieser alle bei ihm registrierten Observer über die Neuerungen. Anschaulicher ist dieses Vorgehen anhand der aktuellen Umsetzung in LAOTSE. In diesem Fall existiert das ObserverListObservable. Dies ist das abstrakte Observable, anhand dessen später die eigentlichen Observables definiert werden. Als „Vorlage“für die Observer dient die klassische Observer Klasse. Diese sieht eine `update` Methode vor, die aufgerufen wird, wenn das Observable eine Nachricht sendet, dass sich etwas geändert hat.
Im System selbst existieren zahlreiche implementierte Observables und Observer. Grundsätzlich

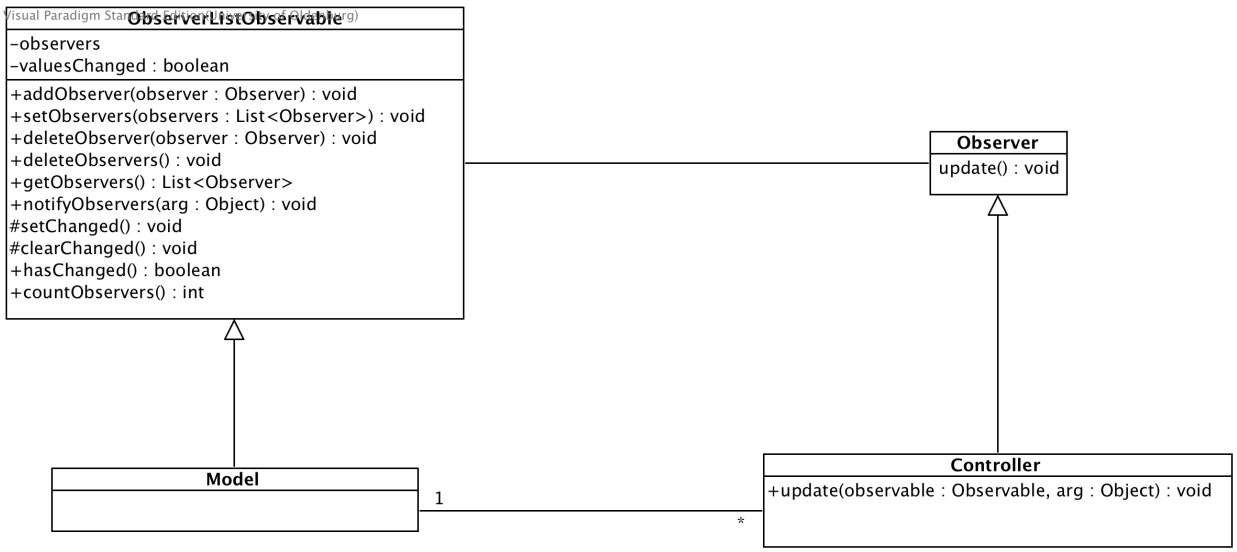


Abbildung 7.12.: Observer Pattern in LAOTSE

ist es dabei so, dass jedes Model, das innerhalb von LAOTSE genutzt wird, von der Observable Klasse erbt. Bei den Models registrieren sich dann die dazugehörigen Controller als Observer. Diese implementieren die Observer-Klasse. Das bedeutet, dass bei jeder Änderung an einem Model eine Nachricht an alle Observer herausgeschickt wird, dass sich das Model geändert hat. Die Controller können dann auf diese Änderungen reagieren und dem Benutzer das Model korrekt visualisieren. Allerdings ist es in LAOTSE nicht so, dass nur Controller als Observer agieren, sondern auch andere Klassen benötigen Informationen, dass sich am Model etwas geändert hat.
Ein Beispiel dafür ist die Klasse `Lists`, in der alle Sensoren angezeigt werden können. Allerdings ist eine Bearbeitung des Sensors möglich und damit auch eine Umbenennung. Daher muss die Liste ebenfalls informiert werden, wenn sich an einem Sensor etwas ändert, denn für diesen Fall müsste in der Liste ein anderer Name angezeigt werden, damit es beim Benutzer nicht zu Irritationen kommt.
In LAOTSE ist es also so, dass alle Models als Observable fungieren, damit diese den Controllern, die an Ihnen interessiert sind Informationen über eine Aktualisierung zukommen lassen können. Dadurch kann garantiert werden, dass die GUI immer auf dem aktuellen Stand ist und keine Fehlinformationen angezeigt werden.

7.4.3. Client/Server-Kommunikation

KB

Die Kommunikation zwischen Server und Client ist essentiell für das System. Dabei geht es vor allem darum, dass der Client Daten aus der Datenbank oder aus Mosaik vom Server abrufen kann. Wie eine solche Kommunikation zwischen Client und Server aussehen kann, zeigt Abbildung 7.13.
Durch Benutzereingabe werden dabei die entsprechenden `sendMessage` Methoden im `LaotseWebSocketClient` aufgerufen. Durch diese weiß der Client, welche Daten er an den Server übergeben muss. Die fertig zusammengebaute Nachricht wird dann an den `LaotseWebSocketServer` gesendet. Dieser empfängt die Nachricht in seiner `OnMessage` Methode. In-

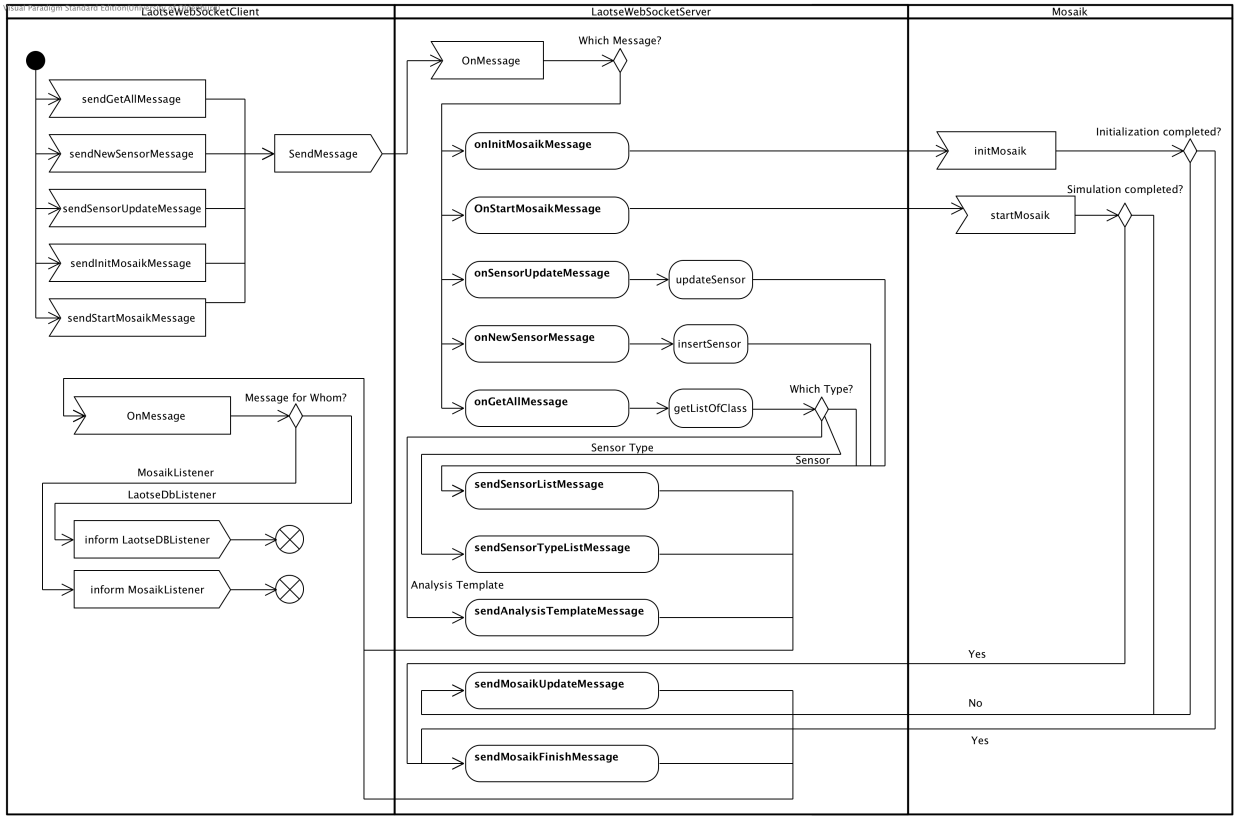


Abbildung 7.13.: Kommunikation zwischen Client und Server

nerhalb dieser Methode überprüft der Server, was für eine Nachricht der Client ihm gesendet hat und kann dementsprechend die richtigen Methoden aufrufen um auf diese Nachrichten zu reagieren.

Kommt die Nachricht, dass ein Sensor hinzugefügt oder aktualisiert werden soll, oder dass eine Liste eines bestimmten Typs aus der Datenbank abgerufen werden soll, dann wird die gewünschte Operation auf der Datenbank ausgeführt. Im Anschluss sendet der Server dem Client eine Nachricht mit der entsprechenden Liste, einer Sensor ID oder der Nachricht, dass der Vorgang abgeschlossen ist. Der Client kann danach dementsprechend die Benutzeroberfläche anpassen und dem Benutzer somit visualisieren, dass der Vorgang erfolgreich war.

Bei den beiden Messages, die Mosaik betreffen verhält sich die Kommunikation etwas anders. Der Benutzer kann wie gewohnt aus der GUI die Initialisierung, oder wenn diese bereits abgeschlossen wurde, die Simulation starten. Die Nachricht, welche Simulation bzw. Initialisierung gestartet werden soll sendet der Client dann wiederum an den Server. Dieser empfängt die Simulationsinformationen und startet im `MosaikController` die Initialisierung bzw. die Simulation selbst. Während der Initialisierung und der Simulation sendet Mosaik an den Server Update Nachrichten zurück, wie weit die Simulation fortgeschritten ist. Ist die Simulation bzw. die Initialisierung abgeschlossen, sendet Mosaik an den Server eine `FinishMessage`.

Der Server reagiert dann auf diese Nachrichten, indem er die Informationen über den Stand der Simulation an den `LaotseWebSocketClient` weitersendet, damit dieser dem Benutzer anzeigen kann, wie weit die Simulation fortgeschritten ist. Im Grunde genommen reagieren Server und Client in diesem Anwendungsfall ähnlich wie bei den anderen Nachrichten auch, allerdings wird der Weg über Mosaik gegangen, da hier die Simulation gestartet werden muss.

Der Client erhält die vom Server gesendeten Nachrichten wiederum in der `onMessage` Methode. Innerhalb dieser wird vom Client überprüft, an wen die Informationen weitergegeben werden müssen. Im Client selbst können sich Listener reagieren. Es wird dabei zwischen `DbListenern` und `MosaikListenern` unterschieden. Kommen beim Client die Nachrichten vom Server an, wird also überprüft, ob die vom Server gesendeten Daten an die registrierten `DBListener` oder an die `MosaikListener` weitergegeben werden müssen. Abschließend sendet der Client den definierten Listenern dann die Daten, die diese dem Benutzer anzeigen.

7.5. Odysseus

DN

Das Datenstrommanagementsystem Odysseus ist leicht erweiterbar. Damit Datenströme in Odysseus definiert und ausgeführt werden können, werden Operatoren benötigt, die durch die Modularität von OSGi-Anwendungen durch das Hinzufügen neuer Bundles integriert werden können. Neben den schon in Odysseus zur Verfügung stehenden Operatoren wurde eine Datensenke für Kafka benötigt, die im Rahmen der Projektgruppe entwickelt und dem Odysseus-Projekt zur Verfügung gestellt wurde. Für den Zugriff auf OPC UA-Server waren zwei Operatoren vorhanden, die die Kommunikation mit einem gegebenem OPC UA-Server zunächst nicht aufnehmen konnten. Nach Anpassung eines Operators war der Zugriff möglich.

In Odysseus werden Datenströme von aneinander gehängten Operatoren verarbeitet. Nach außen gibt es den Access-Operator, der Daten in Odysseus einspeisen kann. Zur Ausgabe von Daten wird ein Sender-Operator zur Verfügung gestellt. Die Operatoren können unterschiedliche Verbindungen mit unterschiedlichen Protokollen aufbauen. Dafür wird das Access-Framework [Odyb] genutzt. Zur Definition von Access- oder Sender-Operator werden die Angabe eines DataHandlers, ein ProtocolHandler und ein TransportHandler benötigt. Für die Datenverarbeitung in Odysseus wird gewöhnlich der DataHandler `Tuple` verwendet. Zur Ein- und Ausgabe von JSON-Objekten wird der DataHandler `KeyValueObject` verwendet. Damit die Daten von Operatoren verarbeitet werden können, werden die Daten zunächst in eine `Tuple`-Struktur umgewandelt. Ein TransportHandler implementiert die Anbindung von Odysseus an andere Programme. Er ermöglicht den Datenaustausch. Damit die Umwandlung der Datenstruktur zwischen externem Aufbau und internem DataHandler durchgeführt werden kann, werden ProtocolHandler verwendet. Im Folgenden werden die Implementierungen der TransportHandler für Kafka und OPC UA beschrieben.

7.5.1. Kafka Operator

In LAOTSE werden Daten an Druid übergeben. Druid kann die Daten über das Nachrichtensystem Apache Kafka aufnehmen. Damit Odysseus die Sensordaten an Druid übergeben kann, wurde daher ein KafkaWrapper implementiert, der einen TransportHandler zur Ausgabe von Daten an Kafka realisiert.
Zur Datenübergabe an Kafka wird der Kafka Producer verwendet. Dieser kann sich an ein Kafka-Cluster verbinden und Nachrichten zu einer definierten „Topic“ veröffentlichen.

LAOTSE soll die Sensordaten einer Menge von Sensoren verarbeiten können. Es ist allerdings nicht sinnvoll für jede einzelne Datenquelle einen eigenen Kafka Producer zu erzeugen. Vielmehr lohnt es sich, einen Producer zu erzeugen und ihn wiederzuverwenden, wenn die Konfigurationen der einzelnen Producer identisch wären. Das Datenbank-Feature von Odysseus [Odyc] verwendet die Möglichkeit, eine Verbindung aufzubauen und mithilfe eines weiteren Operators diese Verbindung wieder zu verwenden. Der `KafkaTransportHandler` arbeitet in ähnlicher Weise. Hier wird allerdings implizit nach einer äquivalenten Producer-Konfiguration gesucht. Die Klasse `KafkaProducerRegistry` hält dafür alle erzeugten Kafka Producer.

Kafka kann empfangene Nachrichten in Partitionen einsortieren, um diese parallel zu verarbeiten. Für die Partitionierung können eigene Partitionierer implementiert werden. Die Partitionierer werden zur Laufzeit über den Klassennamen geladen. Der Klassenlader wird aus dem Bundle `org.apache.kafka` aufgerufen. Um einen eigenen Partitionierer zu verwenden, wurde ein OSGi-Fragment `de.uniol.inf.is.odysseus.wrapper.kafka.partitioner` erstellt, dass den Partitionierer `KafkaPartitioner` enthält. OSGi-Fragmente sind einem OSGi-Plugin angehängt und teilen sich den Klassenlader. Der `KafkaPartitioner` implementiert eine einfache Hashfunktion.
Nachrichten, die über den `KafkaTransportHandler` versendet werden, müssen JSON-Objekte sein, da der Schlüssel, über den die Nachrichten partitioniert werden, aus der Nachricht mithilfe eines JSON-Parsers ermittelt wird.

Listing 7.1: Beispiel KafkaTranportHandler

```
KafkaOut = SENDER({wrapper='GenericPush',
transport='kafka',
protocol='JSON',
dataHandler='KeyValueObject',
SINK="SENDERjson1",
options=[['metadata.broker.list', 'localhost:9092'],['request.
   required.acks','1'],['producer.type','async'],['partitioner.
   class','de.uniol.inf.is.odysseus.wrapper.kafka.KafkaPartitioner
   '],['serializer.class','kafka.serializer.StringEncoder'],['
   keyname','sensor'],['topicName','laotse'],['shareProducer','
   true'],['shareProducer','true'],['sampleSize','1']]
},OTHEROPERATOR)
```

Listing 7.1 zeigt den Ausschnitt einer beispielhaften PQL-Anfrage, die Nachrichten an Kafka sendet. Die `options` haben dabei folgende Bedeutung:

- `topicName`: Definiert den Namen der Topic, unter der die Nachrichten in Kafka gegeben werden soll.
- `shareProducer`: Falls dieser Wert auf `'true'` gesetzt ist, kann ein Producer mit anderen Instanzen des `KafkaTransportHandler` geteilt werden. Sonst wird ein exklusiver Producer erzeugt. Parameter ist optional, der Standardwert ist `'false'`.
- `sampleSize`: Nachrichten können gesammelt werden, bevor sie an einen Kafka-Server übergeben werden. Dieser Parameter bestimmt die Größe der Liste an Elementen, die jeweils übergeben wird.

- `keyName`: Definiert den Namen des JSON-Elements, das zur Partitionierung an einen Partitionierer übergeben wird

Die folgenden Optionen werden für die Konfiguration des Kafka-Producers verwendet. Die Bedeutung ist in der Kafka-Dokumentation [Kaf] nachzulesen.

- `metadata.broker.list`
- `request.required.acks`
- `producer.type`
- `partitioner.class`
- `serializer.class`

Der `KafkaTransportHandler` kann durch folgende Änderungen verbessert werden:
Es kann ermöglicht werden, beliebige Konfigurationsparameter an den Kafka-Producer weiter zu geben und nicht eine feste Liste. Zur Zeit werden nur Nachrichten im JSON-Format unterstützt. Für andere Anwendungsfälle ließe sich die Partitionierungsstrategie so anpassen, dass der Wert, aus dem der Schlüssel gebildet wird, nicht aus der Nachricht geparst werden muss.

7.5.2. OPC UA Operator

In Odysseus sind zwei Operatoren vorhanden, die den lesenden Zugriff auf OPC UA-Server ermöglichen.

Im Bundle *de.uniol.inf.is.odysseus.server.incubation.opcua* liegt eine Implementierung, die eine Subscription von Werten auf dem OPC UA-Server ermöglichen. Der Operator wurde innerhalb des Odysseus-Projekts veröffentlicht. Im Rahmen der Projektgruppe konnte er aufgrund des darunter liegenden Stacks allerdings nicht eingesetzt werden. Der Grund dafür war, dass die Endpoint-Adresse des OPC UA-Servers aus dem Hostnamen bestand, der Server aber nur über die IP-Adresse kontaktiert werden konnte, da kein Domain Name System (DNS) innerhalb des verwendeten Netzwerks zur Verfügung stand. Dem Stack wurde die IP-Adresse mitgegeben, intern wurde nach einem ersten Verbindungsaufbau die Endpoint-Adresse ermittelt, über die eine Verbindung aufgebaut werden sollte. Ohne DNS konnte die Verbindung nicht aufgebaut werden.

Das Bundle *de.uniol.inf.is.odysseus.opcua* stellt eine zweite Implementierung bereit.
Da der Stack, auf dem die Implementierung beruht, nicht frei verwendbar ist, ist das Bundle nicht veröffentlicht. Innerhalb der Projektgruppe durfte es verwendet werden, um zu zeigen, dass Laotse grundsätzlich auch Daten von OPC UA-Servern verarbeiten kann. Der zur Verfügung gestellte TransportHandler ruft die Daten eines zu konfigurierenden Knotens und dessen Unterknoten von einem OPC UA-Server ab. Dabei wird keine Subscription verwendet. Die Daten werden durch Polling in einem konfigurierbaren Intervall abgerufen. Damit der Operator auf die zur Verfügung stehenden OPC UA-Server zugreifen konnte, musste dieser angepasst werden. Ursprünglich war vorgesehen, dass die Verbindung verschlüsselt werden sollte. Da ein dafür benötigtes Server-Zertifikat nicht zur Verfügung stand, wurde der TransportHandler so angepasst, dass eine Verbindung ohne Verschlüsselung aufgebaut werden konnte. Außerdem war es möglich, die Endpoint-Adresse von der Adresse, über die der Server erreichbar ist, zu trennen. Damit der

TransportHandler verwendet werden kann, muss der DataHandler KeyValueObject zur Verfügung stehen.

Listing 7.2: Beispiel OPCUATransportHandler

```
#PARSER PQL
#ADDQUERY
device=ACCESS({source='device',
wrapper='GenericPull',
protocol='None',
transport='OPCUA',
dataHandler='KeyValueObject',
options=[
['endpoint','opc.tcp://VMIPRO-50-50.offis.uni-oldenburg.de:16664/4
   CEUAServer'],
['parentnode','ns=2;s=Q1Control/B10PTRC1'],
['pulldelay','1000'],
['connecturl','opc.tcp://172.20.50.50:16664/']
]})
```

Listing 7.2 zeigt eine PQL-Anfrage, die den Abruf der Werte des Knotens Q1Control/B10PTRC1 ermöglicht.

7.5.3. Odysseus Produkte

In LAOTSE wird Odysseus mit Erweiterungen eingesetzt. Die Erweiterungen wurden auf Seite des Odysseus-Servers benötigt. Damit Odysseus mit diesen Erweiterungen ausgeliefert werden kann, wurden die OSGi-Products von Odysseus Server und Odysseus Monolithic angepasst. Odysseus Server stellt die Server-Plattform bereit. In dieser werden neben Verwaltungsaufgaben die Datenstöme verarbeitet. Odysseus Monolithic vereint den Odysseus Server mit dem Odysseus Client. Beim Start wird die graphische Oberfläche geladen, mit deren Hilfe der Odysseus Server konfiguriert und unterschiedliche Visualisierungen der Datenströme durchgeführt werden können. In einer Produktivumgebung von Laotse würde der Server bevorzugt Anwendung finden, da er keine GUI auf einem entfernten System anzeigt. Möchte man die Funktionen des Odysseus Clients verwenden, so kann man einen Odysseus-Client verwenden, der sich über den WebService verbindet.
Die angepassten Products enthalten zusätzlich die folgenden Features:

- `de.uniol.inf.is.odysseus.keyvalue.feature`:

  Erlaubt die Verarbeitung von Key-Value-Objekten. Key-Value-Objekte werden zum einen von Mosaik gesendet. Zum anderen werden Daten in Druid über Key-Value-Objekte eingefügt.

- `de.uniol.inf.is.odysseus.wrapper.kafka.feature`: Dieses Feature beinhaltet den zuvor beschriebenen Kafka-Wrapper und dessen Abhängigkeiten.

- `de.uniol.inf.is.odysseus.wrapper.mosaik.feature`: Für den Abruf der Simulationsdaten aus Mosaik war dieses Feature bereits im Odysseus-Projekt veröffentlicht.

7.6. Druid R API

DN

Der *Druid connector for R* [Rdr] ermöglicht implementiert die Druid-Abfragetypen [Drua] TimeSeries, TopN und GroupBy. Diese Anfragen geben keine Rohdaten zurück, die Daten werden durch wählbare Granularitäten gruppiert. Auf jede Gruppe wird eine Aggregationsfunktion angewendet. Ist die Granularität der Gruppierung so angelegt, dass jede Gruppe maximal ein Element enthält, so können die Rohdaten wieder erhalten werden. Die kleinste Granularitätsstufe ist eine Millisekunde. Wenn es Datenquellen gibt, die in einer Millisekunde mehr als einen Datensatz liefern, ist das Auslesen der Rohdaten nicht möglich.

In Druid steht eine experimentelle Abfrage Select [Drub] zur Verfügung. Diese Anfrage erlaubt es, alle Daten in der Reihenfolge, in der sie in Druid geladen wurden, auszugeben. Da die Zahl an Ergebnissen sehr groß werde kann, unterstützt dieser Anfragetyp Pagination. Das heißt, dass das Ergebnis in Blöcke mit konfigurierbarer Größe aufgeteilt wird. Mit jeder Anfrage wird ein Block zurückgeliefert, beginnend mit den neuesten Daten, die in Druid vorhanden sind. Jedem Ergebnisblock wird eine Information angehängt, mit der der nächste Block abgefragt werden kann.

Der Druid connector for R wurde insofern angepasst, als dass das Zusammenbauen der Anfrage im JSON-Format angepasst wurde, da alle Anfragetypen durch unterschiedliche Parameter konfiguriert werden. Druid liefert das Anfrageergebnis als JSON-Objekt. Damit die Daten weiter verarbeitet und zum Beispiel für Analysen genutzt werden kann, müssen die Daten in ein anderes Format gebracht werden. Da der Aufbau der Anfrageergebnisse vom jeweiligen Anfragetyp abhängt, musst dass parsen des JOSN-Objekts ebenfalls angepasst werden.

Zur Zeit ist in der Erweiterung keine Unterstützung für Pagination implementiert. Zur Zeit muss die Blockgröße mindestens der Anzahl an Zeilen sein, die sich innerhalb des abgefragten Intervalls befinden. Die Erweiterung könnte insofern erweitert werden, als das zum einen signalisiert wird, wenn ein Block nicht alle Elemente beinhalten konnte. Weiter könnte der Abruf von folgenden Blöcken ermöglicht werden.

7.7. Hilfsprogramme

DN

Neben dem Hauptsystem LAOTSE, das in einzelne Bundles unterteilt ist und den Erweiterungen, die an Odysseus vorgenommen wurden, wurden weitere Programme implementiert, die bei der Entwicklung des Zielsystems unterstützten. Die Implementierung dieser Hilfsprogramme wird im Folgenden erläutert.

7.7.1. Datenbanktester

DN

Der `Datenbanktester` kann Tests auf unterschiedlichen Datenbanken durchführen. Ein Test wird aus Testteilen zusammengesetzt. Ein Testteil übernimmt dabei eine bestimme Aufgabe, zum Beispiel das Einfügen einer Menge von Daten oder der Abruf von Daten. Abbildung 7.14 zeigt ein vereinfachtes Klassendiagramm des Datenbanktesters. Die Klasse `TestApplikation` verwaltet die Testläufe. Sie initialisiert Testteile und startet danach jeden Testteil. Ein Testteil erbt von der Klasse `AbstractTestPart` und führt eine bestimmte Aufgabe durch. Dabei wird die Dauer der Ausführung gemessen. Das Ergebnis wird von der `TestApplikation` abgerufen. Die Ergebnisse der Tests werden abgerufen und zusammen in einer Logdatei gespeichert. Wenn alle Testteile abgeschlossen sind, wird ein Verteiler über die Klasse `Mailer` gesendet. Sollte während der Tests ein Fehler auftreten und dadurch eine Exception geworfen werden, wird ebenfalls eine Mail an diesen Verteiler geschickt. So können die Tests unbeaufsichtigt laufen.

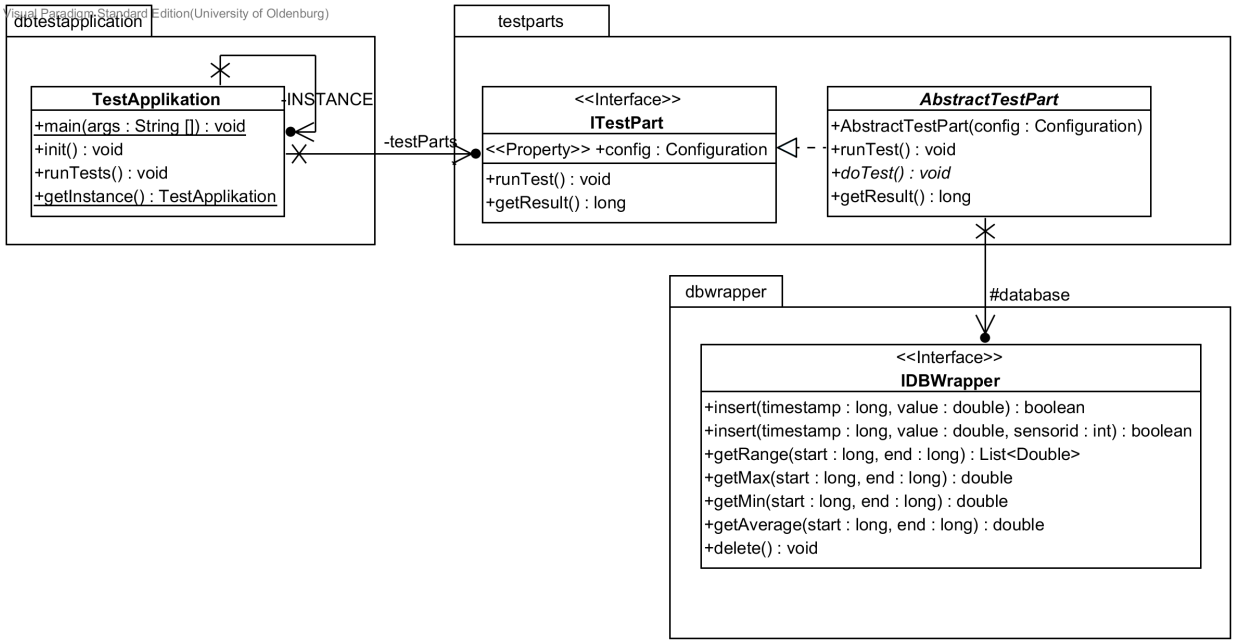


Abbildung 7.14.: Datenbanktester Klassendiagramm

Eine Instanz des `IDBWrapper` implementiert die notwendigen Methoden, um Daten in eine Datenbank zu schreiben oder Anfragen an die Datenbank zu stellen. Folgende DBWrapper wurden implementiert: Aerospike, Cassandra, Druid, Influx, Kafka, und VoltDB. Daneben wurde ein `DummyDbWrapper` implementiert, der lediglich Aufrufe der Methoden loggt. Es gibt Methoden zum Speichern eines Paares aus einem Wert vom Datentyp `double` und einem Zeitstempel. Weitere Methoden implementieren den Abruf von Daten in einem konfigurierbaren Intervall. Das ist entweder eine Liste der Daten oder das Minimum, das Maximum oder das arithmetische Mittel aller Werte in dem gegebenem Intervall sein. Die Mehtode `delete()` soll die verwendete Datenbank leeren.
Folgende Testteile wurden implementiert:

- `InsertTest`: Fügt Daten in die Datenbank ein. Generiert jeweils paarweise einen Zeitstempel und einen Zufallswert mithilfe von `Math.random()`. Der Wertebereich der Zu-

fallszahlen ist konfigurierbar. Beginn und Ende des Zeitstempelintervalls und der Abstand zwischen zewi Zeitstempeln ist ebenfalls konfigurierbar.

- `GetRangeTest`: Ruft Daten in einem gegebenem Intervall von der Datenbank ab. Ruft die Methode `getRange()` des `DBWrapper` auf.

- `GetAverageTest`: Ruft das arithmetische Mittel der Daten in einem gegebenem Intervall von der Datenbank ab. Ruft die Methode `getAverage()` des `DBWrapper` auf.

- `GetMaxTest`: Ruft das Maximum der Daten in einem gegebenem Intervall von der Datenbank ab. Ruft die Methode `getMax()` des `DBWrapper` auf.

- `GetMinTest`: Ruft das Minimum der Daten in einem gegebenem Intervall von der Datenbank ab. Ruft die Methode `getMin()` des `DBWrapper` auf.

- `Truncate`: Löscht den Inhalt der Datenbank.

- `Wait`: Pausiert die Ausführung des Testlaufs. Die Dauer der Pause wird in Millisekunden angegeben. Intern wird `Thread.sleep()` aufgerufen.

Die Kombination der Testteile und die Konfiguration derselben wird mithilfe von Konfigurationsdateien ermöglicht. Die Konfigurationsdateien müssen im Ordner „testConfigs“ abgelegt werden. Die Datei „main“ übernimmt eine Sonderrolle. Sie definiert zum einen, auf welche Implementierung des `IDBWrapper` verwendet werden soll. Zum anderen wird eine Liste der Namen der Konfigurationsdateien angegeben, die die Testteile definieren. Jeder Testteil benötigt eigene Parameter, die zu konfigurieren sind. Entsprechende Musterkonfigurationsdateien liegen im Jar-Archiv.

7.7.2. Mosaik Starter

LS

Das Projekt MosaikStarter enthält zwei Java Klassen: `MosaikStarterServer` und `TestClient`.
Der `MosaikStarterServer` kann Mosaik Szenarios ausführen lassen. Dazu braucht er den Namen einer Mosaik Szenario Python Datei, die in einem bestimmten Ordner auf der Maschine liegt, wo der Server gerade läuft. Dieser Ordner kann im Code des Servers angepasst werden.
Empfängt der Server nun solch einen Dateinamen, startet er eine Bash-Session, setzt in dieser die virtuelle für die Ausführung notwendige mosaik-Umgebung (ebenfalls anpassbar im Code) und führt das Script aus. Während der Ausführung gibt er alle Ausgaben des Prozesses an den verbundenen Client weiter.
Die Klasse `TestClient` kann zum Testen des `MosaikStarterServers` benutzt werden, wenn z.B. eine Anpassung der Pfade stattgefunden hat, ohne die GUI neu starten zu müssen. Aus ähnlichen Gründen befindet sich ein einfaches Python Skript *test.py* im Projekt, das die Zahlen von 1 bin 10 ausgibt, so dass getestet werden kann ohne mosaik, und im Verlauf auch Odysseus, zu starten.

7.7.3. Datengeneratoren

DN

Die Datengeneratoren, die in LAOTSE eingesetzt wurden, wurden mithilfe eines Frameworks[Ody15], das im Odysseus-Projekt entstanden ist, realisiert. Das Framework befindet sich innerhalb des Bundles `de.uniol.inf.is.odysseus.generator`. Dieses Bundle stellt die Klasse `StreamServer` bereit. Einem StreamServer wird eine Implementierung eines `AbstractDataGenerator` übergeben, der jeweils Datentupel liefert. Diese Tupel werden vom StreamServer über einen TCP-Stream an Odysseus übermittelt. Je nach Aufgabe des jeweiligen Datengenerators wurden angepasste `AbstractDataGenerators` entwickelt. Die einzelnen Implementierungen werden in den folgenden Abschnitten näher erläutert.
Allen Datengeneratoren ist gemein, dass die Klasse `RaspberryServerApplication` einen neuen `StreamServer` erstellt, der sich auf Port 12345 bindet. Es kann ausgewählt werden, ob für jede eingehende Verbindung ein eigener Datengenerator erstellt werden soll. Intern wird in einer Endlosschleife ständig `next()` aufgerufen. Die Methode gibt eine Liste an Datentupeln zurück. Wenn der Durchsatz eines Datengenerators beschränkt werden soll, kann innerhalb der Implementierung der Methode ein `Thread.sleep()` eingesetzt werden. Alle Datengeneratoren erzeugen pro Tupel einen zu berechnenden Wert und einen Zeitstempel über die Zeit der Erstellung des Tupels. Der Zeitstempel ist vom Typ `long` und gibt die Anzahl der Millisekunden seit dem 1. Januar 1970, 00:00:00 GMT, (vgl. `Date.getTime()`).

Konstantzahlengenerator

Der Konstantzahlengenerator ist in der Klasse `ConstantDoubleGenerator` implementiert. Bei jedem Aufruf von `next()` wird ein Tupel erzeugt, das aus einem konstanten Wert vom Datentyp `double` ud dem Zeitstempel besteht.

Zufallszahlengenerator

Der Zufallszahlengenerator ist in der Klasse `RandomDoubleGenerator` implementiert. Bei jedem Aufrud von `next()` wird ein Tupel mit einer Zufallszahl vom Datentyp `double` im Intervall [0,1) (vgl. `Math.random()`).

Temperatursensor

Der Temperaturdatengenerator ist in der Klasse `TemperatureDataGenerator` implementiert. Der Datengenerator erhält seine Messwerte von dem Sensor DS18B20, der an den Raspberry Pi über 1-Wire-Verbindung angeschlossen wir [Ds1]. Zusätzlich müssen die Kernelmodule `w1-gpio` und `w1_therm` geladen werden. Dies kann realisiert werden, indem der Datei */boot/config.txt* die Zeile „dtoverlay=w1-gpio“ hinzugefügt und danach der Raspberry Pi neu gestartet wird. Dadurch wird eine Datei unter dem Dateipfad */sys/bus/w1/devices/28-001414b338ff/w1_slave* erzeugt. Über diese Datei kann die gemessene Temperatur ausgelesen werden. Listing 7.3 zeigt den Aufbau der Datei. Die gemessene Temperatur ist in Zeile 2 mit t=20125 in m°C angegeben.

Listing 7.3: cat /sys/bus/w1/devices/28-001414b338ff/w1_slave

```
1 42 01 55 00 7f ff 0c 10 07 : crc=07 YES
  42 01 55 00 7f ff 0c 10 07 t=20125
```

Der Datengenerator parst die Temperaturangabe aus der Datei und rechnet in °C um. Der Wert wird als Datentyp `double` in das Datentupel eingefügt.

Audiosensor

Der Audiodatengenerator ist in der Klasse `AudioDataGenerator` implementiert. Er verwendet das Java-Audiosystem, um einen Datenstrom aus Schallpegeldaten zu erzeugen. Auf dem Raspberry Pi ist ein Cirrus Logic Audio Board [Cirb] montiert, dessen dessen Mikrofon/Line-In-Eingang als Datenquelle genutzt wird.
Damit die Schallpegeldaten abgerufen werden können, muss das Audiosystem vorbereitet werden. Das heißt, dass eine `TargetDataLine` erzeugt werden muss. Darüber lässt sich ein `AudioInputStream` erzeugen, der die Daten in einen `ByteBuffer` laden kann. Damit eine `TargetDataLine` erzeugt werden kann, werden Parameter benötigt, die beispielsweise die Kodierung des Audiosignals definieren. Der Datengenrator nutzt standardmäßig eine PCM-SIGNED-Kodeierung mit einer Samplerate von 16 ksps und einer Sample-Größe von 16 bit pro Kanal. Dabei müssen zwei Kanäle aufgenommen werden, da das Cirrus Logic Audio Board keine mono-Aufnahme unterstützt. Da ein Sample 16 Bit groß ist, muss über die Reihenfolge der Bytes entschieden werden. Aufgrund der Visualisierung der Audiodaten wurde hier Big-Endian ausgewählt.
Bei einem Aufruf von `next()` wird zunächst geprüft, wie viele Bytes aktuell zum Abruf zur Verfügung stehen. Diese Anzahl von Bytes wird in den `ByteBuffer` geladen. Der Zusammenschluss von Bytes, die zum gleichen Zeitpunkt erfasst wurden, wird Frame genannt. In einem Frame werden zunächst die Bytes des linken Kanals und dann die Bytes des rechten Kanals abgelegt. Damit die Daten als ein Wert übertragen werden kann, werden die Bytes zunächst in einen Datentypen mit größerem Wertebereich eingefügt. Hier wird der aus zwei Bytes bestehende Datentyp `short` gewählt. Beide Kanäle werden zu einem einzigen Kanal zusammengeführt, indem jeweils paarweise der Durchschnitt gebildet wird.
Zu Beginn der Entwicklung war der Datengenerator generischer aufgebaut. Um zum Beispiel Datenformat zu ändern, wird die Datei `audio.conf` verwendet, die neben dem ausführbaren Jar-Archiv platziert wird.

Spannungssensor

Der Spannungsdatengenerator wurde in der Klasse `VoltageDataGenerator` implementiert. Die Daten werden von dem Analog-Digital-Wandler adafruit ADS1015 12-Bit ADC - 4 Channel with Programmable Gain Amplifier [Ads] empfangen. Der Analog-Digital-Wandler wird über die I2C-Schnittstelle mit dem Raspberry Pi verbunden. Der Zugriff auf die Schnitttelle von Java wird von der Bibliothek Pi4J [Pi4] bereit gestellt.
Abbildung 7.15 zeigt einen Ausschnitt der Klassen. Klasse `Adafruit_ADS1015` wird für den Datenabruf verwendet. Die Implementierung ist an Adafruit's Raspberry-Pi Python Code Library [Adaa] angelehnt.

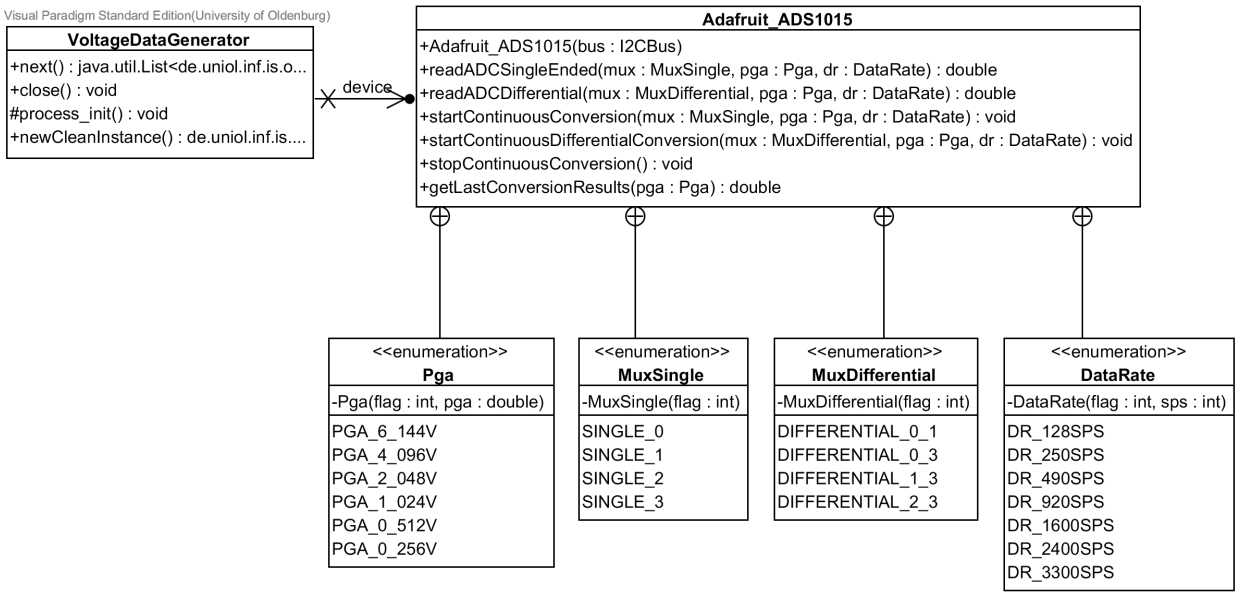


Abbildung 7.15.: VoltageDataGenerator Klassendiagramm

Der ADS1015 arbeitet folgendermaßen: Es gibt ein Konfigurationsregister, in das eine Konfiguration über den I2C-Bus geschrieben werden kann. Dabei werden unter anderem folgende Einstellungen verwendet:

- Multiplexer: Bestimmt zwischen welchen Eingängen die Spannung gemessen wird. Zum einen kann an jedem Eingang eine Single-ended-Messung durchgeführt werden. Das heißt, dass eine gemeinsame Masse als Referenz verwendet wird. Alternativ kaan eine Differenzialmessung zwischen zwei Eingängen durchgeführt werden. Mögliche Kombinationen sind in den Enums `MuxSingle` bzw. `MuxDifferential` aufgelistet.

- Datenrate: Definiert die Datenrate in Samples pro Sekunde. Eine höhere Datenrate verringert den Energieverbrauch pro erhobenem Messwert. Eine niedrigere Messrate verringert auftretende Messfehler. Mögliche Datenraten könne in Enum `DataRate` ausgewählt werden.

- Programmable Gain Amplifier (PGA): Der ADS1015 liefert Ergebnisse mit einer Breite von 12 Bits. Die maximal darstellbaren Messwerte sind $\pm 6,144V$. Wenn die zu Messende Spannung bekanntermaßen kleiner ist, lohnt es sich, den Wertebereich zu verkleinern und dabei dessen Auflösung zu erhöhen. Damit der Messwert interpretiert werden kann, muss diese Einstellung beim Abruf der Daten berücksichtigt werden. Das Enum `Pga` liefert die möglichen Einstellungen.

- Mode: Es gibt zwei Modi, mit denen der ADS1015 betrieben werden kann. Zum einen gibt es einen „Continuous Conversion Mode“, in dem ständig aktuelle Messwerte in ein „Conversion Register“ mit der eingestellten Datenrate geschrieben werden. Zum anderen gibt es einen „Power-down single-shot mode“, in dem ein Messwert nur erhoben wird, wenn ein weiteres Bit im Konfigurationsregister umgesetzt wird. Zwischen den einzelnen Aufrufen begibt sich der Sensor in einen Stromsparmodus.

Über die Methoden `readADCSingelEnded()` und `readADCDifferetial()` wird jeweils ein einziger Messwert erhoben und ausgelesen. Um die Messdaten kontinuierlich zu er-

heben, werden die Methoden `startContinuousConversion` bzw. `startContinuous-DifferentialConversion` angeboten. Dieser Modus kann über die Methode `stopContinuousConversion` beendet werden. Werden Messwerte kontinuierlich erhoben, so kann der jeweils aktuellste Messwert mithilfe der Methode `getLastConversionResults` abgerufen werden.
Es ist wünschenswert, einen Wert immer dann abzurufen, wenn ein neuen Ergebnis in das Conversion Register eingetragen wurde. Der ADS1015 stellt einen Ausgang „ALERT/RDY“ bereit, der nach einer Messwerterhebung kurz den Zustand ändert. Diese Zeit ist allerdings zu kurz, um an einem GPIO-Pin eines Raspberry Pis als Event erkannt zu werden. Experimente mit der entwickelten Anwendung Interupttest (angelehnt an [Gpi] zeigten, dass zum Beispiel das Schließen eines Tasters als Signal erkannt wurde, beim Anschluss des ALERT/RDY-Pins wurden keine Events getriggert.

Der Datengenerator wurde so konzipiert, dass seine Datenrate möglichst hoch ist. Somit wurde eine Datenrate von 3300 sps gewählt. Der I2C-Bus kann in unterschiedlichen Geschwindigkeiten betrieben werden. Standardmäßig arbeitet der Raspberry Pi mit 100kHz. Die Datenrate kann auf 400 kHz erhöht werden. Das wird erreicht, indem in der Datei */boot/config.txt* die Zeile

```
dtparam=i2c1\_baudrate=400000
```

eingefügt wird.

Der Datengenerator erhebt die Spannung in einer Differentialmessung zwischen den Eingängen A1 und A0. Dabei wird der maximale Wertebereich von $\pm 6,144V$ erlaubt.

8. Qualitätssicherung

UG, LS

Der Bereich der Qualitätssicherung in einem Softwareprojekt behandelt die Sicherstellung des Einhalts gewisser, vorher festgelegter, Qualitätskriterien. Diese sollen einen möglichst langen Lebenszyklus des Endproduktes begünstigen, da durch erhöhte Softwarequalität auch die Wartbarkeit entscheidend verbessert wird.
Ein bekanntes Vorgehensmodell um derartiges zu erreichen ist Continuous Integration, welches im Folgenden in Grundzügen geschildert wird.

8.1. Anforderungsqualität

UG

Die anfänglich erarbeitete Anforderungsdefinition bildet die Grundlage für die Realisierung des Projekts. Neben der Reduktion des Entwicklungsaufwandes ermöglicht diese die Validierung und Verifikation der Projektergebnisse und legt den Rahmen zur Überprüfung der Projektziel-Erreichung fest. Aufgrund der fortlaufenden Erfahrungen der Projektbeteiligten und den Änderungen in der Umwelt sowie bei den Stakeholdern unterliegen die Projektbedingungen einem ständigen Wandel. Diese resultierenden Anforderungsänderungen werden erst mittels Requiremetns-Management-Techniken systematisch erfass- und hadhabbar, um bei der Realisierung des Projektes Berücksichtigung zu finden.

Im Rahmen des Anforderungsmanagements geht es um die projektbegleitende Verfolgung, Verwaltung und Änderung der Anforderungen um die Projektziele bestmöglich zu erreichen. Neben der Auswahl und Bewertung von Technologien, resultierten die meisten Änderungen aus der fortlaufenden Konkretisierung der Anwendungsfälle.

Die Anforderungen sind kontinuierlich überprüft und bei Änderungen mit den Auftraggebern in den wöchentlichen Sitzungen verhandelt worden. Resultierende Änderungen wurden in den Sitzungsprotokollen festgehalten und sind folgend zusammengefasst und begründet.

05.08.2015 Hardware für Sensoren wurde festgelegt

Am 05.08.2015 sind die Sensoren für die Realisierung der folgenden Anforderungen mit den Auftraggebern verhandelt und verabschiedet worden.

- NFA30 (hochfrequente Daten bis 30 kHz)
- L-FA4 (Sensordaten über Einplatinencomputer verarbeiten)

Die folgenden vier angegebenen Sensorarten sind im Rahmen des Projekts zu berücksichtigen:

- Adafruit ADS1015 12-Bit ADC 4-Kanal (Spannungsmessung)
- Adafruit INA169 AnalogDC (Stromstärkesensor)
- Test-Soundsignal über eine Soundkarte erzeugt (Audiosensor)
- (Temperatursensor)
- 2 x Raspberry PI 2 B und ein Raspberry PI B (zur Sensoranbindung)

18.09.2015 Festlegung des DBMS

Dem Auftraggeber wurde am 18.09.2015 zur Abnahme des favorisierten DBMS Druid das Datenbankauswahlkonzept übergeben. Dieses beinhaltet die betrachteten Datenbanksysteme, deren Testergebnisse und eine abschließenden Bewertung. Druid ist von den Auftraggebern als DBMS angenommen worden.

30.09.2015 Mockup für Laotse GUI wurde angenommen

Gemäß den Anforderungen ist der erste Entwurf der GUI des Zielsystems von dem Auftraggeber angenommen worden.

- L-FA11 (Analysen auf Langzeitdaten) → in der GUI für vorgegebene Standardanalysen (MIN, MAX, AVG, SUM) weitere Anfragen ggf. mittels API extern ausgeben
- L-FA12 (Liveanalysen auf Datenströme ermöglichen) → Nutzung des Odysseus Studio Clients
- L-FA13 (Standardanalysen bereitstellen) → vorgegebene Standardanalysen (MIN, MAX, AVG, SUM)
- L-FA15 (Datenströme visualisieren) → Nutzung des Odysseus Studio Clients
- L-FA16 (Langzeitdaten visualisieren) → Standardanalysen in der GUI und weitere Anfragen ggf. mittels API extern ausgeben
- L-FA17 (Metadaten visualisieren) → in der GUI
- L-FA18 (Analyseergebnisse visualisieren) → Standardanalysen in der GUI
- L-FA19 (Visualisierungen ein- und ausblenden) → Visualisierungen wahlweise ein- und ausblenden
- NFA40 (Visualisierung durch Anwender auswählbar) → wahlweise ein- und ausblendbar
- NFA41 (GUI frei gestaltbar) → Verwendung von JavaFX GUI mit einzelnen Panes (Listpane, Ein- Ausgabepane und Menüleiste)

09.10.2015 erster Prototyp der GUI vorgestellt

Infolge der am 30.09.15 verhandelten Eigenschaften der GUI, ist der erste Prototyp vorgestellt und abgenommen worden.

- L-FA11 (Analysen auf Langzeitdaten) → in der GUI für vorgegebene Standardanalysen (MIN, MAX, AVG, SUM im Ausgabepane darstellbar)
- L-FA13 (Standardanalysen bereitstellen) → vorgegebene Standardanalysen (MIN, MAX, AVG, SUM werden angeboten)
- L-FA16 (Langzeitdaten visualisieren) → die Langzeitdaten lassen sich mittels der Standardanalysen in der GUI darstellen (Ausgabepane)
- L-FA17 (Metadaten visualisieren) → die Metadaten und weitere Eigenschaften der Sensoren sind in der GUI darstellbar (Metadaten, Standort)
- L-FA18 (Analyseergebnisse visualisieren) → Ergebnisse der Standardanalysen sind darstellbar
- L-FA19 (Visualisierungen ein- und ausblenden) → Visualisierungen wahlweise ein- und ausblenden mittels Auswahl der Analysen
- NFA40 (Visualisierung durch Anwender auswählbar) → Analysen sind wahlweise ein- und ausblendbar
- NFA41 (GUI frei gestaltbar) → Verwendung von JavaFX GUI mit einzelnen Panes (Listpane, Ein- Ausgabepane und Menüleiste)

09.10.2015 Verwendung Coding Styles

Die Verwendung der Coding Styles von Google wird verabschiedet.

- NFA24 (Codekonventionen einhalten)

21.10.2015 Vorstellung Zwischenbericht

Der Stand des Zwischenberichtes wurde vorgestellt und von den Auftraggebern abgenommen.

- NFA2 (Abgabe Zwischenbericht)

21.10.2015 Weiterer Stand des Laotse Prototyps wird vorgestellt

Der erweiterte Funktionsumfang der GUI wurde vorgestellt und verabschiedet.

- L-FA13 (Standardanalysen bereitstellen) → vorgegebene Standardanalysen (MIN, MAX, AVG, SUM werden unter dem Reiter Analyses angeboten)
- L-FA16 (Langzeitdaten visualisieren) → die Langzeitdaten lassen sich für einen ausgewählten Sensor mittels der Standardanalysen in der GUI darstellen (Ausgabepane)
- L-FA17 (Metadaten visualisieren) → die Metadaten und weitere Eigenschaften sind für die Sensoren darstellbar (Standort wird über Google Maps angezeigt)
- L-FA18 (Analyseergebnisse visualisieren) → Ergebnisse der Standardanalysen sind darstellbar
- L-FA19 (Visualisierungen ein- und ausblenden) → Visualisierungen wahlweise ein- und ausblenden mittels öffnen und schließen der Analysen

- NFA40 (Visualisierung durch Anwender auswählbar) → Analysen sind wahlweise aufrufbar

28.10.2015 Vorstellung Zwischenbericht

Der Stand des Zwischenberichtes wurde vorgestellt und von den Auftraggebern mit geringen Anmerkungen abgenommen.

- NFA2 (Abgabe Zwischenbericht)

11.11.2015 Funktionen Laotse GUI

Am 11.11.2015 wurde die Anforderung " Analysen sollen über mehrere Sensoren ausführt werden können" verabschiedet.

- L-FA11 (Analysen über mehrere Sensoren auf die Langzeitdaten ermöglichen)
- L-FA16 (analysierte Langzeitdaten mehrerer Sensoren visualisieren)

18.11.2015 Sensortyp Strommodul

Das defekte Strommodul AdafruitINA169AnalogDC wird auf Wunsch der Auftraggeber nicht neu beschafft und aus den Anforderungen entfernt.

- L-FA4 (Strommodul Adafruit INA169 AnalogDC wird nicht mehr berücksichtigt)

25.11.2015 Einbau Sensorhardware im SESALab

Die Sensorhardware soll über ein Hutschienengehäuse in das SESALab eingebaut werden. Dabei ist auf eine ansehnliche und sichere Montage zu achten.

- NFA19 (Sensorhardware soll in das SESALab eingebaut werden)

25.11.2015 Verwendung OPC UA Testserver

Eine Anforderung zur Einbindung eines verbreiteten Standards wurde vom Auftraggeber verhandelt. OPC UA soll als weitere Datenquelle berücksichtigt werden. Dazu sind Testdaten in das Zielsystem zu übertragen.

- L-FA5 (OPC UA soll als weitere Quelle als verbreiteter Standard verwendbar sein)

25.11.2015 Analysen über API ermöglichen

Von den Auftraggebern wurde die Berücksichtigung einer API zur Analyse der Langzeitdaten gewünscht. Die Integration von R API wird als nicht verbindliche Anforderung aufgenommen.

- L-FA11 (Individuelle Analysen werden bei Kompatibilität mit Druid über R API ermöglicht)

09.12.2015 Anforderungen Sensoreinbau

Die Anforderungen für den Sensoreinbau im SESALab wurden ergänzt.

- NFA19 (nicht transparent; für jeden Sensor einzelnes Gehäuse; Komponenten vor Berührung schützen)

09.12.2015 Funktionen Laotse GUI

Der aktuelle Stand der GUI wurde vorgestellt und von den Auftraggebern abgenommen. Das Menü ist um die Funktionalität eines Dashboards zur Auswahl und Anzeige der Analysen für die Sensoren erweitert worden. Des Weiteren wird die Datenverfügbarkeit der Sensoren nun angezeigt.

- L-FA11 (Analysen und Visualisierungen werden über ein Dashboard umgesetzt)
- L-FA16 (analysierte Langzeitdaten mehrerer Sensoren können in einem Dashboard angezeigt werden)
- NFA35 (Analysen und Visualisierungen sollen im laufenden Betrieb ermöglicht werden)

09.12.2015 Erweiterung Odysseus Controller

Eine Umsetzung einer API für das Anlegen der PQL Abfragen neuer Sensoren wird von den Auftraggebern nicht gefordert. Die Bereitstellung eines Templates ist ausreichend und soll hinreichend dokumentiert werden.

- L-FA5 (Die Odysseus Querys werden beim Hinzufügen neuer Sensoren über ein Template hinzugefügt)
- L-FA8 (Das Hinzufügen der Odysseus Query soll im laufenden Betrieb möglich sein)
- NFA31 (Sensorerweiterungen müssen im laufenden Betrieb möglich sein)
- NFA42 (Die Odysseus Querys sollen, wie im Handbuch beschrieben, hinzugefügt werden)

06.01.2016 mosaik Simulationsstarter

Das Starten der mosaik-Simulation über einen Websocket wird verabschiedet. Des Weiteren soll nach Möglichkeit das Auslesen der virtuellen Sensoren aus der Simulation automatisch erfolgen.

- L-FA6 (Das Einlesen der virtuellen Sensoren und der Start der Simulation soll nach Möglichkeit automatisiert erfolgen)

06.01.2016 Funktionen Laotse GUI

Die Durchführung eines Usability Tests, zur Überprüfung der Gebrauchstauglichkeit der GUI wurde verabschiedet.

- NFA37 (Die intuitive Bedienung soll über einen Usability Test überprüft werden)
- NFA38 (Die Bedienung erfolgt von Fachleuten)

13.01.2016 Analysen über API ermöglichen

Die Integration der R API ist abgeschlossen und funktioniert nun.

- L-FA11 (Individuelle Analysen werden mit Druid über R API ermöglicht)

20.01.2016 Anbindung BHKW-Container

Der BKHW-Container wird in absehbarer Zeit nicht mehr in Betrieb genommen, wodurch die Anforderung LF-21 von den Auftraggebern gestrichen wurde. Die Datenübertragung über OPC UA reicht aus.

- L-FA21 (Einbindung BHKW-Container wird nicht mehr umgesetzt)

03.02.2016 Analysen mittels R API

Seit dem letzten Update von HTTR funktioniert die R API aufgrund eines Problems in den JISON lite und HTTR Paketen nicht mehr. Das Problem ist von der Entwickler Community zu lösen und braucht von der PG nicht weiter betrachtet zu werden.

- L-FA11 (Individuelle Analysen werden derzeit nicht mehr mit Druid über R API ermöglicht)

10.02.2016 Datenübertragung OPC UA

Der OPC UA Operator für den Namenszugriff kann aufgrund der DNS-Problematik nicht verwendet werden. Für Testzwecke erfolgt der Zugriff ohne Authentifizierung. Dieser veränderte Operator wird in der Realität nicht verwendet und braucht daher nicht im Repository integriert werden. Die manuelle Demonstration reich aus.

- L-FA5 (OPC UA wird über einen Testserver verwendet)

10.02.2016 Verteilung Odysseus

Die Verteilung von Odysseus braucht von der PG nicht vorgenommen werden, da die Implementierung in Odysseus noch nicht abgeschlossen ist.

- L-FA7 (Odysseus wird in einer Instanz verwendet)

10.02.2016 mosaik Szenario

Im Rahmen der mosaik Simulation wird das bereitgestellte Demoszenario auf einem Testserver ausgeführt und das Hinzufügen weitere Szenarien im Entwicklerhandbuch beschrieben.

- L-FA6 (die Ausführung des bereitgestellten Demoszenarios erfolgt auf einem Testserver)
- L-FA5 (Die Einbindung weiterer Szenarien oder Simulationsserver wird im Entwicklerhandbuch beschrieben)

19.02.2016 Datenrate Audiosensor

Der für die Übertragung der Audiodaten verwendete Datengenerator ermöglicht eine stabile Datenübertragungsrate von max. 16 kHz. Darüber hinaus wird die Übertragung sprunghaft. Da dieser Anwendungsfall nur zu Testzwecken dient und in der Praxis nicht vorkommt, braucht keine weitere Zeit zur Optimierung investiert werden. Wichtig ist die Visualisierungsmöglichkeit der hochfrequenten Daten aus dem Langzeitspeicher.

- NFA30 (die Anforderung wurde auf 16 kHz aufgeweicht)

24.02.2016 Stand Laotse GUI

Der finale Stand der GUI wurde vorgestellt und von den Auftraggebern abgenommen. Die abschließende Abnahme erfolgt über den Usabilty Test, der die Gebrauchstauglichkeit feststellt. Ggf. resultierende Änderungen an der GUI sind mit den Auftraggebern zu verhandeln und bei Bedarf umzusetzen. Gemäß den Anforderungen sind die folgenden Funktionalitäten implementiert:

- Vorhanden Sensoren anzeigen
- Metadaten der Sensoren anzeigen
- Metadaten der Sensortypen anzeigen
- Analyseergebnisse/Messwerte der Sensoren anzeigen (Dashboard)
- Mögliche Sensortypen anzeigen
- Vorgegebene Analysen anzeigen
- Aufzeichnungszeiträume der Sensoren anzeigen
- Aggregationen auf die Analyseergebnisse vornehmen
- Neue Sensoren hinzufügen
- Sensoren bearbeiten (deaktivieren, bearbeiten der Metadaten wie Standort)
- Mosaik Simulationen initialisieren (Sensoren in Laotse einlesen), starten und aufzeichnen
- Hilfe

Und die GUI relevanten Anforderungen wie folgt umgesetzt:

- L-FA11 (Analysen auf Langzeitdaten) → in der GUI für vorgegebene Standardanalysen (MIN, MAX, AVG, SUM) können über ein Dashboard ausgeführt werden.
- L-FA13 (Standardanalysen bereitstellen) → vorgegebene Standardanalysen (MIN, MAX, AVG, SUM) werden unter dem Reiter Analysis angeboten
- L-FA16 (Langzeitdaten visualisieren) → die Langzeitdaten lassen sich für einen ausgewählten Sensor mittels der Standardanalysen in der GUI darstellen (Ausgabepane)
- L-FA17 (Metadaten visualisieren) → die Metadaten und weitere Eigenschaften sind für die Sensoren darstell- und änderbar (Standort wird über Google Maps angezeigt)
- L-FA18 (Analyseergebnisse visualisieren) → Ergebnisse der Standardanalysen sind darstellbar. Dafür können Zeiträume ausgewählt und Aggregationen vorgenommen werden
- L-FA19 (Visualisierungen ein- und ausblenden) → Visualisierungen wahlweise ein- und ausblenden mittels öffnen und schließen der gewünschten Dashboards

- NFA40 (Visualisierung durch Anwender auswählbar) → Analysen/Dashboards sind wahlweise aufrufbar

04.03.2016 Usability Test

Der Usability Test, zur Überprüfung der Gebrauchstauglichkeit der GUI wurde, vorgestellt und abgenommen.

- NFA37 (Die intuitive Bedienung soll über einen Usability Test überprüft werden)
- NFA38 (Die Bedienung erfolgt von Fachleuten)

04.03.2016 Abschlusstest

Der Abschlusstest zur Überprüfung der Anforderungen wurde vorgestellt und von den Auftraggebern abgenommen. Die Überprüfung erfolgt zum Ende des Projekts und das Ergebnis wird als kompakte Tabelle aufbereitet.

- NFA7 / NFA8 (Der Abschlusstest wird angemessen dokumentiert und mit den Auftraggebern verhandelt)

11.03.2016 Abnahme Usability Test und GUI

Die Ergebnisse des Usability Tests wurden dem Auftraggeber vorgestellt und mögliche Änderungen verhandelt. Die GUI ist für den Anwendungsfall problemlos verwendbar und bedarf keiner funktionalen Änderungen. Verbesserungen am Design und der Anordnung gehören nicht zu den verpflichtenden Anforderungen und werden daher nur umgesetzt, wenn dieses zeitlich möglich ist. Die Nutzung der GUI ist ausreichend im Benutzerhandbuch zu dokumentieren.

- NFA37 (Die intuitive Bedienung ist möglich und wird mit einer Hilfe und einem Benutzerhandbuch unterstützt)
- NFA38 (Die Bedienung erfolgt von Fachleuten)
- L-FA11 (Analysen auf Langzeitdaten)
- L-FA13 (Standardanalysen bereitstellen)
- L-FA16 (Langzeitdaten visualisieren)
- L-FA17 (Metadaten visualisieren)
- L-FA18 (Analyseergebnisse visualisieren)
- L-FA19 (Visualisierungen ein- und ausblenden)
- NFA40 (Visualisierung durch Anwender auswählbar)

11.03.2016 Vorstellung Stand Abschlussbericht

Der Stand der Abschlussdokumentation wurde vorgestellt und von den Auftraggebern abgenommen.

- NFA4 (Abgabe Abschlussbericht)

16.03.2016 Erweiterung GUI

Um höher frequente Daten ordnungsgemäß in der GUI anzeigen zu können, ist eine Auswahl einzelner weniger Datenpunkte nötigt. Aus diesem Grund soll eine Pagination implementiert werden, wodurch beispielsweise 100 Werte aus den Langzeitdaten ausgewählt und angezeigt werden können. Die Umsetzung ist nicht verpflichtend, aber sehr gewünscht.

- L-FA11 (Analysen auf Langzeitdaten)
- L-FA16 (Langzeitdaten visualisieren)

22.03.2016 Zwischenstand Abschlussbericht

Der Zwischenstand des Abschlussberichtes wurde den Auftraggebern zur Verfügung gestellt und um Anmerkungen und Kommentare gebeten.

- NFA4 (Abgabe Abschlussbericht)

8.2. Codequalität

LS

Dieser Abschnitt beschreibt die zur Unterstützung der Qualitätssicherung verwendeten Tools.

8.2.1. SonarQube

SonarQube ist ein hilfreiches Werkzeug der Firma Sonarsource für statische Codeanalysen im Continuous Integration-Prozess, welches u.a. für die Integration in einen CI-Server entwickelt wurde, vgl. die zugehörige Website [Son].
SonarQube kommt mit einer Reihe von built-in Standardanalysen, die sich mit den häufigsten Ursachen für schlechte Codequalität befassen, die nach eigener Aussage in [Sqq] wie folgt lauten:

1. **Code-Clones** Duplizierter Code ist eine häufige Quelle für Defekte im späteren Produkt, da ein einmal vorhandener Mangel durch unkontrolliertes Copy-Paste leicht vervielfältigt wird und somit auch die Wahrscheinlichkeit steigt, dass selbst wenn ein Defekt erkannt wird, nicht alle Duplikate gefunden werden. Diesem Problem ist natürlich leicht mit dem Generalisieren von Methoden bzw. Klassen abzuhelfen.

   SonarQube ist nach Website ([Sqq], Duplications) in der Lage Level 2, d.h. bis auf Isomorphie und Kommentare identische, Klone zu identifizieren.

2. **Codestandards** SonarQube kann ähnlich wie Checkstyle Codingstandards kontrollieren, vgl. Abschnitt 8.2.2.

3. **Testabdeckung** SonarQube überprüft und dokumentiert die Testabdeckung von committetem Quellcode.

4. **Potentielle Fehler** SonarQube kann bestimmte Defekte im Code identifizieren, die unter bestimmten Voraussetzungen zu einem Fehler im System führen können, wie z.B. Null-Pointer-Fehler oder Schleifenrandfälle.

5. **Komplexität** Große Klassen mit vielen Funktionen etc. sind i.d.R. schwer zu durchblicken. Manchmal sind sie unausweichlich. Aber eine hohe Anzahl von extrem komplexen Klassen ist eher ein Zeichen von schlechter Qualität und sollte daher vermieden werden. Gegenmaßnahmen sind das Aufspalten in mehrere Klassen o.ä.

6. **Dokumentation** SonarQube achtet auf das Vorhandensein von Javadoc zu jeder Klasse und Methode, wobei es gleichzeitig eine zu hohe Anzahl von Kommentaren markiert, da dies die Übersichtlichkeit verringert.

7. **Spaghetticode** (aka. Maintenance nightmare) Beim Programmieren ist es wichtig, auf eine möglichst geringe Vernetzung der Klassen untereinander zu achten. Ist dies dennoch der Fall, so wird das Ändern oder gar Austauschen einer Komponente zu einer sehr zeit- und fehlerträchtigen Angelegenheit. Dem kann u.a. mit der fachgerechten Nutzung von Interfaces und einer guten Architektur abgeholfen werden. SonarQube überprüft den Vernetzungsgrad von Klassen und gibt ggf. Warnungen aus.

Im Rahmen dieses Projektes wurde SonarQube auf demselben Server wie das SVN-Repository installiert, sowie das SonarQube-Plugin dem Jenkinsserver hinzugefügt, so dass dieser derart konfiguriert werden konnte in Folge jeden Commits eine Analyse anzustoßen.

Ergebnisse

Dieser Abschnitt wertet den Stand der SonarQube Analysen nach dem letzten Änderungen am 28.03.2016 aus.
Verschiedene Metriken, nach [GAC14, 11 ff.,116 ff.] und nach [Sqm], die sich auf die geschilderten Code Probleme beziehen, wurden unter anderem mit SonarQube auf den LAOTSE Projekten ausgewertet und im Folgenden näher erläutert:

Größe Diese Metriken geben sowohl allgemeine Angaben zur physikalischen Größe des Projektes, also der Anzahl der Dateien, Zeilen etc., als auch programmiersprachen spezifische Angaben, wie die Anzahl der Funktionen, Klassen, LOC an.

Accessors Zusammenfassung von Java Gettern und Settern.

LOC (Lines Of Code) gibt die Anzahl von Zeilen, die effektiven Java Code enthalten, d.h. die weder Kommentare noch Leerzeilen sind.

Dokumentation Metriken zur Dokumentationsabdeckung:

% Dokumentation Prozentuale Dokumentation von Klassen, Methoden etc.

% Comments Prozentualer Anteil von Kommentarzeilen an LOC.

Public API Anzahl von mit dem Keyword `public` bezeichneten Java Definitionen. Dabei wird jede Definition nur einmal gezählt, also würde eine öffentliche Methode in einem Interface aber nicht in der implementierenden Klasse aufgeführt werden. Getter und Setter sind ebenfalls ausgeschlossen.

Komplexität Für die Berechnung der Komplexität in Java Projekten wird in SonarQube die McCabe Metrik verwendet, die vereinfacht ausgedrückt bei jedem Keyword, das die Verschachtelung verstärkt um 1 erhöht wird. Genauer kann die Funktionsweise der McCabe Metrik auf der SonarQube Dokumentations Website [Sqm] nachgelesen werden. Alle Methoden und Klassen haben eine initiale McCabe Metrik von 1.

Package Struktur Diese Metriken geben an, in wie weit Klassen mit Klassen aus anderen Packages verzahnt sind und Abhängigkeitskreise bilden.

Directory Tangle Index (DTI) Diese Metrik gibt den Prozentsatz an Packages an, die in Abhängigkeitskreisen mitwirken.

Cycles Effektive Anzahl von Abhängigkeitskreisen.

Dependencies to Cut Abhängigkeiten, die entfernt werden sollten, um Kreise zu entfernen.

Probleme Dieses Widget zeigt von SonarQube identifizierte Codequalitätsprobleme an.

Technical Debt Die von SonarQube errechnete Zeit, die für die Ausbesserung der im Code vorhandenen Schwachstellen benötigt wird. Dabei wird je nach Kategorie des Issues mehr Zeit eingeplant: Diese Kategorein lauten wie folgt:

Blocker Fehler, sollte sofort behandelt werden, *blockiert* andere Features.

Critical *Kritische* Codestelle, ist wahrscheinlich ein Fehler.

Major Möglicher Programmierfehler, ist wahrscheinlich nicht so vom Programmieren beabsichtigt gewesen.

Minor Funktionierende aber unschöne Lösung von Programmieraufgabe.

Info Keine fehleranfällige aber z.B. stilistisch abweichende Lösung.

Technical Debt Ratio Das von SonarQube errechnete Verhältnis, zwischen der Technical Debt und der Zeit, die es kosten würde das Projekt neu zu schreiben.

SQALE Rating Die Bewertung des Technical Debt Ratio mit A als bester und E als schlechtester Note.

Da SonarQube an Jenkins angeschlossen wurde, sind die Ergebnisse bzgl. der 3 Hauptprojekte Client-, Server- und Modell-Projekt und ihren jeweiligen Unterprojekten aufgeteilt. Das `ResourceParent` wird nicht betrachtet, da nur Drittbibliotheken eingebunden wurden deren Qualität nicht untersucht wird.

Client Im gesamten Client-Projekt ist vor allem die Dokumentation und das Fehlen von, von SonarQube erkannten, Duplikaten positiv hervorzuheben.
Auch die Technical Debt und daraus folgend das SQALE-Rating sind im höchstmöglichen Bereich angesiedelt.
Die Komplexität liegt mit 2.0 pro Funktion durchschnittlich im sehr guten Bereich, allerdings gibt es, wie in der Abbildung 8.1 zu sehen, einige wenige sehr komplexe Funktionen.
Weniger gut fällt der DTI aus, dies liegt vor allem an den Controllern der LaotseGui, die eng verflochten sind, was aber durch den generellen JavaFX Aufbau vorgeschrieben wurde.

Server Wie zu sehen in Abbildung 8.2 ist ähnlich wie beim Client die Dokumentationsabdeckung und die Komplexität beim Server-Projekt sehr gut.
Es gibt ungefähr das Doppelte an Technical Debt, was auch mit dem sehr geringen Clientwert zusammenhängt, aber dennoch ein SQALE-Rating mit der Bestnote.
Zusätzlich sind die beiden höchsten Issue Kategorien überhaupt nicht vertreten und der Großteil der Issues befindet sich in den untersten Kategorien.

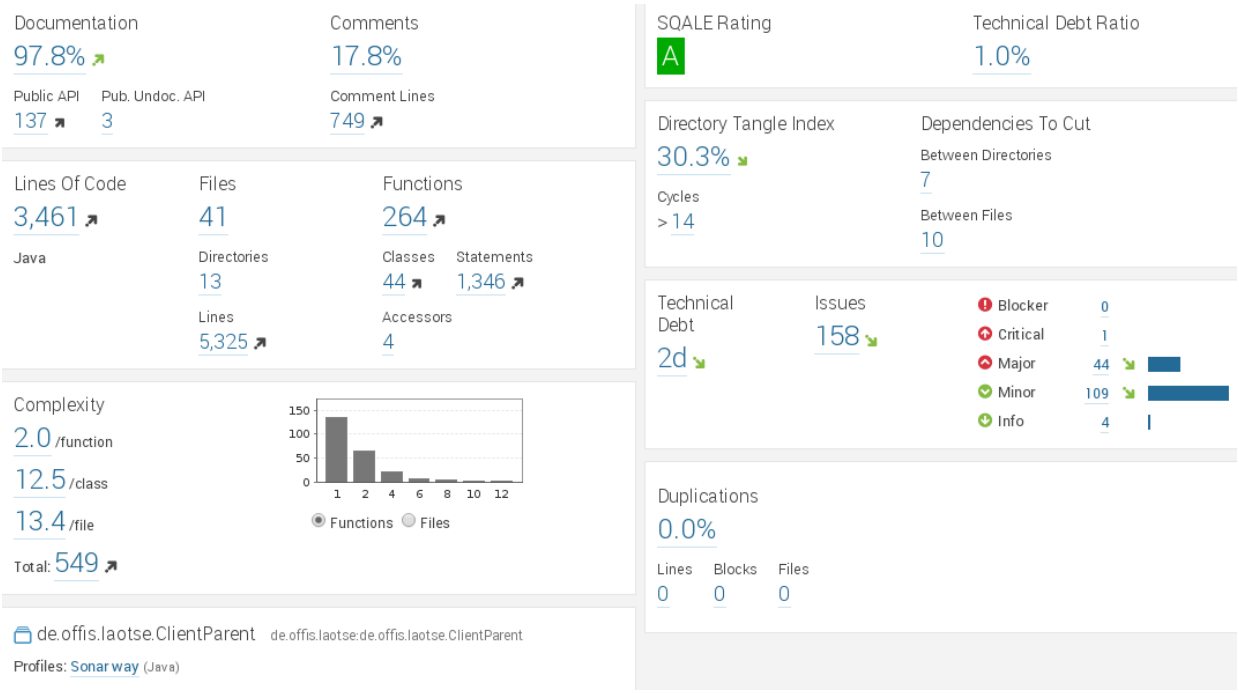


Abbildung 8.1.: SonarQube Ergebnisse Client-Projekt

Das Duplications-Widget zeigt an, dass SonarQube Code-Clone gefunden hat. Dabei handelt es sich um das `ColumnHead`, der ModelDaos, da dieses in jedem Dao implementiert wird (vgl. Abschnitt 7.3.6).
Im Vergleich zum Client ist der DTI beim Server deutlich kleiner und es gibt nur 2 Abhängigkeitskreise.

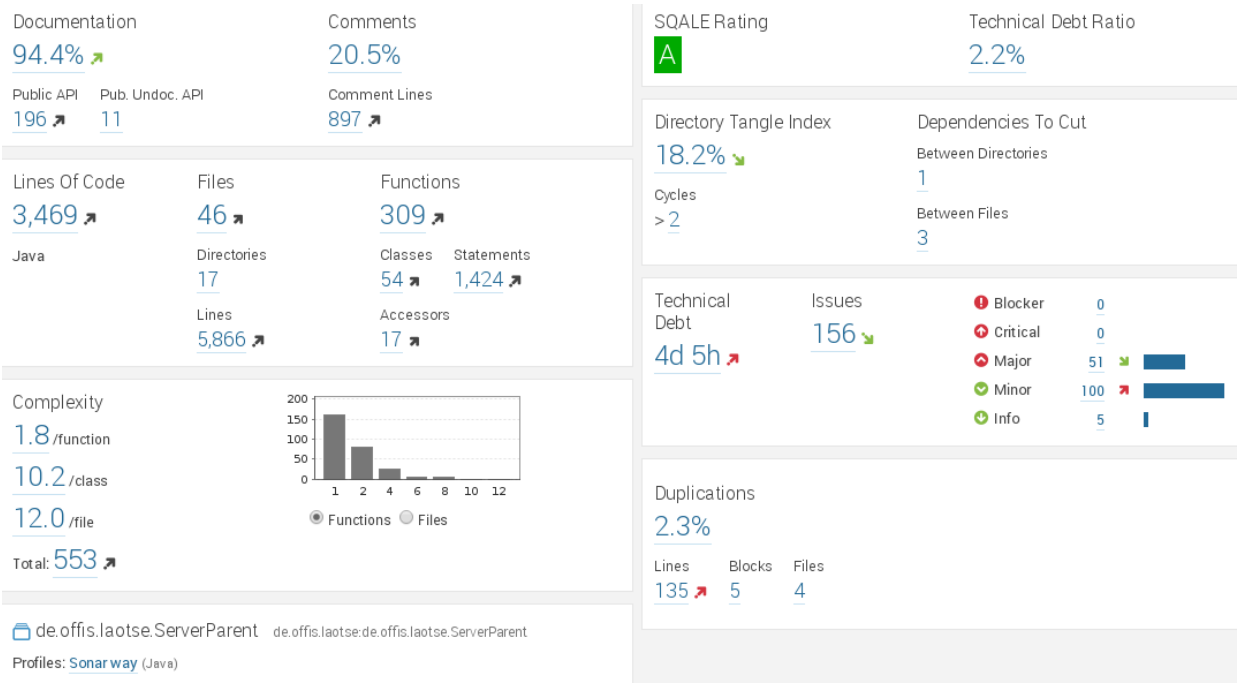


Abbildung 8.2.: SonarQube Ergebnisse Server-Projekt

Model Im Modell-Projekt ist nach Abbildung 8.3 ebenfalls eine sehr gute Dokumentationsabdeckung und Komplexität erreicht worden und eine generelle Abwesenheit von nachweisbaren Duplikaten.
Der DTI ist ähnlich gering wie der vom Server-Projekt und die Issue-Anzahl sehr niedrig, was in dem besten SQALE-Rating resultiert.

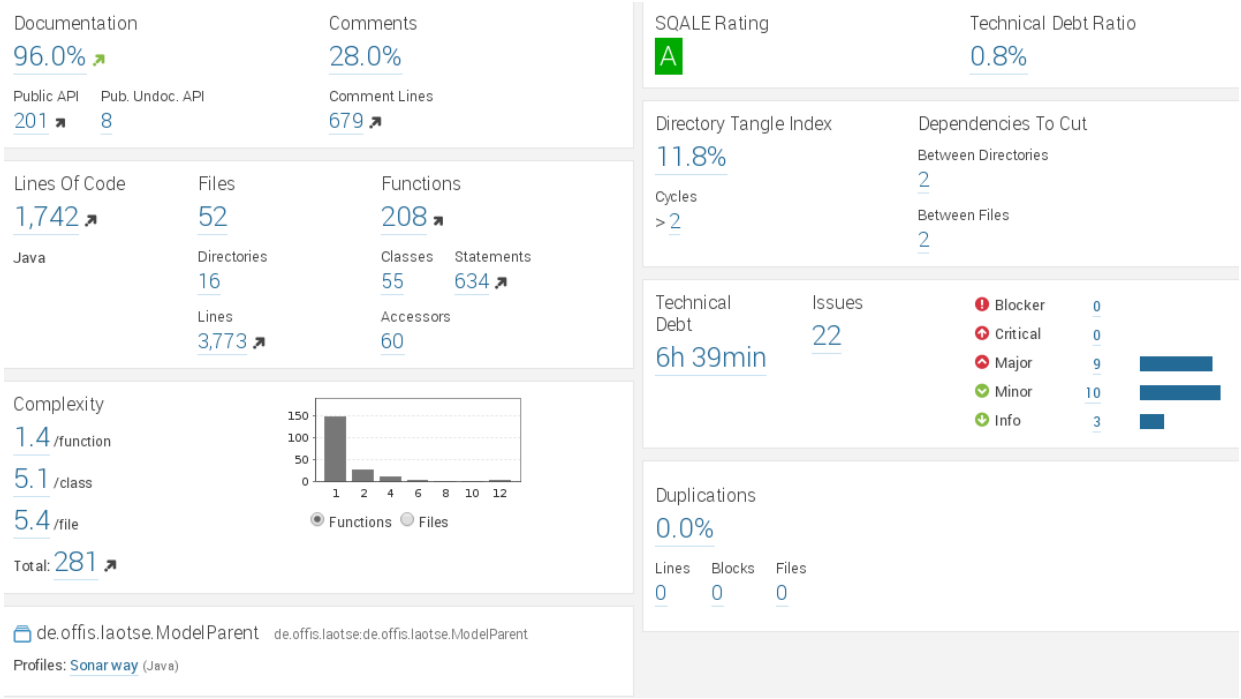


Abbildung 8.3.: SonarQube Ergebnisse Modell-Projekt

Insgesamt sind alle Projekte von SonarQube sehr gut bewertet worden, was auch mit dem durchgehenden Einsatz von Checkstyle während der Projektlaufzeit, durch alle Programmierer zusammenhängt, das im folgenden Abschnitt erläutert wird.

8.2.2. Checkstyle

Checkstyle ist ein Eclipse-Plugin, das den Einhalt von Codingstandards überprüft. Im Checkstyle-Plugin kann ein bestimmter Codestyle eingestellt werden; in diesem Fall der Google Java Style, detailliert beschrieben auf der zugehörigen Website ([Goo]). Dieser wurde in diesem Projekt gewählt, da er relativ neu ist und allgemeinen Konsens gefunden hat.
Solch ein Codingstyle schreibt diverse syntaktische Regeln, wie Benennung aller Art, Formatierung sowie zu vermeidende Java-Konstrukte etc. vor.
Checkstyle markiert bei allen Java-Klassen den Verstoß gegen eine Regel mit einer gelben Warnung in der Eclipse-IDE, so dass der Entwickler beim Programmieren direkt reagieren und seinen Code anpassen kann, bevor er ihn commitet.

8.3. Unit Tests

LS

Unit Tests sind ein bewährtes Mittel, um in sich abgeschlossene Komponenten (Units), z.B. Klassen, Methoden, hinsichtlich ihrer Funktionalität zu überprüfen.
In LAOTSE wurden Unit Tests hauptsächlich in Bezug auf Datenbankzugriffe eingesetzt, aus dem einfachen Grund, dass viele GUI basierte Tests schwierig automatisiert durchgeführt oder überprüft werden können, z.b. Überprüfung der Anpassung der Fenstergröße etc., und in der LAOTSE-GUI allgemein sehr wenig unabhängige Komponenten vorhanden sind.

8.3.1. JUnit

Da LAOTSE in Java programmiert ist bietet sich JUnit, als weit verbreitetes und von Eclipse unterstütztes Testframework, zur Nutzung an. Detailliertere Beschreibungen von JUnit können auf der zugehörigen Website [JUn16] nachgelesen werden.
Eine JUnit Testklasse sollte sich auf genau eine Java Klasse beziehen und deren Methoden testen. Für eine beispielhafte Java Klasse `ExampleClass` könnte ein zugehöriger einfacher Testklassenrumpf wie in Listing 8.1 aussehen.

Listing 8.1: JUnit Test Beispiel

```
1  public class ExampleClassTest {
2
3      @BeforeClass
4      public static void setUp() {
5      }
6
7      @Before
8      public void prepareTest() {
9      }
10
11     @Test
12     public void testFunctionality1() {
13     }
14
15     @Test
16     public void testFunctionality2() {
17     }
18
19     @After
20     public void cleanUp() {
21     }
22
23     @AfterClass
24     public static void tearDown() {
25     }
26 }
```

Dabei benutzt JUnit mehrere Java Annotations, um verschiedenen Methodentypen zu unterscheiden, die im Folgenden näher erläutert werden:

@Test Die folgende Methode ist ein Test und von JUnit durchzuführen.

@BeforeClass Die so überschriebene Methode soll ausgeführt werden bevor die Klasse selbst geladen wird, aus diesem Grund ist sie als `static` deklariert, und kann für einmalige vorbereitende Aktionen wie z.B. den Aufbau einer Socketverbindung genutzt werden.

@Before Diese Methode wird vor jedem Test ausgeführt und kann z.B. für die Erzeugung eines immer gleichen Grundzustands genutzt werden, ein Reset.

@After Diese Methode wird nach jedem Test ausgeführt, z.B. um eine Datenbank wieder aufzuräumen.

@AfterClass Diese Methode wird nach der Ausführung aller Testfälle einmalig ausgeführt, um z.B. eine Verbindung wieder zu trennen oder eine Datenbank zu leeren.

Alle solchen Testklassen, die sich im *src/test/java*-Package befinden, werden automatisch bei jedem Maven build mit ausgeführt.

8.4. Continuous Integration

LS

Continuous Integration ist eine Technik zur Codeverwaltung in größeren, von mehreren Teams bearbeiteten, Softwareprojekten. Sie resultiert aus dem Vorgehensmodell des Extreme Programming.
In diesem agilen Verfahren wird zumeist Pair Programming, also das Programmieren in Teams aus jeweils 2 Personen, genutzt. Beim Ineinanderfügen der Codes aller Teams treten häufig Probleme auf und bis die gesamte Software wieder lauffähig ist, wird viel Zeit in die *Integration* aller Teilergebnisse investiert.
Um diesen Aufwand so gering wie möglich zu halten, ist es sinnvoll jedes abgeschlossene Inkrement sofort in die *Mainline*, dem zentralen Code z.B. auf einem Server, zu integrieren.
Natürlich sollte sichergestellt werden, dass der neu integrierte Code zum einen nicht bereits vorhandene und funktionierende Features zerstört und zum anderen selbst wie gewünscht arbeitet.
Um dies sicherzustellen, müssen, sobald neuer Code eingefügt wird, alle vorhandenen Unit-Tests erneut, und bevorzugt automatisiert, durchgeführt werden, damit mögliche Fehler sofort erkannt und auf einen bestimmten Commit eingeschränkt werden können.
Ein so arbeitender *Integration Server* lässt sich mit einem VCS verbinden und sollte auf jeden Commit mit dem automatischen Anstoßen eines integrierten Build-Tools, sowie Durchführung der Testfälle reagieren.
Anschließend bietet er die Möglichkeit des direkten Deployments in die Zielumgebung (oder einer Kopie davon).
Solch ein Server übernimmt also einen großen Teil der Aufgaben von Continuous Integration.
In diesem Projekt wurde ein Jenkins-Server aufgesetzt, der diese Tätigkeiten übernimmt.

8.4.1. Jenkins

DN

In diesem Projekt wird ein Jenkins Server als Integration Server genutzt, da dieser für Maven-basierte Projekte mit sehr geringem Arbeitsaufwand eingerichtet werden kann und darüber hinaus die am weitesten verbreitete Software ist.
Des Weiteren bietet Jenkins eine ganze Reihe von nützlichen Erweiterungen an, die zur Erhaltung der Codequalität beitragen können, wie z.B. SonarQube.
Jenkins ist so konfiguriert, dass die Laotse-Module Client, Model, Server und Resource jeweils in einem eigenen Job gebaut werden. Jenkins lädt die benötigten Dateien aus dem Subversion-Repository. Bei Änderungen in einem der Module wird der Build-Prozess automatisch angestoßen, da Jenkins regelmäßig nach neuen Revisionen sucht. Am Ende des Builds eines Moduls werden davon abhängige Module gebaut.
Neben den Laotse-Modulen wird außerdem der Mosaik-Starter und die an LAOTSE angepassten Odysseus-Anwendungen gebaut.

Wenn die Build-Jobs über eine Maven-Konfigurationsdatei geladen werden, leitet Jenkins den Build-Prozess an Maven weiter. Maven unternimmt die üblichen Schritte, die über die angegebenen Maven-Goals definiert sind. Jenkins kann die Artefakte, die durch Maven erzeugt werden nutzen. Ergebnisse von Testfällen, die durch JUnit erzeugt wurden, werden von Jenkins aufbereitet. Jenkins erzeugt in der Übersicht eines Jobs einen Link zur Javadoc-Dokumentation des Moduls, wenn diese erzeugt wurde. Durch weitere Plugins lassen sich weitere Aufgaben automatisiert ausführen. Jenkins wurde so konfiguriert, dass nach den Builds der Laotse-Module eine Codeanalyse durch SonarQube durchgeführt wird.

Jenkins wurde eingesetzt, um die Integration von Änderungen am Code auf Fehler zu überprüfen. Es wurde so konfiguriert, dass alle Benutzer, die Änderungen an einem Modul vorgenommen haben, dessen Build fehlschlägt, eine Mail erhalten. Dadurch ist es möglich, schnell auf Fehler zu reagieren. Sobald der Build-Job wieder erfolgreich ausgeführt wird, erhalten alle Nutzer, die bis dahin im Mailverteiler waren darüber eine Nachricht. Build-Jobs können als unstable markiert werden, wenn nicht alle Tests mit Erfolg ausgeführt wurden. In diesem Fall werden ebenfalls Mails nach gleicher Regelung verteilt.

8.5. Usability Test

UG

Der Usability Test umfasst eine Reihe verschiedener Methoden, welche die Gebrauchstauglichkeit der zu entwickelnden Software überprüfen und optimieren. Damit das Zielsystem gemäß den Erwartungen und Anforderungen des Auftraggebers zielgerichtet entwickelt werden kann, ist bereits frühestmöglich mit der Überprüfung und Einbeziehung der Auftraggeber begonnen worden. Bereits im fünften Sprint wurde mit der Entwicklung der GUI begonnen. Anfangs wurden die in der GUI zu realisierenden Funktionalitäten bestimmt und mittels der Card-Sorting Methode eine benutzerfreundliche Menüstruktur und ein erstes Mockup erstellt. Nach der Vorstellung mit anschließendem Feedback der Auftraggeber, erfolgte die prototypische Implementierung. Die

Weiterentwicklung der GUI erfolgte fortlaufend mit regelmäßiger Vorstellung beim Auftraggeber. Abschließend soll die Gebrauchs- und Benutzertauglichkeit der Laotse-GUI vollständig überprüft werden, um die Erwartungen und Anforderungen des Auftraggebers gerecht zu werden.

Dazu wurde ein moderierter Usability Test durchgeführt, bei dem die Probanden der Reihe nach typische Aufgaben durchführt haben. Vor der Aufgabenbearbeitung wurde mit Hilfe eines sogenannten 5 Sekundentest, die Erwartung und Vermutung zum Lösungsweg, unvoreingenommen von der Realisierung in der GUI, ermittelt. Bei Fragen oder Unklarheiten griff der Moderator ein und gab bei Bedarf Hilfestellung. Nachfolgend ist die Zusammenfassung der Ergebnisse des Usability Tests aufgelistet. Der gesamte Testablauf ist mit dem Testleitfaden und der vollständigen Auswertung im Anhang F Usability Test einsehbar.

8.5.1. Zusammenfassung

Das primäre Ziel der Funktionstauglichkeit wurde von allen vier Testprobanden bestätigt. Die GUI ist für den Anwendungsfall verwendbar und bedarf keiner funktionalen Änderungen. Die Hälfte der Testpersonen fanden die GUI nach erster Nutzung intuitiv, einer empfand die GUI sofort mittels Ausprobieren intuitiv und einer könnte sich die Nutzung mit einer Hilfe problemlos vorstellen. Alle Kandidaten fanden die GUI übersichtlich und gut strukturiert. In vielen Bereichen wurde die Benutzerinteraktion durch geeignete visuelle Hilfsmittel gut unterstützt und Fehleingaben mit Warnmeldungen und kontrollierten Eingaben abgefangen. Lediglich im Bereich der Visualisierung gab es leichte Kritik bezüglich der Fenstergröße und Design im Untermenü der Metadaten und Eigenschaften, einer nicht ganz konsistenten Umsetzung der Verwendung von Anklicken und Doppelklick sowie des Kontrastes der GUI durch die Verwendung von schwarzem Hintergrund und weißer Schrift.

Zusammenfassend sind alle Anmerkungen, welche von 50 % der Probanden genannt wurden aufgelistet.

Allgemein Funktionstauglichkeit:

- reicht für Anwendungsfall aus, keine funktionalen Änderungen nötig (4 Personen)
- nach erster Nutzung ist GUI intuitiv (2 Personen)

Design:

- Schrift/Hintergrund: Kontrast (schwarz auf weiß) nicht so gut (2 Personen)
- linkes Pane muss auch in Breite verstellbar sein (lange Sensornamen) (2 Personen)
- Aufteilung und Logik der Fenster allgemein dokumentieren insbesondere bei Metadaten und Eigenschaften (2 Personen)
- Übersicht gut (2 Personen)

Sensoren:

Sensoren Anzeigen:

- Weg: Sensors → Show Sensors (wird neu Laden erwartet) (4 Personen)

Metadaten/Eigenschaften anzeigen + editieren:

- Weg: Sensors → Doppelklick (3 Personen)
- Map-Darstellung zu klein beim Öffnen und Zoom erwartet (4 Personen)
- Bearbeiten der Metadaten über Edit plausibel (4 Personen)
- Fenstereigenschaften der Metadaten (Öffnen und Schließen der Panes) + Edit/Save/Cancel Buttons besser anordnen (3 Personen)
- Mouseover der Metadaten (Werte und Funktion nicht intuitiv ersichtlich) (2 Personen)
- Map zeigt den Standort(Marker) nicht mittig bei Google Maps an (2 Personen)
- Doppelklick vs. Einfachklick (konsistent umsetzen) (2 Personen)

Sensor hinzufügen:

- Pflichtdaten nicht ersichtlich (Speichern/Anlegen erst mit Angabe der Pflichtdaten möglich) (4 Personen)
- Kontext der Fields der Metadaten wie z.B. Gewicht des Sensors oder Active und Autostart nicht intuitiv (4 Personen)
- Vorgehen über Sensors → Add Sensors schlüssig (3 Personen)
- Eingabevalidierung intuitiv (3 Personen)
- Eingabe intuitiv (3 Personen)

Aufzeichnungszeitraum der Sensoren:

- Weg: Sensors → Doppelklick → Data (3 Personen)
- gutes Mapping mit Balken bei Verfügbarkeit (3 Personen)
- gut das Daten auch aus der Verfügbarkeitsansicht für den Zeitraum angesehen werden können (2 Personen)

Dashboard:

Erstellen:

- Weg: Dashboard → new Dashboard (2 Personen)
- Sensor → new Dashboard (gut das Sensor dann bereits ausgewählt ist) (2 Personen)

Angabe der Eingabedaten:

- Auswahl durch Eingabe (3 Personen)
- Auswahl durch Kalender (1 Person) → Zeitauswahl zu klein und dunkel
- Granularität per Dropdown gut (4 Personen)
- Start des Dashboards intuitiv über Button (4 Personen)
- Fortschritt über Ladebalken gut (3 Personen)
- Warnmeldung bei großer Granularitätsauflösung gut (2 Personen)

Darstellung Dashboard:

- Darstellung der Messwerte als Diagramm mit Datums- und Zeitangabe gut (3 Personen)
- Konsolenausgabe verständlich Ermittlungszeitraum einer Analyse ermittelt (3 Personen)
- Anpassen der Fenstergröße gut (3 Personen)
- Übersicht der vordefinierten Analysen (MIN, MAX, AVG, Werteanzahl) gut (2 Personen)

Darstellung Dashboard:

- Darstellung der Messwerte als Diagramm mit Datums- und Zeitangabe gut (3 Personen)
- Konsolenausgabe verständlich Ermittlungszeitraum einer Analyse ermittelt (3 Personen)
- Anpassen der Fenstergröße gut (3 Personen)
- Übersicht der vordefinierten Analysen (MIN, MAX, AVG, Werteanzahl) gut (2 Personen)

Analysetypen anzeigen:

- Query nicht intuitiv (2 Personen)
- Bedeutung der Attribute unklar besonders das Parsing über Position (2 Personen)

Mosaik:

- Aufruf und Nutzung verständlich (zuerst Verbindung zum Simulator über Initialisierung) → Simulation Starten (4 Personen)

Neue Simulation (Initialisierung):

- Mosaik → neue Simulation (4 Personen)
- Parameterangabe für Initialisierung verständlich (4 Personen)
- Start der Initialisierung über Button verständlich (4 Personen)
- Start der Simulation für Datenaufzeichnung mittels Button Start Simulation verständlich (4 Personen)

Anzeige der virtuellen Sensoren:

- Anordnung unter Mosaik-Child nicht plausibel (2 Personen)
- Mosaik Aufbau intuitiv (2 Personen)

9. Evaluation

KB

In diesem Abschnitt findet eine Evaluation des angefertigten Systems statt. Dabei soll überprüft werden, ob die ermittelten Anforderungen erfüllt wurden und zu welchem Grad. Ziel des Projektes war es ein System zu schaffen, dass hochfrequente Daten aus mosaik und von echten Sensoren als Datenstrom interpretiert und diese Daten durch das Datenstrommanagementsystem Odysseus in das ausgewählte Datenbanksystem Druid weitergibt. In Druid sollen die Daten persistent gespeichert werden, aber auch nahezu in Echtzeit abrufbar sein. Diese gespeicherten Daten sollen in einer angemessenen Zeit abrufbar und visualisierbar sein, damit der Benutzer Analysen auf diesen Daten ausführen kann und gewisse Schlüsse aus den ermittelten Daten ziehen kann.
Als Anwendungsfall für diese Evaluation dient ein mosaik-Szenario, dass 136 Sensoren enthält. Die Daten dieser Sensoren sollen mittels Odysseus in Druid gespeichert werden und von dort abgerufen und über das entwickelte Dashboard visualisiert werden. Dadurch kann ermittelt werden, ob das System fähig ist hochfrequente Daten zu speichern und diese in einer angemessenen Zeit abrufen und visualisieren zu können.
Im Abschlusstest wurde genau dieser Anwendungsfall betrachtet und überprüft, ob bei der Durchführung keine Daten verloren gehen und diese Daten somit jederzeit verfügbar sind. Außerdem wurde überprüft, innerhalb welches Zeitraumes die Daten in der Datenbank Druid angelangt sind und von dort abgerufen werden können.
Über das entwickelte System konnte die mosaik-Simulation initialisiert werden. Die Sensoren der Simulation wurden ermittelt und im System angelegt. Alle Sensoren der Simulation waren in der LAOTSE-GUI visualisiert. Nach Abschluss der Initialisierung, konnte die Simulation gestartet werden. Während der Simulation wurde der Fortschritt in der GUI angezeigt und in Odysseus alle für den Abruf der Daten nötigen Querys angelegt.
Demnach gelangten die Daten der mosaik-Sensoren als Datenstrom zu Odysseus und von dort weiter an Kafka. Kafka hat zunächst die eintreffenden Daten gespeichert, um Datenverlust zu vermeiden und daraufhin die Daten entsprechend an Druid für die persistente Speicherung weitergegeben.
Über die Druid Benutzeroberfläche war ersichtlich, dass der Realtime-Knoten von Druid zu arbeiten begann und nach einer gewissen Zeit Buckets an Cassandra für die Langzeitdatenspeicherung weitergab. Durch Veränderung der Verteilung, z.B. Hinzufügen eines weiteren Cassandra-Knotens oder eines weiteren Realtime-Knotens konnte dieser Vorgang beschleunigt und ein Überlauf verhindert werden. Über die LAOTSE-GUI waren die Daten aus Druid abruf- und über das Dashboard visualisierbar. Die Daten der Simulation wurden demnach in Druid persistent gespeichert.
Dieser Test hat gezeigt, dass das System hochfrequente Daten verarbeiten kann und diese Daten nach der Speicherung wieder abrufbar sind, um sie zu visualisieren oder zu analysieren. Während dieses Tests wurden weitere reale Sensoren eingeschaltet, um die Datenlast weiter zu erhöhen und zu zeigen, dass das System in der Lage auch reale Sensordaten zu verarbeiten.
Wie dieser Test gezeigt hat, werden die ermittelten Kernfunktionalitäten durch das System erfüllt,

da hochfrequente Daten gespeichert werden können und für eine Visualisierung wieder abrufbar sind.
Allerdings erzeugt das gegebene mosaik-Szenario nicht, die in den Anforderungen ermittelte, Datenlast und kann somit nicht definitiv beweisen, dass das System mit einer solchen Datenlast umgehen kann. Andere Tests waren allerdings während der Laufzeit des Projektes nicht möglich, da die gegebene Hardware nicht ausreichte, um eine solche Datenlast zu verarbeiten. Da das SESA-Lab nicht im Dauerbetrieb läuft und solche hochfrequenten Daten aktuell im SESA-Lab nicht erzeugt werden, bestand allerdings auch keine Datengrundlage, um das System im Dauerbetrieb über mehrere Tage zu testen.
Der Test hat gezeigt, dass bei weiterer Hinzunahme von Knoten des Druid-Clusters und von Cassandra die ankommenden Daten schneller verarbeitet und alle Daten gespeichert werden konnten. Wenn das Cluster also vergrößert wird, können dementsprechend mehr Daten aufgenommen werden. Wenn das System beliebig skalierbar wäre, sollte auch eine deutlich höhere Datenlast verarbeitet werden können, als der Test gezeigt hat. Dieser Anwendungsfall konnte aus zeitlichen Gründen jedoch nicht mehr durch einen Test bewiesen werden und diese Funktionalität kann damit nur theoretisch angenommen werden.
Alles in allem hat der Test mit dem gegebenen Anwendungsfall gezeigt, dass das System mit hochfrequenten Daten umgehen kann und diese Daten ohne Verlust gespeichert und in einer angemessenen Zeit wieder abgerufen werden können. Der Test wurde allerdings mit einer nicht so hohen Datenlast durchgeführt wie ursprünglich angenommen, da für diese Datenlast nicht genügend Ressourcen zur Verfügung standen. Außerdem konnte kein dauerhafter Betrieb über mehrere Tage getestet werden, da das SESA-Lab nicht dauerhaft Daten liefert. Dennoch kann davon ausgegangen werden, dass das System auch mit höherfrequenten Daten umgehen kann, sofern genügend Hardwareressourcen zur Verfügung stehen, denn mit Vergrößerung des Clusters konnten auch mehr Daten aufgenommen werden. Das entwickelte System erfüllt also die ermittelten Hauptfunktionen und kann somit als System gesehen werden, dass den Anforderungen entspricht.

9.1. Ergebnis

LS

Dieser Abschnitt beschreibt die Ergebnisse des Abschlusstests von LAOTSE.
Dabei wurden die funktionalen Anforderungen anhand der finalen Softwareversion evaluiert.
Es folgen die Anforderungen jeweils mit einer Kennzeichnung bezüglich ihrer Erfüllung.
Mit einem (-) gekennzeichnete Anforderungen sind dabei, in Absprache mit den Betreuern, im Laufe des Projektes gestrichen worden.
Eine detaillierte Begründung zu den jeweiligen Ergebnissen befindet sich im Anhang (siehe Dokument G).

9.1.1. Funktionale Anforderungen

Ergebnisse Funktionale Anforderungen:

Erg	ID	Prio	Anforderung
✓	L-FA1	A	Das System soll die eingehenden Sensordaten (Roh- und Metadaten) persistent speichern können.

✓	L-FA2	A	Das System soll die eingehenden Daten verlustfrei verarbeiten können.
✓	L-FA3	A	Das System soll den Messzeitpunkt (Timestamp) zu den gespeicherten Daten speichern können.
✓	L-FA4	A	Das System soll Sensordaten von Raspberry PIs verarbeiten können.
✓	L-FA5	A	Das System soll die Möglichkeit bieten weitere Quellen, über aktuell verbreitete Standards, hinzufügen zu können.
✓	L-FA6	A	Das System muss Sensordaten aus dem SESA-Lab, die über mosaik bereitgestellt werden, verarbeiten können.
✓	L-FA7	A	Das System muss Datenströme mit Hilfe des vorhandenen DSMS Odysseus verarbeiten können.
✓	L-FA8	B	Das System soll die Möglichkeit, bieten Systemerweiterungen (Hardware, Software) im laufenden Betrieb ermöglichen zu können.
✓	L-FA9	B	Wenn Datenquellen ausfallen, muss das System fähig sein, unterbrechungsfrei weiterlaufen zu können.
✗	L-FA10	B	Wenn ausgefallene Datenquellen wieder verfügbar sind, soll das System diese automatisch wieder einbinden können.
✓	L-FA11	B	Das System soll Analysen auf den gespeicherten Langzeitdaten durchführen können.
✓	L-FA12	B	Das System soll Liveanalysen auf den Datenströmen durchführen können.
✓	L-FA13	B	Das System soll fünf vorimplementierte Standardanalysen, bereitstellen können.
✓	L-FA14	B	Das System soll um weitere Analysen, durch den Benutzer (Forscher), ergänzt werden können.
✓	L-FA15	B	Das System soll die Möglichkeit bieten, Datenströme visualisieren zu können (Beispielsweise die aktuell erzeugte Leistung eines Windparks).
✓	L-FA16	B	Das System soll die Möglichkeit bieten, Langzeitdaten visualisieren zu können (bspw. die erzeuge Leistung des letzten Jahres).
✓	L-FA17	B	Das System soll die Möglichkeit bieten, Metadaten visualisieren zu können (bspw. welche Sensoren aktuell Daten an das System senden).
✓	L-FA18	B	Das System soll die Möglichkeit bieten, Analyseergebnisse aus den Langzeitdaten und Datenströmen, visualisieren zu können (z.B. Mittelwert der letzten 3 Wochen).
✓	L-FA19	B	Das System soll die Möglichkeit bieten, Visualisierungen ein- und ausblenden zu können.
✓	L-FA20	B	Das System soll um weitere Visualisierungen, durch den Benutzer (Forscher), ergänzt werden können.
-	L-FA21	C	Das System soll die Daten, des über Fernwirkprotokolle angeschlossenen BHKW-Containers in der Lesumstraße in Oldenburg, verarbeiten können. (Wenn dieser nicht rechtzeitig in Betrieb genommen wird, erfolgt die Einbindung eines bereit gestellten Dummy-BHKWs.)

✗	L-FA22	C	Das System soll den Ausfall von Sensoren melden können (z.B. als visuelle Hervorhebung oder Nachricht).
✗	L-FA23	C	Das System soll Meldungen unterschrittener Schwellwerte ausgeben können (z.B. als visuelle Hervorhebung oder Nachricht).
✗	L-FA24	C	Das System soll um weitere Meldungen erweitert werden können.
✓	L-FA25	C	Das System soll die zu speichernden Daten komprimieren können.

9.1.2. Nicht-Funktionale Anforderungen

Ergebnisse Nicht-Funktionale Anforderungen:

Projektanforderungen / Anforderungen an Durchführung und Entwicklung

Erg	ID	Prio	Kategorie	Anforderung
✓	NFA1	A	Budget	Wenn Investitionen zur Realisierung des Projektes erforderlich sind, muss das Projektteam Angebote zur Auswahl bereitstellen.
✓	NFA2	A	Lieferumfang und Termine	Das Projektteam muss nach der Hälfte der Projektzeit einen Zwischenbericht, welcher neben dem Grobkonzept, das Feinkonzept enthält, vorlegen.
✓	NFA3	A	Lieferumfang und Termine	Das Projektteam soll nach der Hälfte der Projektzeit einen Prototyp bereitstellen.
✓	NFA4	A	Lieferumfang und Termine	Das Projektteam soll am 31.03.2016 den Abschlussbericht, welcher neben dem Zwischenbericht, das Feinkonzept, ein Installations- sowie Benutzerhandbuch enthält, bereitstellen.
-	NFA5	C	Lieferumfang und Termine	Bei allen Dokumenten soll nach Möglichkeit der Autor hervorgehen.
✓	NFA6	A	Lieferumfang und Termine	Das Projektteam führt monatliche Reviews mit dem Auftraggeber durch.
✓	NFA7	B	Testkonzept	Das Testkonzept soll von Beginn an erstellt und angemessen dokumentiert werden.
✓	NFA8	B	Testkonzept	Das Testkonzept kann bei Bedarf und in Absprache mit den Betreuern erweitert werden.
✓	NFA9	B	Evaluierung	Systemkomponenten sollen mit umfangreicher Alternativenbetrachtung, Begründung und Dokumentation ausgewählt werden.
-	NFA10	A	Dokumentation	Die Dokumentation muss fortlaufend und präzise während der gesamten Projektzeit erfolgen.
✓	NFA11	A	Handbücher	Die Handbücher müssen für Fachleute und in der deutschen Sprache erstellt werden.

Rechtliche Anforderungen

Erg	ID	Prio	Kategorie	Anforderung
✓	NFA12	A	Datenschutz	Die bereitgestellten Daten und Systeme müssen vertraulich, ohne absichtliche Weitergabe an Außenstehende, behandelt werden.
✓	NFA13	B	Rechtliches	Urheber- und Lizenzrechte müssen gewahrt werden.
✓	NFA14	B	Rechtliches	Anforderungsänderungen können nur durch Zustimmung des Auftraggebers durchgeführt werden.
✓	NFA15	B	Rechtliches	Weitere rechtliche Rahmenbedingungen müssen nicht beachtet werden.

Technische Anforderungen

Erg	ID	Prio	Kategorie	Anforderung
✓	NFA16	A	Technologie	Die Technologie kann grundsätzlich frei gewählt werden, außer Vorgaben oder vorhandene Systeme widersprechen dem.
✓	NFA17	B	Technologie	Die Technologieauswahl muss immer ordnungsgemäß evaluiert und dokumentiert werden.
✓	NFA18	A	Technologie	Die Technologie soll unter Berücksichtigung der Ausbaufähigkeit und des langfristigen Einsatzes ausgewählt werden.
✓	NFA19	B	Systemaufbau	Raspberry PIs sollen als Hardwareaufbau realisiert und in das System eingebunden werden.
✓	NFA20	A	Systemaufbau	Die Systemkomponenten des DSMS sollen auf bereitgestellten VM-Instanzen realisiert werden.
✓	NFA21	B	Schnittstellen	Wenn Standards für die Schnittstellen vorhanden sind, sollen diese nach Möglichkeit verwendet werden.
✓	NFA22	A	Schnittstellen	Wenn Schnittstellen bereitgestellt werden, muss deren Spezifikation und Beschreibung erstellt werden.
✓	NFA23	C	Programmiersprache	Die Programmiersprache kann frei gewählt werden.
✓	NFA24	B	Programmiersprache	Codekonventionen sollen eingehalten werden.
✓	NFA25	C	Speicherkapazität	Die Speicherkapazität für die Langzeitspeicherung steht aufgrund der immer weiter sinkenden Kosten ausreichend zur Verfügung.
✓	NFA26	B	Speicherkapazität	Wenn eine Kompression möglich ist, soll diese aus Performance- und Speicherauslastungsgründen implementiert werden,
✓	NFA27	C	Hardware- und Softwarekomponenten	Das Projektteam muss den Ausfall einzelner Systemkomponenten wie DB-Ausfälle oder sonstige Katastrophen nicht berücksichtigen.

✓	NFA28	B	Hardware- und Softwarekomponenten	Das System soll nach Möglichkeit modular aufgebaut werden, um System- und Kapazitätserweiterungen zur Laufzeit zu ermöglichen.
✓	NFA29	A	Hardware- und Softwarekomponenten	Das System soll die Möglichkeit bieten Daten, von bis zu 1000 Sensoren, zu verarbeiten.
✓	NFA30	A	Hardware- und Softwarekomponenten	Das System soll fähig sein hochfrequente Daten, von bis zu 30 kHz, zu verarbeiten.

Qualitätsanforderungen

Erg	ID	Prio	Kategorie	Anforderung
✓	NFA31	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb hinzugefügt werden können.
✓	NFA32	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb modifiziert werden können.
✓	NFA33	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb deaktiviert werden können.
✓	NFA34	C	Verfügbarkeit	Analysen sollen im laufenden Betrieb möglich sein.
✓	NFA35	C	Verfügbarkeit	Visualisierungen sollen im laufenden Betrieb möglich sein.
✓	NFA36	C	Verfügbarkeit	Kapazität- und Systemerweiterungen sollen im laufenden Betrieb möglich sein.
✓	NFA37	C	Bedienbarkeit	Das System soll von Fachleuten bedient werden können.
✓	NFA38	C	Bedienbarkeit	Das System soll möglichst intuitiv bedienbar sein.
✓	NFA39	B	Bedienbarkeit	Analysen sollen durch den Anwender auswählbar sein.
✓	NFA40	B	Bedienbarkeit	Visualisierungen sollen durch den Anwender auswählbar sein.
✓	NFA41	C	Bedienbarkeit	Das GUI kann frei gestaltet werden.
✓	NFA42	A	Bedienbarkeit	Quellen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.
✓	NFA43	C	Bedienbarkeit	Analysen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.
✓	NFA44	C	Bedienbarkeit	Visualisierungen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.
✓	NFA45	B	Performanz	Das System soll auf Anfragen, die einen Zeitraum von maximal zwei Jahren umfassen, innerhalb einer Stunde das Ergebnis liefern können.

Insgesamt lässt sich festhalten, dass sämtliche Anforderungen mit der Prioriät A, die am Ende des Projektes noch bestanden, von der Projektgruppe erfüllt wurden und nur eine B, sowie drei C Anforderungen nicht erfüllt werden konnten.
Damit erfüllt LAOTSE trotz der geringen Gruppengröße noch 38 von 42 bestehenden Anforderungen.

10. Resumee

UG

Das Ziel des Projekts SESAdata war die Entwicklung einer Datenmanagementumgebung, um Forschungsprojekten im Bereich Smart Grid eine Plattform zur Speicherung von Energiedaten bereitzustellen. Dafür mussten die zum Teil sehr hochfrequenten und von heterogenen Sensoren stammenden Daten mit Zeitstempeln verlustfrei verarbeitet und persistent gespeichert werden.
Darüber hinaus soll das Zielsystem Analysen und Visualisierungen der gespeicherten Langzeitdaten ermöglichen, um Erkenntnisse aus diesen Daten zu erzielen.
Die größte technische Herausforderung lag dabei in der hochfrequenten Verarbeitung und verlustfreien Speicherung der Messwerte mit ihren genauen Messzeitpunkten.
Zur Lösung der Aufgabenstellung war das Zusammenwirken vieler Themenbereiche und Technologien erforderlich. Zunächst musste das Anwendungsgebiet in der Energiedomäne aufgrund der unpräzisen und offenen Aufgabenstellung umfassend analysiert werden. Die Verarbeitung und Speicherung der Messwerte mittels Datenströmen erforderte die Einarbeitung in unterschiedliche Technologien wie DSMS, DBMS, Datenübertragungsprotokolle und der Quellsysteme (Sensoren, mosaik). Dazu kamen noch weitere Schwerpunkte wie die Entwicklung einer GUI zur Visualisierung und Steuerung sowie die Analyse und Darstellung der gespeicherten Daten.
Die Vielseitigkeit der Aufgabenstellung hat den Projektmitgliedern viel Können abverlangt und dabei umfangreiche neue Kenntnisse geschaffen.

Aufgrund des Ehrgeizes der Projektgruppe, die vorgegebenen und im Projektverlauf erweiterten Ziele zu erreichen, entstand mit dem entwickelten System LAOTSE ein sehr guter Ansatz zur Speicherung und Analyse der Massendaten in der Energiedomäne. Dieser für die Forschung so wichtige Grundstein zeigt das Potential und die Möglichkeiten der verwendeten Technologien. Die Kernaufgabe der persistenten Speicherung, der aus verschiedenen Datenquellen stammenden, zum Teil hochfrequenten Daten, ist sehr erfolgreich gelöst worden. Mit LAOTSE ist die Speicherung der Messwerte mit den zugehörigen Zeitstempeln für alle geforderten Datenquellen möglich. Eine Analyse und Visualisierung der Langzeitdaten wird komfortabel mittels Dashboards in der LAOTSE-GUI ermöglicht.
Ebenso ist das Zielsystem wie gefordert zukünftig erweiterbar. LAOTSE ist wie im Entwicklerhandbuch siehe Anhang I beschrieben, um weitere neue Datenquellen erweiterbar. Neue Sensoren oder mosaik-Simulationen, bereits existierender Quellentypen, können komfortabel über die GUI hinzugefügt und bearbeitet werden.

Die GUI gehörte nicht zu den Kernaufgaben, ist aber aufgrund der zentralen Bedeutung für die Bedienung und Steuerung des Systems professionell entwickelt worden.

Projekte dieser Größe lassen sich nur dann erfolgreich bewältigen, wenn alle Projektmitglieder engagiert und koordiniert zusammenarbeiten. Aufgrund der geringen Anzahl von 5 Mitarbeitern

erforderte dieses eine sehr gute Planung und Abstimmung. Das anfangs entwickelte Vorgehensmodell nach Scrum ermöglichte eine optimale Planung und Verteilung der Aufgaben unter regelmäßiger Einbeziehung der Auftraggeber.
Rückblickend hat die Teamarbeit dank der regelmäßigen Treffen und der projektbegleitenden Teambildungsmaßnahmen, wie Social Events und gemeinsamer Mittagessen, sehr gut funktioniert. Die gemeinsamen Arbeitstreffen ermöglichten eine enge und effiziente Zusammenarbeit, wodurch sofort wichtige Details, Ideen oder Probleme besprochen und gemeinsam gelöst werden konnten.
Aufgrund der wöchentlichen Treffen (Weekly Scrum) mit dem Auftraggeber, ist dieser ausgiebig in den Entwicklungsprozess integriert worden. Diese trugen zur Transparenz bei und sorgten dafür, dass Probleme und Änderungen frühzeitig erkannt wurden.
Für den Projektverlauf war das ausführliche Lastenheft und daraus resultierenden Arbeitspakete im Product-Backlog ein zentrales Element. In diesem Dokument wurden alle Anforderungen detailliert aufgegliedert, sodass eine realistische Einschätzung des Arbeitsaufwandes ermöglicht wurde. Dennoch gab es während des Projektverlaufs hin und wieder Verzögerungen durch fehlende Ressourcen oder Änderungen. Durch eine entsprechende Umplanung der Arbeitspakete, konnte die Fertigstellung des Produkts zum genannten Endtermin trotzdem gewährleistet werden.
Des Weiteren war die Rollenverteilung der Mitglieder für die gut funktionierende Teamarbeit von großer Bedeutung. So gab es die Rollen Product-Owner, Scrum-Master / Projektleiter, Administrator und für jedes wichtige Themengebiet einen Spezialisten.

Abschließend lässt sich sagen, dass die intensive Mitarbeit und hohe Motivation des gesamten Teams, zum erfolgreichen Abschluss des Projekts führten. Die Projektgruppe hat sowohl fachlich als auch organisatorisch einiges gelernt, welches der zukünftigen Bearbeitung größerer Projekte sehr dienlich ist.

11. Ausblick

DN

Mit Projektabschluss bleiben weitere Ansätze, wie das System erweitert und verbessert werden kann, offen. Einige dieser Ansätze werden in diesem Abschnitt vorgestellt.
Zunächst kann die Funktionalität der GUI erweitert werden. Zur Zeit können Sensoren über die GUI hinzugefügt und bearbeitet werden. Die gleichen Funktionalitäten können für Sensortypen und Analyse-Templates erweitert werden, damit die Konfiguration unabhängiger vom direkten Zugriff auf die MySQL-Datenbank wird.

Für den Einsatz von mosaik wurden Composite-Sensoren entwickelt, da die Daten von mosaik gesammelt an Odysseus übergeben werden. Diese Composite-Sensoren werden intern als virtuelle Sensoren behandelt, welche jeweils einen Sensorwert beinhalten. Diese Technik lässt sich generalisieren, um so beliebige Kombinationen von virtuellen Sensoren zu ermöglichen und beliebig große Einheiten zu bilden. So könnten auch baumartig angeordnete Knoten von OPC UA-Servern sinnvoll dargestellt werden.

Zur Konfiguration von Sensoren werden Odysseus-Anfrage-Templates benötigt, damit die entsprechenden Anfragen in Odysseus gestartet werden können. Zur Entwicklung der Anfragen muss der Anwender in die Sprachen CQL oder PQL eingearbeitet sein. Es wäre gut, wenn es Unterstützung für den Anwender gibt, so dass er den Quelltext nicht selbst eingeben muss, sondern dieser aus Modellen generiert oder mithilfe einer GUI erstellt wird.

In der Projektgruppe wurde der Ausfall von Sensoren und sonstige Katastrophen nicht betrachtet. Odysseus ist dahingehend erweiterbar, dass die Zuverlässigkeit der Datenströme überprüft werden kann. So können Events ausgelöst werden, falls ein Sensor ausfällt oder eine Verbindung abbricht. Sollten innerhalb von Operatoren Exceptions auftreten, sollten diese standardisiert behandelt werden, um bspw. einen Neustart der Anfrage oder andere Recovery-Methoden zu verwenden. Damit einhergehend kann eine Komponente für „Alarme und Events“ entwickelt werden, die in Ausnahmefällen Informationen versendet, so dass ein Datenverlust für den Anwender sichtbar wird und dieser entsprechend reagieren kann.

Die Verteilung der Systemkomponenten ist noch nicht ausgereizt. Weitere Versuche können zeigen, was die Verteilung einzelner Komponenten bewirkt. Innerhalb der Projektgruppe wurde gezeigt, dass die Erstellung der Druid-Datensegmente sehr resourcenintensiv ist.
Anstelle des Realtime-Knotens kann auch der sogenannte Indexing Service verwendet werden. Das dazu benötigte Tool Tranquility war während der Entwicklung von LAOTSE nicht an die aktuelle Druid-Version angepasst. Es lohnt sich, ein Konzept zu erarbeiten und zu evaluieren, welches die gleichzeitige Replizierung und Indizierung auf mehrere Knoten verteilt ermöglicht, um eine Ausfallsicherheit zu gewährleisten.

Vor kurzer Zeit wurde Odysseus Net veröffentlicht. Es verteilt die Datenströme auf unterschiedliche Odysseus-Instanzen. Es wäre sinnvoll zu evaluieren, wie die Performanz des Systems verbessert wird, wenn Odysseus mit dieser Technik verteilt ausgeführt wird.
Die Verteilung von Knoten wurde in der Projektgruppe manuell geplant und eingerichtet. Im Produktivbetrieb kann es vorkommen, dass der Resourcenbedarf schwankt. Es ist sinnvoll eine Lösung zu finden, die bestimmte Knoten dynamisch in das Cluster einbringt, wenn diese gebraucht werden. Damit die Konfiguration der einzelnen Knoten vereinfacht wird, bieten sich Containerlösungen, zum Beispiel mithilfe von Docker [Doc], an. Die vorkonfigurierten Container können dann von Orchestrierungstools dynamisch zur Laufzeit in das Cluster geladen werden.

Der Schwerpunkt der Projektgruppe lag auf der Speicherung von Sensordaten. Das Dashboard ermöglicht einfache Visualisierungen der gespeicherten Daten. Für komplexere Analysen wurde die Druid R API getestet. Die darin neu entwickelte Select-Anfrage kann dahingehend erweitert werden, dass Pagination unterstützt wird und so Analysen über quasi unbegrenzt große Datensätze möglich sind. Außerdem könnte ein Konzept entwickelt werden, welches Analysen vorberechnet und zur schnellen Abfrage in Odysseus ablegt. Interessant sind solche Vorberechnungen, wenn zum Beispiel Analysen über bestimmte Zeiträume regelmäßig abgerufen werden.

Zu guter Letzt wurde LAOTSE für relativ abstrakte Anwendungsfälle entwickelt. Mit der Integration des Systems in das SESA-Lab ergeben sich weitere Ansätze, wie das System verbessert werden kann, um zusätzlichen Nutzen zu bieten.

Literatur

[Adaa] *Adafruit's Raspberry-Pi Python Code Library*. Zugriff: 19.03.2016. 2016. URL: https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.

[Adab] *Bild Adafruit ADS1015 Spannungsmessung*. Zugriff: 30.10.2015. 2015. URL: http://www.flikto.de/products/ads1015-12-bit-adc-4-channel-with-programmable-gain-amplifier.

[Adac] *Bild Adafruit INA169 Stromstärkemessung*. Zugriff: 30.10.2015. 2015. URL: http://www.flikto.de/products/adafruit-ina169-analog-dc-current-sensor-breakout-60v-5a-max.

[Ads] *ADS1015 12-Bit ADC - 4 Channel with Programmable Gain Amplifier*. Zugriff: 19.03.2016. 2016. URL: https://www.adafruit.com/products/1083.

[Aer15] Inc. Aerospike. *Aerospike*. Webseite. letzter Zugriff: 14.07.2015. Juli 2015. URL: http://www.aerospike.com.

[Agi] *Manifesto for Agile Software Development*. letzter Zugriff: 10.08.2015. 2001. URL: http://agilemanifesto.org.

[Apaa] *Apache Commons Configuration 1.10*. Zugriff: 18.03.2016. 2016. URL: https://commons.apache.org/proper/commons-configuration/index.html.

[Apab] *Apache Kafka*. Zugriff: 29.03.2016. 2016. URL: http://kafka.apache.org/.

[Apa15] ApacheSoftwareFoundation. *ApacheWiki - Cassandra*. Webseite. letzter Zugriff: 14.07.2015. Juli 2015. URL: http://wiki.apache.org/cassandra/FrontPage.

[App+12] H.-Jürgen Appelrath u. a. "Odysseus: A Highly Customizable Framework for Creating Efficient Event Stream Management Systems". In: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*. DEBS '12. New York, NY, USA: ACM, 2012, S. 367–368. ISBN: 978-1-4503-1315-5. DOI: 10.1145/2335484.2335525. URL: http://doi.acm.org/10.1145/2335484.2335525.

[BGK14] R. Bachmann, T. Gerzer und D. G Kemper. *Big Data - Fluch oder Segen? - Unternehmen im Spiegel gesellschaftlichnen Wandels*. Mitp Verlag., 2014.

[Cira] *Bild Cirrus_Logic_Audio_Rasp Sounderweiterung*. Zugriff: 30.10.2015. 2015. URL: https://www.rasppishop.de/Cirrus-Logic-Audio-fuer-Raspberry-Pi-Hat-Modelle.

[Cirb] *Cirrus Logic Audio Card*. Zugriff: 19.03.2016. 2016. URL: https://www.element14.com/community/community/raspberry-pi/raspberry-pi-accessories/cirrus_logic_audio_card.

[Cra] *Kommerzielle digitale Überwachung im Alltag*. Zugriff: 23.03.2016. 2016. URL: http://crackedlabs.org/dl/Studie_Digitale_Ueberwachung.pdf.

[Dao] *Core J2EE Patterns - Data Access Object*. Zugriff: 14.03.2016. 2016. URL: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html.

[DF15] Ted Dunning und Ellen Friedman. *Time Series Databases - New Ways to Store and Access Data*. Sebastopol: O'Reilly Media, Inc., 2015.

[Doc] *Docker - Build, Ship, and Run Any App, Anywhere*. Zugriff: 30.03.2016. 2016. URL: https://www.docker.com/.

[Drua] *Druid Documentation - Querying*. Zugriff: 22.03.2016. 2016. URL: http://druid.io/docs/0.8.3/querying/querying.html.

[Drub] *Druid Documentation - Select Queries*. Zugriff: 22.03.2016. 2016. URL: http://druid.io/docs/0.8.3/development/select-query.html.

[Druc] *Druid Segments*. Zugriff: 28.03.2016. 2016. URL: http://druid.io/docs/latest/design/segments.html.

[Dru15] Druid. *druid*. Webseite. letzter Zugriff: 14.07.2015. Juli 2015. URL: http://druid.io.

[Ds1] *DS18B20 Temperature Sensor with Raspberry Pi*. Zugriff: 18.3.2016. 2016. URL: http://www.reuk.co.uk/DS18B20-Temperature-Sensor-with-Raspberry-Pi.htm.

[Dw1] *Usability-Tests selber durchführen*. Zugriff: 03.03.2016. 2016. URL: www.drweb.de/magazin/usability-tests-selber-durchfuhren/.

[Fen] *Beschreibung des Fensteroperators im Odysseus-Wiki*. Zugriff: 29.04.2015. 2015. URL: http://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/Aggregation+and+Window.

[GAC14] Patroklos P. Papapetrou G. Ann Campbell. *SonarQube in Action*. Manning Publications Co., 2014.

[Gar] *Big data is high-volume, high-velocity and high-variety in formation assets that demand cost- effective, innovative forms of information processing for enhanced insight and decision making*. Zugriff: 23.03.2016. 2016. URL: http://www.gartner.com/it-glossary/big-data.

[Glu15] Ivan Glushkov. *NewSQL Overview*. Presentation. letzter Zugriff: 10.08.2015. Feb. 2015. URL: http://de.slideshare.net/IvanGlushkov/newsql-overview.

[Goo] *Google Java Style*. letzter Zugriff: 24.10.2015. 2015. URL: https://google.github.io/styleguide/javaguide.html.

[Gpi] *GPIO State Listener Example using Pi4J*. Zugriff: 22.03.2016. 2016. URL: http://pi4j.com/example/listener.html.

[Gra14] Marco Grawunder. *Datenstromverarbeitung*. Vorlesung im Mastermodul: Informationsmanagement in verteilten Systemen (Informationssysteme III). 2014.

[Hau08] Erich Hau. *Windkraftanlagen : Grundlagen, Technik, Einsatz, Wirtschaftlichkeit*. 4., vollst. neu bearb. Aufl. Berlin [u.a.] : Springer, 2008. ISBN: 3540721509 9783540721505 9783540721512.

[Inf15] InfluxDB. *InfluxDB*. Webseite. letzter Zugriff: 11.08.2015. Juli 2015. URL: https://influxdb.com.

[JUn16] JUni. *JUnit Website*. Zugriff: 19.03.2016. 2016. URL: http://junit.org/.

[Kaf] *Kafka 0.9.0 Documentation*. Zugriff: 22.03.2016. 2016. URL: http://kafka.apache.org/090/documentation.html#producerconfigs.

[Lit] *Oracle JavaFX Scene Builder*. Zugriff: 30.03.2016. 2014. URL: http://www.oracle.com/pls/topic/lookup?ctx=javase80&id=JSBGS164.

[LR09] Bernhard Lahres und Gregor Rayman. *Praxisbuch Objektorientierung*. Galileo Press, 2009.

[MN14] Mark Milster und Dennis Nowak. *Datenstrommanagement zur Überwachung und Kurzzeitprognose der Leistung von Windparks*. Bachelorabschlussarbeit in der Abteilung Informationssysteme der Universität Oldenburg. 2014.

[Mos] *Mosaik Dokumentation*. letzter Zugriff: 29.10.2015. 2015. URL: http://mosaik.readthedocs.org/en/latest/overview.html#mosaik-s-main-components.

[Mys] *MySQL 5.7 Reference Manual*. Zugriff: 22.03.2016. 2016. URL: http://dev.mysql.com/doc/refman/5.7/en/.

[Mü+13] Sven C. Müller u. a. "Einbindung von intelligenten Entscheidungsverfahren in die dynamische Simulation von elektrischen Energiesystemen". In: *Informatik Spektrum* 36.1 (2013), S. 6–16. ISSN: 0170-6012. DOI: 10.1007/s00287-012-0668-6.

[Odya] *Internetauftritt des Odysseus Projekts*. Zugriff: 29.04.2015. 2015. URL: http://odysseus.informatik.uni-oldenburg.de/cms/.

[Odyb] *Odysseus Dokumentation - Access Framework*. Zugriff: 22.03.2016. 2016. URL: http://odysseus.informatik.uni-oldenburg.de:8090/display/ODYSSEUS/Access+framework.

[Odyc] *Odysseus Dokumentation - Database Feature*. Zugriff: 22.03.2016. 2016. URL: http://odysseus.informatik.uni-oldenburg.de:8090/display/ODYSSEUS/Database+Feature.

[Odyd] *Wiki von Odysseus*. Zugriff: 29.04.2015. 2015. URL: http://odysseus.informatik.uni-oldenburg.de:8090/display/ODYSSEUS/Odysseus+Home.

[Pi4] *The Pi4J Project*. Zugriff: 19.03.2016. 2016. URL: http://pi4j.com/.

[Pls] *PL/SQL User's Guide and Reference*. Zugriff: 30.03.2016. 2016. URL: https://docs.oracle.com/cd/A97630_01/appdev.920/a96624/toc.htm.

[Rac15] Rachspace. *BlueFlood DB*. Webseite. letzter Zugriff: 11.08.2015. Juli 2015. URL: http://blueflood.io.

[Rdr] *Druid connector for R*. Zugriff: 22.03.2016. 2016. URL: https://github.com/druid-io/RDruid.

[Rei14] R. Reichert. *Big Data: Analysen zum digitalen wandel von Wissen, Macht und Ökonomie*. transcript Verlag, 2014.

[RR15] J. Rose und J. Reimann. *Eclipse SCADA: The definite guide*. Zugriff: 29.04.2015. 2015. URL: http://books.google.de/books?id=GDsWAwAAQBAJ.

[Sca] *ScaleDB Webseite*. Zugriff: 1.12.2015. 2015. URL: http://www.scaledb.com/.

[Scr] *Scrum Einführung in Scrum*. letzter Zugriff: 10.08.2015. 2015. URL: http://www.scrum-kompakt.de/einfuehrung-in-scrum/.

[Ses] *Smart Energy Simulation and Automation Lab*. Zugriff: 30.10.2015. 2015. URL: http://www.offis.de/f_e_bereiche/energie/gruppen/simulation_und_automatisierung_komplexer_energiesysteme/sesa_lab.html.

[Sin] *OODesign Singleton*. Zugriff: 30.03.2016. 2016. URL: http://www.oodesign.com/singleton-pattern.html.

[Sma] *Einsatz von Sensoren für Energieflussanalysen für Energienetze, Smart Grids, Smart Metering*. Zugriff: 29.03.2016. 2016. URL: http://www.fvee.de/fileadmin/publikationen/Workshopbaende/ws2013/ws2013_05_01.pdf.

[Son] *SonarQube Website*. letzter Zugriff: 24.10.2015. URL: http://www.sonarqube.org/.

[Sou] *Bild Sound_Blaster Sounderweiterung*. Zugriff: 30.10.2015. 2015. URL: http://www.amazon.de/gp/product/B0028RZ23I?*Version*=1&*entries*=0.

[Sou15] Prometheus SoundCloudLimited. *Prometheus - Dokumentation*. Webseite. letzter Zugriff: 11.08.2015. Juli 2015. URL: http://prometheus.io/docs/introduction/overview/.

[Sqm] *SonarQube Metriken*. letzter Zugriff: 19.03.2016. 2016. URL: http://docs.sonarqube.org/display/SONAR/Metric+Definitions#MetricDefinitionsDocumentation.

[Sqq] *SonarQube Qualitätskritierien*. letzter Zugriff: 24.10.2015. 2015. URL: http://docs.sonarqube.org/display/HOME/Developers%27+Seven+Deadly+Sins.

[Stu] *Status Quo Agile (2014)*. letzter Zugriff: 16.06.2015. 2014. URL: http://www.gpm-ipma.de/fileadmin/user_upload/Know-How/studien/PM_2_14_S40.pdf.

[Tie10] Ernst Tiemeyer. *Handbuch IT-Projektmanagement: Vorgehensmodelle, Managementinstrumente, Good Practices*. Hanser Verlag, 2010.

[Vol15] Inc. VoltDB. *Using VoltDB*. Documentation. letzter Zugriff: 11.08.2015. Juli 2015. URL: http://downloads.voltdb.com/documentation/UsingVoltDB.pdf.

[Wir11a] Ralf Wirdemann. *Scrum- Mit User Stories*. Hanser Verlag, 2011.

[Wir11b] Ralf Wirdemann. *Scrum- Mit User Stories*. Hanser Verlag, 2011.

[Wüt+08] Gerd Wütherich u. a. *Die OSGi-Service-Platform : eine Einführung mit Eclipse Equinox*. 1. Aufl. Heidelberg: Heidelberg : dpunkt.Verl., 2008. ISBN: 389864457X.

[Zoo] *Apache Zookeeper*. Zugriff: 29.03.2016. 2016. URL: https://zookeeper.apache.org.

[Ody15] Odysseus Team. *Build your own generator*. letzter Zugriff: 05.08.2015. März 2015. URL: http://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/Build+your+own+generator.

Anlagen

A. Lastenheft

Lastenheft

PG SESAdata

31. Oktober 2015

Inhaltsverzeichnis

1 Einleitung

Konzepte sind Entscheidungsgrundlagen und Handlungsvorgaben, welche bei der Umsetzung komplexer Software-Projekte unverzichtbar sind. Gerade bei Projekten, bei denen zu Beginn oft nicht mehr als eine vage Vorstellung bekannt ist, ist ein Grobkonzept oder auch Lastenheft genannt unabdingbar.
Bevor mit der Umsetzung des Projektes begonnen werden kann, muss ein gemeinsames Verständnis von Anforderungen, Prozessablauf und Umsetzungsalternativen hergestellt werden. Daher wird in der ersten Phase des Projektes, nach dem offiziellen Start durch das Projektkickoff, mit der Erarbeitung des Grobkonzeptes/Lastenheftes (Requirements Engineering) begonnen. In den ersten Gesprächen werden dazu, zusammen mit den Auftraggebern, die Anforderungen an das System ermittelt und anschließend für den Kunden verständlich und widerspruchsfrei formuliert.
Dieses Lastenheft gibt die Richtung für das Softwareprojekt der Projektgruppe SESAdata vor und steckt fachlich sowie technisch den Rahmen des Projektes ab.
Der Name des Projektes ist **L**IVE **A**NALYSIS AND LONG-TIME **S**TORAGE **E**NVIRONMENT (LAOTSE) und wird im folgenden auf das Projekt verweisen.
Zunächst wird die Aufgabenstellung beschrieben, um einen Überblick über das im Rahmen der Projektgruppe umzusetzende Hard- und Softwaresystem zu bekommen. Darauf bauen die zu erreichenden Ziele auf, welche die Aufgabenstellung verdeutlichen und den zu erstellenden Umfang grob umschreiben. Um die Aufgabenstellung und die Projektbestandteile weiter zu konkretisieren, wird im nächsten Schritt ein erster globaler Überblick über die Funktionalitäten des Gesamtsystems und deren Akteure gegeben. Neben der Festlegung von Rahmenbedingungen erfolgt die Abgrenzung der Systembestandteile gegen die Systemumwelt. Auf der Grundlage der Informationen aus den vorherigen Kapiteln werden die konkreten Anforderungen an das System erstellt, welche in funktionale und nicht-funktionale Anforderungen untergliedert sind. Die funktionalen Anforderungen beziehen sich auf die konkreten Funktionalitäten, die das System anbieten soll. Während nicht-funktionale Anforderungen sich hingegen auf die Eigenschaften des Systems und der Projektdurchführung beziehen. Nach dem die Aufgabenstellung und die Anforderungen an das zu entwickelnde System erarbeitet sind, wird in der Technologieauswahl auf die erforderlichen Komponenten eingegangen. Im Vorletzten Kapitel ist ein erster Entwurf der erforderlichen Datenstruktur dargestellt. Weitere identifizierte wichtige Rahmenbedingungen werden im Abschluss des Lastenheftes zusammengefasst.

2 Aufgabenstellung

Dieses Kapitel beschreibt zunächst kurz die Hauptziele die im Rahmen dieses Projektes erreicht werden sollen. Dann wird der Systemeinsatz durch die Beschreibung der Anwendungsbereiche und Zielgruppen konkretisiert.

2.1 Zielbestimmung

Im Rahmen des Projektes soll das SESA-Lab des OFFIS erweitert werden. Um das Projektumfeld besser einordnen zu können, erfolgt zunächst eine kurze Beschreibung des SESA-Labs.
Das SESA-Lab (Smart Energy Simulation And Automation Laboratory) ist eine Echtzeit Co-Simulationsplattform unter Einbindung von Mosaik für intelligente Stromnetze (sogenannten Smart Grids), welches vom OFFIS in Zusammenarbeit mit der Universität Oldenburg aufgebaut wurde. Das Testlabor ermöglicht es die Komponenten komplexer Energieversorgungssysteme unter realitätsnahen Bedingungen zu testen.

Ziel des Projektes ist die Schaffung einer Datenmanagementumgebung für das SESA-Lab, bei der die zum Teil sehr hochfrequenten und von unzähligen Sensoren stammenden Daten, mit genauen Zeitstempeln, in Echtzeit verarbeitet und persistent gespeichert werden können. Darüber hinaus sollen Analysen und Visualisierungen sowohl auf die Echtzeitdaten als auch auf die gespeicherten Langzeitdaten ermöglicht werden. Dabei liegt die größte technische Herausforderung in der hochfrequenten Einspeisung der Daten in eine persistente Datenbank.

2.2 Systemeinsatz

Die Zielgruppe des Systems sind fachkundige wissenschaftliche Mitarbeiter des OFFIS aus dem Bereich Energie. Zunächst sollen für Forschungszwecke Analysen und Visualisierungen sowohl auf Echtzeitdaten, als auch auf gespeicherten Langzeitdaten ermöglicht werden. Die zukünftige wirtschaftliche Nutzung sollte aber bedacht werden. Die Datenquellen sind bisher Simulationsdaten des SESA-Labs, welche über Mosaik bereitgestellt werden, und Messwerte von Raspberry PI' s. Darüber hinaus sollen über Fernwirkprotokolle angebundene externe Systeme, wie der BHKW-Container in der Lesumstraße in Oldenburg, angebunden werden. Neben der Einbindung dieser drei vorhandenen Datenquellen, soll das System zukünftig anhand gängiger Standards erweiterbar sein.
Dabei sollen neben der Anzeige von Metadaten der Quellen, die Rohdaten wie Einzelmesswerte aber auch aggregierte Daten über beispielsweise Zeitreihen ermöglicht

werden. Es sind sinnvolle und hilfreiche Analysen (Durchschnitt, Verbrauchsspitzen, ...) und Visualisierungen zu entwickeln, welche aber individuell ausgewählt und ggf. angepasst werden können. Da die Umgebung zukünftig erweiterbar sein soll, wäre es hilfreich auch neue Analysen und Visualisierungen bereitzustellen und ggf. vorhandene zu modifizieren. Die Erstellung von Warnmeldungen und Reports bei bestimmten Ereignissen, wie beispielsweise der Ausfall eines Sensors, wäre eine sehr hilfreiche Funktion.
Als Datenstrommanagementsystem ist Odysseus zu verwenden. Da dieses System eine Eigenentwicklung der Uni Oldenburg ist, können erforderliche Erweiterungen und Änderungen im Fachbereich vorgenommen werden. Bei der Verarbeitung hochfrequenter Daten, im kHz Bereich, sind die Grenzen von Odysseus nicht genau bekannt. Diese gilt es herauszufinden und zu validieren in wieweit die Datenmanagementumgebung bei der Vielzahl der Datenquellen zuverlässig und stabil eingesetzt werden kann.
Die In-Memory-Datenbank und die Langzeitarchivierung sind entsprechend der Anforderungen frei wähl- und gestaltbar. Dabei ist ein Mittelweg zwischen der Schreib- und Lesegeschwindigkeit zu wählen.
Die folgende Grafik aus dem Projektkickoff, zu sehen in Abbildung 2.1, liefert den Einstieg in das Projekt. Dargestellt ist der theoretische Datenfluss und die Aufgaben der Projektgruppe.

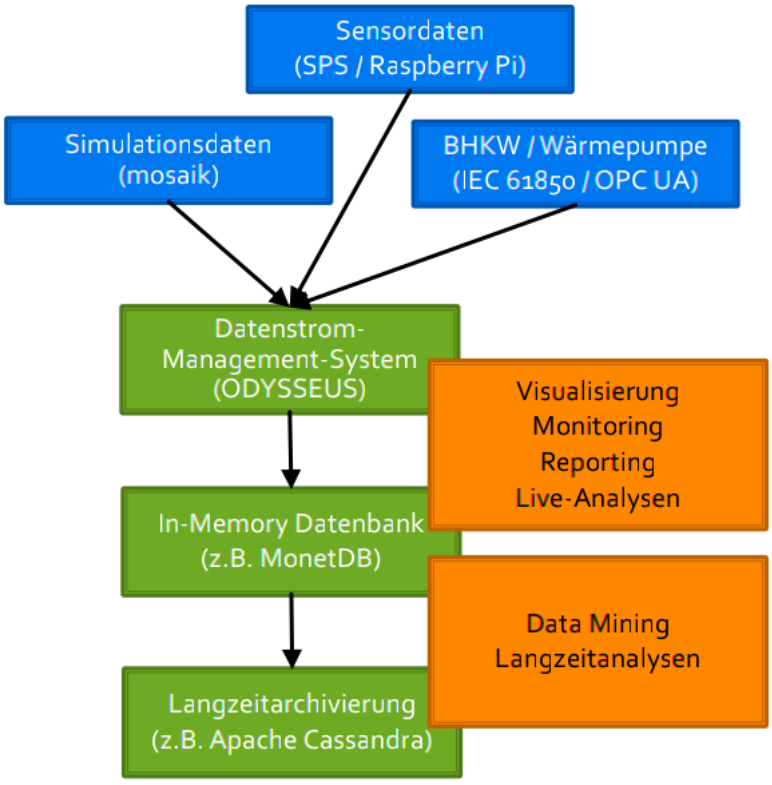


Abbildung 2.1: SESAdata grobe Datenflussbeschreibung

3 Hauptfunktionen

In den ersten Meetings sind zusammen mit den Auftraggebern, die Anforderungen erhoben worden. Diese wurden mit der Kreativitätstechnik Brainstorming gemeinsam erarbeitet und priorisiert.
Folgend die Teilziele mit den dazugehörigen Ergebnissen:

Prio:	**Teilziele:**	**Ergebnisse:**
hoch	Eingangsdaten verarbeiten (kHz, 1000 Sensoren)	Metainformationen, Messwerte
hoch	Odysseus einbinden	Datenstrom der Quellen verarbeiten
hoch	Zeitstempel mitführen	Synchronisierte Zeitreihen
hoch	Anbindung Mosaik	Simulationsdaten aus Mosaik
hoch	Daten persistent speichern	Rohdaten, aggregierte Daten
hoch	Erweiterbarkeit um Standardprotokolle	weitere Datenquellen können hinzugefügt werden.
mittel	Modularität/Erweiterbarkeit	Komponenten können ausgetauscht werden
mittel	Übersicht vorhandener Sensoren	Liveanzeige + Langzeitdatenbank
mittel	Anzeige/Analyse einzelner Sensoren	Auswahlmöglichkeit bei der Darstellung
niedrig	Anbindung eines BHKW	Verarbeitung Sensordaten BHKW-Dummys
niedrig	Reports über Ausfälle/Störungen	Meldungen über Sensorausfall oder ganzer Datenbank
niedrig	Analyse in Echtzeit und Langzeitdaten	Reports aus Odysseus und Langzeitdatenbank
niedrig	Kompression von Daten	geringere Speicherausnutzung

Die Ziele lässt sich in sechs voneinander abhängige Hauptaufgaben überführen, die folgend kurz dargestellt werden:

Datenquellen einbinden: Diese Aufgabe gehört zu den Kernfunktionen und besteht aus der Einbindung der drei vorgegebenen Datenquellen (Mosaik, Raspberry PIs

und über Fernwirkprotokolle angebundene externe Komponenten) in die Datenstrommanagementumgebung.

Datenströme verarbeiten: Mit dieser Kernfunktion sollen die Datenströme der Datenquellen verarbeitet werden. Primär geht es um die möglichst schnelle Zusammenführung der Daten (Sensordaten und Metadaten) als synchronisierte Zeitreihen für die Langzeitdatenbank. Darüber hinaus können eingehende Daten für Analysen und Visualisierungen genutzt werden.

Daten persistent speichern: Diese Kernfunktion umfasst die Überführung der eingehenden Sensordaten in eine geeignete Datenstruktur, sowie deren dauerhafte Speicherung. Dabei ist ein Mittelweg zwischen der Schreib- und Lesegeschwindigkeit zu wählen.

Analysen: Diese optionale Funktion beinhaltet die Bereitstellung von einigen nützlichen Analysen, die sowohl die Langzeitdaten als auch eingehende Datenströme berücksichtigen müssen.

Visualisierungen: Mit dieser optionalen Funktion soll eine ansprechende, übersichtliche und geeignete Darstellung der Langzeitdaten und der eingehenden Datenströme ermöglicht werden. Dabei sind nützliche Daten und ggf. Datenreihen zu bestimmen sowie nützliche Darstellungsformen zu wählen.

Erweiterbarkeit: Das System soll zukünftig erweiterbar sein, womit das Hinzufügen weiterer Quellen und deren Datenverarbeitung ermöglicht werden muss. Bei steigendem Leistungsbedarf, muss das System um weitere Hardware- und Softwarekomponenten wie Server und Datenbanken erweiterbar sein. Das Hinzufügen von Analysen und Visualisierungen ist nicht vorrangig und daher als Nebenfunktion anzusehen. Alle Erweiterungen des Systems, sollen zur Laufzeit ohne Unterbrechung möglich sein.

4 Systemumgebung und Akteure

In diesem Kapitel erfolgt die Systemübersicht, in der die Umgebung und die Anwendungsfälle veranschaulicht werden.

4.1 Systemakteure

Bei den Akteuren handelt es sich um alle Personen, Organisationen, Geräte, Systeme oder Dienste, welche mit dem System in Verbindung stehen und damit interagieren.

Akteur	**Ausprägung**	**Beschreibung**
Einfacher Anwender	Person	Bedient das zu entwickelnde System. Kann Analysen und Visualisierungen auswählen sowie ein- und ausblenden.
Fortgeschrittener Anwender	Person	Kann, über die Funktionalitäten des einfachen Anwenders hinaus, neue Datenquellen anlegen.
Administrator	Person	Administriert das zu entwickelnde Datenmanagementsystem. Dazu gehört neben der Aufrechterhaltung des Betriebes, auch die fortlaufende Systemerweiterung um einen stabilen Betrieb zu gewährleisten. Darüber hinaus legt er bei Bedarf neue Analysen und Visualisierungen an oder modifiziert vorhandene.
Entwickler	Person	Wird nicht betrachtet.
Externe Datenquellen (Raspberry PI, SESA-Lab Mosaik,BHKW-Container)	Datenquelle	Liefert Meta- sowie Sensordaten, welche vom Datenmanagementsystem verarbeitet werden. Pro Sensor können Einzelwerte übertagen werden. Die Übertragungsgeschwindigkeit geht bis in den kHz Bereich.

Das Diagramm in Abbildung 4.1 zeigt eine graphische Darstellung der Akteure in Beziehung zum System.

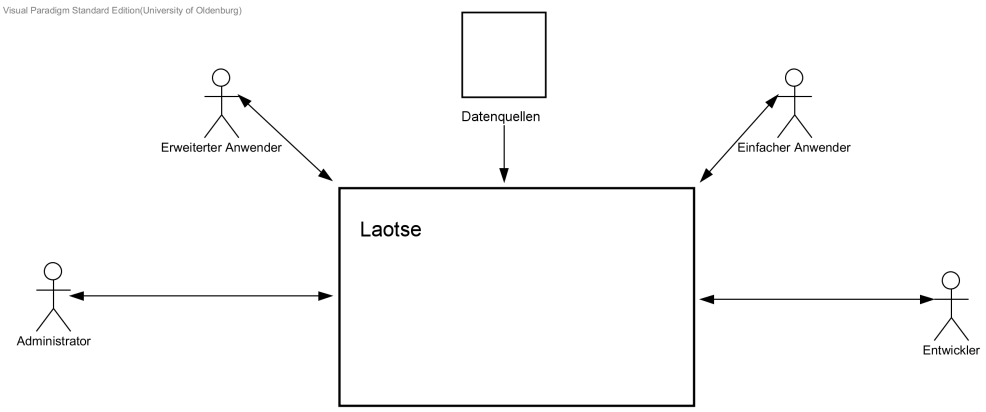


Abbildung 4.1: LAOTSE Umweltdiagramm

4.2 Use-Cases

In dem folgenden Use- Case- Diagramm werden die wesentlichen Komponenten des zu entwickelnden Systems und die Interaktion der in Kapitel 2 beschriebenen Akteure dargestellt.

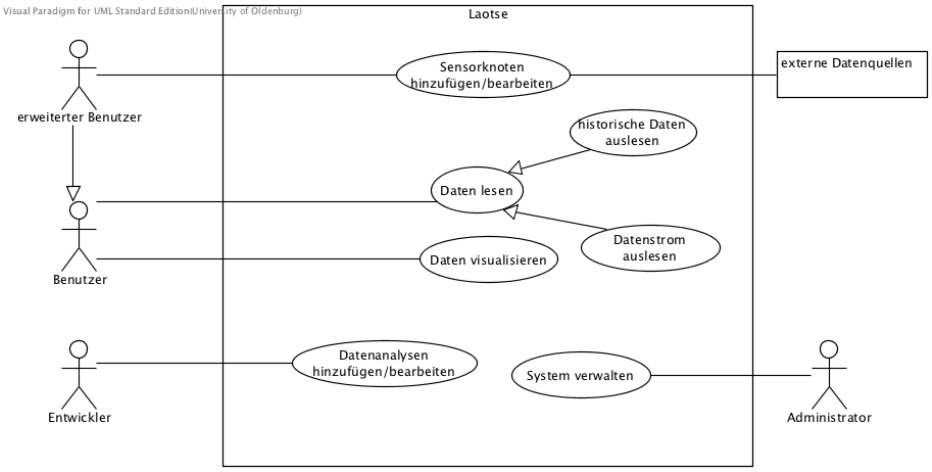


Abbildung 4.2: Use-Case Diagramm zu LAOTSE

Titel	**Sensorknoten hinzufügen/bearbeiten**/entfernen
Kurzbeschreibung	Der erweiterte Benutzer fügt einen Sensorknoten hinzu oder bearbeitet (entfernt) einen bereits vorhandenen Sensorknoten.
Akteure	erweiterter Benutzer
Auslöser	Ein Sensor soll hinzugefügt oder bearbeitet (entfernt) werden.
Vorbedingung	Der erweiterte Benutzer ist im System angemeldet und das System ist in einem konsistenten Zustand.
Nachbedingung	Das System befindet sich in einem konsistenten Zustand.
Ergebnis	Ein Sensor wurde hinzugefügt oder bearbeitet (entfernt).
genereller Ablauf	1. Der erweiterte Benutzer benutzt die Odysseus-GUI um einen Sensor hinzuzufügen/bearbeiten/entfernen. 2. Der erweiterte Benutzer gibt Metadaten bzgl. des Sensors ein. 3. Der erweiterte Benutzer wählt Analysen auf den Messdaten des Sensors aus. 4. Das erfolgreiche Hinzufügen/Bearbeiten/Entfernen wird dem Benutzer angezeigt.

Titel	**historische Daten auslesen**
Kurzbeschreibung	Der Benutzer liest (aggregierte) Messwerte aus dem System aus.
Akteure	Benutzer
Auslöser	Der Benutzer hat einen Informationsbedarf.
Vorbedingung	Der Benutzer ist im System angemeldet und das System ist in einem konsistenten Zustand.
Nachbedingung	Das System befindet sich in einem konsistenten Zustand.
Ergebnis	Daten wurden ausgegeben.
genereller Ablauf	1. Der Benutzer benutzt die LAOTSE-GUI um eine Anfrage auszuwählen. 2. Der Benutzer gibt die Parameter der Anfrage ein. 3. Das System gibt das Ergebnis der Anfrage aus.

Titel	**Datenstrom auslesen**
Kurzbeschreibung	Der Benutzer liest (aggregierte) Messwerte aus dem System aus.
Akteure	Benutzer

Auslöser	Der Benutzer hat einen Informationsbedarf.
Vorbedingung	Der Benutzer ist im System angemeldet und das System ist in einem konsistenten Zustand.
Nachbedingung	Das System befindet sich in einem konsistenten Zustand.
Ergebnis	Daten wurden ausgegeben.
genereller Ablauf	1. Der Benutzer benutzt die Odysseus-GUI um eine Anfrage auszuwählen. 2. Der Benutzer gibt die Parameter der Anfrage ein. 3. Das System gibt das Ergebnis der Anfrage aus.

Titel	**Daten visualisieren**
Kurzbeschreibung	Der Benutzer wählt einen Betrachtungsgegenstand und ggf. einen zugehörigen Zeitraum aus.
Akteure	Benutzer
Auslöser	Der Benutzer möchte Daten in visueller Form betrachten.
Vorbedingung	Es sind visualisierbare Daten vorhanden.
Nachbedingung	Das System bleibt unverändert.
Ergebnis	Daten wurden in visueller Form angezeigt.
genereller Ablauf	1. Der Benutzer benutzt die LAOTSE-GUI um einen Betrachtungsgegenstand auszuwählen. 2. Das System gibt dem Benutzer ggf. die Möglichkeit eine Visualisierungsart auszuwählen. 3. Dem Benutzer werden die Daten visualisiert angezeigt.

Titel	**Datenanalyse hinzufügen/bearbeiten.**
Kurzbeschreibung	Der Benutzer (Forscher) fügt eine Datenanalyse hinzu oder bearbeitet eine bereits vorhandene Datenanalyse.
Akteure	Benutzer (Forscher)
Auslöser	Eine Datenanalyse soll hinzugefügt oder bearbeitet werden.
Vorbedingung	Der Benutzer (Forscher) ist im System angemeldet und das System ist in einem konsistenten Zustand.
Nachbedingung	Das System befindet sich in einem konsistenten Zustand.
Ergebnis	Eine Datenanalyse wurde hinzugefügt oder bearbeitet.

genereller Ablauf	1. Eine Datenanalyse wird vom Entwickler im Java-Code hinzugefügt/bearbeitet. 2. Der Entwickler stellt dies dem Administrator zur Verfügung.

Titel	**System verwalten**
Kurzbeschreibung	Der Administrator verwaltet das System durch Rechte-/Infrastruktur-Anpassung.
Akteure	Administrator
Auslöser	Das System erfordert Veränderungen.
Vorbedingung	Das System befindet sich in einem konsistenten Zustand.
Nachbedingung	Das System befindet sich in einem konsistenten Zustand.
Ergebnis	Das System wurde verwaltet.
genereller Ablauf	1. Der Administrator nutzt seine GUI, um entsprechende Systemänderungen durchzuführen. 2. Die Änderungen treten in Kraft.

4.3 Komponentendiagramm

In diesem Abschnitt wird das in Abbildung 4.3 zu sehende Komponentendiagramm von LAOTSE vorgestellt.

4.3.1 Komponenten

Das LAOTSE System wird ein sehr komplexes System werden, bei dem viele verschiedene Komponenten miteinander agieren müssen. Um alle Komponenten und ihre Zusammenhänge darstellen zu können, werden UML Komponentendiagramme eingesetzt.
Ein solches Diagramm ist in Abbildung 4.3 dargestellt. Es enthält alle geplanten Komponenten des LAOTSE Systems und welche Beziehungen die Komponenten untereinander haben.
Die nachfolgenden Ausführungen sollen dabei noch einmal genauer die einzelnen Komponenten und ihre Funktion innerhalb des Systems darstellen und erläutern.
Die Komponenten lassen sich in drei Kernbereiche unterteilen. Zum einen gibt es Komponenten, die sich mit der Datenhaltung beschäftigen. Auf diese Komponenten greifen dann die Komponenten aus dem Bereich der Logik zu, die für die Datenverarbeitung

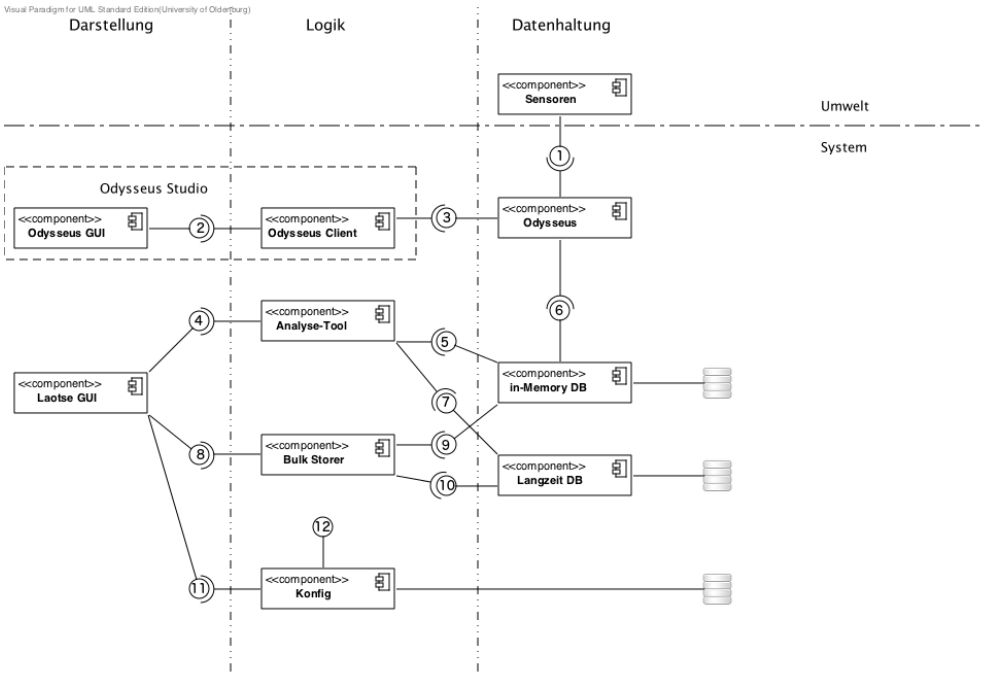


Abbildung 4.3: Komponentendiagramm des Systems mit Darstellung der geplanten Schnittstellen

und Datenanalyse zuständig sind. Im letzten Schritt sollen diese Daten für den Benutzer visualisiert werden. Dafür sind die Komponenten aus dem Bereich Darstellung zuständig, welche die Benutzeroberfläche für das System darstellen.

Sensoren Nicht zum LAOTSE System gehören die Sensoren, diese liegen außerhalb des Systems, sind in dem Komponentendiagramm allerdings dennoch aufgeführt, da diese die Daten für das System liefern, ohne die das System nicht arbeiten könnte und hinfällig wäre.

Odysseus Server Die erste Komponente, die dem Bereich der Datenhaltung zuzuordnen ist, ist der Odysseus Server, der die Daten, welche die Sensoren liefern, in Echtzeit verarbeiten kann. Odysseus ist also für den Datenstrom zuständig und ist somit die Grundlage für die Live-Datenanalyse.

In-Memory Datenbank Odysseus speichert die gegebenen Daten des Datenstroms in eine In-Memory Datenbank, auf die nahezu in Echtzeit zugegriffen werden kann. Diese In-Memory Datenbank hält weitere Daten für Analysen bereit, die auf den Daten ausgeführt werden sollen. Die In-Memory DB hält die Daten allerdings nur so lange, wie Analysen auf den Daten durchgeführt werden oder bis die Daten aus der In-Memory Datenbank in die Langzeit Datenbank geschrieben wurden.

Langzeit Datenbank Die Langzeit Datenbank ist zuständig für die Langzeitarchivierung der Daten und soll es ermöglichen, neue Daten mit historischen Daten vergleichen zu können. Sie ist somit dafür zuständig Daten über einen langen Zeitraum zu speichern und sie für einen späteren Zeitpunkt zur Verfügung zu stellen, damit sie dann gegebenenfalls noch analysiert werden können.

Zu beachten ist bei den beiden Komponenten, der Langezeit Datenbank und der In-Memory Datenbank, dass die in Abbildung 4.3 dargestellten Komponenten nicht die Datenbank an sich darstellen, sondern die Datenbank Handler, welche die Zugriffe auf der jeweiligen Datenbank ausführen. Die Datenbanken selbst werden durch die Symbole rechts davon dargestellt.
Die 3 Komponenten Odysseus Server, In-Memory Datenbank und Langzeit Datenbank organisieren also die Datenhaltung innerhalb des LAOTSE Systems. Damit die gegebenen Daten jedoch auch organisiert und verarbeitet und nicht nur gespeichert werden können, müssen auch Logikkomponenten existieren, die dies umsetzen.

Konfiguration Als erste muss dabei die Konfigurationskomponente erwähnt werden, die Konfigurationsdaten für das System in einer Datenbank speichert und für die anderen Logikkomponenten zur Verfügung stellt. So enthält diese Komponente zum Beispiel die Metadaten der einzelnen angeschlossenenen Sensoren. Somit kann aus diesen Daten ermittelt werden, welche Sensoren aktuell an das System angeschlossen sind und es können über diese Komponenten ebenfalls Sensoren hinzugefügt werden. Aus diesem Grund sind alle anderen Logikkomponenten des Systems Observer dieser Konfigurationskomponente, um auf Änderungen und neue Sensoren reagieren zu können und so zu gewährleisten, dass eine korrekte Analyse durchgeführt wird.

Odysseus Client Für die Odysseus Komponenten existiert dementsprechend auch eine eigene Logikkomponente. Diese Komponente ist der Odysseus Client, der auf dem Datenstrom des Odysseus Servers anfragen laufen lassen kann und auf diese Art und Weise die Daten analysieren oder aggregieren kann. Gemeinsam mit der dazugehörigen Odysseus GUI ist diese Komponenten im Odysseus Studio vereint, dass eben Anfragen auf den Datenstrom aus Odysseus machen kann und diese analysierten Daten im nächsten Schritt visualisieren kann. Da der Odysseus Client ebenfalls ein Observer der Konfigurationskomponente ist, kann das Odysseus Studio vor allem dazu genutzt werden um zu visualisieren welche Sensoren aktuell Daten liefern und gegebenenfalls, ob Warnzustände vorliegen. Bei der Umsetzung ist aus Performance- und Austauschbarkeitsgründen nach Möglichkeit die Trennung der Clients nach Funktionalitäten vorzunehmen. Eine denkbare Unterteilung wäre:

1. **Sensor-Client (write)**
2. **Analyse-Client (read)**
3. **Visualisierungs-Client (read)**

Odysseus GUI Die Odysseus GUI visualisiert dabei die im Odysseus Client ermittelten Daten und stellt somit die Komponente für die Echtzeitüberwachung dar.

Analyse-Tool Für die eigentliche Analyse der eingehenden Daten ist das Analyse-Tool zuständig. Dieses hat neben den Konfigurationsdaten sowohl Zugriff auf die In-Memory Datenbank und somit echtzeitnahe Daten, als auch auf die Langzeit Datenbank. In dieser Komponenten können also alle früheren und aktuellen Daten vereint betrachtet werden und somit komplexe Analysen gefahren werden um die Daten zu analysieren.

Bulk Storer Der Bulk Storer als die Letzte Logikkomponente ist dazu da, von Zeit zu Zeit die Daten aus der In-Memory Datenbank auszulesen und in die Langzeitdatenbank hineinzuschreiben. Damit kann die In-Memory Datenbank entlastet werden und neue Daten aufnehmen und die Langzeit Datenbank hält diese Daten für spätere Analysen weiter. Als zusätzliche Aufgabe kann der Bulk Storer der wichtigsten Darstellungskomponente der LAOTSE GUI Statusmeldungen senden, auf welche die LAOTSE GUI dann reagieren kann.

LAOTSE GUI Die angesprochene LAOTSE GUI ist die Haupt-Benutzerschnittstelle des Systems. Sie ermöglicht es dem Benutzer beispielsweise Analysen auf dem Analyse-Tool auszuführen oder neue Analysen hinzuzufügen, die dann in der Konfigurationsdatenbank gespeichert werden und zu einem späteren Zeitpunkt erneut ausgeführt werden können. Ebenso bietet die LAOTSE GUI dem Benutzer die Möglichkeit neue Sensoren im System zu registrieren oder bestehende Sensoren zu bearbeiten, damit diese für spätere Analysen genutzt werden können. Dabei gibt die LAOTSE GUI die neuen Informationen an die Konfigurationskomponente, die diese Daten dann speichert und die anderen Logikkomponenten bekommen Nachricht über die Änderung und können sich die neuen Daten holen und damit arbeiten. Ebenso ist die LAOTSE GUI aber auch dazu da, die Ergebnisse der durchgeführten Analysen darzustellen und sie dem Benutzer so auf eine geeignete Art und Weise anschaulich zu machen.

4.3.2 Schnittstellen

In Abbildung 4.3 ist das Komponentendiagramm des Systems mit den Schnittstellen dargestellt. Es werden verschiedene Schnittstellen eingesetzt. Dadurch können heterogene Komponenten, wie z.B. verschiedene Sensoren standardisiert eingebunden und genutzt werden. Auch wird dadurch Flexibilität sichergestellt und es können Komponenten ausgetauscht werden, ohne andere Komponenten zu bearbeiten. Die Schnittstellen sind fortlaufend nummeriert und werden in diesem Abschnitt beschrieben:

1. Die Sensoren implementieren eine Schnittstelle, die Metadaten und ggf. Datenformate der Messwerte beschreibt. Über diese Schnittstelle können Daten von den Sensoren an das System übermittelt werden und das System kann Metainformationen über die Sensoren abfragen.

2. Die grafische Benutzeroberfläche des Odysseus-Studio implementiert eine Schnittstelle, die grafische Funktionen bereitstellt. Es werden Benutzereingaben an Odysseus und Visualisierungen von Odysseus verarbeitet.

3. Der Odysseus-Server implementiert eine Schnittstelle, die vom Client des Odysseus-Studio genutzt wird. Über diese Schnittstelle lässt sich der Server steuern und z.B. Abfragen an die Datenströme erzeugen, starten und stoppen.

4. Analog zu Schnittstelle 2 implementiert die LAOTSE GUI drei Schnittstellen, die von der Logik zur Visualisierung von Daten genutzt werden können. Dabei stellt jede Schnittstelle nur die für die jeweilige Komponente notwendigen Funktionen bereit.

5. Die Komponente der in-Memory DB implementiert eine Schnittstelle, die vom Analyse-Tool genutzt werden kann, um auf die Daten der Datenbank zuzugreifen.

6. Analog zu Schnittstelle 5 können über diese Schnittstelle Daten in die in-Memory DB geschrieben werden.

7. Analog zu Schnittstelle 5 implementiert die Langzeit-DB eine Schnittstelle, um die Daten zur Analyse bereit zu stellen.

8. Siehe Schnittstelle 4

9. Siehe Schnittstelle 5

10. Analog zu Schnittstelle 6 können hier Daten in die Langzeit DB geschrieben werden.

11. Siehe Schnittstelle 4

12. Die Konfigurationskomponente stellt eine Schnittstelle bereit, die von beliebigen anderen Komponenten genutzt werden kann. Die anderen Komponenten werden dann über Änderungen in der Konfiguration benachrichtigt. Dies ist z.B. über das Observer-Pattern umsetzbar, indem die Konfiguration ein Observable darstellt und die anderen Komponenten sich als Beobachter registrieren.

5 Anforderungen

In diesem Kapitel werden die funktionalen und nicht-funktionalen Anforderungen an LAOTSE aufgelistet. Die Prioritäten geben Auskunft über die Wichtigkeit für das Projekt und deren Umsetzung. Anforderungen der Priorität A müssen im Rahmen des Projektes umgesetzt werden. Die mit der Priorität B sollen nach Möglichkeit umgesetzt werden. Die Berücksichtigung der Anforderungen der Wichtigkeit C wäre wünschenswert, wenn der zeitliche Rahmen dieses ermöglicht.

5.1 Funktionale Anforderungen

In der folgenden Tabelle sind die Funktionalen Anforderungen an LAOTSE aufgelistet.

ID	**Priorität**	**Anforderung**
L-FA1	A	Das System soll die eingehenden Sensordaten (Roh- und Metadaten) persistent speichern können.
L-FA2	A	Das System soll die eingehenden Daten verlustfrei verarbeiten können.
L-FA3	A	Das System soll den Messzeitpunkt (Timestamp) zu den gespeicherten Daten speichern können.
L-FA4	A	Das System soll Sensordaten von Raspberry PI's verarbeiten können.
L-FA5	A	Das System soll die Möglichkeit bieten weitere Quellen, über aktuell verbreitete Standards, hinzufügen zu können.
L-FA6	A	Das System muss Sensordaten aus dem SESA-Lab, die über Mosaik bereitgestellt werden, verarbeiten können.
L-FA7	A	Das System muss Datenströme mit Hilfe des vorhandenen DSMS Odysseus verarbeiten können.
L-FA8	B	Das System soll die Möglichkeit, bieten Systemerweiterungen (Hardware, Software) im laufenden Betrieb ermöglichen zu können.
L-FA9	B	Wenn Datenquellen ausfallen, muss das System fähig sein, unterbrechungsfrei weiterlaufen zu können.
L-FA10	B	Wenn ausgefallene Datenquellen wieder verfügbar sind, soll das System diese automatisch wieder einbinden können.

L-FA11	B	Das System soll Analysen auf den gespeicherten Langzeitdaten durchführen können.
L-FA12	B	Das System soll Liveanalysen auf den Datenströmen durchführen können.
L-FA13	B	Das System soll 5 vorimplementierte Standardanalysen, bereitstellen können.
L-FA14	B	Das System soll um weitere Analysen, durch den Benutzer (Forscher), ergänzt werden können.
L-FA15	B	Das System soll die Möglichkeit bieten, Datenströme visualisieren zu können. (Beispielsweise die aktuell erzeugte Leistung eines Windparks.)
L-FA16	B	Das System soll die Möglichkeit bieten, Langzeitdaten visualisieren zu können. (Beispielsweise die erzeuge Leistung des letzten Jahres.)
L-FA17	B	Das System soll die Möglichkeit bieten, Metadaten visualisieren zu können. (Beispielsweise welche Sensoren aktuell Daten an das System senden.)
L-FA18	B	Das System soll die Möglichkeit bieten, Analyseergebnisse aus den Langzeitdaten und Datenströmen, visualisieren zu können. (Z.B. Mittelwert der letzten 3 Wochen)
L-FA19	B	Das System soll die Möglichkeit bieten, Visualisierungen ein- und auszublenden zu können.
L-FA20	B	Das System soll um weitere Visualisierungen, durch den Benutzer (Forscher), ergänzt werden können.
L-FA21	C	Das System soll die Daten, des über Fernwirkprotokolle angeschlossenen BHKW-Containers in der Lesumstraße in Oldenburg, verarbeiten können. (Wenn dieser nicht rechtzeitig in Betrieb genommen wird, erfolgt die Einbindung eines bereit gestellten Dummy-BHKW's.)
L-FA22	C	Das System soll den Ausfall von Sensoren melden können. (Z.B. als visuelle Hervorhebung oder Nachricht)
L-FA23	C	Das System soll Meldungen unterschrittener Schwellwerte ausgeben können. (Z.B. als visuelle Hervorhebung oder Nachricht)
L-FA24	C	Das System soll um weitere Meldungen erweitert werden können.
L-FA25	C	Das System soll die zu speichernden Daten komprimieren können.

5.2 Nicht-funktionale Anforderungen

Dieses Kapitel beschreibt die Nicht-funktionalen Anforderungen an LAOTSE.

5.2.1 Projektanforderungen / Anforderungen an Durchführung und Entwicklung

Die folgenden Anforderungen beziehen sich auf die Projektdurchführung.

ID	Prio	Kategorie	Anforderung
NFA1	A	Budget	Wenn Investitionen zur Realisierung des Projektes erforderlich sind, muss das Projektteam Angebote zur Auswahl bereitstellen.
NFA2	A	Lieferumfang und Termine	Das Projektteam muss nach der Hälfte der Projektzeit einen Zwischenbericht, welcher neben dem Grobkonzept, das Feinkonzept enthält, vorlegen.
NFA3	A	Lieferumfang und Termine	Das Projektteam soll nach der Hälfte der Projektzeit einen Prototyp bereitstellen.
NFA4	A	Lieferumfang und Termine	Das Projektteam soll am 31.03.2016 den Abschlussbericht, welcher neben dem Zwischenbericht, das Feinkonzept, ein Installations- sowie Benutzerhandbuch enthält, bereitstellen.
NFA5	C	Lieferumfang und Termine	Bei allen Dokumenten soll nach Möglichkeit der Autor hervorgehen.
NFA6	A	Lieferumfang und Termine	Das Projektteam führt monatliche Reviews mit dem Auftraggeber durch.
NFA7	B	Testkonzept	Das Testkonzept soll von Beginn an erstellt und angemessen dokumentiert werden.
NFA8	B	Testkonzept	Das Testkonzept kann bei Bedarf und in Absprache mit den Betreuern erweitert werden.
NFA9	B	Evaluierung	Systemkomponenten sollen mit umfangreicher Alternativenbetrachtung, Begründung und Dokumentation ausgewählt werden.
NFA10	A	Dokumentation	Die Dokumentation muss fortlaufend und präzise während der gesamten Projektzeit erfolgen.
NFA11	A	Handbücher	Die Handbücher müssen für Fachleute und in der deutschen Sprache erstellt werden.

5.2.2 Rechtliche Anforderungen

Die folgende Tabelle zeigt die rechtlichen Anforderungen.

ID	Prio	Kategorie	Anforderung
NFA12	A	Datenschutz	Die bereitgestellten Daten und Systeme müssen vertraulich, ohne absichtliche Weitergabe an Außenstehende, behandelt werden.
NFA13	B	Rechtliches	Urheber- und Lizenzrechte müssen gewahrt werden.
NFA14	B	Rechtliches	Anforderungsänderungen können nur durch Zustimmung des Auftraggebers durchgeführt werden.
NFA15	B	Rechtliches	Weitere rechtliche Rahmenbedingungen müssen nicht beachtet werden.

5.2.3 Technische Anforderungen

Die folgende Tabelle zeigt die technischen Anforderungen.

ID	Prio	Kategorie	Anforderung
NFA16	A	Technologie	Die Technologie kann grundsätzlich frei gewählt werden, außer Vorgaben oder vorhandene Systeme widersprechen dem.
NFA17	B	Technologie	Die Technologieauswahl muss immer ordnungsgemäß evaluiert und dokumentiert werden.
NFA18	A	Technologie	Die Technologie soll unter Berücksichtigung der Ausbaufähigkeit und des langfristigen Einsatzes ausgewählt werden.
NFA19	B	Systemaufbau	Raspberry Pis sollen als Hardwareaufbau realisiert und in das System eingebunden werden.
NFA20	A	Systemaufbau	Die Systemkomponenten des DSMS sollen auf bereitgestellten VM-Instanzen realisiert werden.
NFA21	B	Schnittstellen	Wenn Standards für die Schnittstellen vorhanden sind, sollen diese nach Möglichkeit verwendet werden.
NFA22	A	Schnittstellen	Wenn Schnittstellen bereitgestellt werden, muss deren Spezifikation und Beschreibung erstellt werden.
NFA23	C	Programmiersprache	Die Programmiersprache kann frei gewählt werden.
NFA24	B	Programmiersprache	Codekonventionen sollen eingehalten werden.

NFA25	C	Speicherkapazität	Die Speicherkapazität für die Langzeitspeicherung steht aufgrund der immer weiter sinkenden Kosten ausreichend zur Verfügung.
NFA26	B	Speicherkapazität	Wenn eine Kompression möglich ist, soll diese aus Performance- und Speicherauslastungsgründen implementiert werden,
NFA27	C	Hardware- und Softwarekomponenten	Das Projektteam muss den Ausfall einzelner Systemkomponenten wie DB-Ausfälle oder sonstige Katastrophen nicht berücksichtigen.
NFA28	B	Hardware- und Softwarekomponenten	Das System soll nach Möglichkeit modular aufgebaut werden, um System- und Kapazitätserweiterungen zur Laufzeit zu ermöglichen.
NFA29	A	Hardware- und Softwarekomponenten	Das System soll die Möglichkeit bieten Daten, von bis zu 1000 Sensoren, zu verarbeiten.
NFA30	A	Hardware- und Softwarekomponenten	Das System soll fähig sein hochfrequente Daten, von bis zu 30 kHz, zu verarbeiten.

5.2.4 Qualitätsanforderungen

Die folgende Tabelle zeigt die Qualitätsanforderungen.

ID	Prio	Kategorie	Anforderung
NFA31	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb hinzugefügt werden können.
NFA32	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb modifiziert werden können.
NFA33	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb deaktiviert werden können.
NFA34	C	Verfügbarkeit	Analysen sollen im laufenden Betrieb möglich sein.
NFA35	C	Verfügbarkeit	Visualisierungen sollen im laufenden Betrieb möglich sein.
NFA36	C	Verfügbarkeit	Kapazität- und Systemerweiterungen sollen im laufenden Betrieb möglich sein.
NFA37	C	Bedienbarkeit	Das System soll von Fachleuten bedient werden können.
NFA38	C	Bedienbarkeit	Das System soll möglichst intuitiv bedienbar sein.
NFA39	B	Bedienbarkeit	Analysen sollen durch den Anwender auswählbar sein.
NFA40	B	Bedienbarkeit	Visualisierungen sollen durch den Anwender auswählbar sein.
NFA41	C	Bedienbarkeit	Die GUI kann frei gestaltet werden.
NFA42	A	Bedienbarkeit	Quellen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.
NFA43	C	Bedienbarkeit	Analysen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.
NFA44	C	Bedienbarkeit	Visualisierungen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.
NFA45	B	Performanz	Das System soll auf Anfragen, die einen Zeitraum von maximal 2 Jahren umfassen, innerhalb einer Stunde das Ergebnis liefern können.

6 Technologieauswahl

Dieser Abschnitt beschäftigt sich mit den ausgewählten Technologien die für das System und die Erstellung des Systems ausgewählt wurden.

6.1 Entwicklungsumgebung

Dieses Kapitel beschreibt alle Technologien, die für die Erstellung des Systems benötigt werden und begründet dabei, aus welchem Grund diese Tools gewählt wurden und keine Vergleichbaren.

6.1.1 SVN

Für die Versionsverwaltung wurde Subversion (SVN) gewählt. SVN zählt zusammen mit GIT zu den bekanntesten Versionsverwaltungstools. Die Entscheidung ist für dieses Projekt auf SVN gefallen, da die meisten der Projektteilnehmer sicherer im Umgang mit SVN als mit GIT sind. Da beide Tools nahezu den identischen Umfang bieten war dies der ausschlaggebende Punkt für Subversion.

6.1.2 JIRA

JIRA ist eine webbasierte Projektmanagement-Software, die sowohl für das Aufgabenmanagement, als auch die Zeiterfassung genutzt werden kann. Es können also sowohl Aufgabenpakete für die einzelnen Benutzer erstellt werden, als auch Zeitschätzungen zu diesen Aufgapenpaketen gemacht werden. Dadurch weiß der Entwickler, wie lange er ungefähr für die gegebenen Aufgaben investieren muss und kann so bereits im voraus seinen Tagesablauf planen. Ebenso kann der Entwickler nach Abschluss der Aufgabe erfassen, wie viel Zeit er wirklich benötigt hat und somit kann der Arbeitsaufwand des Entwicklers eingesehen werden. Gerade im Bereich an Projektmanagement-Software gibt es eine große Auswahl, die von ganz wenigen Funktionen bis nahezu allen Funktionen alles abdecken und dementsprechend auch kostenfrei oder kostenpflichtig sind. Die Wahl ist in diesem Fall auf JIRA, als ein Tool, dass eine große Bandbreite an Möglichkeiten zur Verfügung stellt, gefallen, da die Universität Oldenburg bereits eine Lizenz für JIRA besitzt und dadurch für diese Projektgruppe einfach ein neues Projekt angelegt werden konnte, ohne dass dies mit einem großen Mehraufwand für die Projektteilnehmer verbunden war.

6.1.3 Confluence

Confluence ist eine Wiki-Software, die für die zentrale Speicherung von Informationen, wie zum Beispiel Protokollen genutzt werden kann. Die Wahl des Wikis ist dabei, wie auch schon bei JIRA, auf Confluence gefallen, da die Universität Oldenburg bereits eine Lizenz besitzt und so eine einfache Erstellung des Wikis gewährleistet war. Ebenfalls hat Confluence aber den Vorteil gegenüber anderen Wikis, dass Confluence und JIRA integriert werden können und so eine Verzahnung der beiden Tools gewährleistet ist. In unserem Projekt wird Confluence für die Speicherung von Sitzungsprotokollen und die Erstellung von Stundenzetteln benötigt, da jeder Entwickler dokumentieren soll, wie viele Stunden er an dem Projekt gearbeitet hat.

6.1.4 Latex

Dokumente wie dieses müssen mit einem Texteditor erstellt werden. Da die zu erstellenden Dokumente jedoch häufig sehr umfangreich sind und mehrere Benutzer gleichzeitig Teile des Dokuments schreiben, sollte die Wahl in diesem Fall auf eine Textverarbeitungssoftware fallen, die diesen Punkt besonders berücksichtigt. Daher ist die Wahl für dieses Projekt auf Latex gefallen, da Latex die Möglichkeit bietet einzelne Kapitel des Dokuments in externe Dateien auszulagern. Dadurch kann jeder Benutzer, der ein Kapitel des Dokuments schreiben will an einer eigenen Datei arbeiten und es kommt nicht zu Konflikten innerhalb der Versionsverwaltung. Außerdem bietet Latex gute Möglichkeiten den Text auf eine ansprechende Art und Weise zu formatieren und mit Hilfe von verschiedenen Packages Bilder und mathematische Formeln einfach und sauber darzustellen.

6.1.5 eclipse

Bei der Wahl der Entwicklungsumgebung für Java fällt die Entscheidung meist auf eine der beiden bekanntesten Entwicklungsumgebungen eclipse und NetBeans. Da für die Umsetzung des Projektes Odysseus verwendet werden soll und dieses auf dem OSGI-Framework Equinox basiert, ist eclipse geeigneter. Eclipse basiert nämlich auch auf Equinox, wodurch eine bessere Nutzung und Entwicklung von Odysseus möglich ist. Darüber hinaus arbeiten nahezu alle Projektteilnehmer hauptsächlich mit eclipse, was zusätzlich für die Verwendung spricht. Insgesamt ist eclipse eine Entwicklungsumgebung, die nahezu alle Möglichkeiten mit Hilfe von Plug-Ins bietet. So kann in eclipse auch gleichzeitig die SVN Anbindung gewährleistet und dadurch ein besserer Arbeitsablauf geboten werden.

6.2 System

Die folgenden Ausführungen beschäftigen sich mit den Technologien, die für das System ausgewählt wurden.

6.2.1 Odysseus

Wie schon im Abschnitt zu eclipse erwähnt soll in diesem Projekt das Datenstrommanagementsystem Odysseus verwendet werden. Odysseus ist dabei dafür zuständig die Daten, die in Echtzeit von den Sensoren geliefert werden zu verarbeiten und in eine Datenbank zu schreiben. Ebenso soll Odysseus die Möglichkeit bieten bestimmte Dinge in Echtzeit anzuzeigen. Die Wahl des Datenstrommanagementsystems ist auf Odysseus gefallen, da Odysseus an der Universität in Oldenburg entwickelt wird und somit für das Projekt eine gute Unterstützung und schneller Support bei auftretenden Fehlern gewährleistet werden kann. Außerdem wurde Odysseus als einzusetzendes Tool von dem Auftraggeber vorgegeben.

6.2.2 Java

Als Programmiersprache für die Entwicklung des Systems ist die Entscheidung auf die Programmiersprache Java gefallen, da hiermit eine plattformübergreifende Entwicklung gewährleistet werden kann. Außerdem haben alle Projektteilnehmer die meisten Erfahrungen mit Java gemacht und das Datenstrommanagementsystem Odysseus wurde ebenfalls in Java entwickelt. Da Odysseus im Zuge dieses Projektes weiterentwickelt werden soll, muss zwangsläufig Java eingesetzt werden. Da die Entwicklung mit zwei verschiedenen Programmiersprachen nicht förderlich für das Projekt ist, wird deshalb für das gesamte System Java als Programmiersprache verwendet.

6.2.3 mosaik

mosaik ist ein Smart Grid Simulationsframework. Es dient also dazu Smart Grids zu simulieren und die Simulationsdaten für eine Analyse zur Verfügung zu stellen. Die Sensordaten des SESA-Labs werden häufig mit mosaik erweitert, sodass ein großes Netz simuliert wird. mosaik ist dabei als Simulationsframework von dem Auftraggeber vorgeschrieben worden. Da es aber auch eine Java Anbindung bietet, ist es für dieses Projekt sehr gut einsetzbar und da es bereits mit dem SESALab zusammenarbeitet, ist keine Integration notwendig und mosaik das richtige Tool für den Einsatz in dem zu entwickelnden System.

6.2.4 Raspberry Pis

Ebenso wurde die Nutzung von Raspberry Pis von dem Auftraggeber vorgeschrieben. Diese sollen als Knotenpunkte innerhalb des Netzes fungieren und somit einzelne Sensoren darstellen und simulieren. Zum Beispiel könnte ein Raspberry Pi dazu genutzt werden ein Blockheizkraftwerk (BHKW) zu simulieren, um die spätere Nutzung eines solchen zu gewährleisten. Die entwickelten Raspberry Pies werden dann im SESA-Lab integriert und dienen als Datenquelle für das zu entwickelnde System.

6.2.5 Virtuelle Maschinen

Das später fertige System muss auf einer oder mehreren virtuellen Maschine laufen. Dementsprechend werden virtuelle Maschine auf Linux Basis verwendet, auf denen das lauffähige System kompiliert werden kann. Ebenso wird eine virtuelle Maschine für die Entwicklung des Systems verwendet, auf der zum Beispiel das SVN läuft. Diese virtuellen Maschinen sind also unverzichtbar, sowohl für die Entwicklung des Systems, als auch für das später fertige System.

6.2.6 In-Memory Datenbank

Eine geeignete in-Memory Datenbank wird im Projektverlauf nach ausführlicher Aufstellung von Technologie-Alternativen ausgewählt.

6.2.7 Lösung für die persistente Datenhaltung

Eine geeignete Lösung für persistente Datenhaltung wird im Projektverlauf nach ausführlicher Aufstellung von Technologie-Alternativen ausgewählt.

7 Datenstruktur

Dieser Abschnitt gibt einen ersten groben Überblick über die Datenstruktur des Systems. Dazu wird zunächst ein Entwurf des Datenflusses dargestellt. Des weiteren wird kurz auf die internen Datenformate und der zur Speicherung eingegangen.

7.1 Datenflussbeschreibung

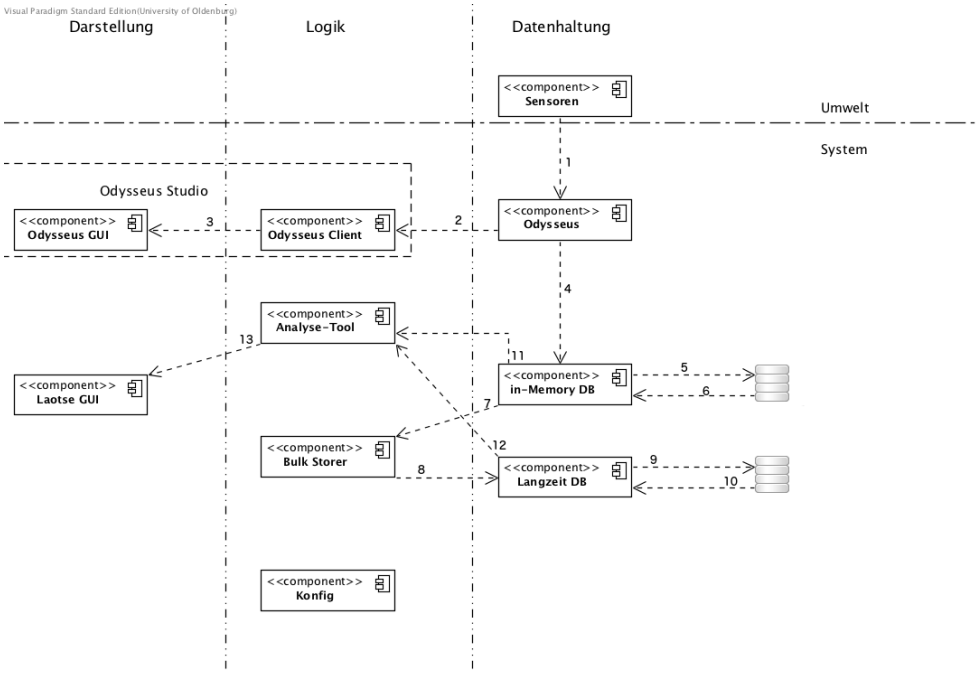


Abbildung 7.1: Komponentendiagramm des Systems mit Darstellung des geplanten Datenflusses der rohen und verarbeiteten Messwerte der Sensoren

In Abbildung 7.1 ist der Datenfluss der Messwerte dargestellt, die von Sensoren an LAOTSE übertragen werden. Im folgenden sind diese Datenflüsse beschrieben und dabei verweise auf die nummerischen Bezeichner angegeben.
Zuerst treffen sie als Datenstrom im DSMS Odysseus ein (1) und werden dort live verarbeitet. Wenn es gewünscht ist, können Ergebnisse einfacher live-Analysen oder

die Rohdaten an das Odysseus-Studio gesendet (2) und dort visualisiert (3) werden.
Anschließende werden alle Rohdaten in die in-Memory DB geschrieben (4). Dabei puffert Odysseus ggf. den Datenstrom entsprechend der Verarbeitungsgeschwindigkeit der Datenbank und überträgt die Daten Stückweise.
Die in-Memory DB Komponente handhabt, das reale schreiben (5) und lesen (6) aus der Datenbank. Die entsprechende Komponente der Langzeit DB arbeitet analog (9 und 10).
Die Messwerte werden aus der in-Memory DB über die Komponente in definierten Intervallen in den Bulk Storer geladen (7) und von diesem in die Langzeit DB geschrieben (8). Dadurch wird die Langzeit DB, die große Datenmengen verarbeitet, allerdings langsamer als die in-Memory DB arbeitet nicht überlastet und entsprechend mit den Messwerten versorgt.
Wenn Analysen auf den aufgezeichneten Daten durchgeführt werden sollen, werden die gewünschten Daten aus der in-Memory DB (11) und Langzeit DB (12) in das Analyse-Tool geladen. Die Analyseergebnisse können in der LAOTSE GUI visualisiert (13) werden.

7.1.1 Datenformate intern

Die intern genutzten Datenformate sollten nach Möglichkeit auf Java-Standarddatentypen aufsetzen, die auch von Odysseus unterstützt werden. Dies erleichtert zukünftige Erweiterungen oder den Austausch von Komponenten, die auf die betroffenen Schnittstellen zugreifen.

7.1.2 Datenformate zur Speicherung

Die Datenformate zur Speicherung, insbesondere im Zusammenhang mit der Langzeitdatenhaltung, sollten nach Möglichkeit 2 verschiedenen Anforderungen genügen:

1. Sie sollten möglichst schnell in die DB ein- bzw. aus ihr auszulesen sein.
2. Sie sollten aufgrund des hohen Datenvolumens möglichst komprimiert abgespeichert werden können.

Der zweite Punkt würde eine auf den eingehenden Daten anwendbare bijektive Komprimierungsfunktion voraussetzen, deren Durchführung wiederum Zeit kosten wird, was einen Widerspruch zur ersten Anforderung darstellt. Aus diesem Grund muss ein Kompromiss zwischen den beiden Anforderungen gefunden werden. Hierbei liegt die Priorität eindeutig auf der Schnelligkeit des schreibenden/lesenden Zugriffs auf die DB. Da Speicherplatz praktisch in unbegrenztem Maße zur Verfügung steht ist die zweite Anforderung weniger zu berücksichtigen.

8 Weitere Rahmenbedingungen

In diesem Abschnitt werden weitere wichtige, identifizierte Rahmenbedingungen zusammengefasst.

8.1 Skalierbarkeit

Das System muss große Datenmengen, die in einer hohen Frequenz eingespeist werden verarbeiten können. Dafür wurden entsprechende Anforderungen erhoben, die z.B. die Rechenleistung und die Speicherkapazität betreffen. Beides muss entsprechend hoch sein um die Aufgabe zu erfüllen. Es ist gewünscht, dass die Architektur erweiterbar ist.
Eine Möglichkeit diese Anforderungen zu erfüllen ist es, eine horizontale Skalierbarkeit im System umzusetzen. Das bedeutet, dass eine Lastverteilung auf verschiedene Ressourcen erfolgt. Wenn die Kapazität des Systems der anfallenden Last in Form von Rechenleistung für den Datenstrom oder Speicherkapazität für die Datenmengen nicht mehr gerecht werden kann, sollen weitere Knoten integriert werden können, die weitere anfallende Last aufnehmen können. Die Fähigkeit eines Systems zur Integration weiterer Knoten und Lastverteilung auf diese, charakterisiert die horizontale Skalierbarkeit. Der Vorteil gegenüber einer vertikalen Skalierbarkeit, bei der vorhandene Knoten erweitert werden würden, ist, dass der horizontalen Skalierbarkeit theoretisch keine Grenzen gesetzt sind. Voraussetzung dafür ist, dass die Implementierung dies Unterstützt.
Konkret muss zur Lastverteilung der Verarbeitung der Datenströme Odysseus horizontal skalierbar sein. Für die Lastverteilung der Datenmenge muss ein skalierbares DBMS für die Komponenten der Langzeit-Datenbank und der in-Memory-Datenbank gewählt werden.
Durch die Skalierung kann gewährleistet werden, dass das System sowohl bei geringem Datenaufkommen, als auch bei hohem Datenaufkommen zuverlässig in den geforderten Bearbeitungszeiten arbeitet.

8.2 Protokolle für Datenquellen

In der Energieinfrastruktur werden eine Vielzahl an Protokollen eingesetzt, um Daten zwischen einzelnen Geräten zu übertragen. Laotse soll die Daten von verteilten elektrischen Geräten sammeln und speichern können. Als Kommunikationsmedium steht Ethernet zur Verfügung. Es gibt zahlreiche Protokolle, die Daten über Ethernet zu übermitteln. Von der IEC wurden Standards zur Kommunikation innerhalb der Energieinfrastruktur veröffentlicht. Laotse soll über die gängigen Standards Daten von den

Sensoren abrufen können. Da Odysseus beliebige Eingabequellen einbinden kann, sollten prinzipiell auch beliebige Standards unterstützt werden können. Ziel ist, Laotse mit mindestens einem Standardprotokoll zu entwickeln. Die heute üblichen Standards werden in den folgenden Abschnitten beschrieben.

8.2.1 OPC UA

OPC UA wurde von der OPC Foundation entwickelt. Es setzt auf Ethernet und ermöglicht eine Client-Server-Kommunikation. Es beinhaltet Komponenten zu Auslesen von Daten (Data Access), Auslesen von historischen Daten (Historical Data Access), Bereitstellung von Meldungen (Alarms & Subscriptions). Dabei wird beschrieben, welche Services die Server zur Verfügung stellen. Clienten können beispielsweise bestimmte Daten abonnieren, damit neue Daten automatisch übertragen werden.

8.2.2 IEC 61850

IEC definiert eine Datenstruktur, in der die Daten von Sensoren eingeordnet werden. Die Geräte in einem physikalischen Energienetz werden nach IEC 61850 in logische Geräte eingeteilt. Diese logischen Geräte stellen zum einen definierte Dienste zur Verfügung, beispielsweise die Übermittlung von Messwerten, zum anderen ist aber auch definiert, welche Arten von Messwerten ein bestimmtes logisches Gerät bereithalten kann.
IEC 61850 kann auf unterschiedlichen Kommunikationsprotokollen realisiert werden. Im eigentlich Standard wird ein Mapping auf die Manufacturing Messaging Specification (MMS) vorgesehen. Es gibt auch die Möglichkeit, ein Mapping auf OPC UA einzusetzen.
Neben den Mappings auf Client/Server-Protokolle stehen weitere Übertragungsmedien zur Verfügung. GOOSE kann dazu genutzt werden, bestimmte Werte an mehrere Adressen zu senden.
Die Konfiguration wird durch SCL realisiert. Es wäre denkbar, dass die Daten aus SCL-Dateien dazu genutzt werden, Anfragen zu generieren, die die Datenquellen in Odysseus einbinden.

8.2.3 Einzusetzende Protokolle in Laotse

Odysseus kann bereits den OPC UA Stack der OPC Foundation einsetzen. Es ist daher sinnvoll, dieses Protokoll bei der Implementierung von Laotse zu verwenden. Wenn OPC UA eingesetzt wird, wird wahrscheinlich ein Konzept benötigt, den übermittelten Daten eine Bedeutung zu geben. IEC 61850 definiert klar, welche Daten von bestimmten Geräten übermittelt werden. Neben den Nutzdaten werden eine Reihe an Metadaten festgelegt. Es ist zu überlegen, ob die strikte Einteilung in den Einsatzbereich passt. Laotse soll in Forschungsprojekten eingesetzt werden können. Hier sind mehr Daten zu messen, als zur Steuerung und Überwachung des Energienetzes erforderlich sind. Eine reine Implementierung nach IEC 61850 ist nicht sinnvoll. Odysseus müsste so erweitert werden, dass Kommunikation nach IEC 61850 möglich ist.

B. Sprintplanung

Dieser Abschnitt beschreibt kurz die vorgesehene Ablaufplanung des Projektes welche zu Beginn erarbeitet wurde. Dabei sollen die wesentlichen vorgenommenen Ziele der einzelnen Sprints dargestellt werden, um später einen Soll- IST-Vergleich mit den tatsächlich erreichten Ergebnissen erstellen zu können.
Im ersten Sprint sollen die Anforderungen erhoben und die Projektumgebung eingerichtet werden. Da die Anforderungsanalyse sehr wichtig für den erfolgreichen Projektverlauf ist und in den agilen Vorgehensmodellen meistens vernachlässigt wird, muss diese zu Beginn des Projektes sehr umfangreich durchgeführt werden. Durch die ausgiebige Auseinandersetzung mit dem zu erstellenden Zielsystem, soll das Anwendungsgebiet und der Leistungsumfang des Zielsystems frühestmöglich präzisiert und Projekteinschränkungen erkannt werden. Damit die Bearbeitung des Projektes von Beginn an zielgerichtet erfolgen kann, sind das Vorgehensmodell der Projektgruppe und die Projektumgebung nach Möglichkeit im ersten Sprint einzurichten.
Um den Auftraggebern, nach der ausgiebigen gemeinsamen Analyse, einen ersten Überblick über die Vision des zu entwickelnde Zielsystem zu geben, soll im zweiten Sprint ein Grobkonzept erstellt werden. Dieses wird die Grundlage für die weitere Entwicklung im Projektverlauf bilden und ein gemeinsames Verständnis des Zielsystems zwischen Auftraggeber und -nehmer sicherstellen. Damit die Lieferobjekte und Leistungen des Auftraggebers zum Projektende überprüfbar und klar festgelegt sind, werden diese widerspruchsfrei, überprüfbar und verbindlich in einem Lastenheft festgehalten.
Nachdem die Vision des zu entwickelnden Zielsystems gefestigt ist und die Anforderungen vom Auftraggeber abgenommen sind, soll im dritten Sprint mit der Einarbeitung in die Grundkomponenten Odysseus, den Sensoren sowie den Datenbanken begonnen werden. Ziel ist es einen ersten Datenflussdurchstich zu erzielen, indem der Datenstrom eines ersten Sensors verarbeitet werden kann. Darüber hinaus soll die benötigte Hardware ausgewählt und bestellt werden, damit diese frühestmöglich zur Verfügung steht und keine Projektverzögerungen durch Wartezeiten entstehen können. Die Datenbankauswahl soll zunächst mit der Recherche der verschiedenen Datenbankarten anhand ihres Speicherprinzips beginnen und für die Eignung des Zielsystems untersuchen. Auf dieser Grundlage sind dann geeignete In-Memory und Langzeitdatenbanken auszuwählen, welche im nächsten Sprint zu vergleichen sind. Des Weiteren soll das Testkonzept für den direkten Vergleich der in Frage kommenden Datenbanklösungen erstellt werden. Da die Speicherung der vielen und zum Teil hochfrequenten Daten den wichtigsten Teil des Projektes ausmachen, wird für die Auswahl und Optimierung der In-Memory- und Langzeitdatenbank ausreichend Zeit kalkuliert. Denn nur mit einer ausreichend performanten Datenbanken ist das Zielsystem überhaupt realisierbar.
Der vierte Sprint dient primär zur Auswahl der Zieldatenbanken. Dazu sind die im dritten Sprint ausgewählten Alternativen zunächst zu installieren und dann gemäß des erstellten Testkonzeptes zu untersuchen. Zum Ende des Sprints soll die In-Memory- und Langzeitdatenbank ausgewählt sein, damit diese im folgenden Sprint umgesetzt und ggf. noch optimiert werden kann. Darüber hinaus wird mit der Einbindung der Sensoren und dem Ausbau der Programmierumgebung begonnen.

Sprint fünf dient im Wesentlichen der Einrichtung der Grundkomponenten, für die Erstellung eines ersten Prototypen, zur Hälfte der Projektzeit. Dazu sollen die Datenbanksysteme eingerichtet und ggf. optimiert, ein erster GUI-Prototyp erstellt, die Sensoren eingebunden und ein geeignetes Datenübertragungsprotokoll für die Datenströme der Sensoren zu Odysseus ausgewählt werden.
Ziel des sechsten Sprints ist es einen ersten funktionierenden Prototypen des Zielsystems bereitzustellen, in dem die bisher umgesetzten Grundkomponenten miteinander interagieren. Dabei sollen die Messwerte der Sensoren persistent gespeichert und über die GUI Analysen auf den Daten ausgeführt und deren Ergebnisse ausgegeben werden. Weiteres wichtiges Aufgabenpaket ist die Erstellung des Zwischenberichtes, der zum Ende des Sprints abzugeben ist.
In Sprint sieben wird damit begonnen, einzelne prototypische Komponenten zu verbessern. Zunächst wird die Konfigurationskomponente dahingehend erweitert, dass sie die Konfiguration des prototypischen Systems wiederspiegelt. Danach kann der Odyssseus-Controller entwickelt werden, der die Anfragen in Odysseus steuert. Der Abruf von Analysendaten in der GUI wird verbessert.
In Sprint acht soll die Anbindung von Sensoren an eine Odysseus-Instanz verbessert werden. Dazu soll ein Protokoll bestimmt werden, dass die Übertragung von Messwerten als Einzelwerte oder Samples erlaubt. Das Protokoll muss dann entsprechend implementiert werden.
In Sprint neun kann die Verteilung von druid optimiert werden. In diesem Sprint werden verbesserte Einstellungen für eine performante Verteilung gesucht. Außerdem wird ermittelt, in welchem Mengenverhältnis die Knoten von Druid und Kafka aufgesetzt werden sollten.
Im zehnten Sprint wird das Gesamtsystem getestet und mögliche Verbesserungen werden durchgeführt. Hier werden optionale Anforderungen erfüllt. Beispielsweise könnten Liveanalysen auf den Datenstrom ausgeführt werden, die den Ausfall von Sensoren melden.
In Sprint 11 soll das Projekt abgeschlossen werden. Dazu wird die Dokumentation fertiggestellt. Außerdem soll das Projektergebnis evaluiert werden. Dazu wird geprüft, ob die Anforderungen von dem System erfüllt werden.

C. Hardwareauswahl

Hardwareauswahl PG SESAdata

UG

Ziel dieser Hardwareauflistung ist die Auswahl der benötigten Komponenten zur Realisierung des Projektes. Zur Bewertung werden neben dem Preis und der Leistung auch die Erweiterbarkeit sowie die Verbreitung der Komponenten berücksichtigt.

Zunächst werden die Anforderungen an die Komponenten aus dem Projekt dargestellt, auf deren Grundlage die Auswahl getroffen wird. Bisher besteht der Bedarf an Komponenten zur Signalerzeugung und -verarbeitung der Sensoren, welche an Einplatinencomputer wie Raspberry PIs anzuschließen werden sind. Diese Sensordaten sollen an das Datenmanagementsystem (DMS) übertragen und dort persistent gespeichert und bei Bedarf analysiert werden. Im Rahmen der Anforderungserhebung wurde die Einbindung der folgenden drei Sensortypen festgelegt.

Temperatur: Die Erfassung der Umgebungstemperatur soll mit einer geringen Abtastrate im Herzbereich erfolgen. Ein Temperatursensor ist bereits aus einem vorherigen Projekt vorhanden und bei Bedarf verwendbar. Der vorhandene Sensor bietet die Anschlussmöglichkeit über eine I^2C-Schnittstelle.

Schallwellen: Die Aufzeichnung und Übertragung hochfrequenter analoger Signale soll im 30 kHz-Bereich erfolgen.

Spannungswerte: Spannungsgrößen sollen über Spanungswandler nach Möglichkeit im kHz-Bereich abgetastet und an das DMS übertragen werden. Dabei sind genormte analoge Eingangssignale von 0 - 10V und 4 - 20 mA zu berücksichtigen.

Funktionale Anforderungen:

ID	Priorität	Anforderung
H-FA1	A	Die Hardwarekomponenten müssen die Messwerte verlustfrei verarbeiten.
H-FA2	A	Das System muss die Messwerte in ihrer Entstehungsgröße speichern.
H-FA3	A	Die Hardwarekomponenten müssen Zeitstempel liefern.
H-FA4	B	Die Hardwarekomponenten sollen um weitere Quellen erweiterbar sein.

Nicht-funktionale Anforderungen:

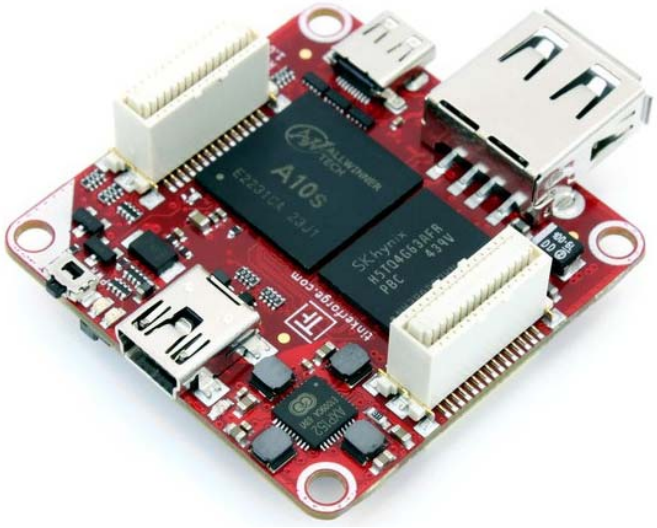
ID	Priorität	Anforderung
H-NFA1	A	Die Hardwarekomponenten müssen hochfrequente analoge Signale (Schallwellen) bis 30 kHz verarbeiten können.
H-NFA2	A	Die Hardwarekomponenten müssen analoge Eingangssignale von 0 – 10V verarbeiten können.
H-NFA3	A	Die Hardwarekomponenten müssen analoge Eingangssignale von 4 – 20 mA verarbeiten können.
H-NFA4	A	Die Hardwarekomponenten sollen die Spanungswerte im kHz-Bereich abtasten.
H-NFA5	B	Die Hardwarekomponenten müssen zuverlässig und erprobt sein.
H-NFA6	B	Die Hardwarekomponenten sollten nach Möglichkeit standardisierte Schnittstellen bereitstellen.
H-NFA7	B	Die Hardwarekomponenten sollen nach Möglichkeit als fertige Komponenten beschafft werden.
H-NFA8	C	Die Hardwarekomponenten sollen nach Möglichkeit über eine Hutschiene montierbar sein.

Auswahl Einplatinencomputer:

Unter Berücksichtigung der Anforderungen kamen die folgenden drei Geräte in die engere Auswahl. Der Raspberry PI 2 B und der Banana Pi ähneln sich von der Bauform und den Leistungsmerkmalen sehr. Beide sind vollwertige Einplatinencomputer, welche über ein Mindestmaß an Schnittstellen verfügen. Der Tinkerforge ist dagegen sehr modular aufgebaut und besitzt in seiner Grundform, dem so genannten Brick, nicht einmal eine Ethernetschnittstelle. Folgend werden die drei Geräte näher betrachtet:

Tinkerforge:

Dieses System ist ein modular aufgebauter Einplatinencomputer. Der Sogenannte Brick bildet das Herzstück des Tinkerforge-Baukastens an dem die verschiedenen Erweiterungsmodule (Bricklets) für Sensoren, Aktoren und Ein- / Ausgabe angeschlossen werden können. Darüber hinaus gibt es noch so genannte Master Extensions um das System mit einer Ethernet-Schnittstelle oder WIFI zu erweitern. Das System besitzt für alle Module eine durchdachte API, welche für zahlreiche gängige Programmier- und Skript-Sprachen umgesetzt ist. Dank der guten Dokumentation und der vielen Beispiele für alle Module, ist eine recht unkomplizierte und schnelle Verwendung möglich.



Spezifikation:

CPU:	Single-Core 900 MHz
RAM:	512 MB DDR 3 SDRAM
Netzwerk:	10/100/1000 Mbits
Analoge Eingänge:	optional
Digitale Eingänge:	16 Pin GPIO per Unit
USB:	1 Anschluss
Mikrofonanschluss:	nein
Spannungsversorgung:	5 V – 27 V
Preis	69,99 €

Benötigte Module:

Ethernet Master Extension (Netzwerkanschluss):

- Geschwindigkeit 10/100/1000 Mbit/s
- Preis 39,99 €

Master Brick (Anschluss der Erweiterungsmodule):

- Bis zu **vier** Bricklets per USB steuerbar
- Grundlage um Bricklets anzuschließen
- Preis 29,99 €

Temperature Bricklet (Temperaturmessung):

- Misst Umgebungstemperatur mit **0,5°C** Genauigkeit
- Temperaturbereich von -40°C bis 125°C
- Ausgabe in 0,1°C Schritten (12Bit Auflösung)
- Preis 6,99 €

Sond Intensity Bricklet (Schallmessung):

- Misst Schallintensität
- bietet Schaltzustand bei gewissem Geräuschpegel
- 12Bit Auflösung
- Preis 12,99 €

Voltage/Current Bricklet (Spannungsmessung):

- Misst Leistung Spannung und Strom bis zu 720W/36V/20A
- Auflösung 12 Bit 1mW, 1mV, 1mA über kompletten Messbereich
- Preis 16,99 €

4 x Hutschienenbefestigung: 4 x 5,99 € = 23,96 €
6 x Befestigungskit: 6 x 1,69 € = 10,14 €
4 x Verbindungskabel Bricklets geschirmt: 4 x 2,49 € = 9,96 €

Gesamtpreis: 226,99 €

Raspberry PI 2 B:

Dieser vollwertige Einplatinencomputer ist mit den gebräuchlichsten Schnittstellen versehen und kann direkt ohne Erweiterung autark betrieben werden. Überzeugend ist die Leistungsstärke von insgesamt vier Cortex-A7-Kernen mit bis zu 900 MHz. Ein weiteres Leistungsplus verschafft der vergrößerte Arbeitsspeicher von 1024 MB.

Technische Spezifikation:

CPU:	Quad-Core 900 MHz
RAM:	1 GB LPDDR2 SDRAM
Netzwerk:	10/100 Mbits
Analoge Eingänge:	nein
Digitale Eingänge:	40 Pin GPIO / SPI / UART / I²C
USB:	4 Anschlüsse
Mikrofonanschluss:	nein
Spannungsversorgung:	5 V
Preis:	38,50 €

Benötigte Module:

Temperatursensor:

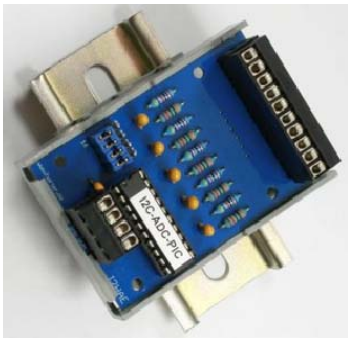
- Ist bereits aus einem vorherigen Projekt vorhanden und kann direkt an der I²C- Schnittstelle betrieben werden.

Spannungsmessung Variante 1

AD-Wandler I²C-AI418M (Spannungsmessung):

- Misst Spannung und Strom bis zu 0 – 10 V/ 0 – 40 mA
- 4 analoge Eingänge
- Schnittstelle I²C Bus (100 Khz/ 400 Khz/ 3,4 Mhz)
- Auflösung 10 Bit
- Datenrate 100 Khz
- Preis 34,45 €
- Komplettset incl. Gehäuse mit Hutschienenbefestigung

Spannungsmessung Variante 2

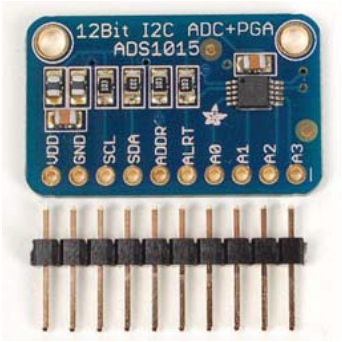


AD-Wandler I2C-Analog Input Variante 2 (Spannungsmessung)

- Misst Spannung und Strom bis zu 0 – 10 V/ 0 – 40 mA
- 5 analoge Eingänge
- Schnittstelle I²C Bus (100 Khz/ 400 Khz/ 3,4 Mhz)
- Auflösung 10 Bit
- Datenrate in Klärung
- Preis 19,90 €
- Montagerahmen 3,50 €
- Bausatz

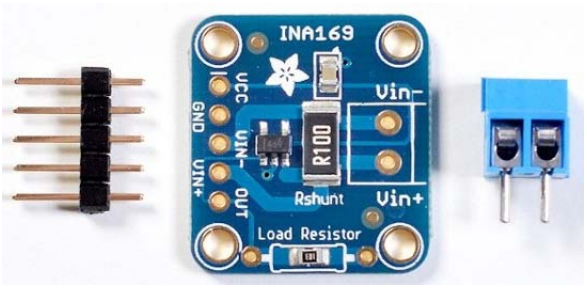
<u>Spannungsmessung Variante 3</u> (neu aufgenommen da im Ausland nicht bestellt werden kann)

Adafruit ADS1015 12-Bit ADC - 4-Kanal (Spannungsmessung)



- Misst Spannung 0 – 5 V
- 4 analoge Eingänge
- Schnittstelle I²C Bus (100 Khz/ 400 Khz/ 3,4 Mhz)
- Auflösung 12 Bit
- bis zu 3300 sps
- Preis 9,95 €
- keine Strommessung

Adafruit INA169 Analog DC (Stromstärkesensor PCB 60 V 5A max.)



- Misst Ströme bis +5A
- 1 analoger Ausgang
- Messung High Side bei bis zu +60VDC
- Ausgangsspannung 1V pro Ampere (max. 5 V)
- Preis 10,45 €

<u>Schallwellen Variante 1:</u>

LK-GeräuschSEN (Schallwellen):

- Misst Schallintensität
- bietet Schaltzustand bei gewissem Geräuschpegel
- 12Bit Auflösung
- Preis 7,99 €

Linker Kit Baseboard:

- ermöglicht den Anschluss der Linker-Kit-Module
- Baseboard mit GPIO-Anschluss und 8 Linker-Kit-Anschlüssen
- Preis 19,95 €

Linker Kit Verbindungskabel:

1 x Verbindungskabel 3,99 €

<u>Schallwellen Variante 2:</u>

USB-Soundkarte mit Mikrofon Daffodil US01:

- Linuxgeeignet
- 30 – 16.000 Hz
- Preis 12,94 €

Gehäuse Raspberry PI 2 B: 6,95 €

Gesamtpreis Variante1: <u>111,83 €</u> **Gesamtpreis Variante2: <u>92,84 €</u>**

Banana Pi:

Dieser vollwertige Einplatinencomputer ist dem Raspberry PI sowohl von der Bauform als auch von den Leistungsmerkmalen sehr ähnlich. Nur der Prozessor ist in seiner Dual-Core Ausführung nicht ganz so leistungsstark. Zur Erweiterung bietet das Gerät dieselben Schnittstellen wie der Raspberry PI, wodurch die vorherig aufgelisteten Erweiterungsmodule verwendet werden können. Preislich ist das Gerät mit 3,60 € nur minimal günstiger als der Raspberry PI. Neben der geringeren Prozessorleistung sprechen auch die geringere Verbreitung, Dokumentation und verbreiteten Anwendungsbeispiele des Gerätes gegen den Einsatz in der Projektgruppe.

Technische Spezifikation:

CPU:	Dual-Core 1 GHz
RAM:	1 GB DDR3 DRAM
Netzwerk:	10/100/1000 Mbits
Analoge Eingänge:	nein
Digitale Eingänge:	26 Pin GPIO / SPI / UART / I²C
USB:	2 Anschlüsse
Mikrofonanschluss:	nein
Spannungsversorgung:	5 V
Preis:	34,90 €

Gesamtpreis Variante 1: 108,23 €

Gesamtpreis Variante 2: 89,24 €

Auswahl:

Die Auswahl erfolgt über eine Nutzwertanalyse, bei der die wichtigsten Anforderungen an die Hardwarekomponenten in die Bewertung einfließen. Zunächst erfolgt ein kurzer Überblick der wichtigsten Eckdaten der drei Komponenten:

Leistungsmerkmal	Raspberry PI 2 B	Banana PI	Tinkerforge
CPU:	Quad-Core 900 MHz	Dual-Core 1 GHz	Single-Core 900 MHz
RAM:	1 GB LPDDR2 SDRAM	1 GB DDR3 DRAM	512 MB DDR 3 SDRAM
Netzwerk:	10/100 Mbits	10/100/1000 Mbits	10/100/1000 Mbits

Analoge Eingänge:	optional	optional	optional
Digitale Eingänge:	40 Pin GPIO / SPI / UART / I²C	26 Pin GPIO / SPI / UART / I²C	16 Pin GPIO per Unit
USB:	4 Anschlüsse	2 Anschlüsse	1 Anschluss
Mikrofonanschluss:	nein	nein	nein
Spannungsversorgung:	5 V	5 V	5 V – 27 V
Gesamtpreis:	104,88 € / 74,84 €	101,28 € / 71,24 €	226,99 €

Nutzwertanalyse:

Bewertung: von 1 bis 3 (3 am besten)

Kriterien	Gewicht	Raspberry PI 2 B	Banana PI	Tinkerforge
Datenrate	**25 %**	**3 * 0,25 = 0,75**	**3 * 0,25 = 0,75**	**1 * 0,25 = 0,25**
Analoge Signale	**20 %**	**2 * 0,20 = 0,40**	**2 * 0,20 = 0,40**	**3 * 0,20 = 0,60**
Hardwareanbindung	**10 %**	**2 * 0,10 = 0,20**	**2 * 0,10 = 0,20**	**3 * 0,10 = 0,30**
Dokumentation / Support	**15 %**	**3 * 0,15 = 0,45**	**2 * 0,15 = 0,30**	**3 * 0,15 = 0,45**
Erweiterbarkeit	**15 %**	**3 * 0,15 = 0,45**	**3 * 0,15 = 0,45**	**3 * 0,15 = 0,45**
Preis	**10 %**	**3 * 0,10 = 0,30**	**3 * 0,10 = 0,30**	**2 * 0,10 = 0,20**
Verfügbarkeit	**5 %**	**2 * 0,05 = 0,10**	**2 * 0,05 = 0,10**	**3 * 0,05 = 0,15**
Summe	**100 %**	**2,65**	**2,5**	**2,40**

Unter Berücksichtigung der Kriterien ist die Auswahl auf das Raspberry PI 2 B System mit den folgend aufgelisteten Erweiterungskomponenten gefallen. Nachfolgend sind die Preise mit den direkten Bestelllinks und den Lieferanten angegeben.

Bestellung 1 reichelt elektronik

Raspberry PI 2 B:

- Lieferant: reichelt elektronik
- Preis: 38,50 €
- Artikel-Nr.: RASPBERRY PI 2 B
- Versand: 5,60€
- Bestelllink:http://www.**reichelt**.de/?ARTICLE=152728&PROVID=2788&wt_mc=amc14152666004589&gclid=CNKNgNzpgscCFSrpwgod8EsOdQ

Raspberry PI 2 B Gehäuse:

- Lieferant: reichelt elektronik
- Artikel-Nr.: TEK-BERRY+ TR
- Preis: 6,95 €
- Versand: Bestellung zusammen mit dem Raspberry
- Bestelllink: http://www.reichelt.de/TEK-BERRY%20TR/3/index.html?&ACTION=3&LA=446&ARTICLE=150184&artnr=TEK-BERRY%2B+TR&SEARCH=RASPBERRY+PI+2+B+geh%E4use

Summe mit Gehäuse: 51,05 €
Summe ohne Gehäuse: 44,10 €

Bestellung 2 ERE Company Limited (10250 THAILAND)

AD-Wandler I²C-AI418M (Spannungsmessung)

- Lieferant: ERE Company Limited (10250 THAILAND)
- Artikel-Nr.: I2C-AI418S
- Preis: 28,64 $ / 26,16 €
- Versand: 31,13 $ / 28,43 € (FedEx Priority Express)
- Bestelllink: https://www.ereshop.com/shop/i2c-420ma-010v-adc-p-805.html

AD-Wandler DIN rail 2pcs Foot 72x42.50mm

- Lieferant: ERE Company Limited (10250 THAILAND)
- Artikel-Nr.: DIN rail 2pcs Foot
- Preis: 7,83 $ / 7,15 €
- Versand: zusammen mit dem AD-Wandler
- Bestelllink: https://www.ereshop.com/shop/enclosures-c-169/din-rail-2pcs-foot-72x4250mm-p-817.html

Summe: 61,74 €

Bestellung 3 amazon:

USB-Soundkarte Daffodil US01:

- Lieferant: amazon
- Artikel-Nr.: /
- Preis: 5,95 €
- Versand: 3,00 € (Prime kostenlos)
- Bestelllink: http://www.amazon.de/Daffodil-US01-Soundkarte-Mikrofon-Lautsprecher-Anschluss/dp/B002FI7GWK

Hama Notebook Mini-Mikrofon:

- Lieferant: amazon
- Artikel-Nr.: /
- Preis: 6,99 €
- Versand: 3,00 € (Prime kostenlos)
- Bestelllink:http://www.amazon.de/gp/product/B000GIGR4O/ref=pd_luc_rh_sim_01_02_t_ttl_lh?ie=UTF8&psc=1

Raspberry PI 2 B Gehäuse:

- Lieferant: amazon
- Artikel-Nr.:
- Preis: 9,95 €

- Versand: 4,95 €

Bestelllink: http://www.amazon.de/Hutschienen-Geh%C3%A4use-f%C3%BCr-Raspberry-Pi-10-001225-RPI/dp/B00LUDYJPI/ref=pd_sim_23_1?ie=UTF8&refRID=1XDA3A0XPMEQAB74FACX

Summe ohne Hutschinengehäuse: 18,94 €
Summe mit Hutschinengehäuse: 33,84 €

Bestellung 4 flikto elektronik: (Ersatzbestellung für Bestellung 2)

Adafruit ADS1015 12-Bit ADC - 4-Kanal (Spannungsmessung)

- Lieferant: flikto elektronik
- Flikto-Nr.: flik.to/9 Hersteller-Nr.: 1083
- Preis: 9,95 €
- Versand: 3,50 €
- Bestelllink: http://www.flikto.de/products/ads1015-12-bit-adc-4-channel-with-programmable-gain-amplifier

Adafruit INA169 Analog DC (Stromstärkesensor PCB 60 V 5A max.)

- Lieferant: flikto elektronik
- Flikto-Nr.: flik.to/392 Hersteller-Nr.: 1064
- Preis: 10,45 €
- Versand: zusammen mit ADC
- Bestelllink: http://www.flikto.de/products/adafruit-ina169-analog-dc-current-sensor-breakout-60v-5a-max

Jumper-Wires - 15cm (F/F, 20 Stück)

- Lieferant: flikto elektronik
- Flikto-Nr.: flik.to/872 Hersteller-Nr.: PRT-12796
- Preis: 1,95 €
- Versand: zusammen mit ADC
- Bestelllink: http://www.flikto.de/products/jumper-wires-connected-6-f-f-20-pack

Gesamtpreis mit normalem Gehäuse inc. Versand: 131,73 €
Gesamtpreis mit Hutschinengehäuse inc. Versand: 139,68 €

Unter Berücksichtigung der Ersatzbestellung:
Gesamtpreis mit normalem Gehäuse inc. Versand: 95,84 €
Gesamtpreis mit Hutschinengehäuse inc. Versand: 103,79 €

D. Gehäuseauswahl

UG

Komponentenauswahl zur Montage der Sensorhardware ins SESALab

UG

Anforderungen:

- das Gehäuse soll nicht transparent sein
- die Hardware soll vor Berührungen geschützt werden
- die Gehäuse sollen an Hutschienen befestigt werden können
- das Netzteil soll die Hardware mit ausreichend Leistung versorgen und über 230 V oder 24 V betrieben werden können

Gehäuse Raspberry PI 2B

Hutschienengehäuse für den Raspberry PI 2B

- Typ: Standard Hutschienengehäuse
- Ausführung: RPI CASE DINRAIL
- Breite: 90
- Höhe: 80 mm
- Tiefe: 58 mm
- Einbau: Schraubenlos
- Technologie: 35 mm DIN Schienensystem
- Preis:10,60 €
- Lieferstatus: z.Zt. ausverkauft
- Lieferant: Reichelt
- Bestellink: http://www.reichelt.de/RPI-CASE-DINRAIL/3/index.html?&ACTION=3&LA=446&ARTICLE=160286&artnr=RPI+CASE+DINRAIL&SEARCH=hutschinengeh%E4use

Hutschienengehäuse für Raspberry PI 2 Modell B / B+

- Typ: Standard Hutschienengehäuse
- Ausführung: RPI CASE DINRAIL
- Breite: nicht angegeben
- Höhe: nicht angegeben
- Tiefe: nicht angegeben
- Einbau: schraubenlos
- Technologie: 35 mm DIN Schienensystem
- Preis: 11,70 €
- Lieferant: sertronics
- Bestellink: http://www.avc-shop.de/epages/64272905.sf/de_DE/?ObjectPath=/Shops/64272905/Products/RPI2-DINC

Gehäuse Sensoren

Standard Hutschienengehäuse als Kit, Größe 12

- Typ: Standard Hutschienengehäuse
- Ausführung: Gehäuse - Kit, Größe 12
- Breite: 212 mm
- Höhe: 90 mm
- Tiefe: 58 mm
- Technologie: 35 mm DIN Schienensystem
- Schutzart: UL94-V0
- Preis: 13,40
- Lieferant: Reichelt
- Bestellink: http://www.reichelt.de/CB-HUTKIT-12/3/index.html?&ACTION=3&LA=446&ARTICLE=133942&artnr=CB+HUTKIT+12&SEARCH=hutschinengeh%E4use

Hutschienen-Gehäuse 90 x 160.0 x 58 Polycarbonat Axxatronic CNMB-9-KIT-CON 1 St.

- Breite: 160.0 mm
- Ausführung: ohne Lüftungsschlitze
- Kategorie: Hutschienen-Gehäuse
- Länge: 160 mm
- Breite: 90 mm
- Höhe: 58 mm
- Prüfzeichen: UL 94 V-0
- Pole: 54
- Material: Polycarbonat
- Preis: 12,33 €
- Lieferant: Conrad
- Bestellink: https://www.conrad.de/de/hutschienen-gehaeuse-90-x-1600-x-58-polycarbonat-axxatronic-cnmb-9-kit-con-1-st-531447.html

Spannungsversorgung

Micro-USB Steckernetzteil, 5V, 1,2A

- Typ: USB-Ladegerät
- Ausführung: für micro USB Geräte
- Strom: 1200 mA
- Anschluss: Netz
- Betriebsspannung: 100 - 240 V AC V
- Technologie: eco friendly
- Output: 5 VDC
- Farbe: schwarz

- Preis: 3,99 €
- Lieferant: Reichelt
- Bestelllink: http://www.reichelt.de/NT-MICRO-USB-1-2/3/index.html?&ACTION=3&LA=446&ARTICLE=134957&artnr=NT+MICRO+USB+1%2C2&SEARCH=netzteil+raspbarry+pi

Micro USB Netzteil - 2000mA 2,0A / 5V

- kompaktes Steckernetzteil
- Micro USB B Stecker
- Input 100-240V AC
- Output 5V / 2000mA
- grüne Betriebs-LED
- Kabellänge ca. 1,2m
- Lieferant: sertronics
- Bestellink: http://www.avc-shop.de/epages/64272905.sf/de_DE/?ObjectPath=/Shops/64272905/Products/4260220079705

Micro USB Netzteil - 1200mA 1,2A / 5V

- kompaktes Steckernetzteil
- Micro USB B Stecker
- Input 100-240V AC
- Output 5V / 1200mA
- rote Betriebs-LED
- Kabellänge ca. 1,4m
- Preis: 3,50 €
- Lieferant: sertronics
- Bestellink: http://www.avc-shop.de/epages/64272905.sf/de_DE/?ObjectPath=/Shops/64272905/Products/4053271019017

Verbindungskabel:

40pin Jumper / Dupont Kabel Female - Female trennbar

- besteht aus 40 Adern
- Female (Buchse) <-> Female (Buchse)
- Adern können auch getrennt werden
- 2,54mm Pinabstand (Pitch)
- Bleifrei sowie RoHs kompatibel
- Preis: 3,90 €
- Lieferant: sertronics
- Bestellink: http://www.avc-shop.de/epages/64272905.sf/de_DE/?ObjectPath=/Shops/64272905/Products/DUPONT-FF

Aufstellung:

Anzahl	Artikel	Preis	Gesamt
2	Hutschienengehäuse für Raspberry PI 2 Modell B / B+	11,70 €	23,40 €
1	Hutschienen-Gehäuse 90 x 160.0 x 58 Polycarbonat Axxatronic CNMB-9-KIT-CON	12,33 €	12,33 €
2	Micro USB Netzteil - 1200mA 1,2A / 5V	3,50 €	7,00 €
1	40pin Jumper / Dupont Kabel Female - Female trennbar	3,90 €	3,90 €
		Summe	**46,63 €**

E. Datenbankauswahl

MM

Datenbankauswahl

Projektgruppe SESAdata

31. März 2016

MM

In diesem Dokument wird die Datenbankauswahl begründet, die im Rahmen der Projektgruppe SESAdata getroffen wurde. Zuerst werden die Datenbanken genannt, die getestet wurden. Anschließend werden die Testergebnisse dargestellt und aufgrund einer Auswertung begründet, warum die Entscheidung auf Druid gefallen ist.

1 Testkandidaten

MM

Hier werden alle Datenbanken aufgeführt, an denen die Testläufe erfolgreich abgeschlossen werden konnten. Einige andere Kandidaten, die zuerst in Betracht gezogen wurden konnten bereits vor der Testphase ausgeschlossen werden. Hierzu ist mehr in der Projektdokumentation zu lesen.

Folgende in-Memory-Datenbanken wurden getestet:

- **VoltDB** ist eine NewSQL Datenbank. Sie wirbt damit, dass SQL genutzt werden kann aber trotzdem Echtzeitdaten in Millisekunden analysiert werden können. Es sollen schnelle Datenströme verarbeitet werden können [Vol15].
- **Aerospike** ist eine NoSQL Datenbank, die einen Key-Value Store nutzt. Dabei ist Aerospike keine reine in-Memory Datenbank. Es werden nur die Indizes im Hauptspeicher gehalten. Die eigentlichen Daten werden auf einer SSD gehalten [Aer15].

Folgende Langzeitdatenbanken wurden getestet:

- **InfluxDB** ist eine Zeitreihenbasierte Datenbank, die eigenständig lauffähig ist. Sie ist spezialisiert auf Sensordaten und soll Analysen in Echtzeit unterstützen. Dabei wird eine SQL-ähnliche Anfragesprache bereitgestellt. Aktuell befindet sich InfluxDB noch in der Alphaphase. Außerdem gibt es bereits Internetapplikationen, die z.B. eine Visualisierung der Daten ermöglichen [Inf15].

- **Cassandra** ist eine NoSQL Datenbank, die hauptsächlich auf Skalierbarkeit und hohe Verfügbarkeit ausgelegt ist. Die Performance soll ebenfalls sehr hoch sein. Allerdings werden Aggregationen nur teilweise und Cursor noch nicht unterstützt [Apa15].

Datenbank, die sowohl in-Memory, als auch Langzeitknoten hat:

- **Druid** ist eine zeitreihenbasierte Datenbank. Die Besonderheit bei Druid ist, dass sie eingehende Daten direkt aus einem Datenstom akzeptiert und in einer in-Memory Datenbank speichert. Es gibt spezielle Broker-Knoten, die Daten aus der in-Memory Datenbank in einen angeschlossenen historischen Speicher schreiben können. Für diesen kann das Dateisystem oder auch eine andere Langzeitdatenbank genutzt werden. Ein Client kann Anfragen an die Broker-Knoten stellen, die zur Beantwortung auf die in-Memory Datenbank und die Langzeitdatenbank zurückgreifen können. Damit sind bereits einige Funktionalitäten des Bulk-Storers und des Analysetools aus LAOTSE von der Datenbanklösung, die in-Memory und Langzeitdatenbank vereint gegeben. Allerdings werden einige externe Komponenten benötigt um z.B. Metadaten über die Knoten zu speichern. Insgesamt ist Druid als sehr schnell und skalierbar beworben, da alle Funktionalitäten verteilt implementiert sind. Die Anfragesprache ist für Analysen ausgelegt und bietet zahlreiche Filter und Berechnungen. Druid befindet sich momentan in der Version 0.8.0 wird aber trotzdem bereits von zahlreichen großen Unternehmen, wie z.B. Netflix, Cisco, Yahoo, Paypal und eBay produktiv eingesetzt [Dru15].

2 Testergebnisse

MM

Hier werden die Testergebnisse aufgeführt und analysiert.

2.1 Insert

Es wurden 50 Millionen Tupel in die Datenbanken eingefügt und die benötigte Zeit in Millisekunden gemessen. In Abbildung 2.1 sind die Testergebnisse dargestellt.

Während des Testlaufs ist in Aerospike ein reproduzierbarer Speicherüberlauf aufgetreten, obwohl über 20 GB Hauptspeicher zur Verfügung standen. Damit ist Aerospike für LAOTSE ungeeignet, da es nicht so große Datenmengen verarbeiten kann. Weiter fällt auf, dass die reine in-Memory Datenbank VoltDB beim Einfügen der Daten langsamer ist, als die reine Langzeitdatenbank Cassandra. Der Hybrid Druid liegt nur knapp hinter Cassandra und ist damit die schnellste Datenbank, die in-Memory Fähigkeiten besitzt.

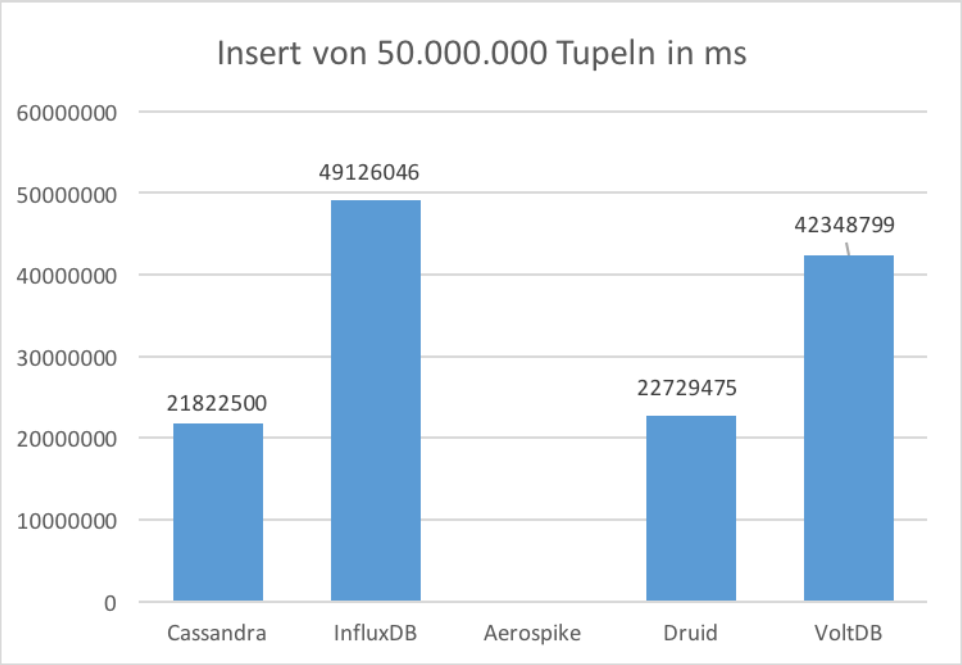


Abbildung 1: Benötigte Zeit in Millisekunden beim Einfügen von 50 Millionen Tupeln.

2.2 Average

Hier wurde ein Durchschnittswert über 90 000 Tupel gebildet. In dieser Abfrage und allen folgenden Abfragen wird Cassandra nicht weiter betrachtet, da die Lesegeschwindigkeit bei Cassandra ca. 100 mal langsamer ist als bei allen anderen Datenbanken. In den folgenden Abbildungen sind die Testergebnisse über die Abfragen zu sehen.

In den Abbildungen 2.2, 2.2 und 2.2 ist zu sehen, dass die Lesegeschwindigkeit bei VoltDB am schnellsten ist. InfluxDB belegt den zweiten Platz und Druid den dritten Platz. Aerospike kann außer acht gelassen werden, das diese Datenbank bereits beim Einfügen der vielen Daten versagte. In Abbildung 2.2 ist zu sehen, dass InfluxDB hier bedeutend länger benötigte. Das liegt daran, dass InfluxDB keine Rangeabfrage nativ unterstützt und dies über zahlreiche einzelne Anfragen imitiert werden musste.

3 Auswertung

MM

Wenn die Testergebnisse mit den entsprechenden Erläuterungen betrachtet werden bleiben final folgende drei Kandidaten zur Verwendung in LAOTSE:

- InfluxDB (Langzeitdatenbank)
- Druid (Langzeit- und in-Memory Datenbank)

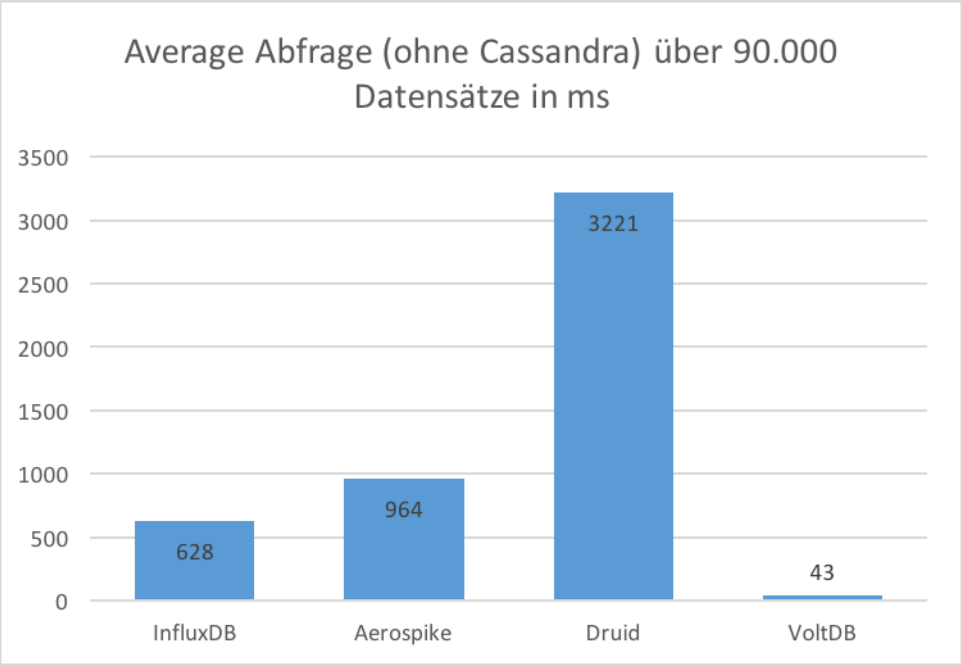


Abbildung 2: Benötigte Zeit in Millisekunden zur Bildung eines Durchschnitts über 90 000 Tupel.

- VoltDB (in-Memory Datenbank)

Die Anforderungen von LAOTSE sehen vor, dass die Geschwindigkeit beim Einfügen von Tupeln oberste Priorität hat. Daher scheint Druid der geeignetste Kandidat zur Verwendung in LAOTSE. Weiter noch sieht die Architektur von LAOTSE die Verwendung einer in-Memory und einer Langzeitdatenbank vor. Die Alternative zur Verwendung von Druid wäre also die Verwendung von VoltDB und InfluxDB. Damit wäre die in-Memory Datenbank VoltDB, die direkt an den Datenstrom aus Odysseus angebunden werden soll, allerdings bereits ein Flaschenhals beim Einfügen der Daten. Die Daten könnten zwar u.U. schneller ausgelesen werden, allerdings gar nicht so schnell neue Daten eingefügt werden. In LAOTSE werden allerdings mehr Daten eingefügt als ausgelesen. Zudem würde die Verwendung eines selbst implementierten Bulk-Storers, der die Daten von VoltDB nach InfluxDB überträgt den Datenfluss noch weiter ausbremsen.

Druid bringt diese Techniken der Übertragung der Daten aus einer eigenen in-Memory Datenbank in eine eigene Langzeitdatenbank bereits mit. Zudem kann Druid einfach skaliert werden, da es als Cluster aufgebaut ist. Das Lesen von Daten kann aus speziellen Knoten erfolgen, die die Verteilung der Daten auf in-Memory Knoten und Langzeitknoten kennen. Somit ist kein Management über die Datenverteilung über die verschiedenen Datenbanken mehr nötig.

Somit fällt die Wahl auf Druid.

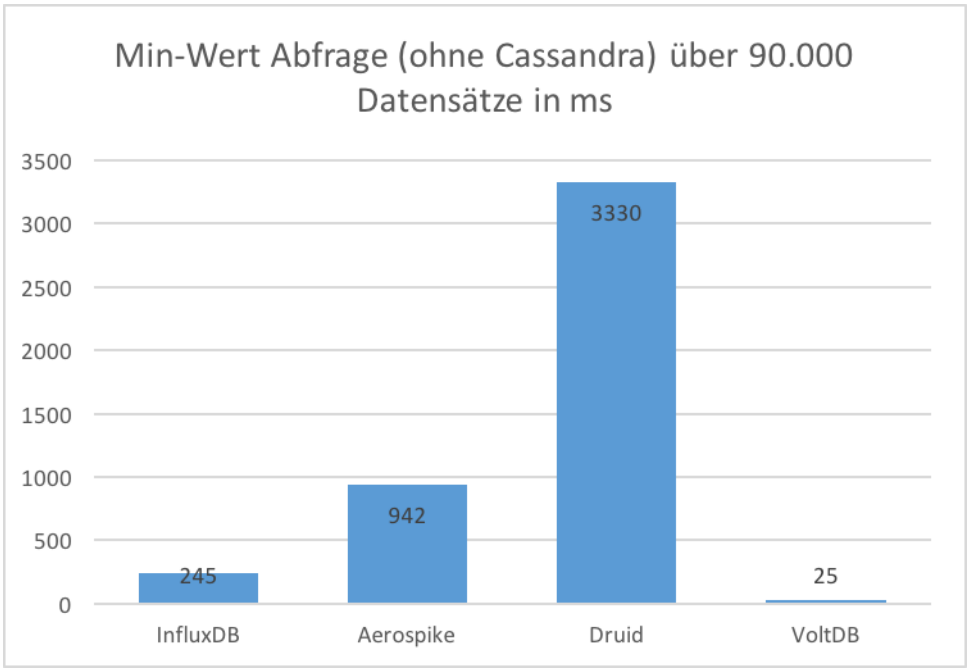


Abbildung 3: Benötigte Zeit in Millisekunden zur Bildung eines Minimums über 90 000 Tupel.

Literatur

[Aer15] AEROSPIKE, Inc.: *Aerospike*. Webseite. `http://www.aerospike.com`. Version: July 2015

[Apa15] APACHESOFTWAREFOUNDATION: *ApacheWiki - Cassandra*. Webseite. `http://wiki.apache.org/cassandra/FrontPage`. Version: July 2015

[Dru15] DRUID: *druid*. Webseite. `http://druid.io`. Version: July 2015

[Inf15] INFLUXDB: *InfluxDB*. Webseite. `https://influxdb.com`. Version: July 2015

[Vol15] VOLTDB, Inc.: *Using VoltDB*. Documentation. `http://downloads.voltdb.com/documentation/UsingVoltDB.pdf`. Version: July 2015

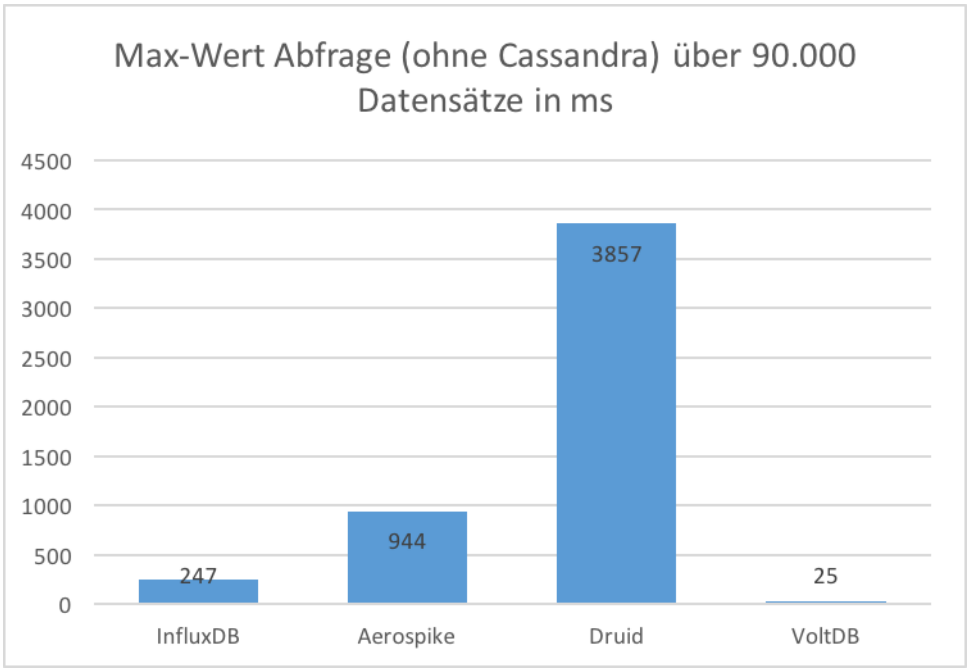


Abbildung 4: Benötigte Zeit in Millisekunden zur Bildung eines Maximums über 90 000 Tupel.

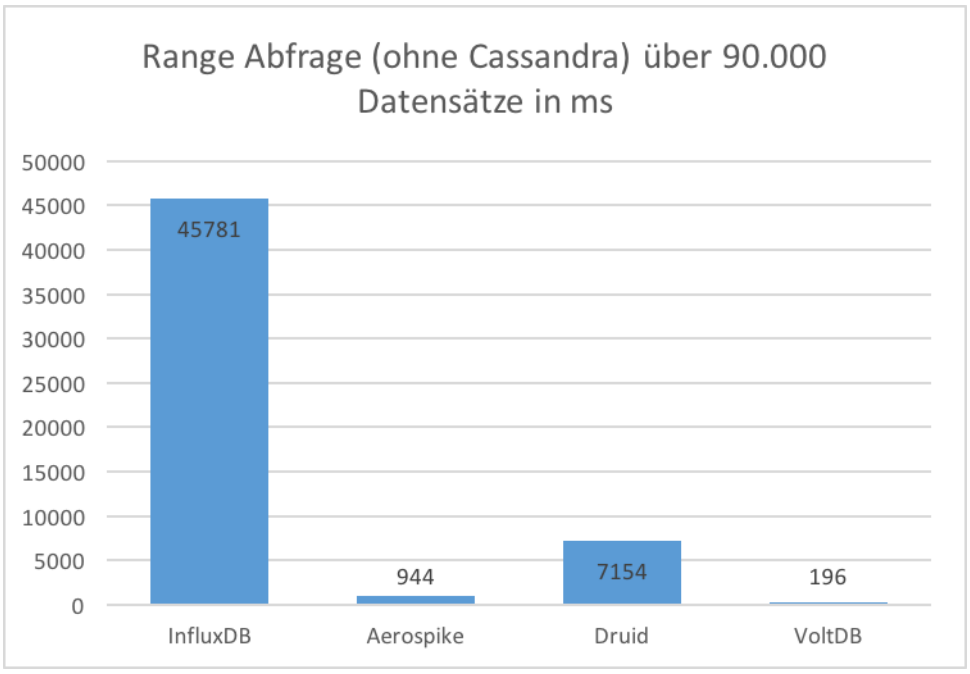


Abbildung 5: Benötigte Zeit in Millisekunden zur Bildung eines Datenumfangs über 90 000 Tupel.

F. Usability Test

UG

In diesem Abschnitt wird das Vorgehen des Usability Tests mit den Methoden und Merkmalen beschrieben.

Vorbereitung

Zu Beginn sind die relevanten Benutzereigenschaften und die Aufgaben des Zielsystems zu identifizieren. Das Zielsystem wird von fachlich versierten Anwendern genutzt, die über ausreichend Hintergrundwissen über das Einsatzgebiet verfügen. Gemäß den Anforderungen werden folgende Aufgaben des Zielsystems bestimmt:

- Vorhandene Sensoren anzeigen
- Metadaten der Sensoren anzeigen
- Metadaten der Sensortypen anzeigen
- Messwerte der Sensoren anzeigen (Dashboard)
- Mögliche Sensortypen anzeigen
- Vorgegebene Analysen anzeigen
- Aufzeichnungszeiträume der Sensoren anzeigen
- Neue Sensoren hinzufügen
- Sensoren bearbeiten (deaktivieren, Bearbeiten der Metadaten wie Standort)
- Mosaik Simulationen initialisieren, starten und aufzeichnen
- Hilfe
- Schließen

Testmethoden

Gemäß der Aufgaben werden die verwendeten Methoden auf Grundlage der Usability-Zielsetzung bestimmt. Um im Testverlauf direkt auf Problemstellungen und weitere Fragen eingehen zu können, wird ein moderierter Usability-Test durchgeführt. Dabei werden den Probanden der Reihe nach typische Aufgaben des Zielsystems vorgegeben, welche eigenständig zu lösen sind. Um auch fehler- und zeitabhängige Messgrößen erheben zu können, wird die Interaktion zwischen Moderator und Testperson auf ein Minimum beschränkt. Nur im Notfall greift der Moderator

ein und gibt Hilfestellung. Zu Beginn jeder Aufgabe wird mittels eines sogenannten fünf Sekunden Tests die Erwartung des Lösungsweges der Kandidaten, unvoreingenommen durch die Lösung des Zielsystems, erfasst. Dazu dürfen die Testpersonen sich das Zielsystem 5 Sekunden lang, ohne scrollen oder anklicken, ansehen. Dann erfolgt die Minimierung der GUI, um die Personen nach den erwarteten und besonders aufgefallenen Inhalten zu befragen. Damit lässt sich feststellen, ob die durch das Menü, Überschriften und Funktionen gesetzten Schwerpunkte, auch ankommen. Gelingt dies nur teilweise, oder werden die wichtigsten Inhalte nicht genannt, sind Änderungen erforderlich. Erst danach erfolgt die Lösung der vorbereiteten Aufgabe.
Um die Gedankengänge der Testpersonen nachvollziehen zu können, werden diese gebeten Ihre Überlegungen bei der Aufgabenerfüllung laut mitzuteilen. Dabei soll der Kandidat nach Möglichkeit sagen, was er beabsichtigt, warum er den Weg beschreitet und was er dabei vom Zielsystem erwartet. Dadurch ist die Verfolgung der Entscheidungswege und ggf. auftretenden Probleme genau möglich.
Während der Aufgabenerledigung durch die Kandidaten werden für jede Aufgabe die Abläufe, Ablaufdauer, Probleme, Emotionen und Erwartungen durch eine weitere Person beobachtet und notiert.
Nach jeder Aufgabe wird der Testproband kurz zum Verlauf, der Gebrauchstauglichkeit des Systems und möglichen Verbesserungsvorschlägen befragt.
Abschließend erfolgt die Auswertung der Usability Tests, um die wahrgenommene Gebrauchstauglichkeit und Qualität des Zielsystems zu überprüfen. Dazu erfolgt die Gruppierung der Aufzeichnungen der Tests, um die Stärken und Schwächen in den betroffenen Funktionen erfassen zu können. Die Ergebnisse werden dem Auftraggeber vorgestellt, um gemeinsam resultierende Verbesserungen festzulegen.

Testmaterial und Einrichtung

Zu Testbeginn müssen alle Unterlagen vorliegen und die Testumgebung funktionieren. Dazu sind die Fragebögen und die Testumgebung vorzubereiten und optimalerweise vorher ein Probedurchlauf durchzuführen.

Teilnehmer

Die erforderlichen Charakteristika der Testpersonen sind Voraussetzung für deren Auswahl. Dabei sind die Anreise zum Testort und die Anzahl der Teilnehmer zu beachten. Generell gilt je mehr Testpersonen zur Verfügung stehen, desto mehr Informationen können über die Benutzerfreundlichkeit des Zielsystems in Erfahrung gebracht werden. Laut Nielsen reichen fünf Testpersonen vollkommen aus, da diese mehr als 83 Prozent aller möglichen Informationen bereitstellen. Je mehr Probanden befragt werden, desto geringer ist der neue Informationsgehalt, weil die wesentlichen Dinge immer wiederholt werden (vgl. [Dw1].
Die Nutzung des Zielsystems erfolgt in der Praxis von fachkundigem Personal, wodurch der Auswahlkreis der Probanden auf das Offis im Bereich Energie fokussiert wird. Nach Möglichkeit soll der Test mit fünf Testpersonen durchgeführt werden.

Einführung

Bevor der Test mit dem Teilnehmer durchgeführt wird, ist eine kleine Heranführung an die Thematik empfehlenswert. Dabei ist der Testperson der Einsatzzweck und die Umgebung des Ziel-

systems zu verdeutlichen. Des Weiteren sollte dem Probanden die Bedeutung seiner Hilfe bei der Überprüfung der Gebrauchstauglichkeit des zu entwickelnden Softwaresystems klar gemacht und die vertrauliche Behandlung seiner Aussagen und des Testvorgehens zugesichert werden.

Test

Im Allgemeinen unterscheiden sich die Testvorgehen hinsichtlich Zweck, Dauer, finanziellen Aufwand und Methoden. Da die GUI bereits während der Entwicklungsphase fortlaufend mit den Anforderungen der Auftraggeber überprüft und angepasst wurde, dient dieser abschließende Usability Test zur letzten Validierung und eventuellen Anpassung. Wie im Abschnitt F Testmethoden erläutert, wird ein moderierter Usability Test verwendet, um die Qualität des Tests zu maximieren und direkt auf Problemstellungen und weitere Fragen eingehen zu können. Damit allen Testpersonen die identische Ausgangssituation gegeben wird und keine Details vergessen werden, erfolgt die Erstellung eines Testleitfadens. Dieser bietet nach den jeweiligen Aufgabenbeschreibungen ausreichend Platz für die Protokollierung. Vor jeder Aufgabe wird mittels eines sogenannten fünf Sekundentest die Erwartungen der Kandidaten und der erste Eindruck unvoreingenommen ermittelt. Während der Testaufgaben ist die Interaktion des Moderators auf ein Minimum zu reduzieren, um das Verhalten der Testperson nicht durch Tipps und Hinweise zu beeinträchtigen. Nach Möglichkeit ist der Teilnehmer durch kurze Hinweise wie “Können Sie uns mitteilen, was sie gerade denken?“ oder “Haben Sie dieses Verhalten erwartet?“ zum lauten Denken zu animieren. Wenn die Informationen der Testperson nicht ausreichend oder zu oberflächlich sind, ist dieses durch gezieltes Nachfragen zu verbessern. Nach jedem Test ist das Ergebnis, durch ein kurzes Zusammenfassen des Protokollanten und der Nachfrage, ob das so richtig wahrgenommen und verstanden wurde, zu überprüfen.

Testleitfaden

Der Testleitfaden dient als roter Faden für den Usability-Test, um allen Teilnehmern die gleiche Ausgangssituation bereitzustellen und keine wichtigen Details zu vergessen. Bevor mit dem Test begonnen wird, erfolgt eine kurze Beschreibung des zu testenden Zielsystems.

Zielbestimmung Systemeinsatz

Ziel des Projektes ist die Schaffung einer Datenmanagementumgebung für das SESA-Lab, bei der die zum Teil sehr hochfrequenten und von heterogenen Sensoren stammenden Daten, für Forschungszwecke analysiert und persistent gespeichert werden können. Die Messwerte sollen dabei mit genauen Zeitstempeln, echtzeitnah und verlustfrei verarbeitet werden. Darüber hinaus sind Analysen und Visualisierungen sowohl auf den Echtzeitdaten, als auch auf die gespeicherten Langzeitdaten zu ermöglichen.
Die Datenquellen sind bisher Simulationsdaten des SESA-Labs, welche über Mosaik bereitgestellt werden und Messwerte von verschiedenen Sensoren (Temperatursensor, AD-Wandler und Audiosensor), welche mittels Raspberry PIs übertragen werden.

Testablauf

Gemäß der drei Hauptkomponenten des Zielsystems erfolgt die Gliederung des Testaufbaus. Zunächst wird der Bereich der Sensoren betrachtet, indem Sensoren editiert, bearbeitet und neu

angelegt werden. Als nächstes erfolgt die Überprüfung typische Aufgabenstellungen zur Analyse und Visualisierung der Daten mittels des Dashboards. Zum Abschluss erfolgt die Überprüfung der Funktionalitäten zur Verarbeitung der Simulationsdaten des SESA-Labs, welche über Mosaik bereitgestellt werden.

Sensoren

1. **Sensoren inklusive Metadaten anzeigen** Zeigen Sie bitte die vorhandenen Sensoren an und überprüfen den Standort des Temperatursensors Thermometer1. Welche Metadaten werden angezeigt?

   5 Sekunden Test:
   Welche Erwartungen haben Sie an den Aufruf der Sensoren und der Anzeige der Metadaten?

   Wie vermuten Sie den Aufruf anhand der gesehenen GUI?

   Notizen Testablauf:

   Positiv:

   Negativ:

   Weg:

Zeit:

Fragen:

Anmerkungen:

2. **Sensor bearbeiten** Bitte ändern Sie beim Temperatursensor Thermometer1 die Metadaten. Tragen Sie Ihren Namen und das Datum als Kommentar ein und verändern Sie den Standort durch die Angabe des Längen- und Breitengrades 42/42.

   5 Sekunden Test:
   Welche Erwartungen haben Sie an die Bearbeitung der Metadaten der Sensoren?

   Wie vermuten Sie die Bearbeitung anhand der gesehenen GUI?

Notizen Testablauf:

Positiv:

Negativ:

Weg:

Zeit:

Fragen:

Anmerkungen:

3. **Neuen Sensor hinzufügen** Bitte legen Sie einen neuen Temperatursensor mit folgenden Metadaten an: (Name: TempBIB; Standort: 53.146733 Breitengrad / 8.183124 Längengrad; Kommentar: Standort Oldenburg; Einheit: ° Celsius ; inaktiv; autostart).

5 Sekunden Test:

Welche Erwartungen haben Sie an das Hinzufügen neuer Sensoren?

Wie vermuten Sie den Ablauf für das Hinzufügen?

Notizen Testablauf:

Positiv:

Negativ:

Weg:

Zeit:

Fragen:

Anmerkungen:

4. **Aufzeichnungszeitraum der Sensoren** Wann wurden am 23.02.16 Messwerte vom Temperatursensor RBTemperature aufgezeichnet?

   5 Sekunden Test:

   Welche Erwartungen haben Sie an diese Funktion?

   Wie Vermuten Sie die Überprüfung der Aufzeichnung anhand der gesehenen GUI?

   Notizen Testablauf:

   Positiv:

   Negativ:

Weg:

Zeit:

Fragen:

Anmerkungen:

Dashboard

1. **Dashboard erstellen für RBTemperature** Bitte erstellen Sie ein Dashboard indem die Messwerte für den Temperatursensor RBTemperature vom 23.02.16 von 15:00 – 16:00 Uhr mit einer Granularität von 1 Min angezeigt werden. Wie ist der MIN, MAX, AVG, Count Wert? Was fällt beim Verlauf auf (Temperatur Anstieg und Abfall)?

   5 Sekunden Test:

   Welche Erwartungen haben Sie an die Analyse und Visualisierung der Messwerte?

Wie Vermuten Sie die Analyse und Visualisierung anhand der gesehenen GUI?

Notizen Testablauf:

Positiv:

Negativ:

Weg:

Zeit:

Fragen:

Anmerkungen:

2. **Analysetypen anzeigen** Lassen Sie sich die vorhandenen Analysen anzeigen. Was für Metadaten werden dort bei Range angezeigt?

   5 Sekunden Test:

   Welche Erwartungen haben Sie an diese Aufgabenstellung?

Wie vermuten Sie die Umsetzung anhand der gesehenen GUI?

Notizen Testablauf:

Positiv:

Negativ:

Weg:

Zeit:

Fragen:

Anmerkungen:

Mosaik (Simulationsdaten aus dem SESA-Lab):

1. **Neue Initialisierung** Bitte wählen Sie das Szenario demoLaotse.py aus, welches im SESA-lab ausgeführt werden soll, damit die Simulationsdaten gespeichert werden. Welche Parameter sind einzugeben und welche sind voreingestellt.

   5 Sekunden Test:

   Welche Erwartungen haben Sie für die Auswahl des Simulationsszenarios?

   Wie vermuten Sie die Auswahl anhand der gesehenen GUI?

Notizen Testablauf:

Positiv:

Negativ:

Weg:

Zeit:

Fragen:

Anmerkungen:

2. **Virtuelle Sensoren der Simulation anzeigen** Bitte lassen Sie sich die virtuellen Sensoren von Mosaik anzeigen.

   5 Sekunden Test:

   Welche Erwartungen haben Sie an den Aufruf der virtuellen Sensoren?

   Wie Vermuten Sie den Aufruf anhand der gesehenen GUI?

   Notizen Testablauf:

   Positiv:

   Negativ:

   Weg:

   Zeit:

Fragen:

Anmerkungen:

Auswertung

Der Einladung des Usability Tests sind 4 Probanden gefolgt. Der Test ist bewusst an einem einzigen Tag durchgeführt worden, um die Austauschmöglichkeit der Testpersonen zu minimieren. Die protokollierten Ergebnisse wurden jeweils nach der Durchführung von zwei Tests ausgewertet und gemäß der Gruppierung des Testleitfadens zusammengeführt. Da die Funktionstauglichkeit der GUI im Vordergrund der Betrachtung steht, werden im ersten Abschnitt die Ergebnisse der abschließenden Frage zur Funktionstauglichkeit dargestellt. Nach der Darstellung des Gesamteindruckes des Designs der GUI, folgen die Ergebnisse der Einzelaufgaben. Zum Abschluss wird eine Zusammenfassung mit den wesentlichen Erkenntnissen des Tests gegeben, welche für eine mögliche Umsetzung näher betrachtet werden.

Allgemeine Funktionstauglichkeit:

- reicht für Anwendungsfall aus, keine funktionalen Änderungen nötig (4 Personen)
- nach erster Nutzung ist GUI intuitiv (2 Personen)
- sofort mittels Ausprobieren intuitiv (1 Person)
- mit Hilfe problemlos butzbar (1 Person)

Design:

- Schrift/Hintergrund: Kontrast schwarz auf weiß nicht so gut (2 Personen)
- linkes Pane muss auch in Breite verstellbar sein (lange Sensornamen) (2 Personen)
- Aufteilung und Logik der Fenster allgemein dokumentieren insbesondere bei Metadaten und Eigenschaften (2 Personen)

- Übersicht gut (2 Personen)
- Farben insgesamt zu dunkel Schrift heller (1 Person)
- Maushover mit Erklärungen für Ein-/Ausgaben (1 Person)
- Fenstergrößen anpassen → Konsole kleiner Ein/Ausgabefenster größer (1 Person)
- Einheiten an Grafik im Dashboard (1 Person)
- Konsoleneingabe zu dominant, kann kleiner (1 Person)

Sensoren:

Sensoren Anzeigen:

Erwartung:

- Nach Auswahl eines Sensors → $Daten$ im rechten Fenster (2 Personen)

Vermutetes Vorgehen:

- Sensoren → $Daten$ Sensor auswählen → $Daten$ Anzeige der Daten (3 Personen)
- Sensor → $Daten$ Show oder anklicken (2 Personen)
- Sensor → $Daten$ Aufruf über Rechtsklick (1 Person)

Test:

- Sensoren Sensors → $Daten$ Show Sensors (wird neu Laden erwartet) (4 Personen)

Metadaten/Eigenschaften anzeigen + editieren:

Erwartung:

- Ausgabe im rechten Fenster (3 Personen)
- Ausgabe im linken Fenster (1 Person)

Vermutetes Vorgehen:

- Sensors → $Daten$ Sensor auswählen → $Daten$ Anzeige der Daten im rechten Fenster → $Daten$ Edit (3 Personen)

Test:

- Doppelklick vs. Einfachklick (konsistent umsetzen)
- Map-Darstellung zu klein beim Öffnen und Zoom erwartet (4 Personen)
- bearbeiten der Metadaten über Edit plausibel (4 Personen)
- Fenstereigenschaften der Metadaten (Öffnen und Schließen der Panes) + Edit/Save/Cancel Buttons besser anordnen (3 Personen)

- Mousehover der Metadaten (Werte und Funktion nicht intuitiv ersichtlich) (2 Personen)
- Map zeigt den Standort (Marker) nicht mittig an (Google) (2 Personen)
- die Query ist nicht verständlich muss über Beispiel und Erklärung angegeben werden (1 Person)
- das Edit für alle Eigenschaften gilt war nicht klar (1 Person)
- Breiten- und Längengrad bei Metadaten erwartet (1 Person)
- bei Map nur die Karte (1 Person)
- Metadaten plausibel (1 Person)

Sensor hinzufügen

Erwartung:

- Eingabemaske im rechten Fenster (2 Personen)
- Eingabe der Metadaten eines Sensors (1 Person)

Vermutetes Vorgehen:

- Sensors → Add Sensor (4 Personen)

Test:

- Pflichtdaten nicht ersichtlich (Speichern/Anlegen erst mit Angabe der Pflichtdaten möglich) (4 Personen)
- Kontext der Fields der Metadaten wie z.B. Gewicht des Sensors oder Active und Autostart nicht intuitiv (4 Personen)
- Vorgehen über Sensors → Add Sensors schlüssig (3 Personen)
- Eingabevalidierung intuitiv (3 Personen)
- Eingabe intuitiv (3 Personen)
- Eingabe der Metadaten wie Einheit °C über Dropdown gewünscht (1 Person)
- Aktivierungsdatum auf Zulässigkeit prüfen (1 Person)
- Speicherung nach Enter bei letzem Pflichtparameter nicht erwartet (1 Person)
- Auch Edit für Eingabe? (1 Person)

Aufzeichnungszeitraum der Sensoren:

Erwartung:

- Aus dem Aufzeichnungszeitraum direkt die Daten öffnen zu können (1 Person)

Vermutetes Vorgehen:

- Sensors → Data (4 Personen)

Test:

- Sensors → Doppelklick → Data (3 Personen)
- gutes Mapping mit Balken bei Verfügbarkeit (3 Personen)
- gut das Daten auch aus der Verfügbarkeitsansicht für den Zeitraum angesehen werden können (2 Personen)
- bei Datenverfügbarkeit nur 1/0 erwartet (1 Person)
- gut das Zeitraum von einen Tag auch für Stunden angesehen werden kann (1 Person)
- zurück über Schließen der Taps nicht so gut? (1 Person)
- Timestamp Grafikfehler (Fenster kurz verschieben dann lesbar) (1 Person)

Dashboard

Erstellen:

Erwartung:

- Sensors → Doppelklick → Data (3 Personen)
- gutes Mapping mit Balken bei Verfügbarkeit (3 Personen)
- Daten über Dashboard visualisiert (2 Personen)
- das im Dashboard Daten grafisch angezeigt werden und in verschiedenen Granularitäten darstellbar sind. (1 Person)

Vermutetes Vorgehen:

- Dashboard → new Dashboard (3 Personen)

Test:

- Weg: Dashboard → new Dashboard (2 Personen)
- Sensor → new Dashboard (gut das Sensor dann bereits ausgewählt ist) (2 Personen)
- Doppelklick auf Balken bei Datenverfügbarkeit (1 Person)
- Daten über Datenverfügbarkeit geöffnet (1 Person)
- das Hinzufügen in die leere Liste war über Sensor → Add to Dashboard unklar (besser über Dropdown oder drag und drop) (1 Person)

Angabe der Daten:

- Auswahl durch Eingabe (3 Personen)
- Auswahl durch Kalender (1 Person) → Zeitauswahl zu klein und dunkel
- Granularität per Dropdown gut (4 Personen)
- Start des Dashboards intuitiv über Button (4 Personen)
- Fortschritt über Ladebalken gut (3 Personen)
- Warnmeldung bei großer Granularitätsauflösung gut (2 Personen)

Darstellung Dashboard:

- Darstellung der Messwerte als Diagramm mit Datums- und Zeitangabe gut (3 Personen)
- Konsolenausgabe verständlich Ermittlungszeitraum einer Analyse ermittelt (3 Personen)
- Anpassen der Fenstergröße gut (3 Personen)
- Übersicht der vordefinierten Analysen (MIN, MAX, AVG, Werteanzahl) gut (2 Personen)
- Konsole kann kleiner dargestellt werden besser Grafik größer (1 Person)
- Zoom gewünscht (1 Person)
- Beschriftung an den Achsen der Grafik (1 Person)

Analysetypen anzeigen:

Erwartung:

- Übersicht der verfügbaren Analysen mit Metadaten und Beschreibung (1 Person)

Vermutetes Vorgehen:

- Dashboard → Anzeige Analysen (2 Personen)
- Dashboard → Anzeige über Show (3 Personen)

Test:

- Query nicht intuitiv (2 Personen)
- Bedeutung der Attribute unklar besonders das Parsing über Position (2 Personen)

Mosaik

Erstellen:

Erwartung:

- Initialisierung des Szenarios (Init-Nachricht an Mosaik) → Simulation starten und Daten aufzeichnen (1 Person)

Vermutetes Vorgehen:

- Mosaik → Initialisierung der Simulation → Simulation starten → Daten aufzeichnen (3 Personen)
- Mosaik starten → Verbindung zum Simulator (Adresse, Port) herstellen → Simulation starten (1 Person)

Test:

- Aufruf und Nutzung verständlich (zuerst Verbindung zum Simulator über Initialisierung) → Simulation Starten (4 Personen)

Neue Simulation (Initialisierung):

- Mosaik → neue Simulation (4 Personen)
- Parameterangabe für Initialisierung verständlich (4 Personen)
- Start der Initialisierung über Button verständlich (4 Personen)
- Start der Simulation für Datenaufzeichnung mittels Button Start Simulation verständlich (4 Personen)
- Mosaik Datei (Szenario) ansehen (1 Person)

Anzeige der virtuellen Sensoren:

- Mosaik → neue Simulation (4 Personen)
- Anordnung unter Mosaik-Child nicht plausibel (2 Personen)
- Mosaik Aufbau intuitiv (2 Personen)
- Namen der virtuellen Sensoren nicht gut (kryptische Namen nicht verwenden (1 Person)
- virtuelle Sensoren unter MosaikChild plausibel (1 Person)
- Frage wie Simulation an Odysseus angebunden wird: Mit Init Simulation wird Query in Odysseus erstellt und Sensoren in DB anlegen. (1 Person)

Zusammenfassung

Das primäre Ziel der Funktionstauglichkeit wurde von allen vier Testprobanden bestätigt. Die GUI ist für den Anwendungsfall verwendbar und bedarf keiner funktionalen Änderungen. Die Hälfte der Testpersonen fanden die GUI nach erster Nutzung intuitiv, einer empfand die GUI sofort mittels Ausprobieren intuitiv und einer könnte sich die Nutzung mit einer Hilfe problemlos vorstellen. Alle Kandidaten fanden die GUI übersichtlich und gut strukturiert. In vielen Bereichen wurde die Benutzerinteraktion durch geeignete visuelle Hilfsmittel gut unterstützt und Fehleingaben mit Warnmeldungen und kontrollierten Eingaben abgefangen. Lediglich im Bereich der Visualisierung gab es leichte Kritik bezüglich der Fenstergröße und Design im Untermenü der Metadaten und Eigenschaften, einer nicht ganz konsistenten Umsetzung der Verwendung von Anklicken und Doppelklick sowie des Kontrastes der GUI durch die Verwendung von schwarzem Hintergrund und weißer Schrift.

Zusammenfassend sind alle Anmerkungen, welche von 50 % der Probanden genannt wurden aufgelistet. Diese werden zusammen mit den Auftraggebern auf Umsetzungsrelevanz überprüft. Die Umsetzung etwaiger Veränderungen der GUI wird aufgrund der geringeren Anforderungspriorität an das Ende der Projektaufgaben gestellt.

Allgemein Funktionstauglichkeit:

- reicht für Anwendungsfall aus, keine funktionalen Änderungen nötig (4 Personen)
- nach erster Nutzung ist GUI intuitiv (2 Personen)

Design:

- Schrift/Hintergrund: Kontrast (schwarz auf weiß) nicht so gut (2 Personen)
- linkes Pane muss auch in Breite verstellbar sein (lange Sensornamen) (2 Personen)
- Aufteilung und Logik der Fenster allgemein dokumentieren insbesondere bei Metadaten und Eigenschaften (2 Personen)
- Übersicht gut (2 Personen)

Sensoren:

Sensoren Anzeigen:

- Weg: Sensors → Show Sensors (wird neu Laden erwartet) (4 Personen)

Metadaten/Eigenschaften anzeigen + editieren:

- Weg: Sensors → Doppelklick (3 Personen)
- Map-Darstellung zu klein beim Öffnen und Zoom erwartet (4 Personen)
- Bearbeiten der Metadaten über Edit plausibel (4 Personen)
- Fenstereigenschaften der Metadaten (Öffnen und Schließen der Panes) + Edit/Save/Cancel Buttons besser anordnen (3 Personen)

- Mouseover der Metadaten (Werte und Funktion nicht intuitiv ersichtlich) (2 Personen)
- Map zeigt den Standort(Marker) nicht mittig bei Google Maps an (2 Personen)
- Doppelklick vs. Einfachklick (konsistent umsetzen) (2 Personen)

Sensor hinzufügen:

- Pflichtdaten nicht ersichtlich (Speichern/Anlegen erst mit Angabe der Pflichtdaten möglich) (4 Personen)
- Kontext der Fields der Metadaten wie z.B. Gewicht des Sensors oder Active und Autostart nicht intuitiv (4 Personen)
- Vorgehen über Sensors → Add Sensors schlüssig (3 Personen)
- Eingabevalidierung intuitiv (3 Personen)
- Eingabe intuitiv (3 Personen)

Aufzeichnungszeitraum der Sensoren:

- Weg: Sensors → Doppelklick → Data (3 Personen)
- gutes Mapping mit Balken bei Verfügbarkeit (3 Personen)
- gut das Daten auch aus der Verfügbarkeitsansicht für den Zeitraum angesehen werden können (2 Personen)

Dashboard:

Erstellen:

- Weg: Dashboard → new Dashboard (2 Personen)
- Sensor → new Dashboard (gut das Sensor dann bereits ausgewählt ist) (2 Personen)

Angabe der Eingabedaten:

- Auswahl durch Eingabe (3 Personen)
- Auswahl durch Kalender (1 Person) → Zeitauswahl zu klein und dunkel
- Granularität per Dropdown gut (4 Personen)
- Start des Dashboards intuitiv über Button (4 Personen)
- Fortschritt über Ladebalken gut (3 Personen)
- Warnmeldung bei großer Granularitätsauflösung gut (2 Personen)

Darstellung Dashboard:

- Darstellung der Messwerte als Diagramm mit Datums- und Zeitangabe gut (3 Personen)
- Konsolenausgabe verständlich Ermittlungszeitraum einer Analyse ermittelt (3 Personen)
- Anpassen der Fenstergröße gut (3 Personen)
- Übersicht der vordefinierten Analysen (MIN, MAX, AVG, Werteanzahl) gut (2 Personen)

Darstellung Dashboard:

- Darstellung der Messwerte als Diagramm mit Datums- und Zeitangabe gut (3 Personen)
- Konsolenausgabe verständlich Ermittlungszeitraum einer Analyse ermittelt (3 Personen)
- Anpassen der Fenstergröße gut (3 Personen)
- Übersicht der vordefinierten Analysen (MIN, MAX, AVG, Werteanzahl) gut (2 Personen)

Analysetypen anzeigen:

- Query nicht intuitiv (2 Personen)
- Bedeutung der Attribute unklar besonders das Parsing über Position (2 Personen)

Mosaik:

- Aufruf und Nutzung verständlich (zuerst Verbindung zum Simulator über Initialisierung) $\rightarrow$ Simulation Starten (4 Personen)

Neue Simulation (Initialisierung):

- Mosaik $\rightarrow$ neue Simulation (4 Personen)
- Parameterangabe für Initialisierung verständlich (4 Personen)
- Start der Initialisierung über Button verständlich (4 Personen)
- Start der Simulation für Datenaufzeichnung mittels Button Start Simulation verständlich (4 Personen)

Anzeige der virtuellen Sensoren:

- Anordnung unter Mosaik-Child nicht plausibel (2 Personen)
- Mosaik Aufbau intuitiv (2 Personen)

G. Abschlusstest

UG, LS

Abschlusstest

PG SESAdata

31. März 2016

Inhaltsverzeichnis

Kapitel 1

Anforderungen

In diesem Kapitel werden die funktionalen und nicht-funktionalen Anforderungen an LAOTSE überprüft. Jede Anforderung wird dabei mit geeigneten Testfällen überprüft und die Ergebnisse notiert. Diese Testfälle bilden die Grundlage für die Abnahme der Projektergebnisse durch den Auftraggeber. Für den Fall von Nichterfüllung wird der Grund dafür genannt und ein Ausblick für die zukünftige Umsetzung gegeben. Die Prioritäten geben Auskunft über die Wichtigkeit für das Projekt und deren Umsetzung. Anforderungen der Priorität A müssen im Rahmen des Projektes umgesetzt werden. Die mit der Priorität B sollen nach Möglichkeit umgesetzt werden. Die Berücksichtigung der Anforderungen der Wichtigkeit C wäre wünschenswert, wenn der zeitliche Rahmen dies ermöglicht.

1.1 Funktionale Anforderungen

In diesem Abschnitt sind die Ergebnisse des Abschlusstests bezüglich der Funktionalen Anforderungen an LAOTSE aufgelistet.

ID	**Prio**	**Anforderung**
L-FA1	A	Das System soll die eingehenden Sensordaten (Roh- und Metadaten) persistent speichern können.

Die persistente Speicherung wird für die folgenden Datenquellen mit festgelegten Intervall überprüft:

- Temperatursensor im SESA-lab für 24 Stunden
  Ergebnis: Die Übertragung wurde anhand von Testdaten überprüft.
- AD-Wandler mit Spannungsgenerator (über Raspberry PI)
  Ergebnis: Die Übertragung wurde anhand von Testdaten überprüft.
- Audiosignal über Raspberry PI
  Ergebnis: Die Übertragung wurde anhand von Testdaten überprüft.
- Mosaik Simulation mit bereitgestelltem Szenario für ein Simulationsdurchlauf
  Ergebnis: Die Übertragung wurde anhand von Testdaten überprüft.

- OPC UA ein Datensatz wird über die Testumgebung (da der benötigte Operator nicht zur Verfügung steht)übertragen
  Ergebnis: Die Übertragung wurde anhand von Testdaten überprüft.

Ergebnis: Diese Anforderung ist erfüllt, da die Messwerte mit Zeitstempel in die Langzeitdatenbank übertragen sind.

ID	Prio	Anforderung
L-FA2	A	Das System soll die eingehenden Daten verlustfrei verarbeiten können.

Nach dem Analogieprinzip überprüft für den Audiosensor wird ein konstanter Datenstrom von 16 kHz vorgegeben, welche auch in der Langzeitdatenbank vorhanden ist.
Ergebnis: Diese Anforderung ist erfüllt, da die Daten in der Langzeitdatenbank der Datenrate des Datenstroms entsprechen.

ID	Prio	Anforderung
L-FA3	A	Das System soll den Messzeitpunkt (Timestamp) zu den gespeicherten Daten speichern können.

Die Messzeitpunkte umfassen die Entstehungszeitpunkte der Messungen und werden von den Datenquellen geliefert. Die Anforderung ist erfüllt, wenn zu jedem Messwert ein Zeitstempel in der Datenbank existiert. Die Überprüfung wird anhand der anfänglich festgelegten Testmessungen vorgenommen.
Ergebnis: Diese Anforderung ist erfüllt, da in Druid zu jedem Datenwert ein timestamp gespeichert wird.

ID	Prio	Anforderung
L-FA4	A	Das System soll Sensordaten von Raspberry PIs verarbeiten können.

Die Messwerte des Temperatursensors, des Audiomoduls und des AD-Wandlers werden über Raspberry PIs an das Zielsystem übertragen. Die Anforderung ist mit L-FA2 erfüllt, wenn die Testdaten der drei Quellen verarbeitet werden.
Ergebnis: Diese Anforderung ist erfüllt, da dies beispielhaft am Temperatursensor erfolgreich getestet wurde.

ID	Prio	Anforderung
L-FA5	A	Das System soll die Möglichkeit bieten weitere Quellen, über aktuell verbreitete Standards, hinzufügen zu können.

Ergebnis: Diese Anforderung ist erfüllt, da Odysseus Standards unterstützt und die Möglichkeit bietet diese zu erweitern.

ID	Prio	Anforderung
L-FA6	A	Das System muss Sensordaten aus dem SESA-Lab, die über mosaik bereitgestellt werden, verarbeiten können.

Die Anforderung gilt mit L-FA2 als erfüllt, wenn die Simulationsdaten der Test-

simulation verarbeitet wurden.
Ergebnis: Diese Anforderung ist erfüllt, da die Initialisierung und Simulation von mosaik erfolgreich durchgeführt wurde sowie Daten aus der GUI abrufbar waren, was nur durch eine erfolgreiche Verarbeitung möglich ist.

ID	Prio	Anforderung
L-FA7	A	Das System muss Datenströme mit Hilfe des vorhandenen DSMS Odysseus verarbeiten können.

Die eingehenden Datenströme der Quellen werden mittels Odysseus verarbeitet und an das DBMS übertragen.
Ergebnis: Diese Anforderung ist erfüllt, da die Daten von den Sensoren über Odysseus nach Druid weitergeleitet werden.

ID	Prio	Anforderung
L-FA8	B	Das System soll die Möglichkeit, bieten Systemerweiterungen (Hardware, Software) im laufenden Betrieb ermöglichen zu können.

Das Zielsystem ist modular aufgebaut und unterstützt die Systemerweiterung (HW / SW) zur Laufzeit. Diese Anforderung gilt als erfüllt, wenn L-FA5 (weitere Quellen einbinden) erfüllt ist und weitere Druid-Knoten zur Laufzeit hinzugefügt werden können. Da die Verteilung von Odysseus noch nicht implementiert ist, braucht diese bei dem Test nicht berücksichtigt zu werden.
Ergebnis: Diese Anforderung ist erfüllt, da man neue Sensoren hinzufügen kann (L-FA5), die automatisch eingebunden werden, und es möglich ist in Druid zur Laufzeit neue Knoten hinzuzufügen.

ID	Prio	Anforderung
L-FA9	B	Wenn Datenquellen ausfallen, muss das System fähig sein, unterbrechungsfrei weiterlaufen zu können.

Der Ausfall einzelner Sensoren darf keine Auswirkungen auf das Zielsystem haben. Die Anforderung ist erfüllt, wenn jeder Sensor zur Laufzeit ohne Auswirkungen ausfallen kann.
Ergebnis: Diese Anforderung ist erfüllt, da das System Übertragungsausfälle einzelner Sensoren tolerieren kann.

ID	Prio	Anforderung
L-FA10	B	Wenn ausgefallene Datenquellen wieder verfügbar sind, soll das System diese automatisch wieder einbinden können.

Sobald ausgefallene Quellen wieder verfügbar sind, sollen die Daten automatisch wieder verarbeitet werden. Um diese Anforderung zu erfüllen, muss die Verarbeitung für jeden Quellentyp automatisch erfolgen.
Ergebnis: Diese Anforderung ist nicht erfüllt, da es möglich ist, dass z.B. Odysseus nicht selbstständig die Verbindung zu Datenquellen wieder herstellen kann, sollten diese ausfallen. Es gibt in Odysseus verschiedene Operatoren, die nicht alle eine Rückverbindungsfunktion implementieren.

ID	Prio	Anforderung
L-FA11	B	Das System soll Analysen auf den gespeicherten Langzeitdaten durchführen können.

Das Zielsystem bietet die Möglichkeit Analysen mit Hilfe des Dashboards und manuell erstellte Abfragen an Druid durchzuführen. Die R API ist derzeit aufgrund eines Fehlers im JSON package aufgrund eines Updates nicht funktional. Die Anforderung gilt als erfüllt, wenn L-FA13 und L-FA14 erfüllt sind.

- Ergebnis L-FA13 (vorimplementierte Analysen): Diese Anforderung ist erfüllt, da die Analysen im Dashboard durchgeführt werden.
- Ergebnis L-FA14 (weitere Analysen): Diese Anforderung ist erfüllt, da der Entwickler weitere Analysen hinzufügen kann.

Ergebns: Insgesamt ist diese Anforderung erfüllt.

ID	Prio	Anforderung
L-FA12	B	Das System soll Liveanalysen auf den Datenströmen durchführen können.

Die Anforderung ist erfüllt, wenn das System Liveanalysen der eingehenden Datenströme ermöglicht.
Ergebnis: Diese Anforderung ist erfüllt, da der Odysseus Client diese Funktionalität anbietet.

ID	Prio	Anforderung
L-FA13	B	Das System soll 5 vorimplementierte Standardanalysen, bereitstellen können.

Die Anforderung ist erfüllt, wenn in der GUI mittels des Dashboards für jeden Datenquellentyp die Funktionen (MIN, MAX, AVG, Aggregation und Count) angezeigt werden können.
Ergebnis: Diese Anforderung ist erfüllt, da diese Daten beim Test des Dashboards berechnet werden.

ID	Prio	Anforderung
L-FA14	B	Das System soll um weitere Analysen, durch den Benutzer (Forscher), ergänzt werden können.

Weitere Analysen und Abfragen sind grundsätzlich komfortabel mit der RAPI möglich. Da diese derzeit aufgrund eines Fehlers im JSON package in Folge eines Updates nicht funktional ist, gilt die Anforderung als erfüllt, wenn manuelle Anfragen an Druid möglich sind.
Ergebnis: Diese Anforderung ist erfüllt, da der Entwickler (=Forscher) mit Zugriff auf den Source-Code weitere Analysen hinzufügen kann.

ID	Prio	Anforderung

L-FA15	B	Das System soll die Möglichkeit bieten, Datenströme visualisieren zu können. (Beispielsweise die aktuell erzeugte Leistung eines Windparks.)

Das System bietet die Möglichkeit die eingehenden Datenströme mittels des Odysseus Studio grafisch anzuzeigen. Darüber hinaus können in der Laotse GUI die Ergebnisse im Dashboard, soblad diese in einem der Druid Knoten eingegangen sind. Die Anforderung ist mit L-FA13 (Analysen Dashboard) und L-FA12 (Liveanalysen auf dem Datenstrom) erfüllt.

- Ergebnis L-FA12 (Liveanalysen auf dem Datenstrom): Diese Anforderung ist erfüllt, siehe L-FA12.
- Ergebnis L-FA13 (Analysen Dashboard): Diese Anforderung ist erfüllt, siehe L-FA13.

Ergebnis: Insgesamt ist diese Anforderung erfüllt.

ID	**Prio**	**Anforderung**
L-FA16	B	Das System soll die Möglichkeit bieten, Langzeitdaten visualisieren zu können. (Beispielsweise die erzeuge Leistung des letzten Jahres.)

Die Anforderung ist erfüllt, wenn in der GUI die Daten für einen auswählbaren Zeitraum für die Quellentypen angezeigt werden können.
Ergebnis: Diese Anforderung ist erfüllt, da im Dashboard Langzeitdaten visualisiert werden.

ID	**Prio**	**Anforderung**
L-FA17	B	Das System soll die Möglichkeit bieten, Metadaten visualisieren zu können. (Beispielsweise welche Sensoren aktuell Daten an das System senden.)

Die Anforderung ist erfüllt, wenn die Metadaten in der GUI angezeigt werden können.
Ergebnis: Diese Anforderung ist erfüllt, da in der Sensor-Detailansicht Metadaten angezeigt werden.

ID	**Prio**	**Anforderung**
L-FA18	B	Das System soll die Möglichkeit bieten, Analyseergebnisse aus den Langzeitdaten und Datenströmen, visualisieren zu können. (Z.B. Mittelwert der letzten 3 Wochen)

Die Anforderung ist erfüllt wenn für jeden Quellentyp die Daten für einen beliebigen Zeitraum und Granularitätsstufe in der GUI darstellbar sind.
Ergebnis: Diese Anforderung ist erfüllt, da Analysen auf Langzeitdaten im Dashboard angezeigt werden und Analysen auf Datenströmen im Odysseus Studio ausgeführt und visualisiert werden können.

ID	**Prio**	**Anforderung**

L-FA19	B	Das System soll die Möglichkeit bieten, Visualisierungen ein- und ausblenden zu können.

Ergebnis: Diese Anforderung ist erfüllt, da mit dem Tabsystem der GUI verschiedene Dashboards ein- und ausgeblendet werden können.

ID	Prio	Anforderung
L-FA20	B	Das System soll um weitere Visualisierungen, durch den Benutzer (Forscher), ergänzt werden können.

Weitere Funktionen können in das Zielsystem für die Visualisierung integriert werden. Das Vorgehen wird im HowTo beschrieben. Die Anforderung ist erfüllt, wenn mit Hilfe der Beschreibung eine Erweiterung umgesetzt werden kann.
Ergebnis: Diese Anforderung ist erfüllt, da im HowTo eine Anleitung zum Hinzufügen von Visualisierungen vorhanden ist.

ID	Prio	Anforderung
L-FA21	C	Das System soll die Daten, des über Fernwirkprotokolle angeschlossenen BHKW-Containers in der Lesumstraße in Oldenburg, verarbeiten können. (Wenn dieser nicht rechtzeitig in Betrieb genommen wird, erfolgt die Einbindung eines bereit gestellten Dummy-BHKW's.)

Das Zielsystem soll die Messwerte Leistung und Temperatur verarbeiten können.
Ergebnis: Die Anforderung konnte nicht umgesetzt werden, da die benötigten Ressourcen (Anbindung BHKW-Container, oder Simulation mittels SPS) dem Projektteam nicht bereitgestellt wurde. Die Anforderung wurde am 20.01.2016 einvernehmlich mit den Auftraggebern gestrichen.

ID	Prio	Anforderung
L-FA22	C	Das System soll den Ausfall von Sensoren melden können. (Z.B. als visuelle Hervorhebung oder Nachricht)

Das Zielsystem soll die Unterbrechung des Datenstroms der Quellen überwachen und in einem Dashboard oder mit einer Warnmeldung mitteilen. Die Anforderung ist erfüllt, wenn das für jeden Quellentyp funktioniert.
Ergebnis: Diese Anforderung ist nicht erfüllt, da die Umsetzung des Messagingservices in Absprache mit den Auftraggebern aus Prioritäts-, Umfangs- und Kapazitätsgründen nicht mehr umgesetzt werden konnte. Die Herangehensweise und Motivation wird im Ausblick veranschaulicht.

ID	Prio	Anforderung
L-FA23	C	Das System soll Meldungen unterschrittener Schwellwerte ausgeben können. (Z.B. als visuelle Hervorhebung oder Nachricht)

Das System soll die Angabe und Überwachung von Schwellwerten für jeden Quellentyp ermöglichen. Das Unterschreiten ist fortlaufend zu überprüfen und mit visueller Hervorhebung oder Nachrichten mitzuteilen.

Ergebnis: Diese Anforderung ist nicht erfüllt, da die Umsetzung der Definition und Überwachung von Schwellwerten in Absprache mit den Auftraggebern aus Prioritäts-, Umfangs- und Kapazitätsgründen nicht mehr umgesetzt werden konnte. Die Herangehensweise und Motivation wird im Ausblick veranschaulicht.

ID	Prio	Anforderung
L-FA24	C	Das System soll um weitere Meldungen erweitert werden können.

Diese Anforderung ist erfüllt, wenn für jeden Quellentyp ein Auslöseereignis mit Benachrichtigung (visuell oder per Nachricht) hinzugefügt werden kann. Das Vorgehen wird im How-to beschrieben.
Ergebnis: Diese Anforderung ist nicht erfüllt, da die Umsetzung des Hinzufügens von Auslöseereignissen mit Benachrichtigung in Absprache mit den Auftraggebern aus Prioritäts-, Umfangs- und Kapazitätsgründen nicht mehr umgesetzt werden konnte. Die Herangehensweise und Motivation wird im Ausblick veranschaulicht.

ID	Prio	Anforderung
L-FA25	C	Das System soll die zu speichernden Daten komprimieren können.

Das DBMS des Zielsystems soll die Daten, im Hinblick auf die langfristige Speicherung, nach Möglichkeit komprimieren. Die Anforderung ist erfüllt, wenn die Herstellerangaben des DBMS die Komprimierung verlässlich angeben oder durch einen Größenvergleich ersichtlich wird.
Ergebnis: Diese Anforderung ist erfüllt, da Druid gemäß verlässlicher Herstellerdokumentation ein komprimiertes Speicherverfahren verwendet.

1.2 Nicht-funktionale Anforderungen

Dieses Kapitel beschreibt die Ergebnisse des Abschlusstests bezüglich der Nichtfunktionalen Anforderungen an Laotse.

1.2.1 Projektanforderungen / Anforderungen an Durchführung und Entwicklung

ID	Prio	Kategorie	Anforderung
NFA1	A	Budget	Wenn Investitionen zur Realisierung des Projektes erforderlich sind, muss das Projektteam Angebote zur Auswahl bereitstellen.

Die Anforderung ist erfüllt, wenn keine Anschaffung ohne Angebot und Genehmigung durch den Auftraggeber getätigt wurde.
Ergebnis: Diese Anforderung ist erfüllt, da entsprechende Auswahldokumente erstellt wurden.

ID	Prio	Kategorie	Anforderung
NFA2	A	Lieferumfang und Termine	Das Projektteam muss nach der Hälfte der Projektzeit einen Zwischenbericht, welcher neben dem Grobkonzept, das Feinkonzept enthält, vorlegen.

Die Anforderung ist erfüllt, wenn der Zwischenbericht inhaltlich vollständig und termingerecht abgegeben wurde.
Ergebnis: Diese Anforderung ist erfüllt, da der Zwischenbericht termingerecht am 31.10.2015 abgegeben wurde.

ID	Prio	Kategorie	Anforderung
NFA3	A	Lieferumfang und Termine	Das Projektteam soll nach der Hälfte der Projektzeit einen Prototyp bereitstellen.

Die Anforderung ist erfüllt, wenn zur Projekthälfte ein funktionierender Prototyp zum derzeitigen Entwicklungsstand vorgeführt wird.
Ergebnis: Diese Anforderung ist erfüllt, da der Prototyp zur Zwischenpräsentation vorgestellt wurde.

ID	Prio	Kategorie	Anforderung
NFA4	A	Lieferumfang und Termine	Das Projektteam soll am 31.03.2016 den Abschlussbericht, welcher neben dem Zwischenbericht, das Feinkonzept, ein Installations- sowie Benutzerhandbuch enthält, bereitstellen.

Die Anforderung ist erfüllt, wenn alle geforderten Dokumente termingerecht zum 31.03.2016 abgegeben werden.
Ergebnis: Diese Anforderung ist erfüllt, da der Abschlussbericht mit allen geforderten Inhalten termingerecht am 31.03.2016 abgegeben wurde.

ID	Prio	Kategorie	Anforderung
NFA5	C	Lieferumfang und Termine	Bei allen Dokumenten soll nach Möglichkeit der Autor hervorgehen.

Die Anforderung ist erfüllt, wenn der Autor aus allen Dokumenten hervorgeht.
Ergebnis: Diese Anforderung wurde im Einvernehmen mit allen Projektbeteiligten gestrichen, da die kollaborative Zusammenarbeit zwischen allen Projektmitgliedern reibungslos funktioniert und so auf dieses Kontrollinstrument verzichtet wird. Im Abschlussbericht wurden allerdings zur besseren Bewertung dennoch Autorenangaben eingefügt.

ID	Prio	Kategorie	Anforderung

NFA6	A	Lieferumfang und Termine	Das Projektteam führt monatliche Reviews mit dem Auftraggeber durch.

Die Anforderung ist erfüllt, wenn zu jedem Sprint ein Review mit den Auftraggebern durchgeführt wurde.
Ergebnis: Diese Anforderung ist erfüllt, da die Reviews wöchentlich zur Zufriedenheit der Auftraggeber durchgeführt wurden.

ID	Prio	Kategorie	Anforderung
NFA7	B	Testkonzept	Das Testkonzept soll von Beginn an erstellt und angemessen dokumentiert werden.

Die Anforderung ist erfüllt, wenn die Testkonzepte in angemessenem Umfang zu den jeweiligen Entwicklungsschritten erstellt, dokumentiert und von den Auftraggebern abgenommen wurden.
Ergebnis: Diese Anforderung ist erfüllt, da eine solche Dokumentation im Abschlussbericht vorliegt.

ID	Prio	Kategorie	Anforderung
NFA8	B	Testkonzept	Das Testkonzept kann bei Bedarf und in Absprache mit den Betreuern erweitert werden.

Die Anforderung gilt als erfüllt, wenn die Testkonzepte gemäß Absprachen mit den Betreuern erweitert wurden.
Ergebnis: Diese Anforderung ist erfüllt, da dies mit den Sitzungsprotokollen gezeigt werden kann.

ID	Prio	Kategorie	Anforderung
NFA9	B	Evaluierung	Systemkomponenten sollen mit umfangreicher Alternativenbetrachtung, Begründung und Dokumentation ausgewählt werden.

Die Anforderung ist erfüllt, wenn die Komponentenauswahl unter Berücksichtigung ausreichender Alternativen und Testvorgehen zur Zufriedenheit der Auftraggeber ausgewählt wurden.
Ergebnis: Diese Anforderung ist erfüllt, da dies mit den im Abschlussbericht enthaltenen Hardwareauswahldokumenten erbracht wurde.

ID	Prio	Kategorie	Anforderung
NFA10	A	Doku	Die Dokumentation muss fortlaufend und präzise während der gesamten Projektzeit erfolgen.

Die Anforderung ist erfüllt, wenn die Dokumentation zur Zufriedenheit der Auftraggeber erstellt und abgenommen wurde.
Ergebnis: austehend.

ID	Prio	Kategorie	Anforderung

NFA11	A	Handbücher	Die Handbücher müssen für Fachleute und in der deutschen Sprache erstellt werden.

Um die Anforderung zu erfüllen, sind die Handbücher von den Auftraggebern abzunehmen.
Ergebnis: Diese Anforderung ist erfüllt, da alle Handbücher in der deutschen Sprache erstellt wurden.

1.2.2 Rechtliche Anforderungen

Die folgende Tabelle zeigt die rechtlichen Anforderungen.

ID	Prio	Kategorie	Anforderung
NFA12	A	Datenschutz	Die bereitgestellten Daten und Systeme müssen vertraulich, ohne absichtliche Weitergabe an Außenstehende, behandelt werden.

Um die Anforderung zu erfüllen, sind die Datenschutzrichtlinien des OFFIS zu beachten. Die Anforderung ist erfüllt, solange keine andauernde Missachtung dieser Vorgaben vorliegt.
Ergebnis: Diese Anforderung ist erfüllt, da alle Datenschutzrichtlinien beachtet wurden.

ID	Prio	Kategorie	Anforderung
NFA13	B	Rechtliches	Urheber- und Lizenzrechte müssen gewahrt werden.

Die Einhaltung der Urheber- und Lizenzrechte ist verbindlich. Um die Anforderung zu erfüllen, dürfen in Abgegebenen Software keine Verletzungen vorliegen.
Ergebnis: Diese Anforderung ist erfüllt, da evtl. Quellen angegeben wurden.

ID	Prio	Kategorie	Anforderung
NFA14	B	Rechtliches	Anforderungsänderungen können nur durch Zustimmung des Auftraggebers durchgeführt werden.

Die Anforderung ist erfüllt, wenn das Anforderungsmanagement immer in Absprache mit den Auftraggebern vorgenommen wurde.
Ergebnis: Diese Anforderung ist erfüllt, da diese Zustimmung in den wöchentlichen Meetings gegeben wurden.

ID	Prio	Kategorie	Anforderung
NFA15	B	Rechtliches	Weitere rechtliche Rahmenbedingungen müssen nicht beachtet werden.

Die anforderung ist erfüllt, wenn die betreuer keine weiteren rahmenbedingungen fordern.
Ergebnis: Diese Anforderung ist erfüllt, da keine weiteren Rahmenbedingungen gefordert wurden.

1.2.3 Technische Anforderungen

Die folgende Tabelle zeigt die technischen Anforderungen.

ID	Prio	Kategorie	Anforderung
NFA16	A	Technologie	Die Technologie kann grundsätzlich frei gewählt werden, außer Vorgaben oder vorhandene Systeme widersprechen dem.

Die Anforderung ist erfüllt, wenn die verwendeten Technologien unter Beachtung der Vorgaben und der vorhandenen Systeme ausgewählt wurden.
Ergebnis: Diese Anforderung ist erfüllt, da die Technologien entsprechend ausgewählt wurden, siehe Abschlussdokumentation (Technologieauswahl).

ID	Prio	Kategorie	Anforderung
NFA17	B	Technologie	Die Technologieauswahl muss immer ordnungsgemäß evaluiert und dokumentiert werden.

Um die Anforderung zu erfüllen, ist jede Auswahlentscheidung ausreichend zu evaluieren, zu dokumentieren und vom Auftraggeber abzunehmen.
Ergebnis: Diese Anforderung ist erfüllt, da die Technologien entsprechend evaluiert wurden, siehe Abschlussdokumentation (Technologieauswahl).

ID	Prio	Kategorie	Anforderung
NFA18	A	Technologie	Die Technologie soll unter Berücksichtigung der Ausbaufähigkeit und des langfristigen Einsatzes ausgewählt werden.

Die Anforderung ist erfüllt, wenn die Ausbaufähigkeit und langfristige Verwendung der ausgewählten Komponenten sichergestellt ist.
Ergebnis: Diese Anforderung ist erfüllt, da die Technologie entsprechend ausgewählt wurde, siehe Abschlussdokumentation (Technologieauswahl).

ID	Prio	Kategorie	Anforderung
NFA19	B	System	Raspberry PIs sollen als Hardwareaufbau realisiert und in das System eingebunden werden.

Die Anforderung ist erfüllt, wenn die Raspberry PIs für den Anschluss der Sensoren im SESA-Lab eingebunden werden.
Ergebnis: Diese Anforderung ist erfüllt, da Raspberry PIs im SESAlab eingebaut wurden.

ID	Prio	Kategorie	Anforderung
NFA20	A	System	Die Systemkomponenten des DSMS sollen auf bereitgestellten VM-Instanzen realisiert werden.

Die Anforderung ist erfüllt, wenn das Zielsystem auf VM-Instanzen realisierbar ist.
Ergebnis: Diese Anforderung ist erfüllt, da das System auf solchen VM-Instanzen realisiert wurde.

ID	Prio	Kategorie	Anforderung
NFA21	B	Schnitt-stellen	Wenn Standards für die Schnittstellen vorhanden sind, sollen diese nach Möglichkeit verwendet werden.

Diese Anforderung gilt als erfüllt, wenn vorhandene Standards in Absprache mit den Auftraggebern verwendet werden.
Ergebnis: Diese Anforderung ist erfüllt, da Standards nach Absprache verwendet wurden, z.B. OPC UA.

ID	Prio	Kategorie	Anforderung
NFA22	A	Schnitt-stellen	Wenn Schnittstellen bereitgestellt werden, muss deren Spezifikation und Beschreibung erstellt werden.

Diese Anforderung ist erfüllt, wenn diese Schnittstellen spezifiziert und deren Verwendung ausreichend beschrieben ist und von den Auftraggebern abgenommen wird.
Ergebnis: Diese Anforderung ist erfüllt, da nach Absprache mit den Betreuern z.B: OPC UA verwendet wurde.

ID	Prio	Kategorie	Anforderung
NFA23	C	Programmier-sprache	Die Programmiersprache kann frei gewählt werden.

Diese Anforderung ist erfüllt, wenn die Auswahl bedacht und begründet erfolgt.
Ergebnis: Diese Anforderung ist erfüllt, da die Auswahl von Java in der Abschlussdokumentation begründet wurde.

ID	Prio	Kategorie	Anforderung
NFA24	B	Programmier-sprache	Codekonventionen sollen eingehalten werden.

Für die Erfüllung der Anforderung ist ein geeigneter Codekonventionstandard konsistent zu verwenden.
Ergebnis: Diese Anforderung ist erfüllt, da die Google-Codekonventionen mithilfe von Checkstyle in den bearbeiteten Projekten angewandt und beachtet wurden.

ID	Prio	Kategorie	Anforderung
NFA25	C	Speicher-kapazität	Die Speicherkapazität für die Langzeitspeicherung steht aufgrund der immer weiter sinkenden Kosten ausreichend zur Verfügung.

Ergebnis: Diese Anforderung ist erfüllt, da von den Betreuern immer ausreichend Speicher zur Verfügung gestellt wurde.

ID	Prio	Kategorie	Anforderung

NFA26	B	Speicher-kapazität	Wenn eine Kompression möglich ist, soll diese aus Performanz- und Speicherauslastungsgründen implementiert werden,

Die Anforderung ist erfüllt, wenn die Kompressionsmöglichkeit der verwendeten DBMS Komponenten umgesetzt werden.
Ergebnis: Diese Anforderung ist erfüllt, da die von Druid bereitgestellte Kompression verwendet wird, um die Datenmenge der Langzeitdaten möglichst klein zu halten.

ID	Prio	Kategorie	Anforderung
NFA27	C	Hardware- und Softwarekomponenten	Das Projektteam muss den Ausfall einzelner Systemkomponenten wie DB-Ausfälle oder sonstige Katastrophen nicht berücksichtigen.

Ergebnis: Diese Anforderung ist erfüllt, da keine katastrophensicherung eingebaut wurde, wie zu sehen im Laotse Quelltext.

ID	Prio	Kategorie	Anforderung
NFA28	B	Hardware- und Softwarekomponenten	Das System soll nach Möglichkeit modular aufgebaut werden, um System- und Kapazitätserweiterungen zur Laufzeit zu ermöglichen.

Diese Anforderung gilt als erfüllt, wenn die Hauptkomponenten des Zielsystems DSMS, DBMS und die Laotse-Komponenten modular aufgebaut sind und somit zur Laufzeit erweitert werden können.
Ergebnis: Diese Anforderung ist erfüllt, da OSGi-Bundles per Definition Module sind und Druid aus verschiedenen Knoten besteht.

ID	Prio	Kategorie	Anforderung
NFA29	A	Hardware- und Softwarekomponenten	Das System soll die Möglichkeit bieten Daten, von bis zu 1000 Sensoren, zu verarbeiten.

Um diese Anforderung zu erfüllen, muss das Zielsystem erweiter- und verteilbar sein. Dem Auftraggeber wird ein Verteilungs- und Erweiterungskonzept vorgelegt, indem die Ausbaufähigkeit erläutert wird.
Ergebnis: Diese Anforderung ist erfüllt, da die Durchführung einer Erweiterung im Entwicklerhandbuch geschildert wird.

ID	Prio	Kategorie	Anforderung
NFA30	A	Hardware- und Softwarekomponenten	Das System soll fähig sein hochfrequente Daten, von bis zu 30 kHz, zu verarbeiten.

Die Anforderung gilt als erfüllt, wenn hochfrequente Daten bis zu 30 kHz vom Zielsystem verarbeitet werden können. Sollte keine geeignete Datenquelle zur Verfügung stehen, ist eine vergleichbare Alternative mit möglichst hoher Übertragungsfrequenz zu nutzen.
Ergebnis: Diese Anforderung ist erfüllt, da das System, mit ausreichend Verarbeitungszeit, in der Lage ist solch hochfrequente Daten abzuarbeiten.

1.2.4 Qualitätsanforderungen

Die folgende Tabelle zeigt die Qualitätsanforderungen.

ID	Prio	Kategorie	Anforderung
NFA31	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb hinzugefügt werden können.

Diese Anforderung ist erfüllt, wenn zur Laufzeit neue Sensoren z.B. mit der GUI hinzugefügt werden können, sowie deren gesendete Daten in die Langzeitdatenbank überführt werden.
Ergebnis: Diese Anforderung ist erfüllt, da das Hinzufügen von Sensoren mit der GUI getestet wurde und deren Messwerte im Dashboard angezeigt wurden, was durch das Vorhandensein dieser Daten in der Langzeitdatenbank zu erklären ist.

ID	Prio	Kategorie	Anforderung
NFA32	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb modifiziert werden können.

Diese Anforderung gilt als erfüllt, wenn die Metadaten der Sensoren zur Laufzeit verändert werden können.
Ergebnis: Diese Anforderung ist erfüllt, da das Editieren von Sensoren z.B. mit der GUI funktioniert.

ID	Prio	Kategorie	Anforderung
NFA33	A	Verfügbarkeit	Sensoren müssen im laufenden Betrieb deaktiviert werden können.

Diese Anforderung ist erfüllt, wenn die Sensoren zur Laufzeit deaktivierbar sind.
Ergebnis: Diese Anforderung ist erfüllt, da z.B. mit der GUI der Zustand eines Sensors auf `active` gesetzt werden kann.

ID	Prio	Kategorie	Anforderung
NFA34	C	Verfügbarkeit	Analysen sollen im laufenden Betrieb möglich sein.

Ergebnis: Diese Anforderung ist erfüllt, da Analysen auf dem Druid-Realtime Knoten via Dashboard möglich sind, kurz nachdem die Daten vom Sensor empfangen wurden.

ID	Prio	Kategorie	Anforderung

NFA35	C	Verfügbarkeit	Visualisierungen sollen im laufenden Betrieb möglich sein.

Ergebnis: Diese Anforderung ist erfüllt, da das Dashboard Analyseergebnisse und die Sensor-Detailansicht der GUI z.B. den Standort des Sensors visualisieren.

ID	Prio	Kategorie	Anforderung
NFA36	C	Verfügbarkeit	Kapazität- und Systemerweiterungen sollen im laufenden Betrieb möglich sein.

Die Anforderung ist erfüllt, wenn die Hauptkomponenten DSMS und DBMS zur Laufzeit erweiterbar sind.
Ergebnis: Diese Anforderung ist erfüllt, da Druid zur Laufzeit um zusätzliche Knoten erweiterbar ist.

ID	Prio	Kategorie	Anforderung
NFA37	C	Bedienbarkeit	Das System soll von Fachleuten bedient werden können.

Die Anforderung ist mit Abnahme des Usability-Tests und der ggf. resultierenden Änderungen erfüllt.
Ergebnis: Diese Anforderung ist erfüllt, da alle Teilnehmer des Usabilitytests, die Bedienbarkeit als mindestens ausreichend eingestuft haben (siehe das Ergebnisdokument vom Usabilitytest).

ID	Prio	Kategorie	Anforderung
NFA38	C	Bedienbarkeit	Das System soll möglichst intuitiv bedienbar sein.

Die Anforderung ist mit Abnahme des Usability-Tests und der ggf. resultierenden Änderungen erfüllt.
Ergebnis: Diese Anforderung ist erfüllt, da alle Teilnehmer die GUI als ausreichend intuitiv eingestuft haben (siehe das Ergebnisdokument vom Usabilitytest).

ID	Prio	Kategorie	Anforderung
NFA39	B	Bedienbarkeit	Analysen sollen durch den Anwender auswählbar sein.

Die Anforderung ist erfüllt, wenn die Analysen für jeden Quellentyp in der GUI ausgewählt werden können.
Ergebnis: Diese Anforderung ist erfüllt, da alle Analysen mit der Auswahl eines Dashboards für Sensoren ausgeführt werden.

ID	Prio	Kategorie	Anforderung
NFA40	B	Bedienbarkeit	Visualisierungen sollen durch den Anwender auswählbar sein.

Die Anforderung ist erfüllt, wenn die Ergebnisse der Analysen für jeden Quellentyp in der GUI angezeigt werden können.
Ergebnis: Diese Anforderung ist erfüllt, da es keiner speziellen Berechtigungen oder Fähigkeiten bedarf, um Visualisierungen durchzuführen und diese mit dem Dashboard möglich sind.

ID	Prio	Kategorie	Anforderung
NFA41	C	Bedienbarkeit	Die GUI kann frei gestaltet werden.

Die Anforderung ist erfüllt, wenn die Gebrauchstauglichkeit der GUI durch den Usability-Test vom Auftraggeber abgenommen wird.
Ergebnis: Diese Anforderung ist erfüllt, da die GUI durch den Usabilitytest abgenommen wurde (siehe Ergebnisdokument Usabilitytest).

ID	Prio	Kategorie	Anforderung
NFA42	A	Bedienbarkeit	Quellen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.

Die Anforderung ist erfüllt, wenn die Quellen wie im Handbuch beschrieben, über die GUI hinzugefügt werden können.
Ergebnis: Diese Anforderung ist erfüllt, da das Hinzufügen von Quellen mit der GUI, wie im Benutzerhandbuch beschrieben, möglich ist.

ID	Prio	Kategorie	Anforderung
NFA43	C	Bedienbarkeit	Analysen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.

Die Anforderung ist erfüllt, wenn weitere Analysen wie im Handbuch beschrieben, hinzugefügt werden können.
Ergebnis: Diese Anforderung ist erfüllt, da das Hinzufügen von Analysen im Quellcode, wie im Entwicklerhandbuch beschrieben, möglich ist.

ID	Prio	Kategorie	Anforderung
NFA44	C	Bedienbarkeit	Visualisierungen sollen, wie in einem Handbuch beschrieben, hinzugefügt werden können.

Die Anforderung ist erfüllt, wenn weitere Visualisierungen, wie im Handbuch beschrieben hinzugefügt werden können.
Ergebnis: Diese Anforderung ist erfüllt, da das Hinzufügen von Visualisierungen im Quellcode, wie im Entwicklerhandbuch beschrieben, möglich ist.

ID	Prio	Kategorie	Anforderung
NFA45	B	Performanz	Das System soll auf Anfragen, die einen Zeitraum von maximal 2 Jahren umfassen, innerhalb einer Stunde das Ergebnis liefern können.

Die Anforderung ist erfüllt, wenn für alle vorimplementierten Analysen die Ergebnisse, über einen Betrachtungszeitraum von zwei Jahren, innerhalb von einer Stunde ausgegeben werden.

Ergebnis: Diese Anforderung ist erfüllt, da dies mit Anfragen, bei denen eine entsprechende Granularität oder Threshold gewählt wurde, möglich ist.

H. Benutzerhandbuch

Benutzerhandbuch zu LAOTSE

PG SESAdata

31. März 2016

Inhaltsverzeichnis

Kapitel 1

Einrichtung

LS, DN

Laotse besteht aus einer Vielzahl von Programmen, die installiert, konfiguriert und gestartet werden müssen. Um diese Arbeit zu erleichtern, wurden Installationsskripte, angepasste Konfigurationsdateien und Startskripte entwickelt, die wiederverwendet werden können. In den nächsten Abschnitten werden diese Hilfsmittel erläutert.

1.1 Installationsskript

LS, DN

Dieser Abschnitt beschreibt das Vorgehen zur Installation von LAOTSE auf einem Debian Linux System, für das vorbereitete Installationsskripte zur Verfügung stehen.

1. **Vorbereitung** Das Skript *install_as_root.sh* installiert von LAOTSE benötigte Drittprogramme und erstellt einen eigenes Nutzerkonto. Da es mit dem System-Paketmanager Programme hinzufügt, sind für die Ausführung dieses Skriptes zwingend Root-Rechte erforderlich.

   Nach der Ausführung sollte folgendes passiert sein:

   - Die Kommandozeilentools *unzip* und *screen* sind installiert.
   - Die Java JDK 8_74 inklusive JVM wurde von der Oracle Website heruntergeladen und installiert.
   - Das systemeigene Kommandozeilenkommando *java* ruft das soeben installierte Java-Executable aus dem *jvm* Ordner auf.
   - MySQL wurde installiert.
   - Der User *standard* wurde im Betriebssystem erstellt.

   Das Skript kann in einem beliebigen Ordner ausgeführt werden.

2. **Installation** Das Skript *install.sh* kann von einem Nutzer ohne Root-Rechte ausgeführt werden und lädt sowie platziert alle Systembestandteile zur späteren Ausführung.

Nach der Ausführung sollte folgendes passiert sein:

- Im HOME-Verzeichnis wurde ein Ordner *Druid* erstellt.
- Druid, Kafka, Zookeeper, Cassandra, Odysseus(Server) und Laotse-Server wurden in diesen Ordner heruntergeladen und entpackt.
- Im HOME-Verzeichnis wurde der Ordner *init* erstellt.
- In *init* befinden sich 13 Shellskripte.

Das Skript kann in einem beliebigen Ordner ausgeführt werden. Das Installationsskript beinhaltet die Verlinkungen zu den Programmen, die zum Abschluss der Projektgruppe zur Verfügung standen. Wenn in Zukunft neuere Versionen installiert werden sollen, ist das Installationsskript anzupassen. Wenn die Anwendungen auf unterschiedliche Rechner verteilt werden sollen, können einzelne Programme installiert werden, indem die entsprechenden Zeilen aus dem Installationsskript entnommen und ausgeführt werden.

Nun lässt sich Laotse mithilfe der Skripte in *init* starten.

1.2 Konfiguration

Nach der Installation der benötigten Anwendungen müssen diese konfiguriert werden. In diesem Abschnitt werden die Einstellungen beschrieben, die für jede Installation angepasst werden müssen, damit das Cluster lauffähig ist. Für weitergehende Informationen zur Konfiguration ist die Dokumentation der jeweiligen Anwendungen zu beachten. Mit Laotse werden angepasste Konfigurationsdateien ausgeliefert, die sich für den Einsatz in Laotse bewehrt haben. Sie sind im Ordner *pgrepo/trunk/configs* abgelegt. Zum Teil enthalten die Startskripte weitere Konfigurationen. Sollen die Anwendungen auf eine andere Art gestartet werden, sollten die Parameter in den Startskripten beachtet werden.

1.2.1 Druid

DN

Die Konfiguration von Druid ist in mehrere Dateien aufgeteilt. Im Ordner *conf/_common* liegt eine Datei, die Einstellungen definiert, die im Cluster einheitlich sein sollten. Jeder Druid-Knoten erhält in anderen Unterordnern von *conf/* weitere spezifische Einstellungen. Neben den Einstellungen muss eine Spezifikation für eine Datenquelle vorliegen. Die zu konfigurierenden Dateien und die sogenannte Spec-Datei werden im Folgenden erläutert.

Konfigurationsdateien

Im folgenden wird eine Liste der zu konfigurierenden Einstellungen mit der Zuordnung zu den jeweiligen Druid-Knoten aufgelistet.

_common

druid.zk.service.host: Der Hostname des zu verwendenden Zookeeper-Servers

druid.metadata.storage.connector.connectURI: Adresse von MySQL

druid.metadata.storage.connector.user: Benutzername MySQL

druid.metadata.storage.connector.password: Passwort MySQL

druid.storage.host: HostName und Port von Cassandra RPC

druid.storage.keyspace: Name des angelegten Keyspace in Cassandra

broker

druid.host: Hostname, an den sich der Knoten bindet

druid.port: Port, an den sich der Knoten bindet

druid.processing.buffer.sizeBytes: Größe des Puffers für Zwischenergebnisse

druid.processing.numThreads: Anzahl der Kerne zur Abfrageverarbeitung

coordinator

druid.host: Hostname, an den sich der Knoten bindet

druid.port: Port, an den sich der Knoten bindet

historical

druid.host: Hostname, an den sich der Knoten bindet

druid.port: Port, an den sich der Knoten bindet

druid.processing.buffer.sizeBytes: Größe des Puffers für Zwischenergebnisse

druid.processing.numThreads. Anzahl der Kerne zur Abfrageverarbeitung

druid.segmentCache.locations: Pfade und Kapazität zu Ordnern, in denen Segmente abgelegt werden

druid.server.maxSize: Maximalgröße an Segmenten, die der Knoten laden darf

realtime

druid.host: Hostname, an den sich der Knoten bindet

druid.port: Port, an den sich der Knoten bindet

druid.processing.buffer.sizeBytes: Größe des Puffers für Zwischenergebnisse

druid.processing.numThreads: Anzahl der Kerne zur Abfrageverarbeitung

Spec-Datei

Die Datei *laotse_realtime.spec* wird im Ordner /Druid/druid_spec erwartet. Sie enthält Informationen über die Druid-DataSource, die für LAOTSE in Druid angelegt werden muss. Listing 1.1 zeigt einen Ausschnitt der Datei. Wichtig ist die Konfiguration der Zookeeper-Adresse in Zeile 48. Unter dieser Adresse muss das von Kafka verwendete Zookeeper-Cluster erreichbar sein. In Zeile 61 wird der Pfad angegeben, unter dem Segmente abgelegt werden, bis sie von einem Historical-Knoten geladen wurden.

Listing 1.1: Auszug *laotse_realtime.spec*

```
    "granularitySpec": {
38    "type": "uniform",
      "segmentGranularity": "FIVE_MINUTE",
40    "queryGranularity": "NONE"
    }
42  },
  "ioConfig": {
44    "type": "realtime",
    "firehose": {
46        "type": "kafka-0.8",
        "consumerProps": {
48        "zookeeper.connect": "172.20.50.93:2181",
          "group.id": "laotse"
50      },
        "feed": "laotse"
52    },
    "plumber": {
54      "type": "realtime"
    }
56  },
  "tuningConfig": {
58    "type": "realtime",
    "maxRowsInMemory": 500000,
60    "windowPeriod": "PT10m",
    "basePersistDirectory": "\/tmp\/realtime\/
        basePersist",
62    "rejectionPolicy": {
      "type": "serverTime"
64    }
  }
66 }]
```

1.2.2 Zookeeper

KB

Zookeeper wird zur Synchronisation von Kafka und Druid benötigt. Wurde Zookeeper installiert, muss an den Einstellungen nichts weiter vorgenommen werden. Die Standardkonfiguration von Zookeeper ist ausreichend für den Anwendungsfall des Systems.

1.2.3 Kafka

KB

Das Messaging-System Kafka wird in Laotse dazu genutzt, die Daten des Datenstroms an Druid weiterzugeben, damit diese dort gespeichert werden können. Damit Kafka funktioniert, müssen zunächst Einstellungen vorgenommen werden. Um die Einstellungen anzupassen müssen die Einstellungen unter *conf/server.properties* angepasst werden. Hier müssen für eine korrekte Arbeitsweise folgende Parameter angepasst werden.

```
1 host.name: [hostName:port]
  logs.dir: [Pfad]
3 zookeeper.connect: [Hostname und Port von Zookeeper]
```

Es muss angegeben werden, auf welchem Server und unter welchem Port Kafka erreichbar ist. Diese Einstellung wird mittels `host.name` vorgenommen. Außerdem muss definiert werden, unter welchem Pfad Kafka die Logs schreiben soll. Diese Einstellung kann mittels `logs.dir` vorgenommen werden. Wird der Standardwert unter */tmp/* verwendet, werden die Logs bei einem Neustart der Server-Maschine gelöscht, wenn dieser nicht anders konfiguriert wurde. Außerdem muss Kafka wissen, wo Zookeeper liegt, da mit Hilfe von Zookeeper die Kommunikation in Druid geregelt wird. Mit Hilfe der Einstellung `zookeeper.connect` kann angegeben werden, auf welchem Server sich Zookeeper befindet und unter welchem Port es erreichbar ist. Wurden diese Einstellungen gesetzt, sollte Kafka korrekt arbeiten können und damit Daten in Druid ankommen.

1.2.4 Cassandra

KB

Der NoSQL-Datenspeicher Cassandra wird als Deep Storage von Druid genutzt. Sie dient als Datenspeicher für die Langzeitdatenspeicherung. Die Daten werden also von Druid in Cassandra hinterlegt und können auch von Druid wieder aus Cassandra ausgelesen werden. Dabei müssen bei Cassandra allerdings zunächst Einstellungen konfiguriert werden.

Die Konfigurationsdatei von Cassandra liegt unter `conf/cassandra.yaml` im Cassandra-Verzeichnis. In dieser Konfigurationsdatei müssen folgende Parameter verändert werden, damit Cassandra arbeiten kann:

hints_directory gibt an, unter welchem Pfad die Cassandra `hints` gespeichert werden sollen.

data_file_directories gibt den Pfad zum Datenordner des Keyspaces an, in dem die Daten des Keyspaces hinterlegt werden sollen.

commitlog_directory gibt den Pfad zum Commitlog an, unter dem die Logs gespeichert werden sollen. Hierfür wird am besten eine andere Festplatte genutzt, als für die Speicherung der Daten.

saved_caches_directory gibt den Pfad für die *save caches* an.

seeds ist wichtig für die Verteilung von Cassandra. Hier wird die Adresse eines Knotens angegeben, der bereits im Cluster vorhanden ist. Wenn ein neues Cluster aufgesetzt wird, muss der erste Knoten immer auf sich selbst zeigen, da noch kein Knoten im Cluster vorhanden ist. Alle weiteren Knoten müssen dann auf einen bereits existierenden Knoten des Clusters zeigen.

listen_address hier muss der *host name* angegeben werden.

rpc_address auch hier muss der *host name* angegeben werden.

Damit Druid Daten in Cassandra ablegen kann, muss in Cassandra ein Keyspace angelegt werden. Um einen Keyspace anzulegen, sollte die `cqlsh-Konsole` genutzt werden. Diese ist aufrufbar über die Cassandra beiliegende Anwendung */Druid/apache-cassandra-3.3//bin/cqlsh <externe IP>*. Befindet man sich in der `cqlsh-Konsole`, kann der Keyspace über folgenden Befehl angelegt werden.

```
CREATE KEYSPACE druid WITH replication = {'class' : '
    SimpleStrategy', 'replication_factor' : 2};
```

Mit diesem Befehl wird der Keyspace druid angelegt, der 2 mal repliziert werden soll, um die Daten zu sichern. Sollte nur eine Cassandra-Instanz verwendet werden, ist die Replikation auf 1 zu setzen. Der Keyspace muss in Druid angegeben werden, damit die Daten in Cassandra gespeichert werden können. Außerdem müssen noch 2 Tabellen in Cassandra angelegt werden. Eine für die indizierte Speicherung und einen `Descriptor Storage`. Um diese Tabellen anzulegen können folgende Befehle genutzt werden.

```
USE druid;
CREATE TABLE index_storage
   (key TEXT, chunk TEXT, value BLOB, PRIMARY KEY (key,
       chunk))
WITH COMPACT STORAGE;

CREATE TABLE descriptor_storage
   (key VARCHAR, lastModified TIMESTAMP, descriptor
       VARCHAR, PRIMARY KEY (key))
WITH COMPACT STORAGE;
```

Zunächst einmal muss der Cassandra Keyspace genutzt werden. Dies geschieht mit Hilfe des `USE-Befehls`. Danach werden die beiden Tabellen *index_storage* und *descriptor_storage* mit ihren Parametern angelegt. Diese Befehle sind nahezu identisch mit den MySQL Befehlen, die so etwas erzeugen würden.

1.2.5 MySQL

MM

Die relationale Datenbank MySQL wird in Laotse für zwei Anwendungsgebiete genutzt. Es werden dafür zwei Datenbanken in MySQL angelegt. Diese können parallel auf einer MySQL-Instanz oder auch auf zwei verschiedenen Instanzen auf unterschiedlichen Maschinen betrieben werden. In diesem Abschnitt wird beschrieben, wie die beiden Datenbanken in MySQL angelegt und konfiguriert werden müssen.

Druid Metadata

Der Druid-Metadatenspeicher wird genutzt, um innerhalb von Druid zu speichern, welche Segmente im System existieren. Die Verwaltung wird nach einmaliger Konfiguration automatisch von Druid übernommen. Für die Konfiguration ist es notwendig, auf der Maschine, auf der die Instanz von MySQL installiert ist, die für den Druid-Metadatenspeicher genutzt werden soll auf eine Kommandozeile zuzugreifen. Über diese wird mit dem MySQL-Root Benutzer auf die Datenbank zugegriffen:

```
mysql -u root
```

In MySQL muss nun eine Datenbank und ein Benutzer für Druid angelegt werden. Dabei kann wenn gewünscht der Benutzername und das Passwort frei gewählt werden. Außerdem ist darauf zu achten, dass dem Benutzer das Recht gegeben wird, von den Maschinen, auf denen Druid liegt, auf die Datenbank zuzugreifen. Es ist also ggf. „localhost“ anzupassen.

```
1 -- create a druid database, make sure to use utf8 as
      encoding
  CREATE DATABASE druid DEFAULT CHARACTER SET utf8;
3
  -- create a druid user, and grant it all permissions on
      the database we just created
5 GRANT ALL ON druid.* TO 'druid'@'localhost' IDENTIFIED
      BY 'diurd';
```

Wenn alle Befehle erfolgreich durchgeführt wurden, ist MySQL für die Verwendung von Druid erfolgreich konfiguriert. In 1.2.1 wird beschrieben, wo in Druid die hier gewählten Daten konfiguriert werden müssen.

Diese Anleitung wurde in Anlehnung an [dru16] erstellt.

Laotse Config

Die Laotse-Config Datenbank wird für die Metadaten der Sensoren und Analysen der Dashboards genutzt. Die Verwaltung wird nach einmaliger Konfiguration automatisch von Laotse übernommen. Für die Konfiguration kann ein SQL-Skript genutzt werden, das die benötigte Datenbank und die Datenstrukturen anlegt.

Mit der Kommandozeile muss in das Verzeichnis navigiert werden, welches die Datei *LaotseDatabaseAndSchema.sql* beinhaltet, die mit Laotse ausgeliefert wird. Es kann dann in diesem Verzeichnis folgender Befehl auf der Maschine genutzt werden, auf der die entsprechende MySQL-Instanz installiert ist.

```
1 mysql -u root < LaotseDatabaseAndSchema.sql
```

Anschließend muss noch ein MySQL-Benutzer für Laotse angelegt werden, der die benötigten Zugriffsrechte erhält. Dies geschieht mit folgendem Befehl:

```
1 GRANT ALL ON laotseConfig.* TO '[user]'@'[
      LaotseServerHost]' IDENTIFIED BY '[PASSWORD]';
```

Dabei muss [user] und [PASSWORD] durch den gewünschten Benutzernamen und das gewünschte Passwort ersetzt werden. Diese müssen später in den LAOTSE-Properties entsprechend eingetragen werden. [LaotseServerHost] muss durch den Hostnamen, bzw. die IP-Adresse des LAOTSE-Servers ersetzt werden. Alternativ kann dafür auch „% “ eingesetzt werden, was einen Zugriff von jedem Client aus ermöglicht und daher bedeutend unsicherer ist. Für die Verwendung der IP-Adresse des LAOTSE-Server muss dieser allerdings eine statische IP-Adresse besitzen. Die Datenbank enthält nun eine Reihe vorkonfigurierter Anfragen und Sensoren. Nicht benötigte Sensoren und Sensortypen können entfernt werden.

1.2.6 Odysseus

DN

Odysseus muss nicht weiter konfiguriert werden, wenn die für LAOTSE gebaute Version verwendet wird. Dieser liegen alle benötigten Features bei. Beim ersten Start von Odysseus wird im Nutzerordner der Ordner *odysseus* angelegt [**?**]. In diesem speichert Odysseus Einstellungen. Dort können zum Beispiel gespeicherte Verbindungskonfigurationen geändert werden. Weitere Informationen sind dem Odysseus Wiki [**?**] zu entnehmen.

1.3 Startskripte

LS

Dieses Kapitel erläutert die Benutzung und die Aufgaben der mitgelieferten Startskripte für die einzelnen Bestandteile von LAOTSE.

1.3.1 Druid

Die Druid Startskripte unterstützen jeweils die drei Befehle **start**, **stop** und **restart**. Zum Starten von Druid müssen diese Skripte in der folgenden Reihenfolge mit dem Befehl **start** ausgeführt werden (Bsp. sh druidX.sh start):

1. druidZookeeper.sh
2. druidKafka.sh
3. druidCassandra.sh
4. druidCluster.sh

Dabei ruft das Skript *druidCluster.sh* jeweils die Startskripte der einzelnen Druid-Knoten auf. Bei einer Verteilung von Druid müssten diese also ggf. per Hand aufgerufen werden. Die Reihenfolge der Skripte ist dabei wie folgt einzuhalten:

1. druidCooredinator.sh
2. druidHistorical.sh

3. druidBroker.sh
4. druidRealtime.sh

1.3.2 Odysseus

Nach dem Start von Druid kann Odysseus mit dem Skript *odysseus.sh* gestartet werden. Dieses reagiert auf folgende Argumente:

start Startet Odysseus.

stop Stoppt Odysseus.

restart Stoppt und startet Odysseus.

status Gibt den aktuellen Status vom Odysseus Service aus.

command <arg> Führt das Kommando *<arg>* in der aktuellen Odysseus Session aus.

Durch den Aufruf des vorher genannten Skripts wird wird das Skript *startOdysseus.sh* aufgerufen.

1.3.3 Datengeneratoren

Das Skript *raspberryAudioServer.sh* zeigt beispielhaft wie eine beliebige *.jar*-Datei als Session-unabhängiger Prozess gestartet und gestoppt werden kann. Dies kann z.B. für Datengeneratoren genutzt werden

1.3.4 Laotse Server

Nach dem Start von Druid und Odysseus kann der Laotse Server mit dem Skript *laotse.sh* gestartet werden. Dieses reagiert auf Argumente analog zu *odysseus.sh*. Zum Start vom Laotse Server wird das Skript *startLaotse.sh* nach dem Befehl **start** aufgerufen.

Kapitel 2

Laotse Konfiguration

In diesem Kapitel wird beschrieben, wie LAOTSE konfiguriert werden kann.

Dazu muss LAOTSE zuerst erfolgreich installiert worden sein. Es werden *.properties*-Dateien genutzt um LAOTSE zu konfigurieren. Bei einem ersten Start von LAOTSE werden diese mit Standardwerten in dem Ordner *LaotseProperties* im Benutzerordner automatisch erstellt. Anschließend kann bei Bedarf LAOTSE beendet und die Konfiguration geändert werden. Diese wird bei einem erneuten Start von LAOTSE automatisch übernommen. Unter *pgrepo/configs/LaotsePropertiesClient* und *pgrepo/configs/LaotsePropertiesServer* sind die Dateien für den Client bzw. den Server abgelegt. Die Inhalte können vor einem ersten Start von LAOTSE in den Ordner *LaotseProperties* im Benutzerordner kopiert werden, damit ein Fehlstart vermieden wird.

Falls *.properties*-Dateien fehlen, werden diese von LAOTSE beim Start automatisch mit Standardwerten neu erzeugt.

Für den Server, den Client und das Model, die Komponenten, die sowohl auf dem Server, als auch auf dem Client verteilt liegen, wird jeweils ein Ordner in *LaotseProperties* erzeugt. Dabei ist zu beachten, dass nur die Ordner erzeugt werden, deren Programm auf dem jeweiligen Rechner ausgeführt wird. Es wird beim Client also Client und Model und beim Server Server und Model erzeugt.

Wenn Einstellungen in dem Model angepasst werden, ist zu beachten, dass die entsprechenden Änderungen sowohl auf dem Client, als auch auf dem entsprechenden Server in den Model-Properties durchgeführt werden müssen. Ansonsten kann es zu einem inkonsistenten Zustand von LAOTSE kommen, der unerwartetes Verhalten zur Folge hätte.

Im Folgenden werden die einzelnen *.properties*-Dateien mit ihren jeweiligen Einstellungen beschrieben. Dabei wird besonderer Wert auf diejenigen Einstellungen gelegt, die bei einer produktiven Verwendung von LAOTSE wahrscheinlich angepasst werden müssen. Andere Einstellungen können angepasst werden, um weitere Konfigurationen vorzunehmen. Dies ist für den Betrieb von LAOTSE aber nicht zwingend erforderlich.

2.1 Server

In dem Ordner *Benutzerordner/LaotseProperties/server* sind die Einstellungen des LAOTSE-Servers zu finden, die im Folgenden im einzelnen beschrieben werden.

2.1.1 Database.properties

Hier werden die Einstellungen der Verbindung zwischen dem LAOTSE-Server und der LAOTSE-Datenbank konfiguriert. Für den Betrieb müssen folgende Einstellungen angepasst werden:

- **db.interface:** Interface, das zur Verbindung genutzt werden soll.
- **db.type:** Typ, der LAOTSE-Datenbank.
- **db.user:** Benutzername, der in der LAOTSE-Datenbank genutzt werden soll. Wichtig ist, dass dieser Benutzer Schreib- und Leserechte für die LAOTSE-Datenbank besitzt.
- **db.password:** Passwort des oben angegebenen Benutzers für die LAOTSE-Datenbank
- **db.host:** Hostname des Datenbankservers.
- **db.database:** Name der Datenbank
- **db.port:** Port der Datenbank, der für eine Verbindung genutzt werden kann.
- **db.connectionDriver:** Treiber, der für die Verbindung genutzt werden soll.

Die weiteren Einstellungen können genutzt werden, falls die Tabellennamen der LAOTSE-Datenbank geändert werden und von den Standardnamen abweichen.

2.1.2 LaotseServer.properties

Hier wird eingestellt, auf welchem Hostname und Port dieser Server seinen Websocket öffnen und für einen Client zur Verbindung bereitstellen soll.

- **server.hostname:** Hostname dieser Maschine.
- **server.port:** Port, auf dem der Websocket geöffnet werden soll.

2.1.3 Mosaik.properties

Hier können Einstellungen von dem `MosaikController` angepasst werden. Es ist keine Anpassung der Standardwerte nötig. Es kann Angepasst werden, Welchen Namen die Mosaik-View in Odysseus tragen soll und in welchem Feld der Mosaik-Sensortypen das Muster für die Odysseus-Query der simulierten Sensoren zu finden ist. Beide Werte müssen nur geändert werden, wenn die Funktionalitäten von Mosaik innerhalb Odysseus angepasst werden sollen.

2.1.4 Odysseus.properties

Hier können die Einstellungen für den Odysseus-Service von LAOTSE konfiguriert werden. Sie müssen an die gewählten Einstellungen der gewünschten Instanz des Odysseus-Server angepasst werden.

Dazu sind folgende Konfigurationen notwendig:

- **odysseus.port:** Port, der auf dem Odysseus-Server für eingehende Verbindungen genutzt werden kann.
- **odysseus.host:** Hostname des Odysseus-Server.
- **odysseus.kafaka.address:** Hostname von Kafka, das als Datensenke in Odysseus genutzt werden soll.

Die Anderen Einstellungen können bei den Standardwerten verbleiben und müssen nur angepasst werden, wenn weitreichende Änderungen an Odysseus vorgenommen werden.

2.2 Client

In dem Ordner *Benutzerordner/LaotseProperties/client* sind die Einstellungen des LAOTSE-Client zu finden, die im Folgenden im einzelnen beschrieben werden.

2.2.1 Gui.properties

Für eine Verwendung von LAOTSE ist keine Konfiguration notwendig. Es können aber weitreichende Konfigurationen an der GUI von LAOTSE vorgenommen werden.

Es kann eingestellt werden, ob die GUI im Vollbildmodus gestartet werden soll, und welche Dateien für das Styling der GUI und die GUI selbst genutzt werden sollen. Dadurch kann, wenn gewünscht ein anderes Layout für die GUI einfach geladen werden. Es können dabei sowohl die *.css*, als auch die *.fxml* und *.html*-Dateien frei gewählt werden.

2.2.2 Help.properties

Für eine Verwendung von LAOTSE ist keine Konfiguration notwendig. Es kann aber angegeben werden, unter welchem Pfad die Hilfedatei zu finden ist. Es muss sich dabei um eine *.pdf*-Datei handeln. So können z.B. eigene Notizen und Arbeitsanweisungen in der Hilfe, die sich über die GUI öffnen lässt, einfach ergänzt werden.

2.2.3 LaotseClient.properties

Hier werden die Verbindungsdaten zum LAOTSE-Server konfiguriert. Es ist notwendig sie, an die im gewünschten LAOTSE-Server gewählten Einstellungen anzupassen. So ist es flexibel gehalten, welcher Server gewählt werden soll.

- **client.serverHostname:** Hostname des LAOTSE-Server.
- **client.serverPort:** Port, auf dem zum LAOTSE-Server verbunden werden kann.

2.2.4 MosaikClient.properties

Für eine Verwendung von Laotse ist keine Konfiguration notwendig. Es kann aber angegeben werden, welche Verbindungsdaten für eine neue Mosaik-Simulation als Standardwerte in der GUI genutzt werden. Diese können zwar in der GUI angepasst werden, allerdings kann es hilfreich sein, z.B. die Adresse und den Port der Maschine anzugeben, auf der häufig Mosaiksimulationen durchgeführt werden. Dann müssen diese nur in Ausnahmefällen in der GUI angepasst werden.

2.2.5 Sensorview.properties

Für eine Verwendung von Laotse ist keine Konfiguration notwendig. Es kann aber angegeben werden, wie viele Tage von heute zurück in die Vergangenheit in der GUI angezeigt werden soll, wann Messwerte für einen Sensor vorhanden sind. Dabei ist es notwendig einen Wert größer 0 zu wählen. Die maximale Größe hängt von der Rechenleistung ab. Je größer der Wert gewählt wird, desto länger benötigt die entsprechende Berechnung.

2.3 Model

In dem Ordner *Benutzerordner/LaotseProperties/model* sind die Einstellungen des Laotse-Model zu finden, die im Folgenden im einzelnen beschrieben werden. Es ist zu beachten, dass dieser Ordner und die entsprechenden Einstellungen sowohl auf dem Client, als auch auf dem Server zu finden sind. Sie müssen für eine erfolgreiche Nutzung von Laotse konsistent gehalten werden.

2.3.1 AnalysisTemplates.properties

Für eine Verwendung von Laotse ist keine Konfiguration notwendig. Es können aber Metadaten für die in Laotse verwendeten Analysen konfiguriert werden. Dies kann wünschenswert sein, wenn ein Entwickler weitere Analysen hinzufügt oder vorhandene Analysen konfiguriert. Weiteres dazu ist dem Entwicklerhandbuch von Laotse zu entnehmen.

2.3.2 Druid.properties

Hier werden die Verbindungsdaten zu Druid konfiguriert. Es ist notwendig, die Konfiguration an die Druid Installation anzupassen.

- **druid.url:** Die URL, des Druidknotens, der für Anfragen genutzt werden soll. Dafür ist der Brokerknoten empfohlen. Es sind aber außerdem der historische und der Echtzeitknoten möglich zu wählen.

2.3.3 Sensortype.properties

Für eine Verwendung von Laotse ist keine Konfiguration notwendig. Es kann aber angepasst werden, welcher Sensortyp für Mosaiksensoren genutzt werden soll. Dies kann hilfreich sein, wenn dafür verschiedene Sensortypen mit unterschiedlichen Werten in den Feldern in Frage kommen, die z.B. unterschiedliches Verhalten in Odysseus zur Folge haben. Dann ist durch Anpassung auf den aktuell gewünschten Sensortypen ein schneller Wechsel möglich.

Kapitel 3

GUI-Benutzerhandbuch

In diesem Abschnitt werden der Aufbau und die Verwendung der Laotse-GUI erläutert.

3.1 Aufbau der GUI

Die Laotse-GUI ist in vier Bereiche unterteilt. Im oberen Bereich befindet sich die **Menüleiste**, welche den Wechsel zwischen den drei Anwendungsbereichen Sensoren, Dashboard und Mosaik ermöglicht. Im linken **Tree-Bereich (Pane)** erfolgt die Auflistung der Ausprägungen des ausgewählten Anwendungsbereichs. Der rechte Bereich dient der **Ein- und Ausgabe** der einzelnen Funktionen. Das untere Pane gibt die **Konsolenausgabe** aus, die unter anderem zur Fortschrittskontrolle dienen kann.

3.1.1 Öffnen der GUI

Der Aufruf der GUI erfolgt mittels der ausführbaren **Laotse**-Datei und ist mit der Eingabebereitschaft der Konsole vollständig gestartet. Damit die Clientanfragen verarbeitet werden können, ist ein gestarteter **LaotseServer** erforderlich. Das Vorgehen wird im Benutzerhandbuch beschrieben.

3.2 Sensoren

In diesem Abschnitt werden die Funktionalitäten der Sensoren vom Anlegen, Editieren bis zur Prüfung der Datenübermittlung beschrieben.

3.2.1 Sensoren anzeigen

Die GUI ermöglicht die Anzeige aller vorhandenen Sensoren über eine Baumstruktur, welche nach dem Start der GUI unmittelbar in dem linken Bereich (Pane) angezeigt wird. Unter jedem Sensortyp sind die vorhandenen Sensoren aufgelistet. Während der Nutzung können die Sensoren über die **Menüleiste** → **Show Sensors** eingeblendet werden.

3.2.2 Metadaten und weitere Eigenschaften eines Sensors anzeigen

Die Anzeige der **Metadaten** eines Sensors wird mittels **Doppelklick** auf diesen aufgerufen und das Ergebnis im rechten Pane ausgegeben. Weitere Details sind mittels aufklappbaren Navigationspunkten gegliedert. Standardmäßig sind die Metadaten geöffnet. Das Öffnen und Schließen der Untermenüs erfolgt durch einen **Linksklick** auf die entsprechende **Menüleiste**. Innerhalb der Untermenüs können die Daten mittels des Button **Edit** zur Bearbeitung frei gegeben werden. Die Änderungen können über **Save** gespeichert oder mit **Cancel** verworfen werden. Zu den Metadaten gehören:

- Name des Sensors
- Sensortyp
- Modellnummer
- Inbetriebnahmedatum
- Kommentar
- festgelegte Attribute wie Einheit
- Aktiv (Der Sensor ist aktiviert und kann vom System eingebunden werden, um Daten zu übertragen. Die automatische Einbindung erfolgt mittels Autostart)
- Autostart (Beim Start des Laotse-Servers werden die Daten automatisch aufgezeichnet.)

In dem Untermenü **Query** ist die **Odysseus Anfrage** angegeben, welche für das Datenstrommanagement benötigt wird.

Im Menü **Map** wird der **Standort** des Sensors anhand des Breiten- und Längengrades in einer Karte angezeigt.

Data gibt eine Übersicht über die **Datenaufzeichnungszeiträume** des Sensors an.

3.2.3 Metadaten eines Sensors bearbeiten

Für die Bearbeitung der Metadaten ist zunächst ein **Doppelklick** auf den gewünschten **Sensor** erforderlich. Die Metadaten werden im rechten Pane angezeigt. Durch die Betätigung des **Edit** -Buttons ist eine Bearbeitung der Metadaten möglich. Mittels **Save** werden diese übernommen oder über **Cancel** verworfen.

3.2.4 Odysseus-Query eines Sensors bearbeiten

Die Odysseus Query ist unter den Sensoreigenschaften zu ändern. Diese werden über einen **Doppelklick** auf den betroffenen **Sensor** geöffnet. Um Änderungen vornehmen zu können, ist der Button **Edit** auszuwählen. Als nächstes ist das

Untermenü **Query** durch ein **Anklicken** zu öffnen, um dort die gewünschten Parameter zu ändern. Die Eingabe kann durch einfache Eingabe oder mittels Rechtsklick ausgewählter Funktionen (Rückgängig, Wiederholen, Ausschneiden, Kopieren, Einfügen, Löschen und alles markieren) erfolgen. Anschließend sind die Änderungen mittels **Save** zu übernehmen oder **Cancel** zu verwerfen. Das Kopieren einer Query ist ohne Editierung möglich.

3.2.5 Standort eines Sensors bearbeiten

Der Standort eines Sensors ist unter den **Sensoreigenschaften** änderbar. Diese werden über einen **Doppelklick** auf den betroffenen **Sensor** geöffnet. Die Modifizierung wird über den Button **Edit** aktiviert. Als nächstes ist das Untermenü **Map** durch ein **Anklicken** zu öffnen, um dort den gewünschten Standort über Breiten- und Längengrad zu ändern. Die Eingabe kann durch einfache Eingabe oder mittels über Rechtsklick ausgewählter Funktionen (Rückgängig, Wiederholen, Ausschneiden, Kopieren, Einfügen, Löschen und alles markieren) erfolgen. Anschließend sind die Änderungen mittels **Save** zu übernehmen oder **Cancel** zu verwerfen.

3.2.6 Sensor hinzufügen

Das Hinzufügen neuer Sensoren ist über die Funktion **Add Sensor** unter dem Menüpunkt Sensors vorzunehmen. Nach der Auswahl erscheint ein Eingabefenster, in dem die Metadaten und Eigenschaften des neuen Sensors angegeben werden. Gültige Eingaben werden über einen grünen Hintergrund bestätigt, während der rote Hintergrund unzulässige Eingaben signalisiert.

Metadaten: Die Angabe (siehe Abschnitt 3.2.2) kann durch einfache Eingabe oder mittels Rechtsklick ausgewählter Funktionen (Rückgängig, Wiederholen, Ausschneiden, Kopieren, Einfügen, Löschen und alles markieren) erfolgen.

Query: Die Query wird entsprechend des vorausgewählten Sensortyps vorgegeben. Diese ist an den markierten Stellen um entsprechende Parameter wie Client-IP-Adresse und Zugriffsport zu modifizieren. Das Vorgehen ist im Benutzerhandbuch beschrieben.

Map: Im Untermenü Map kann der Standort durch Angabe des Breiten- und Längengrades angegeben werden. Die Eingabe kann durch einfache Eingabe oder über Rechtsklick ausgewählter Funktionen (Rückgängig, Wiederholen, Ausschneiden, Kopieren, Einfügen, Löschen und alles markieren) erfolgen.

3.2.7 Sensortypen anzeigen

Über die Menüleiste **Sensors** → **Show Sensortypes** werden die Sensortypen angezeigt.

3.2.8 Metadaten eines Sensortypen anzeigen

Nach Auswahl des gewünschten Sensors über den Sensortree im linken Listpane werden die Metadaten mittels **Doppelklick** auf den gewünschten **Sensortyp**

geöffnet. Die Anordnung der Attribute kann über den Namen, des Datentyps und der Position sortiert werden. Dazu ist die gewünscht Sortierreihenfolge mittels anklicken zu wählen.

3.2.9 Aufzeichnungszeitraum eines Sensors anzeigen

Das Vorhandensein von **Messwerten** kann im Bereich der **Metadaten** und **Data** betrachtet werden. Dazu ist zunächst der gewünschte **Sensor** im Sensortree auszuwählen und mittels **Doppelklick** die Metadaten aufzurufen. Durch das Öffnen des Untermenüs **Data** wird das **Vorhandensein von Messwerten** zunächst für einen Betrachtungszeitraum von einem Monat angezeigt. Vorhandene Daten werden über einen farbigen Balken angezeigt. Durch das **Anklicken eines Balkens** am gewünschten Betrachtungstag, kann der Zeitraum der **vorhandenen Daten näher betrachtet** werden.

3.2.10 Daten eines Sensors anzeigen

Die **gespeicherten Daten** eines Sensors werden mit Hilfe eines **Dashboards** angezeigt. Das Vorgehen ist im Abschnitt 3.3 beschrieben.

3.3 Dashboard

Dieser Abschnitt beschreibt die Nutzung der Dashboardfunktionalitäten, welche für die Analyse und Visualisierung der Daten verwendet wird. Mittels eines Dashboards lassen sich die zu betrachtenden Daten einer Datenquelle auswählen und darstellen.

3.3.1 Neues Dashboard öffnen

Ein neues Dashboard kann über zwei Wege geöffnet werden. Entweder über den Menüpunkt **Dashboard** → **Open New Dashboard**, bei dem kein Sensor zur Analyse ausgewählt ist. Der Sensor wird über das **Kontextmenü** → **Add to Dashboard** hinzugefügt.

Der andere Weg öffnet das Dashboard über den betrachteten Sensor. Dazu wird auf dem gewünschten Sensor mittels **Rechtsklick** → **Open new Dashboard** das Dashboard für den Sensor geöffnet.

3.3.2 Betrachtungszeitraum des Dashboards ändern

Der Betrachtungszeitraum der Analyse wird mittels der **Start- und Endzeit** angegeben. Diese kann direkt eingegeben oder mittels Kalender ausgewählt werden. Die Eingabe kann auch mittels Rechtsklick und Einfügen vorgenommen werden.

3.3.3 Granularität des Dashboards ändern

Über die Granularität wird der Grad der Aggregation der ausgegebenen Daten angegeben. Die Auswahl erfolgt über ein **Dropdownmenü**. Die Granularität ist mit Bedacht zu wählen, da die Analyse ggf. eine längere Zeit andauern

kann. Zur Aggregation werden von Druid sogenannte Buckets (Datensammlungen / -gruppierungen) gebildet. Bei kleinen Granularitäten im ms-Bereich werden Buckets auch ohne Daten erstellt. Je höher die Anzahl der benötigten Buckets, desto größer ist die Erstellungszeit. Für **länger andauernde Ausgaben** wird eine **Warnmeldung** ausgegeben.

3.3.4 Analyse ausführen

Nach der Auswahl des Zeitraums und der Granularität wird die Analyse mittels des Buttons **Start Analyze** gestartet. Nach der Analyse wird das Ergebnis im rechten Ausgabefenster angezeigt.

3.3.5 Analysetypen anzeigen

Die vorhandenen Analysen werden über den Menüpunkt **Dashboard** → **Show Analyses** im linken Pane angezeigt.

3.3.6 Metadaten eines Analysetyps anzeigen

Nachdem die Analysetypen im linken Pane angezeigt werden, wird durch Auswahl der gewünschten Analyse die Query mit den Parametern angezeigt.

3.4 Mosaik

Dieser Abschnitt beschreibt die Nutzung der Simulationsdaten des SESA-Labs, welche über Mosaik bereitgestellt werden.

3.4.1 Mosaiksimulation anzeigen

Die Mosaiksimulationen werden analog der übrigen Sensoren unter dem **Sensortree** im linken Pane angezeigt und sind unter den **Sensortypen Mosaik Query** aufgelistet. Die Simulation wird über einen compositeSensor abgebildet, unter welchem die sogenannten virtuellen Sensoren (Sensoren innerhalb der Simulation) zugeordnet sind.

3.4.2 Metadaten und weitere Eigenschaften eines Sensors anzeigen

Die Anzeige der Metadaten und der weiteren Eigenschaften erfolgt analog der übrigen Sensoren, wie im Abschnitt 3.2.2 beschrieben, über einen Doppelklick auf der gewünschten Simulation. Die Metadaten einer Mosaiksimulation sind um die Verbindungsattribute der Simulation ergänzt:

- subSensorQuery: Odysseus Anfrage die für das Datenstrommanagement benötigt wird
- subSensorTypeId: Zur internen Verwendung durch den mosaik-Controller gesetzt
- hostname:IP-Adresse des Simulationsservers für den Verbindungsaufbau der Simulation

- port: Port für die Weiterleitung der Simulationsdaten
- remoteFilepath: Angabe der Simulationsdatei (Python)

3.4.3 Metadaten der Mosaiksimulation bearbeiten

Die im rechten Pane angezeigten Metadaten können durch die Betätigung des **Edit**-Buttons, wie in den Abschnitten 3.2.4 (Odysseus-Query bearbeiten) und 3.2.5 (Standort bearbeiten) beschrieben, modifiziert werden. Mittels **Save** werden diese übernommen oder über **Cancel** verworfen.

3.4.4 Mosaiksimulation einbinden

Die Einbindung einer Mosaiksimulation für die Verarbeitung der Simulationsdaten eines Szenarios erfolgt über den **Menüpunkt Mosaik → New Simulation**. Dabei erscheint ein neues Tab, in dem Metadaten und weitere Eigenschaften angegeben werden.

Damit die Simulationsdaten eines Szenarios verarbeitet werden können, ist zunächst die Konfiguration der Simulation vorzunehmen.

3.4.5 Konfiguration der Mosaiksimulation

Um die Konfiguration der zu startenden Simulation vornehmen zu können, ist zunächst der Menüpunkt Mosaik → New Simulation auszuwählen. In dem Eingabefenster sind die folgenden Parameter anzugeben:

- Name: Angabe des Simulationsnamen durch **Eingabe** oder **Rechtsklick → Einfügen**
- Sensor: Auswahl der Mosaik Query per **Dropdown**
- Adresse: Eingabe der Serveradresse wo das Szenario abgelegt ist **(Eingabe/Einfügen)**
- Port: Eingabe der Portnummer für das Übermittlungsprotokoll **(Eingabe/Einfügen)**
- Remote Filepath: Angabe des Mosaiksimulationsdatei **(Eingabe/Einfügen)**

Nach der Angabe der erforderlichen Parameter, kann die Initialisierung der Simulation vorgenommen werden.

3.4.6 Mosaiksimulation initialisieren

Mit Abschluss der Konfiguration ist die Simulation über den Button **Initialize Simulation** zu starten. Nach Abschluss der Initialisierung, kann die Simulation gestartet werden. Die Fertigstellung der Initialisierung wird mit Hilfe eines **blau hinterlegten Häkchens** bestätigt.

3.4.7 Mosaiksimulation starten

Nachdem die Initialisierung abgeschlossen ist, kann die Simulation über den Button **Start Simulation** gestartet werden, um die Messwerte aus der Simulation zu überführen. Der Verlauf der Simulation wird über die Fortschrittsanzeige visualisiert und mit Fertigstellung über einen **blau hinterlegten Häkchen** markiert.

3.4.8 Virtuelle Sensoren einer Mosaiksimulation anzeigen

Die Sensoren einer Simulation sind unter **Sensors** → **Show Sensors** → **MosaikQuery** im linken Listenpane auswählbar. Nach **Aufklappen** der entsprechenden **Mosaiksimulation** werden die jeweiligen virtuellen Sensoren des Szenarios angezeigt.

3.4.9 Daten eines virtuellen Mosaiksensorens anzeigen

Die gespeicherten Daten eines virtuellen Sensors lassen sich über ein **Dashboard** anzeigen. Das Vorgehen ist im Abschnitt 3.3 beschrieben.

Kapitel 4

Datengeneratoren

DN

Zum Betrieb der Datengeneratoren müssen unterschiedliche Voraussetzungen erfüllt sein. Alle Datengeneratoren benötigen Java 8 und eine Netzwerkschnittstelle, an die sich der Datengenerator binden kann. Standardmäßig binden sie sich an TCP-Port 12345.

Konstantzahlengenerator Es sind keine weiteren Voraussetzungen vorhanden.

Zufallszahlengenerator Es sind keine weiteren Voraussetzungen vorhanden.

Temperatursensor Dieser Datengenerator läuft auf einem Raspberry Pi und greift auf den Sensor DS18B20 zu. Damit die Daten auslesbar sind, muss der Raspberry Pi entsprechend konfiguriert werden [**?**]. Die Kernelmodule `w1-gpio` und `w1_therm` müssen geladen werden. Dies kann realisiert werden, indem der Datei */boot/config.txt* die Zeile

```
1 dtoverlay=w1-gpio
```

hinzugefügt und danach der Raspberry Pi neu gestartet wird. Mit dem verwendeten Sensor wird dadurch eine Geräte-Datei */sys/bus/w1/devices /28-001414b338ff/w1_slave* erstellt, die vom Datengenerator eingelesen wird. Andere Sensoren besitzen eine andere Kennung nach dem Muster 28-xxxxxxxxxxxx. Der Datengenerator müsste an den entsprechenden Pfad angepasst werden.

Audiosensor Dieser Datengenerator benötigt eine Soundkarte, die über eine Aufnahmequelle verfügt. Das Format, in dem die Audiodaten aufgenommen werden, ist über die Datei audio.conf konfigurierbar. In der Standardeinstellung ist der Datengenerator so konfiguriert, dass er auf einem Raspberry Pi die Daten von einem Cirrus Logic Audio Board [**?**] aufnehmen kann. Diese werden so gespeichert, dass der Schalldruckpegel ohne weitere Transformation der Daten visualisiert werden kann.

Damit die Soundkarte funktioniert, müssen entsprechende Kernelmodule installiert werden. Am einfachsten ist die Einrichtung, indem ein angepasstes Raspbian-Image von der Webseite [**?**] geladen wird. Wichtig ist, dass bei einem Update der Raspberry Pi-Firmware die Kernelmodule nicht mehr funktionieren und neu gebaut werden müssten. Es empfiehlt sich die Firmware dementsprechend nicht zu aktualisieren.
Bevor die Aufnahme gestartet werden kann, muss eine der Shellskripts */home/pi/cirrus/Record_from_lineIn.sh* für die Aufnahme des Line-In-Eingangs oder */home/pi/cirrus/Record_from_lineIn_Micbias.sh* für die Aufnahme vom Mikrofon-Eingang ausgeführt werden. Diese konfigurieren die Soundkarte so, dass die Aufnahme entweder als Line-In-Aufnahme oder als Aufnahme mithilfe eines Mikrofons möglich ist.

Die Konfigurationsdatei *audio.conf* muss neben dem jar-Archiv liegen und kann folgende Einstellungen enthalten:

```
# Konfiguriert die Reihenfolge der Bytes pro Sample.
#big_endian = true

# Konfigueriert die Breite der Samples pro Kanal
#bits_per_sample = 16

# 1: mono, 2: stereo, Cirrus Logic Audio Board bietet nur stereo
#channels = 2

# durchschnittliche Sample-Rate
#sample_rate = 16000

# Konfiguriert, ob die Werte signed oder unsigned sind.
#signed = true

# Konfiguriert den Port, an den sich der Datengenerator bindet.
#port = 12345
```

Wenn die Einstellungen geändert werden sollen, muss das '#' vor den entsprechenden Zeilen entfernt werden.

Spannungssensor Dieser Datengenerator ruft die Daten über den Adafruit ADS1015 ab. Der Datengenerator ist so konfiguriert, dass er die Spannung zwischen A1 und A0 misst.
Der ADS1015 wird mit dem Raspberry Pi wie folgt verbunden:

VDD an 5V Spannung (Pin 2)

GND an Masse (Pin6)

SCL an SCL1 (Pin 5)

SDA an SDA1 (Pin 3)

Kapitel 5

Odysseus Queries

KB

Dieser Abschnitt beschreibt, wie die Odysseus-Queries für einzelne Sensoren, sowie für Mosaik aufgebaut sind. Diese Queries können als Grundlage für Einbindung weiterer Sensoren in LAOTSE verwendet werden.

5.1 Sensor Query

Ein an das System angeschlossener Sensor liefert regelmäßig Daten. Diese sollen in einen Datenstrom gebracht werden, um sie in dieser Form dem System zuzuführen und dort persistent zu speichern. Das bedeutet, dass die vom Sensor gelieferten Daten als Datenstrom definiert werden müssen, durch das Datenstrommanagementsystem durchlaufen und mittels Kafka an die Datenbank Druid weitergegeben werden.

Der Query eines Sensors, in diesem Beispiel des an das System angeschlossenen Temperatursensors, sieht wie folgt aus:

```
#PARSER PQL
#QNAME sensor%sensorid%
#ADDQUERY
sensor%sensorid% = ACCESS({
                      Source='sensor%sensorid%',
                      Wrapper='GenericPush',
                      Schema=[['value','DOUBLE'],['
                          timestamp','STARTTIMESTAMP']],
                      transport='NonBlockingTcp',
                      protocol='SizeByteBuffer',
                      dataHandler='Tuple',
                      Options=[['port','12345'], ['host
                          ','172.20.10.131']]
                  })
MAP2 = MAP({
          EXPRESSIONS=[['sensor%sensorid%.value', 'value
              '], ['sensor%sensorid%.timestamp', '
              timestamp'], ['%sensorid%', 'sensor
              '],['0','ns']]
```

```
15          },
            sensor%sensorid%
17      )
KEYVALUE2 = TUPLETOKEYVALUE(MAP2)
19  KafkaOut = SENDER({
                wrapper='GenericPush',
21              transport='kafka',
                protocol='JSON',
23              dataHandler='KeyValueObject',
                SINK='kafkaSink%sensorid%',
25              options=[['metadata.broker.list', '%address
                    %'],
                        ['request.required.acks','1'],
27                      ['producer.type','async'],
                        ['partitioner.class','de.uniol.inf
                            .is.odysseus.wrapper.kafka.
                            KafkaPartitioner'],[
29                      'serializer.class','kafka.
                            serializer.StringEncoder'],
                        ['keyname','sensor'],
31                      ['topicName','laotse'],
                        ['shareProducer','true'],
33                      ['sampleSize','1']]
            },KEYVALUE2)
```

Bei einem Odysseus Query wird zu Beginn immer definiert, welcher Parser genutzt werden muss, um die Query zu übersetzen. In diesem Beispiel wurde die Procedural Query Language (PQL) genutzt. Als nächstes wird der Query benannt. In diesem Fall bekommt der Query den Namen sensor`sensorid`. Die Sensor Id des Sensors wird also an den Ausdruck sensor angehängt, um einen eindeutigen und identifizierbaren Namen zu bekommen. Danach startet die eigentliche Query. Zunächst wird dazu die Verbindung zum Sensor hergestellt, um die gelieferten Daten zu bekommen. Dazu wird der `ACCESS-Operator` verwendet. Innerhalb dieses Operators müssen folgende Parameter übergeben werden.

Source Name des ACCESS Operators

Wrapper In Odysseus stehen `GenericPush` und `GenericPull` zur Verfügung. Es kann also definiert werden, ob die Daten von der Quelle reingeschoben oder abgerufen werden sollen.

Schema Hier wird das Ausgabe Schema definiert. In diesem Fall der ermittelte Double-Wert und der dazugehörige Timestamp.

transport definiert, welches Transportprotokoll genutzt werden soll. In diesem Fall TCP.

protocol definiert das Applikationsprotokoll, um die ermittelten Ergebnisse umzuwandeln.

dataHandler gibt an, welcher Datentyp für die Daten genutzt werden soll. In diesem Fall soll ein Tupel genutzt werden.

Options In diesem Parameter können verschiedene Optionen angegeben werden. Um auf den Sensor zugreifen zu können muss hier die Serveradresse des Sensors, bzw. des angeschlossenen Raspberry Pis und der Port auf den zugegriffen werden kann, angegeben werden.

Die Daten können mit Hilfe dieser Abfrage vom Sensor an Odysseus übertragen und dort weiterverarbeitet werden. Da der Datenstrom aber nicht in Odysseus beendet werden soll, sondern an Kafka weitergegeben werden soll, muss zunächst ein Mapping der eingehenden Attribute auf neue Attribute gemacht werden, die definieren wie diese Attribute im Ausgabestrom heißen sollen. Da Druid eine Key-Value-Datenbank ist, muss das bisher gegebene Tuple in ein Key-Value-Objekt umgewandelt werden. Dazu wird der Operator `TUPLETOKEYVALUE` genutzt.

Im Anschluss daran kann die Verbindung zum Odysseus `KafkaWrapper` hergestellt werden, der die Daten an Kafka weitergibt, damit diese im Anschluss daran in Druid gespeichert werden können. Dazu wird der `SENDER-Operator` genutzt. Dieser dient dazu Daten an einen oder mehrere Endknoten weiterzugeben. Dazu müssen folgende Parameter definiert werden.

wrapper In Odysseus stehen `GenericPush` und `GenericPull` zur Verfügung.

transport definiert das Transport Protokoll, dass genutzt werden soll. In diesem Fall wird der Kafka `TransportHandler` genutzt.

protocol definiert das Applikationsprotokoll, um die ermittelten Ergebnisse umzuwandeln.

dataHandler gibt an, welcher Datentyp für die Daten genutzt werden soll. In diesem Fall soll ein `KeyValueObject` genutzt werden.

options Parameter, die das Transport Protokoll oder das Applikationsprotokoll betreffen. In diesem Fall werden hier Parameter übergeben, die benötigt werden um die Daten an Kafka weiterzugeben, in Kafka zu partitionieren und allgemein in Kafka zu nutzen.

Sind diese Dinge im Sensor Query definiert, werden die Sensordaten als Datenstrom durch Odysseus laufen und zum Abschluss von Kafka an Druid übergeben und dort gespeichert.

5.2 Mosaik Query

Die mosaik k-Query wird benötigt, um Odysseus mit Mosaik zu verbinden und die Daten, die aus mosaik kommen, zu empfangen. Diese Query erhält dabei alle Daten, die von Mosaik kommen. Im Datenstrom sind also alle Sensoren enthalten, die im gewählten mosaik Szenario Daten liefern. Daher ist diese Query besonders wichtig für die Initialisierung von mosaik im System, da während dieser Initialisierung herausgefunden werden soll, welche Sensoren vorhanden sind und im System angelegt werden sollen. Die mosaik-Query ist folgendermaßen aufgebaut:

```
#PARSER PQL
#RUNQUERY
mosaik%sensorid% := ACCESS({
                      Source='sensor%sensorid%',
                      Wrapper='GenericPush',
                      transport='TCPServer',
                      protocol='Mosaik',
                      dataHandler='Keyvalueobject',
                      Options=[['port','5554'],['
                          cleanstrings','true']]
                    })
#QNAME mosaik%sensorid%
#ADDQUERY
Stream1 = Stream({SOURCE='System.mosaik%sensorid%'})
```

Zunächst einmal wird angegeben, dass die gegebene Query in der Procedural Query Language (PQL) geschrieben wurde. Im `RUNQUERY` Bereich wird dann mit Hilfe des ACCESS-Operators ein TCP Server aufgebaut, mit dem sich mosaik dann verbindet. Für die Generierung eines solchen TCP Servers werden folgende Parameter benötigt:

Source Name des ACCESS Operators

Wrapper In Odysseus stehen `GenericPush` und `GenericPull` zur Verfügung. In diesem Fall wird `GenericPush` genutzt.

transport definiert, welches Transportprotokoll genutzt werden soll, in diesem Fall TCP.

protocol definiert das Applikationsprotokoll, in diesem Fall Mosaik.

dataHandler wie sollen die Daten genutzt werden? Für Mosaik als Key-Value-Objekte.

Options hier können verschiedene Optionen als Parameter übergeben werden.

Mit Hilfe dieses TCP-Servers erhält Odysseus die Daten aus mosaik. Dieser TCP-Server wird in einer Odysseus View gesichert. Da eine View aber keine eigene Query erzeugt, muss diese im weiteren Verlauf noch angelegt werden. Dazu wird zunächst der Name der Query festgelegt und dann die eigentliche Query definiert. In diesem Fall wird dabei nur ein Stream erzeugt, der die Daten aus der View aufgreift. Die definierte View wird hier also in einen Stream gelegt, der als Stream1 gespeichert wird. Dieser Stream kann dann vom System überwacht werden und somit die Daten ausgelesen werden. Im Falle der Initialisierung würden aus diesem Strom alle Sensoren, die in dem mosaik Szenario enthalten sind ausgelesen werden, um diese im System anzulegen.

5.2.1 Mosaik Subquery

Jeder dieser im System angelegten Sensoren erhält dann wiederum eine eigene Odysseus Query, mit Hilfe welcher die Daten der einzelnen Sensoren erfasst und an Kafka und somit auch an das System übertragen werden. Die Query eines solchen Untersensors ist wie folgt aufgebaut:

```
#PARSER PQL
#QNAME mosaik%sensorid%
#ADDQUERY
sensor%sensorid% = Stream({SOURCE='System.mosaik%
    parentSensorid%'})
keyvalue = KEYVALUETOTUPLE({
              SCHEMA=[['%sensorName%','Double']],
              KEEPINPUT='false',TYPE='mosaik'
            },sensor%sensorid%)
rename2 =rename({
            aliases = ['%sensorName%','value'],
            pairs = 'true'
          },keyvalue)
MAP2 = MAP({
        EXPRESSIONS=[['value','value'], ['start', '
            timestamp'], ['%sensorid%', 'sensor'],['0','
            ns']]
       }, rename2)
KEYVALUE2 = TUPLETOKEYVALUE(MAP2)
KafkaOut = SENDER({
              wrapper='GenericPush',
              transport='kafka',
              protocol='JSON',
              dataHandler='KeyValueObject',
              SINK='kafkaSink%sensorid%',
              options=[
                ['metadata.broker.list', '%address%'],
                ['request.required.acks','1'],
                ['producer.type','async'],
                ['partitioner.class','de.uniol.inf.is.
                    odysseus.wrapper.kafka.
                    KafkaPartitioner'],
                ['serializer.class','kafka.serializer.
                    StringEncoder'],
                ['keyname','sensor'],
                ['topicName','laotse'],
                ['shareProducer','true']]
         },KEYVALUE2)
```

Auch in dieser Query wird zunächst angegeben, dass `PQL` genutzt wurde. Im Anschluss daran wird dann ein Name für die Query definiert. Analog zu den Sensoren wird hier mosaik mit der angehängten Sensorid des angelegten Sensors genutzt. Danach folgt die Definition der eigentlichen Query. Dazu wird zunächst auf die View zugegriffen, die durch den „Obersensor“definiert wurde, um auf die Daten von mosaik zuzugreifen. Die Daten liegen dann als Key-Value-Objekte vor. Damit Odysseus die Daten verarbeiten kann sollten, sollten die Daten in Tupeln vorliegen. Daher werden die Daten zunächst in Tupel umgewandelt. Dies geschieht mit Hilfe des `KeyValueToTuple-Operators`. Hier muss angegeben werden, welche Daten wie gespeichert werden sollen. Die Daten, die unter dem Sensornamen ankommen, sollen hier als Double-Werte gespeichert werden, da es Sensorwerte sind. Diese Operation wird auf dem definierten Strom, der aus der View des „Obersensors“erzeugt wurde, ausgeführt. Damit liegen die Daten dann

in Tupel Form vor. Zur besseren Verständlichkeit und damit das System die Daten auf die gleiche Art und Weise abrufen kann, wie bei den Sensoren, wird der Parameter mit dem Sensornamen in `value` umbenannt. Anschließend findet ein Mapping für den Ausgabestrom statt, in dem die Parameter, die übergeben werden sollen, angegeben werden und wie diese im Ausgabestrom heißen sollen. Dies wird über den `MAP-Operator` realisiert.

Damit die Daten der einzelnen Mosaik Sensoren auch in Druid ankommen, müssen die Daten an Kafka weitergegeben werden, von wo die Daten dann zu Druid gelangen. Dies geschieht mittels des `SENDER-Operators`. In diesem Operator müssen dabei einige Parameter angegeben werden.

wrapper In Odysseus stehen `GenericPush` und `GenericPull` zur Verfügung

transport definiert, welcher TransportHandler aus Odysseus genutzt werden soll. In Laotse wird der `KafkaTransportHandler` verwendet.

protocol definiert das Applikationsprotokoll, um die ermittelten Ergebnisse umzuwandeln. In diesem Fall soll `JSON` genutzt werden.

dataHandler wie sollen die Daten genutzt werden? Als Key-Value-Objekt.

SINK Name der Datensenke.

options Verschiedene Optionen, die im Transportprotokoll oder dem Applikationsprotokoll genutzt werden können. In diesem Fall werden hier Kafka Optionen übergeben, die Kafka benötigt um die Daten richtig zu verarbeiten, sie zu Partitionen und sie korrekt zu speichern, damit Druid darauf zugreifen kann.

Kapitel 6

mosaik

LS

Zum Starten einer mosaik-Simulation muss eine entsprechendes Szenario durch ein Python-Skript gestartet werden, siehe dazu die mosaik-Dokumentation [mos15].

Der implementierte `MosaikStarter` ist in der Lage solche Skripte zu starten, sofern sie in einem bestimmten Ordner liegen. Er kann auf einem beliebigen Server installiert werden und bietet folgende Konfigurationsmöglichkeiten seiner Java-Konstanten:

SCRIPT_DIR Der Ordnerpfad zu den Python-Skripten.

Standard: */Code/mosaik-demo/*.

VIRT_ENVIRONMENT Der Pfad zur Ausführbaren, die die virtuelle Umgebung für die erfolgreiche Ausführung des Skriptes aufsetzt.

Standard: */.virtualenvs/mosaik/bin/activate*

PORT Der Part für das `ServerSocket` auf dem der `MosaikStarter` auf Nachrichten wartet.

Standard: 12345.

Für eine erfolgreiche Anbindung an Odysseus muss der Pfad am Anfang des Python-Skriptes in `sim_config` an die genutzte Odysseus Instanz angepasst werden:

```
sim_config = {
    'CSV': {
        'python': 'mosaik_csv:CSV',
    },
    ... ,
   'Odysseus': {
        'connect': '172.20.50.96:5554'
    }
}
```

Literaturverzeichnis

[dru16] *Druid Metadata Storage Installation Guide*. `http://druid.io/docs/latest/dependencies/metadata-storage.html`. Version: 2016. – Zugriff: 25.03.2016

[mos15] *Mosaik Dokumentation*. `http://mosaik.readthedocs.org/en/latest/overview.html#mosaik-s-main-components`. Version: 2015. – Zugriff: 29.10.2015

I. Entwicklerhandbuch

Entwicklerhandbuch zu LAOTSE

31. März 2016

Inhaltsverzeichnis

Kapitel 1

Entwicklung in Eclipse

DN

Die Entwicklung von Laotse wurde mithilfe der IDE Eclipse realisiert. In diesem Kapitel wird beschrieben, welche Schritte notwendig sind, wenn Laotse in einen Eclipse-Workspace importiert werden soll. Zur Entwicklung wurde Eclipse Mars for RCP and RAP Developers [**?**] verwendet.

1.1 Tycho

Die Bundles werden mithilfe von Maven gebaut. Eclipse benötigt den m2e-Connector `Tycho Configurator`, damit es den Build durchführen kann. Dieser kann unter **Window** → **Preferences** → **Maven** → **Discovery** → **Open Catalog** gefunden und installiert werden.

1.2 e(fx)clipse

Für die Laotse-GUI und das Laotse Model wird JavaFX verwendet. Damit diese Bundles gebaut werden können, muss `e(fx)clipse` installiert werden. Es kann über den Eclipse Marketplace (**Help** → **Eclipse Marketplace...**) gefunden und installiert werden.

1.3 Workspace

Damit die Bundles bearbeitet werden können, werden die in den Workspace von Eclipse importiert (**File** → **Import...** → **General** → **Existing Projects into Workspace**). In dem Dialog kann der Ordner, in dem die Bundles liegen ausgewählt werden (*OSGi-Mars*). Ein Klick auf **Finish** importiert die gewählten Eclipse-Projekte.
Die Bundles werden gegen eine Target-Plattform entwickelt. Diese bestimmt, welche Bundles zur Laufzeit zur Verfügung stehen. Damit alle Projekte gebaut werden können, muss die Target-Plattform *Laotse.target* aus dem Bundle `de.offis.laotse.model.Feature` gesetzt werden. Dazu wird die Datei mit einem Doppelklick geöffnet und gewartet, bis die Target-Plattform geladen wurde.

Es kann erforderlich sein, dass die Target-Plattform mit Klick auf **Update** und **Reload** aktualisiert werden muss. Danach kann die Plattform gespeichert und mit Klick auf **Set as Target Platform** gesetzt werden.

1.4 Checkstyle

Damit Codekonventionen von geprüft werden können, kann Checkstyle in Eclipse installiert werden. Das `Checkstyle Plug-In` kann über den Eclipse Marketplace gefunden und installiert werden (**Help** → **Eclipse Marketplace...**).
Im Projekt wurden die Java-Codekonventionen von Google angewendet. Damit Checkstyle diese verwendet, werden die Regeln importiert. Unter **Window** → **Preferences** → **Java** → **Code Style** → **Formatter** können die Regeln importiert werden. Diese liegen unter *pgdoku/CodingStyles/styleguide-gh-pages* in Datei *eclipse-java-google-style.xml*.
Damit die imports in der richtigen Reihenfolge sortiert werden, kann die Ordnung durch die Datei *laotse.importorder* vorgegeben werden. Diese liegt im Ordner *pgdoku/CodingStyles/* und kann unter den Einstellungen **Window** → **Preferences** → **Java** → **Code Style** → **Organize Imports** importiert werden.
Die Bundles in Laotse sind so konfiguriert, dass Checkstyle aktiviert ist. Sollten neue Bundles hinzugefügt werden, muss Checkstyle aktiviert werden (Rechtsklick auf das entsprechende Bundle → **Checkstyle** → **Activate Checkstyle**).

1.5 Bundle erstellen

Wenn ein neues Bundle erstellt wird, muss dieses so konfiguriert werden, dass es gültige OSGi-Bundles ist. Damit es für das Laotse-System gebaut und eingesetzt werden kann, muss es zusätzlich mit einer gültigen Maven-Konfiguration versehen werden. In Laotse werden zwei Arten von Bundles erstellt. Zum einen können Drittanbieter-Bibliotheken in eine OSGi-Bundle gewrapped werden. Zum anderen werden Bundles genutzt, die selbst Quelltext enthalten.

Ein neues Bundle wird unter **File** → **New** → **Plugin Project** erstellt. Dem Bundle wird ein Name und der Ordner, in dem die Dateien abgelegt werden, zugeordnet. Unter **Target Platform** kann **an OSGi framework:** ausgewählt werden, wenn keine speziellen Eclipse-Funktionen verwendet werden müssen. Im folgenden werden weitere Metadaten konfiguriert. Die ID und Version sollten gemerkt werden, da sie im weiteren Verlauf benötigt wird. Ein Klick auf **Finish** erstellt das Bundle.

Im Anschluss wird die Maven-Konfiguration hinzugefügt (Rechtsklick auf das Bundle → **Configure** → **Convert to Maven Project**). Die Artifact Id sollte der ID des Bundles entsprechen. Die Versionsinformation muss an das Bundle angepasst werden. Als Packaging ist `eclipse-plugin` zu wählen.
Damit das Bundle gebaut wird und das Packaging aufgelöst werden kann, wird das Bundle einem der Maven-Parents zugeordnet. Dazu ist die Datei *pom.xml* des Parents zu öffnen. Im Abschnitt **Modules** wird das vorher erstellte Bundle ausgewählt. Der Haken an **Update POM parent section in selected pro-**

jects sollte gesetzt sein. Anschließend muss das erstellte Bundle aktualisiert werden (Rechtsklick auf das Bundle → **Maven** → **Update Project...**).

Wenn Java-Pakete von anderen Bundles genutzt werden sollen, müssen sie in der *MANIFEST.MF* des implementierenden Bundles exportiert werden (unter **Runtime** → **Exported Packages**).

Sollen Bibliotheken von Drittanbietern eingebunden werden, werden diese für gewöhnlich im Bundle im Ordner *lib/* abgelegt. Damit diese zur Verfügung stehen, müssen sie dem Classpath hinzugefügt werden (unter **Runtime** → **Classpath**). Die Java-Pakete müssen dann ebenfalls, wie vormals beschrieben, exportiert werden, um von anderen Bundles genutzt zu werden.

1.6 Laotse bauen

Damit alle Komponenten gebaut werden, sollten die Parents in der folgenden Reihenfolge mit den Goals **clean verify** gebaut werden. Dabei ist folgende Reihenfolge einzuhalten:

1. ResourcesParent
2. ModelParent
3. ClientParent und ServerParent

Die Produkte Client und Server werden gebaut und installiert, wenn zusätzlich das Maven-Profil **product** hinzugefügt wird.

Kapitel 2

Erweiterungen an Laotse

MM

In diesem Kapitel wird erklärt, wie ein Entwickler vorgehen muss, um einige Komponenten von Laotse zu erweitern. Das Vorgehen wird detailliert beschrieben, so dass Fachleute es nachvollziehen können. Es wird beschrieben, wie man das Dashboard erweitern, neue Sensortypen anlegen und neue Properties zur Verfügung stellen kann. Dadurch können neue Funktionen zu Laotse hinzugefügt werden.

2.1 Analysen zum Dashboard hinzufügen

MM

Das Dashboard in der Laotse-GUI dient dazu, Analysen über eine Menge an Sensoren und einen Zeitraum durchzuführen. Dabei wird die Menge der Sensoren und der Zeitraum durch den Nutzer in der Laotse-GUI ausgewählt und die vorgegebenen Analysetypen mit diesen und weiteren durch den Nutzer festgelegten Parameter durchgeführt. Der Entwickler kann die Menge dieser im Dashboard genutzten Analysetypen einfach erweitern. In diesem Abschnitt wird erklärt, wie dazu vorgegangen werden muss.

Es müssen dazu folgende Schritte durchgeführt werden, die nachfolgend genauer beschrieben werden.

1. Neuen Analysetyp in der Datenbank anlegen
2. ggf. Parameter für den neuen Analysetypen in der Datenbank anlegen
3. ggf. Elemente zur Eingabe der Werte für die Parameter zur GUI des Dashboard hinzufügen
4. Elemente zur Anzeige der Ergebnisse zur GUI des Dashboard hinzufügen
5. Quelltext des `DashboardController` um Berechnung und Anzeige der Analyseergebnisse erweitern

Es gibt bereits zwei Arten von Dashboards, die von der abstrakten Klasse `DashboardController` erben.

2.1.1 Neuen Analysetyp in der Datenbank anlegen

MM

In der LAOTSE-Datenbank wird ein neuer Analysetyp hinzugefügt, indem eine Zeile in die Tabelle `analysisTemplates` eingefügt wird. Dazu sind folgende Werte notwendig:

- **id:** Wird automatisch durch die Datenbank gesetzt und muss nicht beachtet werden.
- **name:** Ein String, der als Name für den Analysetypen genutzt wird. Er muss innerhalb dieser Tabelle einmalig sein, da er in dem LAOTSE-Client zur Identifikation genutzt wird.
- **body:** Die Druid Anfrage mit Platzhaltern für die Sensorliste und weitere Parameter als String. Wie Druid-Anfragen aufgebaut werden, wird in Kapitel 3 erläutert. Als Platzhalter für die Sensorliste und die anderen Parameter sollte der String genutzt werden, der in der Datei *LaotseProperties/model/AnalysisTemplates* angegeben ist. Dieser wird von LAOTSE in der Anfrage gesucht und ersetzt. Die Parameter werden nach dem Platzhalter durchnummeriert. Beispielsweise ist bei dem Platzhalter `%` der erste Parameter `%0` und der zweite Parameter `%1`.
- **explanation:** Ein Freitext, der zur Erklärung genutzt werden kann.

2.1.2 Parameter für den neuen Analysetypen in der Datenbank anlegen

MM

In der LAOTSE-Datenbank müssen nun die Parameter, für die im vorherigen Schritt Platzhalter angelegt wurden, spezifiziert werden. Dies geschieht, indem für jedes Auftreten eines Parameters in jedem Analysetypen eine Zeile in die Tabelle `analysisFields` eingefügt wird. Dazu sind folgende Werte notwendig:

- **id:** Wird automatisch durch die Datenbank gesetzt und muss nicht beachtet werden.
- **name:** Ein String, der als Name dient. Er muss für jeden Analysetypen, aber nicht für die gesamte Tabelle einmalig sein.
- **format:** Ein String, der angibt, in welchem Format der Wert des Parameter erwartet wird.
- **position:** Position des Parameters innerhalb des Analysetypen, indem er auftritt. Die Position ist die Nummerierung nach dem Platzhalter.
- **templateId:** Id des Analysetypen, in dem dieser Parameter an der angegeben Position auftritt.

Die Sensorliste braucht hier nicht weiter betrachtet werden, da sie von LAOTSE in jeder Anfrage erwartet wird und automatisch verarbeitet wird.

2.1.3 Elemente zur Eingabe der Werte der Parameter zur GUI des Dashboard hinzufügen

MM

Wenn im neuen Analysetypen ausschließlich Parameter genutzt werden, die bereits in anderen Analysetypen genutzt werden, kann dieser Schritt übersprungen werden. Das ist im Auslieferungszustand von LAOTSE genau dann der Fall, wenn der Name der Parameter einem der Folgenden entspricht:

- Interval
- Granularity
- Threshold

In diesem Fall werden die Werte, die der Nutzer für die Parameter in die GUI eingegeben hat, auch bei der Verwendung in neuen Analysetypen, bereits automatisch gesetzt.

Bei einer neuen Art des Parameters muss die GUI um ein Eingabefeld für den Paramter erweitert werden. Dazu wird die *.fxml* -Datei z.B. mit einem neuen Textfeld erweitert. Anschließend muss dafür gesorgt werden, dass der `DashboardController` den Wert des Nutzers in den neuen Parameter einsetzt. Dazu wird die Implementierung der Methode `checkAndSetFieldValue` erweitert. Es muss der Parameter anhand des Namens identifiziert werden und der vom Nutzer in die GUI eingegebene Wert in den Parameter gesetzt werden. Im Folgenden dazu ein Beispiel für die Granularität:

```
@Override
  protected FieldValue checkAndSetFieldValue(FieldValue
      value) {
    if (value.getField().getName().equals("Granularity")
        ) {
      value.setValue(this.tfGranularity.getText());
    }
    return value;
  }
```

Damit innerhalb des Java Quelltextes auf ein Textfeld der *.fxml* zugegriffen werden kann, muss dieses mit einer id und das Java-Feld mit einer Annotation versehen werden. Dies zählt zum Basiswissen für JavaFX und kann bei Bedarf in einer Anleitung von JavaFX nachgelesen werden, da weitere Erläuterungen dazu den Umfang dieser Dokumentation übersteigen würden.

Durch diese Implementierung kann die neue Art des Parameters in jedem Analysetypen verwendet werden und wird dann jedes mal automatisch erkannt und gesetzt. Der Aufruf der hier beschriebenen Funktion erfolgt automatisch beim Starten der Analysen.

2.1.4 Elemente zur Anzeige der Ergebnisse zur GUI des Dashboard hinzufügen

MM

Es kann jedes beliebige Element, auch selbst implementierte, genutzt werden, um Ergebnisse zu visualisieren. Im Folgenden ein Beispiel für ein Label, das einfachen Text und Zahlen darstellen kann.

Die *.fxml* -Datei des Dashboard muss um ein Label erweitert werden, welches das Ergebnis der Analyse anzeigen soll. Dieses wird mit einer id versehen und als Feld im `DashboardController` instanziiert. Das Feld wird mit der Annotation `@FXML` versehen und damit automatisch vom `FXML-ControllerLoader` mit dem Textfeld geladen.

2.1.5 Quelltext des `DashboardController` um Berechnung und Anzeige der Analyseergebnisse erweitern

MM

Die Implementierung muss so erweitert werden, dass die Analyse instanziiert, durchgeführt und das Ergebnis angezeigt wird.

Die neue Analyse wird automatisch aus der Datenbank mit den Parametern geladen und dem `DashboardController` übergeben.

Falls die Analyse mit dem neuen Analysetypen einzeln für jeden ausgewählten Sensor berechnet werden soll, sollte sich der Entwickler sie analog zur `Range` -Query implementieren. Andernfalls, wenn eine Aggregation über alle ausgewählten Sensoren berechnet werden soll, kann der Entwickler sich an den anderen Analysetypen orientieren. Dann muss der Entwickler die Methode `executeAndDisplayAnalysis` erweitern. Der Analysetyp wird anhand seines Namens identifiziert und anschließend vom `DruidDbWrapperService` durchgeführt. In den meisten Fällen kann dafür die Methode `getAnalysisResult` genutzt werden. Diese erwartet in dem JSON-Ergebnis von Druid das Ergebnis unter dem Wert „value “.

Wenn komplexere Berechnungen durchgeführt werden sollen, muss dafür eine eigene Methode implementiert werden, so wie es bei der `Range` und der `Avg` -Analyse der Fall ist. So können z.B. auch auf der Java-Ebene noch komplexere Berechnungen oder Vergleiche mit anderen Daten, mit den Ergebnissen von Druid implementiert werden. Vor und nach der Berechnung kann der aktuelle Zeitstempel erhoben werden. Diese werden später automatisch für eine Auswertung der Analysedauer genutzt.

Um das Ergebnis in der GUI anzuzeigen, muss der Wert in das Label gesetzt werden, das Element gesetzt werden, das im vorherigen Schritt hinzugefügt wurde. Dabei sollte sichergestellt werden, dass dies im GUI-Thread geschieht, da die Berechnung der Analyse in einem anderen Thread durchgeführt wird. Ein Beispiel aus der `Maximum`-Analyse lautet:

```
1 FxThreadUtil.runSafelyOnUiThread(() -> lblMax.setText(
      result));
```

Abschließend sollte die `DruidDbWrapperException` abgefangen und der Fehler in der GUI visualisiert werden.

2.2 Sensortypen hinzufügen

KB

Sensortypen können nicht über die Benutzeroberfläche hinzugefügt werden, da in diesem Bereich nicht sehr häufig neue Typen dazu kommen. Um einen Sensortypen hinzuzufügen, muss also auf die Datenbank zugegriffen werden. Dazu muss in der Datenbank in der Tabelle `sensorTypes` eine neue Zeile hinzugefügt werden. Diese Tabelle enthält nur die beiden Spalten `Id` und `Name`. Da die Id automatisch hochgezählt wird, muss dementsprechend nur der Name des Sensortypen angegeben werden und das einfügen abgeschlossen werden. Dann existiert ein neuer Sensortyp im System.

2.3 Properties hinzufügen

MM

Was Properties sind, wozu sie dienen und wie sie funktionieren ist im Benutzerhandbuch unter dem Kapitel „Properties “ erklärt. Die Lektüre wird vor der Lektüre dieses Kapitels empfohlen. Hier wird beschrieben, wie ein Entwickler neue Properties zu LAOTSE hinzufügen kann.

Neue Properties sollten immer dann hinzugefügt werden, wenn neue Funktionalitäten in LAOTSE implementiert werden, für die Einstellungen benötigt werden, die nicht in der GUI vom Nutzer angepasst werden können oder sollen. Bei diesen ist es dann möglich bei Bedarf diese in *.properties* -Dateien anzupassen. Es müssen dazu Folgende Schritte durchgeführt werden:

1. Erstellen einer default *.properties* -Datei
2. Implementieren einer Properties-Klasse
3. Nutzen der Properties

Der Entwickler muss sich nicht darum kümmern, dass die neue Datei beim Nutzer angelegt wird, falls er sie noch nicht hat. Dies geschieht von LAOTSE automatisch, wenn sie wie im Folgenden beschrieben angelegt wurde.

2.3.1 Erstellen einer default *.properties*-Datei

MM

Damit beim Nutzer beim Start von LAOTSE die neue *.properties*-Datei angelegt wird, muss lediglich eine Datei mit den gewünschten Standardwerten im entsprechenden Properties Bundle angelegt werden. Dies ist für Properties für den Client: `de.offis.laotse.client.Properties` und für den Server: `de.offis.laotse.server.Properties`. Bei Properties, die sowohl im Client als auch im Server genutzt werden sollen, also z.B. auch im Model, wird das Bundle `de.offis.laotse.model.Properties` genutzt.

In dem Bundle wird in dem Ordner *LaotseProperties* in dem entsprechenden Unterordner *client*, *model* oder *server* eine neue *.properties*-Datei angelegt. In

der Datei werden die Properties nach dem Folgenden Format mit Standardwerten angegeben:

```
# comment
[prefixName].[propertyName] = [defaultValue]
```

Dabei gilt:

- **[prefixName]:** Prefix des Property-Namens, der für die ganze Datei einheitlich ist und zur Identifikation der Properties genutzt wird.
- **[propertyName]:** Name der Property.
- **[defaultValue]:** Standardwert, der genutzt werden soll, wenn der Nutzer *.properties*-Datei noch nicht hat oder nicht konfiguriert hat.

2.3.2 Implementieren einer Properties-Klasse

MM

Es muss eine Klasse implementiert werden, die die im vorherigen Schritt angelegte Datei nutzt. Dazu steht die entsprechende Klasse `ModelProperties`, `ClientProperties` und `ServerProperties` in dem entsprechenden Bundle zur Verfügung. Von dieser Klasse sollte geerbt werden. Dadurch wird der gewünschte Dateipfad richtig genutzt. Außerdem erben alle diese Klassen von `Properties`, die das Dateihandling und Laden der Properties implementiert.

Die neue Klasse sollte wie folgt implementiert werden:

```
public class [ClassName]Properties extends Model/Server/
    ClientProperties {

  public static final String FILENAME = "[ClassName].
      properties";

  public static final String PROPERTY_NAME_PREFIX = "[
      prefix].";

  public static final String [PropertyName1]
      _PROPERTY_NAME = "[PropertyName1]";
  public static final String [PropertyName2]
      _PROPERTY_NAME = "[PropertyName2]";

  private static [ClassName]Properties instance;

  public static [ClassName]Properties getInstance() {
    if ([ClassName]Properties.instance == null) {
      [ClassName]Properties.instance = new [ClassName]
          Properties();
    }
    return [ClassName]Properties.instance;
  }

  @Override
  public String getPropertyNamePrefix() {
```

```
    return PROPERTY_NAME_PREFIX;
  }

  @Override
  public String getFileName() {
    return FILENAME;
  }

}
```

Dabei müssen folgende Variablen ersetzt werden:

- **[ClassName]:** Dateiname der zugehörogen *.properties* Datei ohne Endung. Dabei darf die Schreibweise in der Konstante `FILENAME` und dem Auftreten in dem Klassennamen voneinander abweichen.
- **[prefix]:** Prefix der Properties, der auch in der *.properties* Datei genutzt wurde.
- **[PropertyNameX]:** Namen der einzelnen Properties, die in der *.properties* Datei genutzt wurden. Für jede Property wird hier eine Konstante definiert. [PropertyNameX] steht stellvertretend für einen dieser Namen.

2.3.3 Nutzung der Properties

MM

Wenn der Wert einer Property genutzt werden soll, kann dies durch folgenden Aufruf geschehen:

```
// Fuer einen String:
[ClassName]Properties.getInstance().getString([ClassName
    ]Properties.[PropertyNameX]_PROPERTY_NAME);

// Fuer einen boolschen Wert:
[ClassName]Properties.getInstance().getBoolean([
    ClassName]Properties.[PropertyNameX]_PROPERTY_NAME);

// Fuer ein int Wert:
[ClassName]Properties.getInstance().getInt([ClassName]
    Properties.[PropertyNameX]_PROPERTY_NAME);
```

Dabei sind die Variablen entsprechend der vorherigen Implementierung zu ersetzen. Wenn diese Werte der Properties in Konstanten in den Klassen gehalten werden, die sie betreffen, muss jeder Wert nur ein mal aus den Dateien gelesen werden, was eine höhere Performanz verspricht.

Kapitel 3

Druid Queries

KB

Für den Abruf von Daten aus Druid werden Anfragen im JSON-Format benötigt. In diesem Abschnitt werden die verwendeten Anfragemuster erläutert.

3.1 Range Queries

In diesem Abschnitt werden die beiden Range Queries erklärt, die im System genutzt werden. Im System wird eine Range Query genutzt, die die Daten mit einer bestimmten Granularität aggregiert. Die andere Query bekommt einen Threshold und zeigt dem Threshold entsprechend viele Werte an. Dafür aber die genauen Werte.

3.1.1 Aggregierte Range

Die aggregierte Range nutzt eine Granularität und aggregiert dementsprechend diese Werte. Die Query für diese Analyse sieht wie folgt aus:

```
{"queryType" : "timeseries",
 "dataSource" : "laotse",
 "filter" : {"type" : "selector", "dimension" : "sensor
    ", "value" : "<SENSORLIST>" },
 "dimensions" : [],
 "granularity" : %1,
 "aggregations" : [
   {"type" : "doubleSum", "name" : "sum", "fieldName" :
       "sum"},
   {"type": "longSum", "name" : "count", "fieldName" : "
       count"}
 ],
 "intervals" : ["%0"],
 "context" : {"skipEmptyBuckets": "true"}
}
```

Für diese Anfrage wird ein Timeseries-Query in Druid genutzt. Außerdem müssen noch folgende Parameter angegeben werden.

dataSource Welche Datenquelle soll genutzt werden. Entspricht der Partition von Kafka.

filter Wonach soll gefiltert werden? Entspricht im Grunde genommen dem Where in einer SQL-Abfrage. In diesem Fall soll nach Sensor gefiltert werden und es sollen alle Sensoren abgerufen werden, die im Dashboard in der Sensorliste enthalten sind.

dimensions Hier kann angegeben werden, welche `Dimensions` abgerufen werden sollen. In diesem Fall wird gar nichts angegeben, das bedeutet, dass alle `Dimensions` abgerufen werden. Es werden also alle Daten die vom Sensor vorhanden sind abgerufen.

granularity Hier kann angegeben werden, mit welcher Granularität die Daten aggregiert werden. Wird zum Beispiel `hour` angegeben, wird für jede Stunde ein Wert aggregiert. In diesem Fall steht hier ein Platzhalter, da hier während der Laufzeit der ausgewählte Wert aus dem Dashboard eingesetzt wird.

aggregations Hier kann definiert werden, welche Aggregationen getätigt werden sollen. In diesem Fall wird die Summe aller Werte und die Anzahl der gegebenen Werte mit abgerufen.

intervals das Intervall in dessen Bereich die Werte ermittelt werden. Hier steht wiederum ein Platzhalter, da an dieser Stelle die Werte aus dem Dashboard eingesetzt werden.

context In diesem Fall wird `skipEmptyBuckets` auf `true` gesetzt. Das bedeutet, dass alle Buckets, in denen überhaupt keine Werte liegen ignoriert werden sollen, da diese das Diagramm verfälschen würden, denn sie würden 0 als Wert liefern.

Mit Hilfe dieser Anfrage können also die Werte eines bestimmten Zeitraums aggregiert angezeigt werden.

3.1.2 Range mit Threshold

Die Range-Abfrage mit Threshold ermöglicht die Abfrage einer gegebenen Anzahl an Werten. Diese Anzahl wird als Threshold definiert und kann vom Benutzer angegeben werden. Die Query für diese Abfrage ist wie folgt aufgebaut.

```
{"queryType" : "select",
 "dataSource" : "laotse",
 "granularity" : "all",
 "filter" : {"type" : "selector", "dimension" : "sensor
    ", "value" : <SENSORLIST> },
 "dimensions" : [],
 "intervals" : ["%0"],
 "pagingSpec" : {"pagingIdentifiers" : {}, "threshold" :
     %1}
}
```

Diese Query ist ähnlich aufgebaut wie die im vorherigen Abschnitt beschriebene Query für die aggregierte Range-Abfrage. Statt einer `timeseries`-Query

wird eine `select`-Query genutzt. Der Unterschied zwischen den beiden Abfragen ist, dass die Select-Abfrage die genauen Werte ermittelt und bei der Timeseries-Abfrage die Daten aggregiert werden. Da bei dieser Abfrage die genauen Werte ermittelt werden sollen, wird die Select-Abfrage genutzt. Im Gegensatz zur aggregierten Range Abfrage müssen bei dieser Abfrage nicht die leeren Buckets betrachtet werden. Dafür wird aber der Threshold übergeben, wie viele Werte angezeigt werden müssen. Der Threshold-Wert wird in der `pagingSpec` definiert. In dieser Query steht für den Threshold ein Platzhalter, da der Wert vom Benutzer eingegeben werden kann und somit aus dem Dashboard ausgelesen wird.

3.2 Aggregation Queries

In diesem Abschnitt werden die Queries der anderen Abfragen beschrieben. Da diese sich sehr ähnlich sind wird nur einmal auf die gesamte Query eingegangen und nur die kleinen Unterschiede zwischen den einzelnen Queries beschrieben.

```
{"queryType" : "timeseries",
 "dataSource" : "laotse",
 "filter": {"type": "or", "fields": [<SENSORLIST>]},
 "granularity" : "all",
 "aggregations" : [{"type" : "doubleMin", "name" : "
    value", "fieldName" : "min"}],
 "intervals" : ["%0"]
}
```

Auch für die anderen Abfragen wird die `timeseries`-Query genutzt, da nur aggregierte Werte ermittelt werden sollen. In diesem Beispiel ist dabei die Minimum-Abfrage dargestellt. All diese Queries nutzen die Granularität `all`. Das bedeutet, dass aus allen Daten nur ein Wert ermittelt wird. Damit diese Abfragen richtig funktionieren, muss als Filter ein `Or-Filter` genutzt werden, in dem alle Sensoren der Sensor-Liste eingefügt werden. Ansonsten wird nur der Minimum Wert eines Sensors betrachtet. Als Aggregation muss dann der Anfragetyp gewählt werden, damit Druid bereits automatisiert die Aggregation durchführt. Für die einzelnen Analysen können dabei folgende Aggregationen genutzt werden.

```
Maximum: [{"type": "doubleMax", "name": "value", "
    fieldName": "max"}]

Count: [{"type": "longSum", "name" : "value", "fieldName
    ": "count"}]

Average: [{"type" : "doubleSum", "name" : "sum", "
    fieldName" : "sum"},{"type": "longSum", "name" : "
    count", "fieldName": "count"}]
```

Für alle Anfragen, außer der Average-Anfrage existieren also bereits Aggregationen in Druid, die genau das von der Anfrage geforderte erfüllen. Somit können die Werte aus dem JSON-Ergebnis einfach ausgelesen und dann dem Benutzer visualisiert werden. Eine Durchschnittsberechnung ist in Druid nicht

möglich. Daher müssen für die Durchschnittsberechnung sowohl die Summe aller Werte, als auch die Anzahl der Werte berechnet werden, um aus diesen beiden Werten den Durchschnitt zu berechnen.

3.3 DAC-Queries

Das Data Availability Chart (DAC) zeigt an, an welchen Tagen bzw. in welchen Stunden-Intervallen der gegebene Sensor Daten geliefert hat. Es wird also für ein gegebenes Intervall abgerufen, ob Daten da sind und dann für den Benutzer ein Diagramm gezeichnet um das Ganze zu visualisieren. Die Query für diesen Anwendungsfall ist wie folgt aufgebaut:

```
{"queryType" : "groupBy",
 "dataSource" : "laotse",
 "granularity" : "day",
 "dimensions" : [],
 "filter" : {"type" : "selector", "dimension" : "sensor
    ", "value" : "sensorid" },
 "aggregations" : [{"type" : "longSum", "name" : "count
    ", "fieldName" : "count"}],
 "intervals" : ["interval"],
 "context" : {"skipEmptyBuckets": "true"}}
```

Für diese Abfrage wird eine `GroupBy`-Query genutzt. Dadurch werden die abgerufenen Daten gruppiert. Im Grunde genommen sieht die Query bei dieser Abfrage fast genauso aus, wie bei den im vorherigen beschriebenen `timeseries`-Queries. Allerdings wird in diesem Fall beim Filter nicht eine Sensorliste, sondern nur die einzelne Sensorid übergeben. Außerdem müssen auch bei dieser Query wieder die leeren Buckets außer Acht gelassen werden. Da im DAC sowohl die Tage, als auch die Stunden eines Tages, in denen ein Sensor Daten geliefert hat angezeigt werden können, existieren für diesen Anwendungsfall zwei Queries. Einmal muss die Granularität `day` sein, wenn die Tage angezeigt werden sollen und im anderen Fall muss die Granularität `hour` sein, wenn die Stunden angezeigt werden sollen. Die Übergabe des Intervalls wird aus den Properties entnommen und dann eingesetzt.

Kapitel 4

Datenbank-TestSetup

DN

Zum Testen von Laotse können bestimmte Datenbankinhalte benötigt werden. Der `LaotseDbWrapper` bietet eine Klasse `TestSetup`, die eine Datenbank zurücksetzt und mit Standardwerten füllt.

Damit das TestSetup ausgeführt werden kann, wurde eine Applikation im OSGi-Bundle `de.offis.laotse.test.LaotseDbTestsetup` entwickelt. Es enthält eine Product-Beschreibung, das die Anwendung `TestSetupApplication` startet. Die Anwendung ruft wiederum über die Klasse `TestSetupDbAccess` die Zugangsdaten zur Datenbank ab und führt das TestSetup durch.

Für den Zugriff auf die Datenbank werden die folgenden Attribute in die Klasse `TestSetupDbAccess` eingetragen:

USER Der MySQL-Nutzername, der Zugriffsrechte auf die Konfigurationsdatenbank hat.

PASSW Das Passwort des MySQL-Nutzers

SENSOR_DB_URL Die URl, unter der die Datenbank erreichbar ist. Beispielsweise

```
jdbc:mysql://<hostname>/<databasename>
```

Das TestSetup kann ausgeführt werden, indem die Daten *LaotseDbTestsetup.product* in Eclipse geöffnet wird. Mit Klick auf „ Synchronize“ und dann „Launch an Eclipse Application“ wird das TestSetup ausgeführt.

Kapitel 5

Datengenerator bauen

Dieser Abschnitt erläutert, wie neue Datengeneratoren mithilfe das Datengenerator-Frameworks [**?**] aus dem Odysseus-Projekt ertellt werden können. Damit der Datengenerator entwickelt werden kann, müssen die Eclipse-Projekte `de.uniol.inf.is.odyssseus.generator`[1] und `org.apache.commons.math3`[2] in den Eclipse-Workspace importiert werden. Zum Zugriff auf das Repository sei auf die entsprechende Seite im Odysseus-Wiki verwiesen[**?**].

Der neue Datengenerator wird als neues Java-Projekt unter **File** → **New** → **Project...** → **Java** → **Java Project** erzeugt. Im geöffneten Dialog wird nach Eingabe eines Namens und Auswahl des Ordners, in dem die Projektdateien abgelegt werden sollen, auf **Next** geklickt. Im folgenden Dialog wird unter dem Reiter **Projects** das Projekt `de.uniol.inf.is.odyssseus.generator` hinzugefügt. Ein Klick auf **Finish** erstellt das Projekt und schließt den Dialog.

Als nächstes wird in dem neuen Projekt eine Klasse erzeugt, die einen Datengenerator implementiert. Die Klasse heiße hier zum Beispiel `SampleGenerator`. Diese soll von der Klasse `AbstractDataGenerator` erben. Für die weitere Implementierung der Klasse wird auf das Odysseus-WIki verwiesen [**?**].

Damit der Datengenerator gestartet werden kann, wird eine Klassen benötigt, die einen `StreamServer` erstellt und startet. Der folgende Ausschnitt zeigt eine Main-Methode, die den Datengenerator erzeugt und startet. Der `StreamServer` bindet sich dabei an Port 12345.

```
1  public static void main(String[] args) {
     try {
3      StreamServer server = new StreamServer(12345, new
           SampleGenerator(), false);
       server.start();
5    } catch (Exception ex) {
       ex.printStackTrace();
7    }}
```

[1] http://isdb1.offis.uni-oldenburg.de/repos/odysseus/trunk/application/generator/generator.base/

[2] http://isdb1.offis.uni-oldenburg.de/repos/odysseus/trunk/common/resources/org.apache.commons.math3/

Der Datengenerator kann mit Klick auf **File** → **Export...** → **Java** → **Runnable JAR file** → **Next** exportiert werden. Dazu ist ein Dateipfad und -name und die entsprechende Launch-Configuration, die erzeugt wird, wenn die Starter-Klasse einmal in Eclipse ausgeführt wurde, zu wählen. Es empfiehlt sich, die gepackten Abhängigkeiten in das JAR-Archiv legen zu lassen.