

Konzeption und prototypische Entwicklung einer Webanwendung zur Aggregation und Analyse wissenschaftlicher Daten des Argo-Projektes

Sebastian Schmid

S0543196

Prof. Dr. Christin Schmidt

Prof. Dr.-Ing. Hendrik Gärtner

Bachelorarbeit zur Erlangung des akademischen Grades: Bachelor of
Science. (B.Sc.)

Fachbereich Wirtschaftswissenschaften II

Studiengang Angewandte Informatik

an der Hochschule für Technik und Wirtschaft Berlin

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum

Unterschrift

Danksagung

An dieser Stelle möchte ich meiner Betreuerin Frau Prof. Dr. Christin Schmidt danken. Diese unterstützte mich stets tatkräftig und half mir, diese Abschlussarbeit auszuarbeiten. Auch Herrn Prof. Dr.-Ing. Gärtner möchte ich dafür danken, dass er sich dazu bereit erklärt hat, diese Arbeit zu betreuen.

Außerdem möchte ich auch meinen Eltern danken. Auch wenn diese heute nicht mehr unter uns sind, so waren sie es doch, die den Keim gelegt haben, der mich dazu befähigte den Weg bis zu dieser Stelle zu gehen.

Zu danken habe ich auch Robert M. Pirsig. Sein Buch „Zen und die Kunst ein Motorrad zu warten.“ begleitete mich bei der Entwicklung dieser Arbeit und inspirierte mich immer wieder aufs Neue. Ich könnte mir kein besseres Buch als Begleitung für eine schriftliche Ausarbeitung vorstellen.

Zuletzt und doch ganz besonders richtet sich mein Dank an Denise Peth. Diese begleitete mich auf dem Weg dieser Ausarbeitung und stand mir stets Rat und Tat zur Seite.

Inhalt

Abbildungsverzeichnis	VI
Tabellenverzeichnis	VI
1 Einleitung	1
1.1 Existierende Lösungen	2
1.1.1 Das JCOMMops	2
1.1.2 Data Selection and Visualization Tool des Coriolis GDAC	3
1.2 Alleinstellungsmerkmal & Abgrenzung	3
2 Datengrundlage	4
2.1 Das Argo Programm	4
2.1.1 Daten des Argo Programmes	5
2.1.2 Datenformate	5
2.1.3 Datenstruktur	6
3 Anforderungsanalyse	7
3.1 Systembeschreibung	7
3.2 Ermittlung der Anforderungen	7
3.3 Funktionale Anforderungen	8
3.3.1 Anwendende Perspektive	8
3.3.2 Administrative Perspektive	9
3.4 Nicht-Funktionale Anforderungen	9
3.4.1 Anwendende Perspektive	9
3.4.2 Administrative Perspektive	9
3.5 Benötigte Daten	10
3.6 Technische Anforderungen	11
3.6.1 Verwendete Programmiersprachen	11
3.6.2 Betriebssystem des Servers	11
3.6.3 Webframework	12
3.6.4 Datenbank	13
4 Systementwurf	14
4.1 Modellierung der Datenbank	14
4.2 Architektur	14
4.2.1 Entwurf der Datenaggregation	14
4.2.2 Entwurf der Webapplikation	16

4.2.3	Ausarbeitung der Webrouen	16
4.2.4	Aussehen der Webapplikation	17
5	Grundlagen	19
5.1	Objekt-Relationale Unverträglichkeit	19
5.2	Geographische Informationssysteme ODER	20
5.3	Verwendete Software	20
5.3.1	Network Common Data Format	20
5.3.2	numpy	21
5.3.3	Julian Date	21
5.3.4	Flask	21
5.3.5	JSON?	23
5.3.6	openLayers	23
5.3.7	D3	24
5.4	...	24
6	Implementierung	25
6.1	Datenaggregation	25
6.2	Webapplikation	28
6.2.1	Objektrelationales Mapping	28
6.2.2	Controller	35
6.2.3	Templates	36
6.2.4	Kartendarstellung	37
6.2.5	Zeichnen der Graphen	37
7	Testen	38
8	Demonstration und Auswertung	39
9	Anhang	40
	Quellenverzeichnis	41

Abbildungsverzeichnis

1	Argos Messzyklus - Bildquelle: http://www.argo.ucsd.edu	4
2	Der Werdegang der Argo-Daten - Bildquelle: http://www.argo.ucsd.edu .	5
3	Use case Diagramm der Anforderungen	7
4	Anforderungen und ihre Abhängigkeiten als gerichteter Graph	8
5	Beschreibung der Entitäten der Datenaggregation	14
6	Entwurf der Architektur der Aggregation der Daten	15
7	Grafischer Grobentwurf der Webapplikation	18
8	Architekturbeschreibung	25

Quellcodeverzeichnis

1	test	6
2	Webrouen der API	16
3	Webrouen der APP	17
4	Extraktion und Berechnung des Erstellungsdatums eines NetCDF-Datensatzes	20
5	Minimalbeispiel für die Verwendung von Flask	21
6	Implementierung des Kontextmanagers zur Sicherstellung der richtigen Dateibehandlung.label	26
7	Implementierung des Iterators zur Steuerung der Aggregationssequenzlabel	26
8	Verwendung der Schnittstelle zur Steuerung der Datenaggregationlabel .	27
9	Factory zur Extraktion der Datensätze.	27
10	Modellklasse für ein Argo-Float	28
11	Modellklasse für eine Messung	29
12	Das Schreiben der Daten eines Argo-Floats in die Datenbank	30
13	Anfragen über SQLAlchemy zum Lesen von Daten	31
14	Das Ausführen von SQL Anfragen über SQLAlchemy	32
15	webapp.models.__init__.py	33
16	webapp.__init__.py	35
17	webapp.templates.map.html	36

Tabellenverzeichnis

1	Beschreibung der ausgewählten Daten	10
---	---	----

1 Einleitung

Die im Jahre 1965 von Gordon Moore vorhergesagte Gesetzmäßigkeit, die Komplexität, und damit die Speicherdichte integrierter Schaltkreise, würde sich regelmäßig verdoppeln (vgl. [Moo98]), hat sich seit dem bis zum heutigen Tag bewahrheitet. Damit sieht sich die Menschheit über 50 Jahre später in einer einmaligen Lage. Zum einen verfügen wir über eine noch nie dagewesene Ansammlung an Informationen, zum anderen wurden die Speichermedien durch die wachsende Komplexität immer flüchtiger und schwieriger in der technischen Handhabung. Werden kommende Zivilisationen in der Lage sein, diesen Pool an Informationen für sich zu nutzen oder sollten wir vielmehr annehmen, dass wir eine einmalige Chance haben, die wir nicht vergeuden sollten?

In diesem Kontext scheint es als folgerichtig, dass Bewegungen wie Freie Software, Creative Commons und Open-Access die Hürden für den Zugang zu den Informationen weiter verringern. Viele der Informationen stehen heute frei zur Verfügung und können genutzt werden und doch werden, von einem gefühlt immer größeren Anteil unseres Kulturkreises, Prinzipien zur Bewertung von Quellen abgelehnt. In einer Zeit in der es einfacher den je ist, Fakten zu überprüfen, werden wissenschaftlich bewiesenen Aussagen wie dem Klimawandel einfach nicht geglaubt. Es scheint als wären viele Menschen der Flut an Informationen überdrüssig, als wende sich ein großer Teil überfordert davon ab. Die Frage ist, was kann dazu beitragen dieses Potential mehr zu nutzen? Über welche Mittel verfügt die Informatik, Daten in einen Kontext einzubetten? Wie muss der Kontext gestaltet sein, über die Menschen an wissenschaftliche Arbeit herangeführt werden können? Welche Daten sind geeignet, um die Brisanz und das Potential unserer Zeit einem breiteren Publikum zuzuführen? Mit diesen Fragestellungen motiviert sich die hier vorliegende Arbeit.

Im Rahmen dieser Abschlussarbeit wurde, über die Aggregation wissenschaftlicher Daten, ein exploratives Werkzeug geschaffen, um diese intuitiv erfahrbar zu gestalten. Dies geschieht mit der Hoffnung, dass hier eine Identifikation mit den Messwerten und des wissenschaftlichen Prozesses zu erreicht werden kann. Als Datengrundlage wurden Daten des Argoprogramms verwendet. Unter dem Dach dieses Projektes werden seit dem Beginn dieses Jahrtausends das Wasser der Weltmeere nach den Parametern Temperatur, Salzgehalt und Leitfähigkeit untersucht. Diese Daten stehen unter einer freien Lizenz zur Verfügung und können in eigene Projekte eingebunden werden. Die Messdaten dienen Wissenschaftlern, die Auswirkungen des globalen Klimawandels zu untersuchen. Die Herausforderung besteht darin, die hochkomplexen Messdaten in einer Form aufzubereiten, so dass diese intuitiv erfahrbar werden.

Ein solches Werkzeug könnte zum Beispiel in Schulklassen eingesetzt werden. Die Ler-

nenden könnten, innerhalb einer Kontextvorgabe durch den Lehrenden, die Auswirkungen des Klimawandels auf die Weltmeere untersuchen. Hier könnte auch das Interesse und ein eventueller Berufswunsch im Wissenschaftssektor geweckt werden. Die Applikation richtet sich aber nicht explizit an Heranwachsende. Erwachsene Personen könnten sich hier, eigenes Interesse vorausgesetzt, weiterbilden. Die Applikation kann hier Relevanz der Forschung und die in diesem Programm erhobenen Messwerte erfahrbar machen.

Der schriftliche Teil dieser Arbeit handelt von der Konzeption und Entwicklung eines Prototypen mit der oben genannten Problemstellung. Zu Beginn wird das Argo Programm näher vorgestellt. Es wird auf den Prozess der Datenerhebung und Aufbereitung eingegangen. Im Anschluss daran werden die Anforderungen der Software ausgearbeitet, um im Anschluss darauf gangbare Lösungsansätze und die geeignetsten Werkzeuge ermittelt. Im Anschluss darauf wird das Softwaresystem entworfen. Es werden die geplante Architektur und die wichtigsten Geschäftsprozesse ausgearbeitet und beschrieben. Über das Aufzeigen von gangbaren Alternativen werden die passendsten Werkzeuge und Entwicklungsmuster für diese Problemstellung ermittelt. Nach der Planung folgt eine Beschreibung der vorgenommenen Implementierung vorgenommen. Auch hier werden Implementierungen iterativ verfeinert, so das die passendste Lösung gefunden werden kann.

Um die Qualität der Software beschreiben zu können, folgt im darauf folgenden Kapitel eine Beschreibung der Verwendeten Test-Verfahren. Hier werden Metriken ausgearbeitet, welche erlauben funktionale als auch nicht-funktionale Anforderungen überprüfbar messen zu können. Die Messung erfolgt dabei über Funktions-, wie Unittests, als auch über eine Umfrage, zur Ermittlung der erreichten Benutzbarkeit.

Im Abschluss findet eine Präsentation des entwickelten Prototypen statt. Dabei wird ein möglichst kritischer Blick auf Problemstellung und Lösungen geworfen. Abschließend wird ein Ausblick auf Alternativen und mögliche Weiterentwicklungen der Prototypen gegeben.

1.1 Existierende Lösungen

1.1.1 Das JCOMMops

Das Joint Technical Commission for Oceanography and Marine Meteorology (JCOMM) bietet mit jcommops.org eine Grundlage für die wissenschaftliche Arbeit unter anderem mit den von Argo gesammelten Daten. Neben der Darstellung von Karten erfährt der Nutzer hier, von Sensordaten über den Bautyp der jeweiligen Boje alles was die Bojen zu erzählen haben. Daneben werden über diese Plattform auch redaktionelle Reports veröffentlicht, um der Leserschaft ein Bild der aktuellen Lage unserer Weltmeere zu vermitteln. Über einen Twitter Account werden Änderungen an der Plattform und neu veröffentlichte

Reports veröffentlicht.

Zwar bietet die Plattform durch ihre kartenbasierte explorative Darstellung ein ähnliches Angebot, wie es in ArgoData gemacht wird. Die schiere Fülle der Parameter erfordert vom Benutzer aber die Bereitschaft sich vertieft in die Materie einzuarbeiten. Damit richtet sich das Angebot des Global Ocean Observing Systems an Wissenschaftler und Journalisten. Diesen dient sie als eine hervorragende Datengrundlage für deren Veröffentlichungen. Ein Benutzer aus der hier genannten Zielgruppendefinition wird von einem derartigen Angebot wohl eher abgeschreckt sein, bevor er dessen Vorteile für sich erarbeiten konnte.

1.1.2 Data Selection and Visualization Tool des Coriolis GDAC

Das Data selection and visualization tool der Coriolis GDAC ¹ ermöglicht den Download und von nach verschiedenen Filterkriterien ausgewählten Datensätzen. Ein Betrachter erlaubt zusätzlich Werte einer spezifischen Treibboje anzusehen und ihren bisher zurückgelegten Weg nachzuvollziehen.

Auch wenn die Darstellung hier mit weniger Parameter auskommt, so ist die primäre Aufgabenstellung dieser Plattform wohl das Extrahieren der Daten um diese in einem wissenschaftlichen Paper verwenden zu können. Neben der akademischen Verwendung müsste ein Benutzer aber auch hier viel Neugierde und Zeit für die Einarbeitung in das Thema mitbringen um aus den angebotenen Daten einen Mehrwert für sich zu generieren.

1.2 Alleinstellungsmerkmal & Abgrenzung

Das Ziel dieser Arbeit ist die Darstellung von wissenschaftlichen Daten aus dem Argo-Programm. Im Gegensatz zu den bereits vorhandenen Lösungen sollen die Daten aber nicht zur wissenschaftlichen Verwendung aufbereitet werden. Vielmehr sollen die hochkomplexen Daten auf ein einfach erfahrbares Maß herunter gebrochen werden. Zum Zeitpunkt dieser Arbeit existiert noch kein exploratives Tool für Daten des Argo Programms mit diesem Ansatz.

¹siehe [Arg17c]

2 Datengrundlage

2.1 Das Argo Programm

Zum Ende des vergangenen Jahrtausends verdichteten sich die Hinweise auf einen globalen und durch Menschen verursachten Klimawandel. Um dessen Auswirkungen auf die Weltmeere studieren zu können, wurde unter dem Dach des Global Ocean Observing System das Argo Programm gegründet. Dieses sollte, unterstützt durch das Satellitensystem Iason die Wassersäule der oberen 2000m auf deren chemischen Eigenschaften untersuchen. Dabei werden in ständigen Intervallen Salzgehalt, Druck, Temperatur und (...) gemessen. Die ermittelten Daten werden veröffentlicht, so dass diese durch Wissenschaftler ausgewertet werden können. Zu diesem Zeitpunkt existieren bereits XXYYZZ Publikationen, die sich mit diesen Daten befassen.

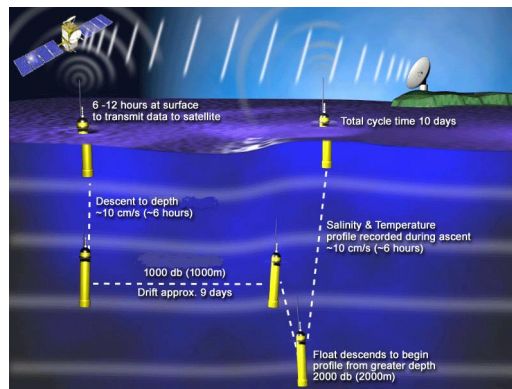


Abbildung 1: Argos Messzyklus

Die Argo Treibbojen werden mit Schiffen an spezifischen Punkten ausgesetzt. Ein Messzyklus beträgt 10 Tage. Die Boje taucht am Anfang des Zyklus auf 1000 Meter Tiefe herab. In dieser Tiefe verbringt die Sonde die nächsten 9 Tage. Anschließend sinkt sie auf die Maximale Tiefe von 2000 Metern herab um anschließend wieder zur Oberfläche aufzusteigen. An der Oberfläche sendet die Boje innerhalb von 6-12 Stunden die Daten über Iason an die Bodenstationen.

Nach der Erhebung werden die Daten auf deren Plausibilität und Qualität überprüft.² Nach diesem Prozess werden die aufbereiteten Daten über die Globalen Data Assembly Center in monatlichen Releasezyklen veröffentlicht.³ Der Werdegang der Daten ist in Abbildung 2 zu sehen.

²Siehe [SG] Seite 3

³Siehe [Arg]



Alle am Argo teilnehmenden Organisationen verpflichten sich auf eine gemeinsame Datenpolitik. So gibt es keine Herrschaft über die erhobenen Daten. Vielmehr stehen diese ab dem Zeitpunkt der Veröffentlichung transparent der Öffentlichkeit zur Verfügung. Nachdem die Daten von Iason aus dem Argo Array an die Data Assembly Centers übermittelt hat, werden diese einer Qualitätskontrolle unterzogen. Die Daten werden auf Plausibilität und Abweichungen überprüft. Nach dieser Qualitätskontrolle werden die Daten nun über die Global Data Assembly Centers in Frankreich und den USA released. Diese können dann über HTTP und FTP abgerufen werden.

Ein mal Im Monat werden die daten als Snapshots, den sogenannten DOI (digital object identifier) released.

Während und vor der Qualitätskontrolle liegen die Daten im TESAC und BUFR Format vor.

Die von den DGACs veröffentlichten Daten stehen dann im Format NetCDF vor. Diese sind unter der Lizenz Attribution 4.0 International (CC BY 4.0) veröffentlicht und dürfen unter der Nennung der Lizenz frei verwendet und dabei auch verändert werden.

5

2.1.3 Datenstruktur

In Listing 1 ist die Ordner Struktur der netCDF Dateien zu erkennen. Über den Ordnernamen aoml ist die Herkunft des DOI zu erkennen. In diesem Fall wurden die Dateien von einem Server der Atlantic Oceanographic & meteorological Laboratory erstellt. Hier befindet sich für jede Messboje ein Unterordner. In diesem Fall sind die Ordner der Bojen 1900200 und 1900201 zu sehen. Die Dateien meta, prof, Rtraj und tech sind eine Quelle für Meta-informationen. (TODO Was steht in den metadaten?) Die Messprofile einer Boje finden sich im Ordner profiles. Hier wird für jeden messzyklus eine Datei angelegt. Dieser startet bei 1 und inkrementiert über jeden Messzyklus um 1.

```
1 ./aoml/1900200/  
2 - 1900200_meta.nc  
3 - 1900200_prof.nc  
4 - 1900200_Rtraj.nc  
5 - 1900200_tech.nc  
6 - profiles  
7   - D1900200_001.nc  
8   ...  
9   - D1900200_215.nc  
10  - D1900200_216.nc  
11 ./aoml/1900201/  
12 ...
```

Listing 1: test

3 Anforderungsanalyse

3.1 Systembeschreibung

Die zu entwickelnde Anwendung verfolgt einen datengetriebenen Ansatz. Daten aus dem Argo-Programm werden erhoben und eine Auswahl davon in ein Datenformat überführt. Zur Darstellung wird eine Webapplikation verwendet. Über diese werden über Positionsmarker auf einer Weltkarte, die letzten Positionen der Messstationen angezeigt. Über einen Klick auf die Repräsentation einer Messboje werden weitere Daten der Messstation präsentiert.

3.2 Ermittlung der Anforderungen

Ausgehend von einem ersten Prototypen wurden Use-Cases entwickelt (Siehe Abbildung 3). Dabei wurden zwei Klassen von Akteuren herausgearbeitet. Darunter finden sich die Benutzenden, also die eigentlichen Anwender der Applikation. Diese erwarten einen reibungslosen Betrieb der Applikation und die Darstellung der Messstationen in der gegebenen Form. Zum anderen existiert die Klasse der Administrierenden. Diese benötigen Werkzeuge um die Daten zu restrukturieren und in die Datenhaltung einzuspielen. Beide Klassen haben verschiedene Sichten auf die Datenhaltung. Daraus entsteht ein Spannungsverhältnis, welches minimiert werden muss.

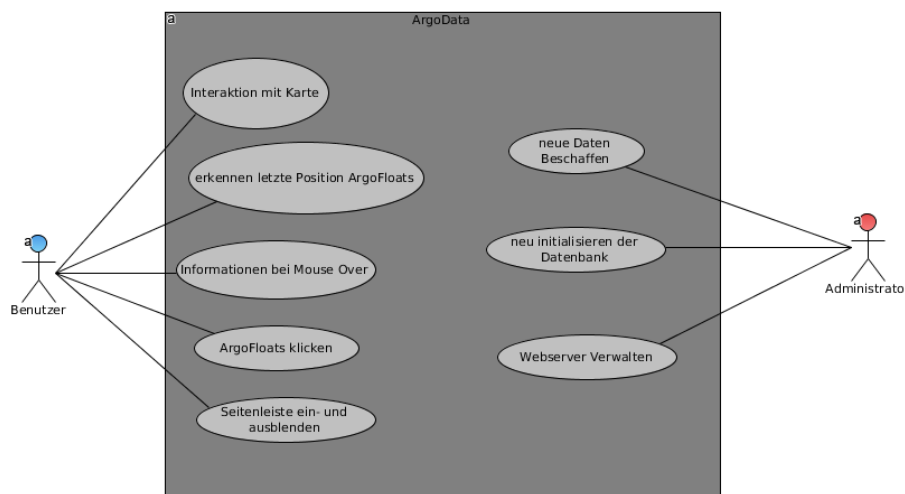


Abbildung 3: Use case Diagramm der Anforderungen

Anhand dieses Modelles wurden Anforderungen ausgearbeitet. Diese wurden in Form eines gerichteten Graphen aufbereitet (Siehe Abbildung: 4). Dabei werden die Anforderungen und deren Nachbedingungen über Knoten des Graphen repräsentiert. Finden sich in einer Anforderung Vorbedingungen zu den Nachbedingungen einer anderen Anforderung, so

wird diese Abhängigkeit über eine Kante dargestellt. Diese Darstellung half dabei, Definitionslücken in den Beziehungen der Anforderungskette besser verstehen zu können und formale Lücken schließen zu können.

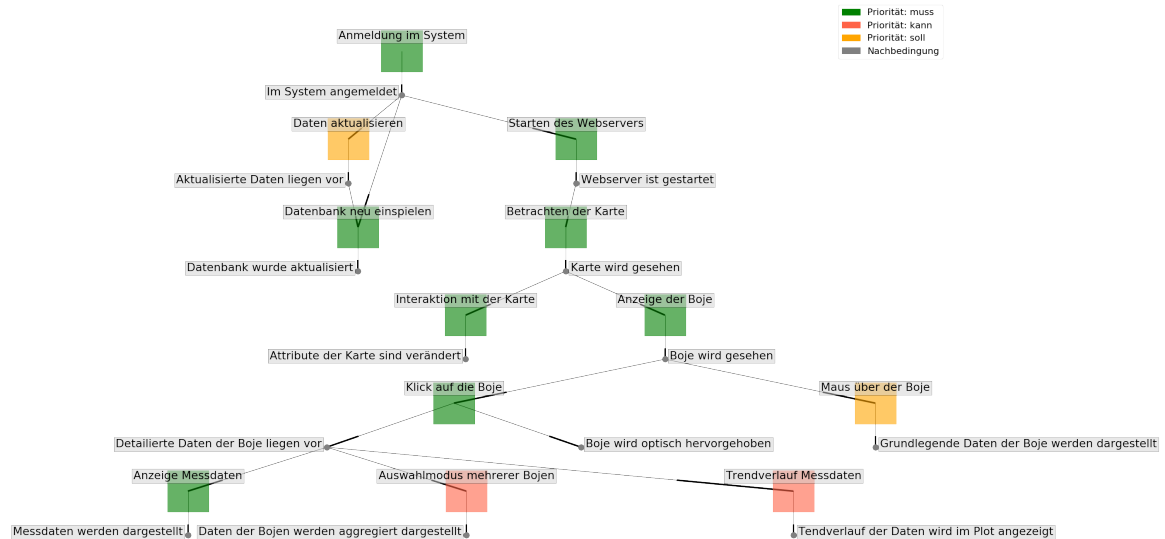


Abbildung 4: Anforderungen und ihre Abhängigkeiten als gerichteter Graph

3.3 Funktionale Anforderungen

3.3.1 Anwendende Perspektive

In der Anwendung wird die Darstellung einer Webapplikation erwartet. Angesteuert durch die zugehörige URL sollen über eine Kartenapplikation die Grenzen der Ozeane und der umliegenden Kontinente erkennbar werden. Eine genauere Darstellung von Landmarken wie Straßen, Städte oder Gebirge ist für den Aussagewert der Applikation nicht erforderlich. Die Kartendarstellung soll über Interaktionsmöglichkeiten, wie dem Einstellen der Zoomstufe, sowie des ausgewählten Kartenbereichs, verfügen. Über den Kartendienst sind die letzte Position jeder Messboje aus dem Datensatz ersichtlich. Um Streuung und Verdichtung an spezifischen Orten sehen zu können, werden alle Bojen des angezeigten Kartenausschnittes dargestellt. Es erfolgt keine Zusammenfassung oder Ausblendung. Wird die Maus über die visuelle Repräsentation einer Messboje geführt, so sollen grundlegende Daten der Messstationen dargestellt werden, gleichzeitig wird diese Station optisch hervorgehoben. Ein Mausklick auf die schematische Darstellung soll hier gemessene, spezifische Daten anzeigen. Dazu werden in einem separierten Darstellungsfeld der Verlauf der Messdaten, sowie weiterer sinnvoller Werte angezeigt. Zusätzlich soll der zurückgelegte Weg der Messtation ersichtlich sein. Eine Trendanalyse der Daten könnte hier ein Maß

für die Richtung der Messwerte ermöglichen. Könnten mehrere Messstationen gleichzeitig ausgewählt werden, so wäre es sinnvoll die akkumulierten Werte darzustellen.

3.3.2 Administrative Perspektive

Es notwendig, dass die Datensätze des Argo Programms ausgelesen und in ein für die Datenbank aufbereitetes Format überführt werden. Neue Daten müssen hierfür aus den Quellen des Argo Programms heruntergeladen werden. Um die Inhalte zu erstellen, werden statt einer journalistischen eine administrative Tätigkeiten erwartet. Durch den Datenge-triebenen Ansatz der Applikation müssen für das Erstellen der Inhalte keine Webmasken wie Texteingabefelder zur Verfügung gestellt werden. Vielmehr ist es hier sinnvoll, die hier benötigten Werkzeuge als Skripte zur Verfügung zu stellen, um die administrativen Aufgaben über die Kommandozeile ausführen zu können. Dies würde eine Abbildung der Prozesse bis hin zu einer Vollautomatisierung erlauben und eine örtliche Trennung von Datenaggregation und -darstellung vereinfachen. Für die administrativen Tätigkeiten werden Werkzeuge benötigt, um neue Daten in die Datenbank überführen können.

3.4 Nicht-Funktionale Anforderungen

3.4.1 Anwendende Perspektive

Für die Benutzenden sind Usability und Benutzbarkeit der wichtigste Aspekt. Die Bedienungsmuster müssen klar ersichtlich und intuitiv erfahrbar sein. Die Darstellung der Applikation soll einfach und schlicht gehalten werden und dem gewohnten Erscheinen modernen Web Applikationen entsprechen. Es soll Wert auf die wichtigsten Aspekte des Datenschutzes Wert gelegt werden. Insbesondere im Hinblick auf Datensparsamkeit sind das Einbinden von Trackern, Loginfunktionen und das Speichern von Logfiles zu bewerten. Sicherheitsaspekten ist aus der Anwendersicht nur eine normale Rolle zugewiesen werden. Dabei sollen die heute normalen Sicherheitsvorkehrungen, wie zum Beispiel Transportverschlüsselung und Schutz gegen SQL-Injections erfolgen.

3.4.2 Administrative Perspektive

Für die Aggregation der Daten ist die Sicherheit ein zentraler Aspekt. Dies umfasst sowie Aspekte von Authentifizierung und Autorisierung der Rolle der Datenaggregation als auch eine Sicherheit um die Integrität der Heruntergeladenen Daten.

Datenfeld	Beschreibung	Datentyp
PLATFORM_NUMBER	Eindeutige Identifikationsnummer einer Messstation	<STRING8>
CYCLE_NUMBER	Fortlaufende und innerhalb einer Messboje eindeutige Identifikationsnummer eines Messprofils	<int32>
JULD	In Julian Date codierter Zeitpunkt der Übertragung (Begin oder ende?) einer Messung	<float64>
N_PARAM	Anzahl der Messsensoren.	<STRING16>
LATITUDE & LONGITUDE	Die Positionsdaten einer Messung zum Zeitpunkt der Übertragung der Werte.	<float64>
PRES	Messvektor des Wasserdrucks	<float32>
TEMP	Messvektor der Wassertemperaturen	<float32>
PSAL	Messvektoren des Salzgehaltes	<float32>

Tabelle 1: Beschreibung der ausgewählten Daten

3.5 Benötigte Daten

Um die Darstellung zu vereinfachen und die Kommunikation mit der Datenbank zu beschleunigen, ist im ersten Schritt eine Auswahl aus den Daten der Argo-Bojen vorzunehmen. Hierbei soll darauf geachtet werden, nur diejenigen Daten zu verwenden, welche für die Erbringung des Dienstes notwendig sind.

Aus diesem Grund findet sich im Folgenden eine Auswahl aus dem Datenkatalog ⁴ von Argo mit der jeweiligen Begründung:

Das Feld PLATFORM_NUMBER wird benötigt um Messprofile eindeutig der Plattform zuordnen zu können. Während die CYCLE_NUMBER eine Messung innerhalb ihrer Plattform eindeutig zuordenbar macht. Das Datum der Messdatenübertragung kann aus dem Feld JULD ermittelt werden, dieses erlaubt es, die Messreihe in ihren zeitlichen Kontext zu sehen. Die Anzahl der Sensoren lässt sich aus dem Feld N_PARAM herleiten. Dies wird benötigt, da einige Messstationen nicht alle Sensoren eingebaut haben. Über die Geokoordinaten wird die lokale Zuordnung der Datenübertragung erreicht. Dies wird benötigt um in der Kartendarstellung der Messung einen Ort zuzuweisen. Aus den Messwerten werden die Felder PRES, TEMP und PSAL ausgewählt. Neue Modelle von Argo Floats besitzen noch das Feld COND über den sich die Leitfähigkeit des umliegenden Wassers herleiten lässt. Dieser ist aber noch nicht bei vielen Messstationen implementiert und wird deswegen hier nicht verwendet.

⁴Siehe Quelle [CKT⁺15] S. 19 ff.

Vereinfachung der Messdaten Die verwendeten Messwerte PRES, TEMP und PSAL eines Messprofils liegen in Vektorieller Form vor. Für die Darstellung über einen univariaten Graphen wird nur ein skalarer Wert pro Messprofil benötigt. Die Daten sollen zusammengefasst werden, bevor diese in die Datenstruktur überführt werden um die Größe der Daten zu verringern.

3.6 Technische Anforderungen

3.6.1 Verwendete Programmiersprachen

Die Darstellung der Webapplikation wird mit HTML und Javascript umgesetzt werden. Diese Sprachen gelten in der Entwicklung von Webseiten als Standard und werden von den gängigen Browsern unterstützt.

Die weiteren teile der Applikation soll mit einer Sprache entwickelt werden, die alle benötigten Teilbereiche umsetzen kann.

Das **Öffnen der netCDF** Dateien und die numerische Berechnung kann mit C, Java/Scala, R und Python erfolgen. Insbesondere die beiden letzten sind in der Datenverarbeitung und Numerik als etablierte Werkzeuge zu sehen.

Die Darstellung der Seite soll durch ein Webframework unterstützt werden. Hier gibt es in beinahe allen Hochsprachen entsprechende Werkzeuge. Wählt man aus der Problemstellung der numerischen Verarbeitung R und Python heraus, so ist die Auswahl der geläufigen Webframeworks bei Python höher anzusehen. Diese Sprache besitzt eine große Anzahl von Bibliotheken für die Anbindung von Datenbanken, sowie einige etablierte Bibliotheken für die Schaffung von Webapplikationen.

Python besitzt in Hinsicht auf Laufzeitkosten und einer nicht strikten Typisierung einige Nachteile gegenüber Sprachen wie C und Java. In Hinsicht die auf Auswahl von Programmbibliotheken, sowie der numerischen Berechnung besitzt die Sprache aber Vorteile und wird deswegen hier für die Implementierung genutzt.

3.6.2 Betriebssystem des Servers

Für die Laufzeitumgebung der Applikation wird ein GNU/Linux eingesetzt werden. Die Software wird so entwickelt das sie unter den gängigen Distributionen lauffähig sein wird. Für diese Applikation wird sich hier für die Distribution Debian-Stable entschieden. Diese zeichnet sich insbesondere gegenüber weiteren Distribution mit einer ähnlichen Verbreitung wie RHEL und Ubuntu durch ihre nicht kommerzielle Ausrichtung aus. Debian ist sehr verbreitet und stark auf Stabilität ausgelegt. Als Nachteil ist der fehlende kommerzielle Support zu sehen. Diese Arbeit sollte voraussichtlich aber in keine Bereiche vordringen, welche einen kostenpflichtigen Support rechtfertigen würde.

3.6.3 Webframework

Für Python gibt es eine große Anzahl an Werkzeugen für die Entwicklung von Webseiten. Hier werden exemplarisch 3 herausgegriffen und für die hier vorliegende Aufgabe bewertet.

Django wird beworben als The web framework for perfectionists with deadlines.

Es wurde 2005 unter einer BSD Lizenz released und entwickelt, die News-Seite des *Lawrence Journal-World* umzusetzen und zu verwalten. Django ist ein etabliertes und häufig verwendetes Webframework. Es ist dynamisch einsetzbar und für eine große Anzahl von Anwendungen verwendbar. Die Bibliothek ist außerdem durch Module erweiterbar. Die Software folgt dem „*batteries included*“ Ansatz und liefert in der Grundausstattung bereits alles nötige mit, um eine Webseite inklusive Login und Eingabemasken für journalistische Tätigkeiten auszubauen.

Flask ist ein sogenanntes Micro-Framework. es verwendet die Toolsammlung *Werkzeug* um Webseiten darstellbar zu machen. Das Framework folgt dem KISS Ansatz und liefert im Grundumfang nur diejenigen Werkzeuge, die man für die Verwaltung von einfachen Webseiten benötigt. Die Software lässt sich über Module erweitern. Einzelne Teilprojekte von Applikationen lassen sich in sogenannte Blueprints modularisieren.

Falcon ist ebenso als Microframework zu sehen. Es wurde insbesondere in Hinblick auf Geschwindigkeit entwickelt. Das Framework erlaubt requests asynchron zu verarbeiten und lässt die auf Geschwindigkeit spezialisierte Python-Laufzeitumgebung pypy zu. Falcon ist ein relativ neues Framework und erfährt in letzter Zeit zur Schaffung von REST-APIs immer mehr Aufmerksamkeit. Es ist aber auch möglich, mit diesem Framework Webapplikationen mit einer Anzeige über HTML-Elementen zu gestalten.

In dieser Applikation wird die Schaffung von Journalistischen und redaktionellen Inhalten eine sehr geringe Rolle spielen. Unter diesem Gesichtspunkt ist die Frage der Komplexität der verwendeten Software zu klären. (..) Unter diesem Aspekt erscheint Django nicht als die ideale Wahl.

Flask und Falcon verfolgen beide den Ansatz eines Microframeworks. Die Laufzeitgeschwindigkeit des Controllers erscheint an dieser Stelle nicht als limitierender Faktor der Applikation. Zwar wäre es wünschenswert, die Datenbeschaffung der Webapplikation bereits im Backend asynchron zu erledigen, doch überwiegt das breitere Spektrum an Modulen und Dokumentationen für die Webentwicklung von Flask.

Damit erscheint Flask als das geeignetste Werkzeug für diese Aufgabe.

3.6.4 Datenbank

Das DBMS soll über Schnittstellen in den Sourcecode des Programmes eingebunden werden. Die datenbank soll einem relationalen schema folgen. Die Verwendung sollte kostenfrei möglich sein und die benötigte Software über die Quellen des betriebssystems verfügbar sein.

In dieser Applikation wurde sich für die Verwendung von PostgreSQL entschieden. Als Object-Realational-Mapper steht für Python SQLAlchemy zur verfügung.

4 Systementwurf

4.1 Modellierung der Datenbank

Um die Daten zu modellieren ist es sinnvoll, sich diese im Kontext der Erhebung zu betrachten. In (1) ist dieser Prozess vereinfacht dargestellt.

Boje \rightarrow (misst_{zu Zeitpunkt}) \rightarrow Messprofile \rightarrow (enthält) \rightarrow Aufzeichnungen (1)

Dabei ist erkennbar, dass dieses Modell eine Verkettung von Entitäten und Ereignissen darstellt. Eine Boje misst über ihre Lebensdauer eine Anzahl von Messzyklen. Jeder dieser Messzyklen besteht aus einer gewissen Anzahl von Messwerten.

Um das Modell weiter fortzuführen, wurde diese Ereigniskette in ein Entitätsschema überführt. In Abbildung 5 ist eine mögliche Modellierung des Prozesses zu sehen.

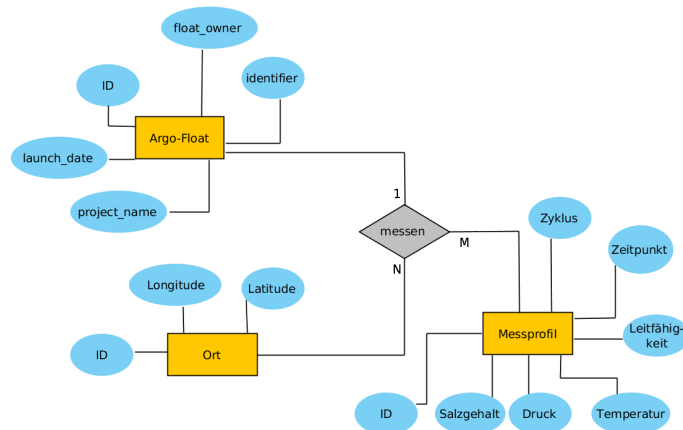


Abbildung 5: Beschreibung der Entitäten der Datenaggregation

4.2 Architektur

Die Applikation besteht aus zwei Grundkomponenten. Ein Teil ist für die Beschaffung und Aufbereitung der Daten zuständig, während der zweite Teil für die Darstellung der Daten zuständig ist.

4.2.1 Entwurf der Datenaggregation

Die Datenaggregation erfüllt zwei Funktionen. Zum einen muss sichergestellt sein, dass die Daten aus den vom Argo Programm bereitgestellten Strukturen gelesen und in ein logisches Format überführt werden. Des weiteren müssen diese Daten in die datenbank der Applikation überführt werden.

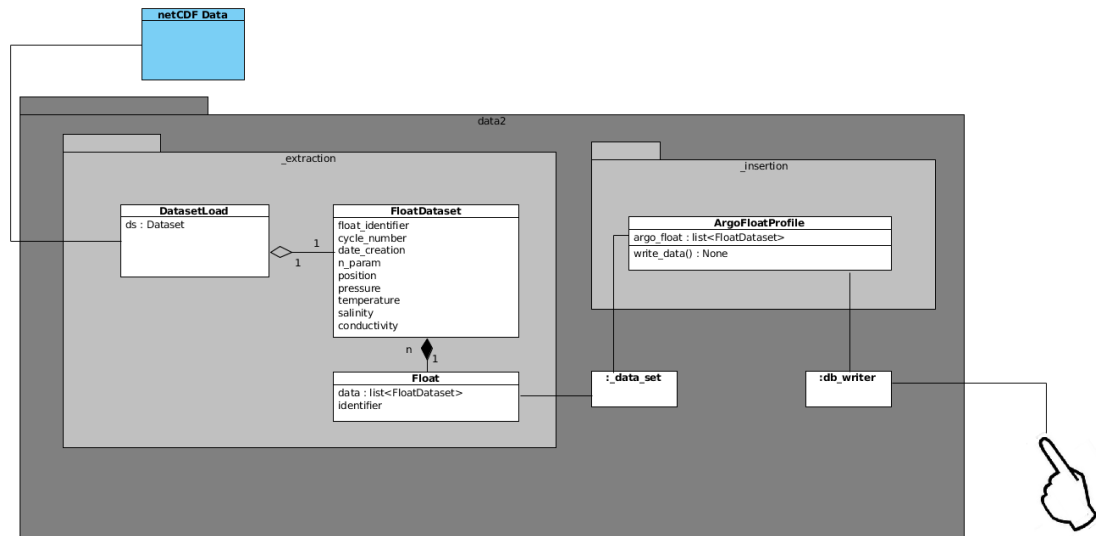


Abbildung 6: Entwurf der Architektur der Aggregation der Daten

Auch in diesem Modell ist die Ereigniskette aus (1) Abzubilden. Jede Messstation wird über ein Objekt abgebildet. Die hier gespeicherten Daten sind für eine Messstation eindeutig. Jeder Messzyklus der Boje wird über ein weiteres, zur Messstation gehöriges Objekt abgebildet. Dieses ist innerhalb des Kontextes eindeutig und trägt die Informationen zur jeweiligen Messung.

Für die Persistierung werden objektorientierte Strukturen für den ORM verwendet. Diese Strukturen teilt sich das Modul mit der Webapplikation. Die Behandlung der Daten sowie eine Steuerung des Prozesses sind als weitere Schnittstellen zu definieren.

Die Aggregation der Daten wird über zwei Teilbereiche abgebildet. Zum einen müssen die benötigten Parameter aus den Dateien ausgelesen und modelliert werden.

Als zweite Ebene der Aggregation ist der Prozess des Schreibens in eine Datenbank zu sehen. Diese verwendet die zuvor ermittelten modellierten Messprofile und schreibt sie Anhand der dort enthaltenen Daten in eine Datenbank.

Es ist an dieser Stelle zu erkennen, dass zwei Stellen existieren, die die intrinsischen Eigenschaften des Modules an dieser Stelle beeinflussen. Hier sollten Schnittstellen geschaffen werden, die eine richtige Handhabung festsetzen

Schnittstellen

1. Daten

- a) Die Datenstruktur ist normiert. Es ist sicherzustellen, dass die bekannte Daten- bzw Ordnerstruktur abgearbeitet wird. Mit Änderungen in dieser Struktur muss nicht gerechnet werden.

- b) Es muss sicher gestellt werden, dass geöffnete Dateien wieder geschlossen werden

2. Steuerung

- a) Daten separiert in die Datenbank zu schreiben, könnte zu Inkonsistenzen führen und soll vermieden werden.
- b) Der Prozess sollte als Sequenz modelliert werden.
- c) Es muss sicher gestellt werden, dass die Daten schrittweise in die Datenbank überführt werden. Würden zu Beginn alle Dateien geöffnet und gemeinsam im flüchtigen Speicher vorgehalten, könnte es zu Problemen führen.

4.2.2 Entwurf der Webapplikation

Die Webapplikation besteht aus zwei Teilkomponenten. Die erste Komponente ist für die Darstellung der Webseite zuständig (app). Durch diese werden Elemente in HTML, Javascript und als Bilder ausgeliefert, welche im Webbrowser der Benutzenden angezeigt werden können. Die Darstellung erfolgt dabei dem Singlepage-Prinzip.

Die zweite Teilkomponente ist dafür zuständig, benötigten Daten bereitzustellen (api). Alle Daten aus datenhaltung werden über diese Schnittstelle angefordert. Die Daten werden über JSON codiert.

4.2.3 Ausarbeitung der Webrouten

Die **api** ist dafür zuständig, die benötigten Daten der Applikation bereitzustellen. Über definierte Webrouten wird über ein GET-Request ein JSON angefordert. Die hierfür ausgearbeitete Struktur ist in Listing 2 zu sehen.

```

1 (1.1) GET      /last_seen
2 (1.2) GET      /last_seen/[force_reload]
3 (1.3) GET      /argo_float/[identifizier]
4 (1.4) GET      /positions/[identifizier]
```

Listing 2: Webrouten der API

(1.1) / (1.2) Über diese Route werden die Daten für die letzte Position der Messstationen aufgerufen. Dieser Datensatz trägt die Daten die zur Anzeige der Positionen der Messstationen auf der Weltkarte benötigt werden und die Zusatzinformationen für das Tooltip der Zusatzinformationen der einzelnen Argo-Floats. Die Daten werden bei jedem Besuch der Webseite ausgeliefert. Da sich diese erst verändern, wenn neue Daten vorliegen, werden diese über einen Caching-Mechanismus vorgehalten,

der optionale Übergabeparameter `force_reload` ermöglicht das neu anlegen des caches. Um Vandalismus vorzubeugen, sollte es sich dabei um einen nicht zu erratenden Token handeln.

- (1.3) Über diese Route werden die Mess-, wie auch Metadaten einer Messstation angefordert. Die Auswahl der Boje erfolgt über den Übergabeparameter `identifizier`, hier wird die eindeutige Identifikationsnummer (siehe `PLATTFORM_NUMBER` in Tabelle 1) der Messstation verwendet.
- (1.4) Um den Positionsverlauf einer Boje anzufordern, dient diese Route. Auch hier wird zur Identifikation der Messstation deren Identifikationsnummer als Parameter übergeben.

Die **app** liefert die Teile aus, die vom Benutzer gesehen werden. Es handelt sich dabei um HTML/JavaScript und Bilddaten. Die hierfür ausgearbeitete Struktur ist in Listing 3 zu sehen.

```

1 (2.1) GET      /
2 (2.2) GET      /chart/[identifizier]
3 (2.3) GET      /info/[identifizier]
```

Listing 3: Webrouten der APP

- (2.1) Über diese Seite ist die eigentliche Applikation zu sehen. Dies umfasst die Karte sowie die angezeigten Messwerte.
- (2.2) Diese Route stellt den Plot der darzustellenden Messwerte als Bild zur Verfügung.
- (2.3) Über diese Route wird ein HTML für die Infoanzeige einer Messstation bereitgestellt.

4.2.4 Aussehen der Webapplikation

Das Aussehen der Applikation soll dem Muster von gängigen Webapplikationen entsprechen. In Abbildung 7 ist ein erster Grobentwurf der Applikation zu sehen.

Zentrales Element der Applikation ist die Darstellung einer Karte. Über diese sollen die letzten Positionen der Karte ersichtlich sein.

Klickt man eine Messstation an, so wird auf der linken Seite ein weiteres Element in die Seite eingefügt. Dieses dient zur Darstellung der Messwerte des jeweiligen ArgoFloats.

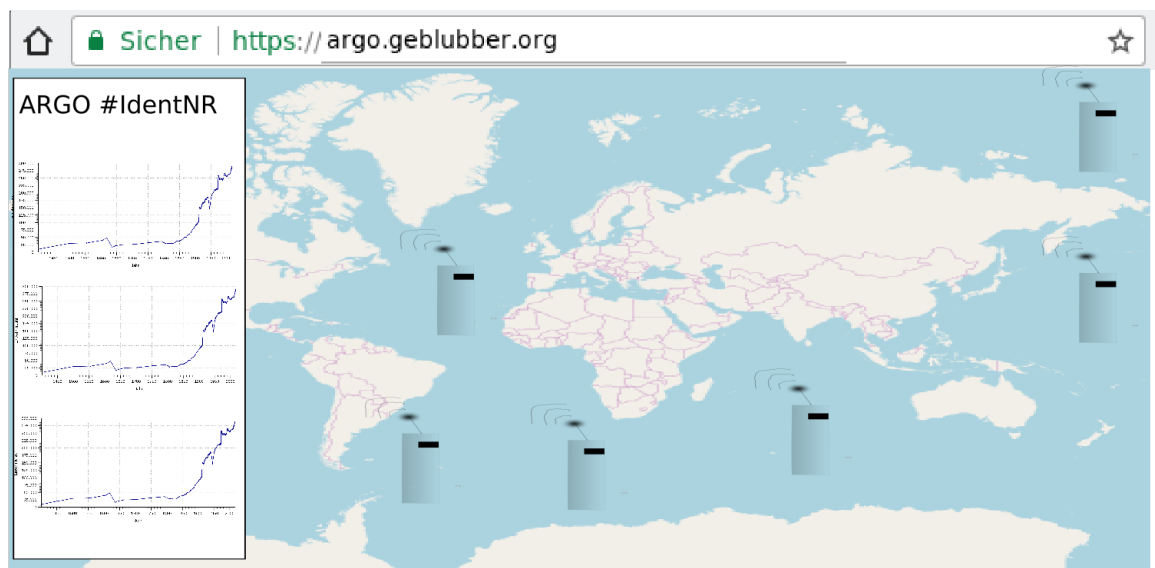


Abbildung 7: Grafischer Grobentwurf der Webapplikation

5 Grundlagen

5.1 Objekt-Relationale Unverträglichkeit

In einem Softwareparadigma manifestiert sich ein bestimmtes Konzept in der Modellierung der Welt. Durch diese konzeptionelle Betrachtungsweise beeinflusst ein Paradigma den Erstellungsprozess sowie die Ergebnisse des Softwaredesigns und zwingt diesem seine Grenzen auf. Probleme bei der Kombination verschiedener Paradigmen werden dabei als Unverträglichkeit (impedance Mismatch) bezeichnet. Objektorientierte Programmierung und eine relationale Abfrage von Datensätzen sind weit verbreitete Paradigmen in der Softwareentwicklung. Damit ist die objekt-relationale Unverträglichkeit (object-relational impedance mismatch) eine häufig auftretende Herausforderung. (vgl. [IBNW09] S. 36-38)

[IBNW09] nennt dabei unter anderem folgende Betrachtungsweisen als Ursachen für die objektrelationale Unverträglichkeit.

Strukturelle Unterschiede Die objektorientierte Programmierung erlaubt das definieren, beliebig komplexer Strukturen aus Methoden und Klassen. Durch Vererbungsstrukturen ist es möglich, Objekte zu spezialisieren und Konzepte für die Erstellung der Klassen zu generalisieren. Spricht man von einer relationalen Algebra, diese wird durch Tupeln, Mengen und Wahrheitswerten definiert. Inhärent wiederholbare Strukturen oder Hierarchien können mit diesen Mitteln nicht umgesetzt werden.

Datenkapselung In der objektorientierten Programmierung können die intrinsischen Attribute eines Objektes verborgen werden. Dieses Konzept ist als Kapselung bekannt und erlaubt es den Zugriff auf die gekapselten Strukturen einzuschränken. In einer relationalen Algebra ist eine derartige Abstraktion über die Daten nicht vorgesehen.

Objektidentität Durch die Instanziierung eines Objektes aus einer Klasse erhält dieses eine eindeutige Identität, somit unterscheiden sich zwei Objekte auch wenn diese Träger eines identischen Datensatzes sind, durch ihre Repräsentation im Arbeitsspeicher. Relationen werden durch den Primärschlüssel, und damit über ihre Daten, definiert. Zwei eigenständige Relationen mit identischem Datensatz ist damit nicht möglich.

Als Hilfsmittel zur Überwindung der oben genannten Probleme werden Objekt-Relationale Mapper (ORM bzw. O/R-Mapper) eingesetzt. Durch dieses Mapping wird eine Schnittstelle oder Abstraktionsebene zwischen Programmteilen aus den jeweiligen Sprachparadigmen definiert, um den impedance Mismatch möglichst transparent zu überwinden.

5.2 Geographische Informationssysteme ODER

5.3 Verwendete Software

5.3.1 Network Common Data Format

Das Network Common Data Format (NetCDF) dient zum Austausch von wissenschaftlichen Daten. Es ist eine Weiterentwicklung des von der NASA entwickelten Common Data Format (CDF). Das Format zeichnet sich dadurch aus, dass es selbstbeschreibend ist. Dadurch wird die Dokumentation mit den Daten mitgeführt. Dies soll die Portabilität des Datensatzes verbessern.⁵

Neben einigen anderen Sprachen, wurde auch eine Bibliothek für Python entwickelt.⁶ Mit dieser ist es möglich, die Binärdaten zu öffnen und zu numpy-Arrays zu extrahieren. In Listing 4 ist die Verwendung exemplarisch an der Extraktion und Berechnung des Datensatzerstellungsdatum aufgezeigt und im Folgenden beschrieben.

```

1 from netCDF4 import Dataset
2 import datetime
3
4 dataset = Dataset('./4900442/profiles/D4900442_042.nc')
5 print(dataset.variables['JULD'])
6
7 # <class 'netCDF4._netCDF4.Variable'>
8 # float64 JULD(N_PROF)
9 #     long_name: Julian day (UTC) of the station relative to
10 #     ↪ REFERENCE_DATE_TIME
11 #     units: days since 1950-01-01 00:00:00 UTC
12 #     conventions: Relative julian days with decimal part (as parts of
13 #     ↪ day)
14 #     _FillValue: 999999.0
15 # unlimited dimensions:
16 # current shape = (1,)
17 # filling off
18
19 julian_date = dataset.variables['JULD'][:,0]
20 dataset.close()
21
22 print(julian_date)
23 # 20062.5483218
24
25 juld_zero = datetime.datetime.strptime('1950-01-01 00:00:00 UTC',
26                                       '%Y-%m-%d %H:%M:%S UTC')
27 date_creation = juld_zero + datetime.timedelta(days=int(julian_date))
28 print(date_creation)
29 # datetime.datetime(2004, 12, 5, 0, 0)

```

Listing 4: Extraktion und Berechnung des Erstellungsdatums eines NetCDF-Datensatzes

Um einen Datensatz zu öffnen, bildet man eine Instanz der Klasse `netCDF4.Dataset`. Die

⁵vgl. [AGS+]

⁶Siehe [net]

Auswahl des Profils gelingt durch die Pfadangabe als Parameter bei der Instanzierung. Über das Attribut `dataset.variables` werden die Datensätze als `OrderedDict` gehalten. Bei der Extraktion eines Parameters erhält man zunächst die Dokumentation des jeweiligen Datensatzes. In diesem Fall handelt es sich um den Zeitpunkt der Sattelitenübertragung des betreffenden Messprofils.

Aus der Dokumentation lassen sich für die Weiterverarbeitung wichtige Parameter entnehmen. So ist der Datensatz in Form des C-Datentyps `float64` codiert. Beim Datumsformat handelt es sich um *Julian day* ab dem Zeitpunkt *01. Januar 1950*.

Um die Werte eines Datensatzes zu extrahieren, wird über die `numpy-slicing` Operation `arr[:, :]` der Datensatz in vektorieller Form extrahiert. Da in diesem Array nur ein skalarer `float64` Wert enthalten ist, kann dieser als nulltes Element extrahiert werden.

Zur Überführung in das Format des durch die ISO 8601 bei uns normierten Gregorianischen Kalenders wird ein Datumsobjekt des Referenzdatums benötigt. Durch die Verwendung von `timedelta` werden die Tage aus dem Feld `JULD` addiert. Da ein Messzyklus 10 Tage andauert, kann an dieser Stelle der Datensatz durch die Entfernung der Dezimalstellen vereinfacht werden. Die Casting-Operation `int()` rundet in jedem Fall ab.

5.3.2 numpy

Numpy ist eine Pythonbibliothek zur diskreten Verarbeitung von ein- oder mehrdimensionalen Arrays.

Nach der Extraktion der CDF-Daten, liegen diese als maskierte Arrays vor. Diese erlauben auch das fehlen einzelner Datensätze.

5.3.3 Julian Date

5.3.4 Flask

Flask wurde im Jahre (YXXX) von Armin Ronacher gegründet.

Die Idee startete Ursprünglich als Aprilscherz.⁷ Der Designansatz wurde aber von vielen Menschen als positiv aufgefasst. Dieser wurde daraufhin unter dem Namen Flask released.

```

1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/')
5 def hello_world():
6     return 'Hello, ArgoData!'

```

Listing 5: Minimalbeispiel für die Verwendung von Flask

⁷[Ron11]

Das Framework verfolgt den Ansatz „*Why things work the way they work*“. Funktionalitäten werden also nicht versteckt, sondern sind dem Entwickler einfach zugänglich. Durch diese Designentscheidung ist es möglich ganze Webapplikationen mit nur wenigen Zeilen Code und in einer einzigen Sourcecode-Datei zu realisieren. Diese Transparenz wird durch ein global verfügbares „magisches“ `!sic` Objekt ermöglicht. Es lassen sich auch weitere Instanzen innerhalb einer Applikation erstellen. Teile der Applikation können über Blueprints modularisiert werden.

Durch die Templateengine **Jinja** erlaubt Flask das generalisieren von HTML-Dokumenten. Durch diese ist es möglich erweiterbare Grundtemplates anzulegen, Variablen abzufragen, zu überprüfen und über Kollektionen zu iterieren.

Durch sogenannte **Blueprints** ist es möglich, eine logische Trennung zwischen Teilbereichen der Applikation vorzunehmen. Diese Modularisierung erlaubt die Definierung von spezifischen Datenordnern sowie für HTML-Templates als auch spezifischer subrouten der Module innerhalb der Applikation.

(...)

In der Grundausstattung der Funktionsumfang von Flask sehr beschränkt. Der KISS-Ansatz bedingt, dass benötigte Funktionalitäten nachgerüstet werden müssen. Im Folgenden wird auf einige in dieser Applikation sinnvolle Erweiterungen eingegangen:

flask_sqlalchemy erlaubt die transparente Anbindung des ORM in die Applikation. In der Grundausstattung besitzt Flask keine Funktionalität um Datenbanken aus der Webapplikation anzusprechen. Durch die Einbindung dieser Erweiterung erhält man ein Framework, das dem Entwurfsmuster MVC entspricht.

Entsprechend den in Flask gängigen Entwurfsmustern entspricht, wird die Anbindung an den ORM durch ein globales Objekt modelliert. Dieses ist eine Instanz der Klasse `flask_sqlalchemy.SQLAlchemy` und stellt die Operationen von SQLAlchemy zur Verfügung.

flask_script rüstet den Funktionsumfang der Webapplikation um eine zentrale Steuerungseinheit auf. Diese Erweiterung besitzt bereits vorgefertigte Steuerungsmodule. Darunter finden sich scripts zum starten der Webapplikation oder zum starten eines Python-REPL mit den Umgebungsvariablen der Webapplikation. Unter Verwendung des Decorator Entwurfsmuster lässt sich das definierte Steuerungsmodul um beliebige Funktionalitäten erweitern.

5.3.5 JSON?

5.3.6 openLayers

OpenLayers ist ein Framework zur Entwicklung webbasierter Geoapplikationen. Dieses ist in JavaScript entwickelt und erlaubt durch eine layerstruktur den Aufbau komplexer Kartenapplikationen. Dabei lassen sich die Elemente der Karten durch folgende Objekte strukturieren:

ol.Map

ol.View

ol.Overlay

ol.layer.Base

ol.interaction.Interaction

ol.control.Control

OpenLayers ist eine Programmbibliothek um interaktive Geoapplikationen zu entwickeln. Das Framework ist in Javascript entwickelt und nimmt alle benötigten Berechnungen auf Clientseite vor.

Die Bibliothek erlaubt es Kartenmaterial aus verschiedensten Quellen zu Rendern. Dabei können Kachel- oder auch Vektorbasierte Materialien eingebunden werden.

Kachel-Layer liefern Bilddaten direkt aus. Dabei werden die Kartenmaterialien in Kacheln aufgeteilt. Dadurch das die Bilder in Teilbereiche untergliedert sind, müssen nur die Bildbereiche ausgeliefert werden, die für die Darstellung des Kartenausschnittes vonnöten sind.

Vektor-Layer sind Beschreibungen von Linien und Polygonen. Die Darstellung wird nicht anhand von Bildmaterial generiert, sondern anhand von geometrischen Beschreibungen gezeichnet.

Neben der reinen Darstellung erlaubt die Bibliothek auch die Definition von „controls“. Dabei lassen sich Zoom-, Rotations- und Mouse-Positions-Effekte umsetzen. Siehe Quelle [GSH15a] S. 296

Zu zeichnende Daten lassen sich über GEOJson-Objekte modellieren. Die Bibliothek folgt einem objektorientierten Entwurfsmuster und erlaubt das transparente Hinzufügen und Ändern von Layern der Karte.

5.3.7 D3

5.4 ...

6 Implementierung

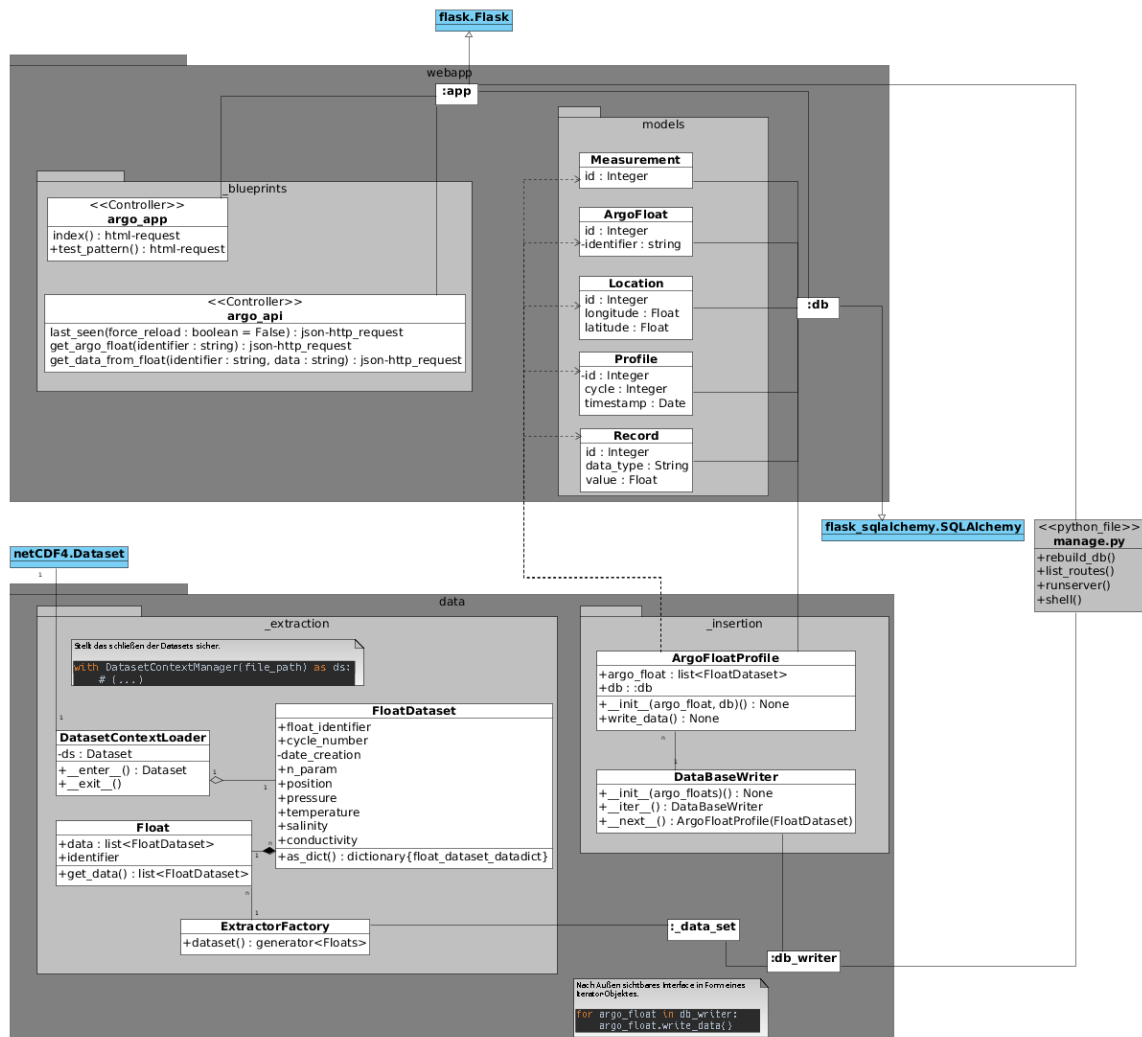


Abbildung 8: Architekturbeschreibung

6.1 Datenaggregation

Bei der Aggregation der Daten aus dem Argo Programm sind Schnittstellen auszuarbeiten. Diese Implementierung ist im folgenden beschrieben.

Dateninput Der Datensatz besteht vor der Verarbeitung aus zahlreichen Dateien. Da die Gefahr besteht, dass geöffnete Dateien nicht wieder ordnungsgemäß geschlossen werden, muss eine Schnittstelle geschaffen werden, die unsachgemäße Verwendung der Dateien verhindert.

Python sieht für diesen Zweck den Kontextmanager vor.

```

1 from netCDF4 import Dataset
2
3 class DatasetContextManager(object):
4     def __init__(self, file_path):
5         self.ds = Dataset(file_path)
6
7     def __enter__(self):
8         return self.ds
9
10    def __exit__(self, exc_type, exc_val, exc_tb):
11        self.ds.close()

```

Listing 6: Implementierung des Kontextmanagers zur Sicherstellung der richtigen Dateibehandlung.label

Steuerung Um die Verarbeitung der Daten sowie den Prozess der Aggregation in der Datenbank richtig zu modellieren, ist es sinnvoll, den Prozess als Sequenz zu modellieren. Dies erlaubt es, die Datenstruktur über einen Generator vorzuhalten. Python sieht dafür den Iterator vor.

Die hier verwendete Implementierung ist in Listing ?? zu sehen. Das Interface wird über die Funktion `def __next__(self)` in Zeile 12 realisiert. Dieses delegiert die Iteration zum Objekteigenen Generator `self.argo_floats`. In dem Moment, in dem ein Objekt aus dem Generator verarbeitet wird, findet die Verarbeitung der dem zugrunde liegenden Datenstruktur statt.

In Zeile 14 wird der Datensatz über ein Objekt ausgeliefert, dass es erlaubt, die Daten in die Datenbank zu überführen.

Damit ist sichergestellt, dass der Prozess nur als Sequenz verwendet werden kann.

```

1 from ._argo_float_profile_writer import ArgoFloatProfile
2
3
4 class DataBaseWriter:
5     def __init__(self, argo_floats, db, app):
6         self.argo_floats = argo_floats
7         self.db = db
8         self.app = app
9
10    def __iter__(self):
11        return self
12
13    def __next__(self):
14        self.argo_float = next(self.argo_floats)
15        return ArgoFloatProfile(self.argo_float, self.db, self.app)

```

Listing 7: Implementierung des Iterators zur Steuerung der Aggregationssequenzlabel

Die Verwendung der Schnittstelle ist in Listing ?? zu sehen. Dabei wird die Sequenz in Zeile 3 iteriert um den Datensatz daraufhin über die daraufhin folgende Zeile in die Datenbank zu überführen.


```

1 from data import db_writer
2
3 for argo_float in db_writer:
4     argo_float.write_data()

```

Listing 8: Verwendung der Schnittstelle zur Steuerung der Datenaggregationlabel

Eine zentrale Problemstellung im Prozess der Überführung der Daten in das Relationale Schema ist die effektive Verwendung des flüchtigen Speichers. Eine klassische, sequenzielle Verarbeitung würde die Daten initial auslesen und diese in Gänze im Ram vorhalten bevor diese über den Mapper in die Datenbank überführt würden. Dieser Prozess wurde durch die Verwendung von Generatoren aufgebrochen. In Listing 9 ist die Implementierung dieser Schnittstelle zu sehen. Da in der Comprehension runde statt eckige Klammern verwendet werden, wird initial nur die logische Struktur als Sequenz vorgehalten. Die inhärenten Float-Objekte des Generators werden zu dem Zeitpunkt erzeugt, wenn diese durch eine Iteration aufgerufen werden. Damit werden die Daten erst zu dem Zeitpunkt ausgelesen wenn diese für die weitere Verarbeitung benötigt werden,

Die Methode `get_data_sets()` erzeugt aus jedem Unterordner im definierten Arbeitsverzeichnis ein Float-Objekt und gibt dieses über das Schlüsselwort `yield` zurück.

Durch diese Klasse wird somit ein Generator definiert, der es erlaubt, alle im Arbeitsverzeichnis definierten ArgoFloat Datenobjekte bereitzustellen, ohne diese bereits bei der Instantiierung kennen und abarbeiten zu müssen.

```

1 import os
2
3 from ._float import Float
4
5
6 class ExtractorFactory:
7     def __init__(self, data_directory):
8         self.extractor_generator = (
9             Float(os.path.join(data_directory,
10                               profile_directory))
11             for profile_directory in os.listdir(data_directory))
12
13         self.__sum_floats = sum([os.path.isdir(os.path.join(
14             ↪ data_directory, d)) for d in os.listdir(data_directory)])
15
16     def get_data_sets(self):
17         return self.extractor_generator
18
19     def float_count(self):
20         return self.__sum_floats

```

Listing 9: Factory zur Extraktion der Datensätze.

6.2 Webapplikation

6.2.1 Objektrelationales Mapping

Als Objektrelationalen Mapper wurde sich hier für SQLAlchemy entschieden. Diese Software gilt als erprobt und wird bereits in einer Vielzahl von Softwaresystemen eingesetzt. So verwendet unter anderem reddit oder die Mozilla Foundation SQLAlchemy als Schnittstelle zur Datenhaltung. SQLAlchemy wird für die Verwendung in Flask empfohlen und es existiert eine Erweiterung um die Verwendung des Mappers in Flask zu vereinfachen (vgl. [Ron11] S. 33).

In SQLAlchemy werden die Tabelleneinträge in Modellklassen abgebildet. In Listing 10 ist die Implementierung des Models für eine ArgoFloat-Messtation zu sehen.

```

1 from . import db
2
3
4 class ArgoFloat(db.Model):
5     __tablename__ = 'argo_floats'
6
7     measurements = db.relationship('Measurement', backref='argo_floats',
8     ↪ lazy='dynamic')
9
10    id = db.Column(db.Integer, primary_key=True)
11    identifier = db.Column(db.String(10))
12    project_name = db.Column(db.String(100))
13    launch_date = db.Column(db.Date)
14    float_owner = db.Column(db.String(100))
15
16    def __init__(self, identifier, project_name, launch_date,
17    ↪ float_owner):
18        self.identifier = identifier
19        self.project_name = project_name
20        self.launch_date = launch_date
21        self.float_owner = float_owner
22
23    def __repr__(self):
24        return f'<Argo Float {self.id!r}>'
25
26 class ArgoFloatTmpTable(ArgoFloat):
27     __bind_key__ = 'data_input'

```

Listing 10: Modellklasse für ein Argo-Float

Die Registrierung des Models erfolgt über die Vererbung einer Metaklasse `db.Model`. Die Eigenschaften der jeweiligen Entitäten werden über Attribute des Objektes implementiert. Diese werden in Instanzen von `db.Column` transferiert. Dies erlaubt das Festsetzen des Datentyps. So lassen sich auch weitere Dateiattribute, wie der Festlegung des Primärschlüssels, definieren. Die Attribute werden durch eine Parameterübergabe in den Initiator

der Klasse mit Werten versehen. Um das Binding zu umzusetzen, wurden Nachfahren der Modellklassen erzeugt. Diese benötigen für die Zuordnung das Attribut `__bind_key__`. Über das Schlüsselwort `db.relationship` werden Beziehungen beschrieben. In diesem Fall besteht eine 1– N Beziehung (`argo_float` \leftarrow `measurements`) zu `measurements` (Siehe auch Figure 5 auf Seite 14). Als Übergabeparameter wird ein Name für die Beziehung erwartet. Der Parameter `backref` gibt den Namen der Klasse auf dem Mapper an. Diese Einstellung erlaubt den Aufruf der Beziehung auch in die andere Richtung. Der Parameter `lazy` definiert die verwendete Strategie für das Lazyloading. In diesem Fall wurde sich für 'dynamic' entschieden. Diese Einstellung gibt bei Lesezugriffen ein vorkonfiguriertes Query-Objekt zurück. Dies erlaubt das Hinzufügen weiterer Filter, vor dem Zugriff auf die Tabellen. Die durch diese Beziehung verbundene Modellklasse `Measurements` ist in Listing 11 zu sehen.

```

1 from . import db
2
3
4 class Measurement(db.Model):
5     __tablename__ = 'measurements'
6
7     id = db.Column(db.Integer, primary_key=True)
8
9     profiles = db.relationship('Profile', backref='measurements', lazy='
    ↪ dynamic')
10
11     argo_float_id = db.Column(db.Integer, db.ForeignKey('argo_floats.id'
    ↪ ))
12     argo_float = db.relationship('ArgoFloat')
13
14     location_id = db.Column(db.Integer, db.ForeignKey('locations.id'))
15     location = db.relationship('Location')
16
17     def __init__(self, argo_float, location):
18         self.argo_float = argo_float
19         self.location = location
20
21     def __repr__(self):
22         return f'<Measurement {self.id!r}>'
23
24
25 class MeasurementTmpTable(Measurement):
26     __bind_key__ = 'data_input'

```

Listing 11: Modellklasse für eine Messung

Auf dieser Seite der Beziehung (`measurement` \rightarrow `argo_float`) unterscheidet sich deren Implementierung. Für die eindeutige Zuordnung des übergeordneten Argo-Floats, wird deren `id` als Fremdschlüssel registriert. Die Beziehung benötigt neben dem Namen der anderen Seite keine weiteren Parameter, da die Beziehung bereits konfiguriert worden ist. Das Schreiben von Daten über den Mapper SQLAlchemy ist Teil der Datenaggregation.

Der dort implementierte Programmcode ist in Listing 12 zu sehen.

```

1 from webapp.models import ArgoFloat, Location, Measurement, Profile
2
3
4 class ArgoFloatProfile:
5     def __init__(self, argo_float, db, app):
6
7         self.argo_float = argo_float
8         self.db = db
9         self.app = app
10
11     def write_data(self, bind=None):
12         session = self.db.create_scoped_session(
13             options={
14                 'bind': self.db.get_engine(self.app, bind),
15                 'binds': {}
16             }
17         )
18         try:
19             argo_float_ = ArgoFloat(
20                 identifier=self.argo_float.identifier,
21                 project_name=self.argo_float.project_name,
22                 launch_date=self.argo_float.launch_date,
23                 float_owner=self.argo_float.float_owner
24             )
25
26             for profile_data in self.argo_float.data:
27                 location_ = Location(
28                     latitude=profile_data.position['latitude'],
29                     longitude=profile_data.position['longitude']
30                 )
31                 measurement_ = Measurement(
32                     argo_float=argo_float_,
33                     location=location_
34                 )
35
36                 session.add(Profile(
37                     cycle=int(profile_data.cycle_number),
38                     timestamp=profile_data.date_creation,
39                     measurement=measurement_,
40                     salinity=profile_data.salinity,
41                     pressure=profile_data.pressure,
42                     temperature=profile_data.temperature,
43                     valid_data_range=profile_data.valid_data_ranges
44                 ))
45
46             session.commit()
47         except Exception as err:
48             session.rollback()
49             raise err

```

Listing 12: Das Schreiben der Daten eines Argo-Floats in die Datenbank

Die hier implementierte Klasse `ArgoFloatProfile` ist für das Schreiben aller Datensätze eines `ArgoFloats` zuständig. Diese verwendet für die Zuordnung der Daten die Modell-Klassen aus der Webapplikation. Durch die Methode `write_data` werden die Daten des durch `__init__` übergebenen Datensatzes in die Datenbank überführt. Um ein dyna-

misches Binding realisieren zu können, wird der betreffende Parameter bind übergeben. Mithilfe dieses Parameters wird eine Session aufgebaut, welcher das Schreiben in die Datenbank erlaubt, die über das binding definiert ist.

Für jeden Datensatz eine Instanz der betreffenden Modell-Klasse instanziiert. Die dafür benötigten Daten werden aus den Datensätzen des Klasseneigenen Generator-Objektes argo_float angefordert und als Parameter übergeben.

Der zuvor definierten Session werden nur die Instanzen der Profile-Klassen übergeben. Da alle zum Messprofil gehörenden Modells in diesem Kontext eindeutig sind, kann SQLAlchemy diese selbstständig zuordnen. Zum Abschluss werden die Daten über session.commit() in die Datenbank überführt. session.rollback() erlaubt es, die Veränderungen, die innerhalb dieser Session an der Datenbank herbeigeführt wurden, wieder auf den Urzustand zurückzuführen.

SQLAlchemy bietet eine eigene Abfrage-Sprache an. In Listing 13 ist eine vereinfachte Implementierung der Abfrage zur Bestimmung der Positionshistorie einer Argo-Float zu sehen, wie sie auch in der Web-API implementiert ist.

```

1 from webapp import db
2 from webapp.models import Measurement, Location, Profile, ArgoFloat
3
4 query = db.session.query(ArgoFloat, Location, Profile) \
5     .join(Measurement) \
6     .join(Location) \
7     .join(Profile) \
8     .filter(ArgoFloat.identifier == '1900037') \
9     .order_by(Profile.timestamp)
10
11 result = db.session.execute(query, None, bind=db.get_engine(app, None))
12
13 data = [row for row in result]
```

Listing 13: Anfragen über SQLAlchemy zum Lesen von Daten

In diesem Beispiel wird eine Projektion auf Elemente von Attribute von ArgoFloat, Location und Profile angefordert. SQLAlchemy erlaubt die Definition von Joins über mehrere Tabellen, diese werden durch die in den Modell-Klassen definierten Beziehungen aufgelöst. Die Selektion erfolgt durch die Methode query.filter(). In diesem Fall werden nur Relationen der Argo-Float mit dem identifier '1900037' ausgewählt. Die Methode query.order_by() erlaubt das Sortieren der Ergebnisse. In diesem Fall werden die Datensätze durch das Attribut Profile.timestamp chronologisch sortiert.

SQLAlchemy ist auch in der Lage, in SQL-Sprache definierte Anfragen zu verarbeiten. In Listing 14 ist eine vereinfachte Implementierung aus der Web-API zu sehen. Diese

extrahiert die letzten ÜPositionen mit den dazugehörigen Zeitpunkten aus der Datenbank.

```

1 from webapp import db
2
3 sql_template = "SELECT DISTINCT ON (f.id)
4     f.identifrier,
5     l.longitude,
6     l.latitude,
7     p.timestamp
8 FROM argo_floats f
9     JOIN measurements m
10        ON f.id = m.argo_float_id
11     JOIN profiles p
12        ON m.id = p.measurement_id
13     JOIN locations l
14        ON l.id = m.location_id
15 ORDER BY f.id, p.cycle DESC;"
16
17 result = db.engine.execute(sql_template)

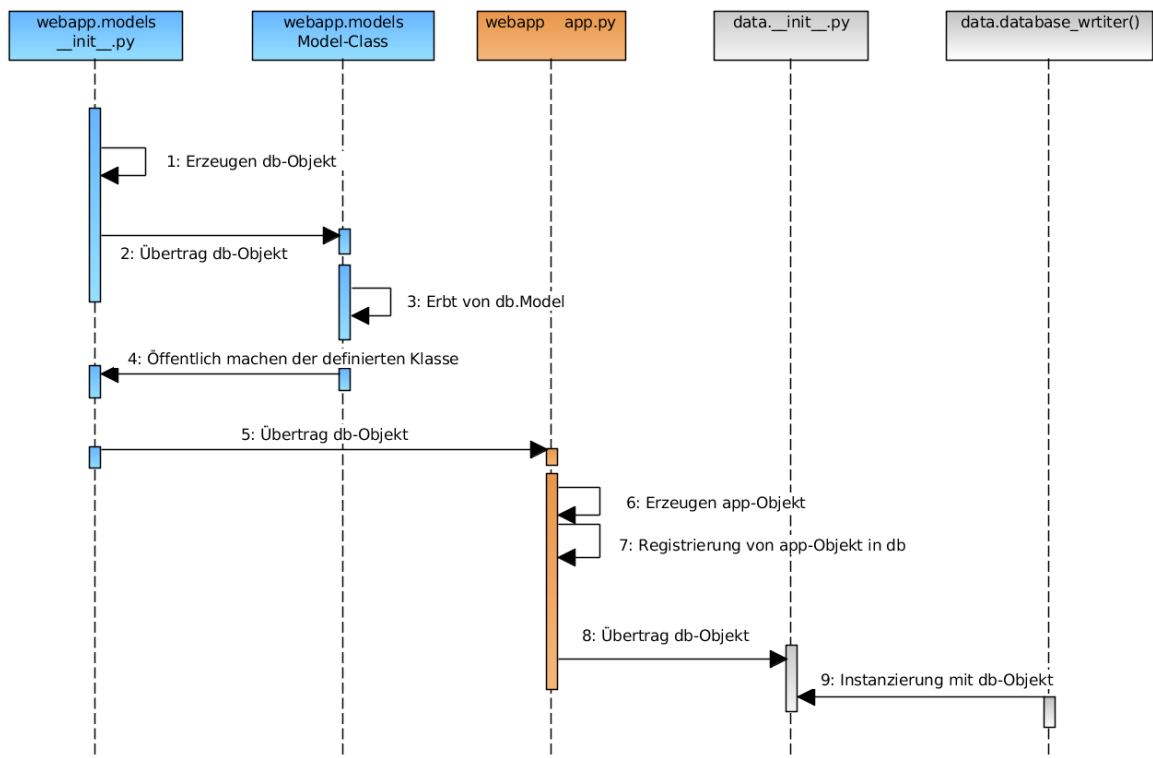
```

Listing 14: Das Ausführen von SQL Anfragen über SQLAlchemy

Über die Methode `db.engine.execute()` können SQL-Queries direkt verarbeitet werden. Dies hat aber zwei entscheidende Nachteile. Der Query-Dialekt beschränkt die Anfrage auf ein bestimmtes DBMS. Diese Anfrage ist für PostgreSQL entworfen und würde auf einem anderen DBMS nicht funktionieren. Zum anderen würden Eingaben nicht gegen SQL-Injections gesichert werden. Eine Abfrage wie `db.engine.execute('SELECT * FROM argo_floats WHERE argo_floats.identifrier=%s' % user_input)` würde also unabschätzbare Sicherheitsrisiken mit sich bringen.

Flask bietet für die Einbindung von SQLAlchemy eine Erweiterung an. Durch diese Erweiterung wird eine SQLAlchemy-Instanz durch ein zentrales und scheinbar globales Objekt (`db`) repräsentiert. Dies ermöglicht eine einfache Datenabstraktion und ist für Flask typisch. Dieser Mechanismus wird durch die zeitkritische und in ihrem Ablauf fest vorgeschriebene Erstellung von Objekten und Submodulen erkauft. Dieser Prozess ist bei der Einbindung von `flask-sqlalchemy` gut sichtbar. Aus diesem Grund wird die Initialisierung dieser Erweiterung hier gesondert dargestellt.

In Figure 6.2.1 ist eine sequentielle Darstellung des Prozesses zu sehen und wird im folgenden beschrieben.



1. In der Datei `webapp.models.__init__.py` muss sichergestellt werden, dass die SQLAlchemy-Instanz noch vor der Initiierung der Modell-Klassen erstellt wurde.
2. Nun kann das db-Objekt in die Modellklassen übertragen werden. Dafür importieren diese db in ihren Scope.
3. Die Modell-Klassen werden in db registriert. Dafür erben diese von der Metaklasse `db.Model`. Siehe auch Listings 10 und 11.

```

1 from flask_sqlalchemy import SQLAlchemy
2
3 db = SQLAlchemy()
4
5 from ._measurement import Measurement
6 from ._argo_float import ArgoFloat
7 from ._location import Location
8 from ._profile import Profile

```

Listing 15: `webapp.models.__init__.py`

4. Die Modell-Klassen dürfen erst nach der Instantiierung von db über den Scope des Submodules heraus bekannt gemacht werden, um sicherzustellen, dass dieses bereits existiert. Die Implementierung des Ablaufes ist in Listing 15 ersichtlich.

5. Die Instanz db kann nun mit den in ihr registrierten Modell-Klassen in aus dem Submodul in die Datei webapp.__init__.py importiert werden.
6. Hier wird nun das globale Objekt der Flask-Instanz (app) erstellt und konfiguriert.
7. Anschließend wird app in db registriert. Die Schritte 5 bis 7 können über das Listing 16 nachvollzogen werden.
8. Die Instanz db aus dem Scope webapp.db ist nun fertig konfiguriert und kann für das Lesen und schreiben in die Datenbank verwendet werden.

6.2.2 Controller

```

1 from flask import Flask, url_for
2 from flask_twisted import Twisted
3
4 from .models import db
5
6 app = Flask(__name__, static_url_path='/static')
7 app.config.from_pyfile('./argo.cfg')
8
9 db.init_app(app)
10
11 from .blueprints import argo_api, argo_app
12
13 app.register_blueprint(argo_api)
14 app.register_blueprint(argo_app)
15
16 from . import models
17
18 twisted = Twisted(app)

```

Listing 16: webapp.__init__.py

Über die globale Instanz des Controllers (app) kann auf Daten zugegriffen und die Konfiguration vorgenommen werden. In Listing 16 ist die Initialisierung des Controllers zu sehen.

Über Blueprints wird die in Kapitel 4.2.3 ausgearbeitete logische Trennung zwischen app und api realisiert. Die hier definierten Module umfassen auch die in diesem Kontext benötigten Hilfsprogramme. Die hier implementierte Modulstruktur ist im Folgenden ersichtlich.

argo_api

query_factory Alle lesenden Zugriffe zur Datenbank werden über eine QueryFactory zentral zusammengefasst. Die hier ausgearbeitete Implementierung ist in Kapitel 6.2.1 ausführlich beschrieben.

json_builder Durch diese Programmabschnitte werden die Daten in Listen und Dictionaries überführt. Die Variablen und Datenstrukturen werden dabei so angelegt, dass diese direkt als JSON bzw geoJSON ausgeliefert werden können.

Dieses Controller-Blueprint fordert die Daten über die query_factory an und bringt sie über den json_builder in das vorgesehene Format. Die Umwandlung in JSON und die Erstellung des Requests erfolgt über flask.jsonify.

argo_app

generate_graph Dieser Programmteil erzeugt einen Plot über die Bibliothek matplotlib. Eine nähere Beschreibung der Erstellung findet sich in Kapitel 6.2.5.

In diesem Blueprint werden Templates zu HTML-Dateien zusammengesetzt. Dafür benötigt dieser Programmteil auf die Verzeichnisse für die HTML- jinja-Templates (webapp/templates) und dem Javascript-Code, CSS und Bilder (webapp/static). Diese Daten werden über `flask.render_template` zusammengesetzt und ausgeliefert.

Die so erstellten Blueprints werden im zentralen Controller (app) über das Schlüsselwort `app.register_blueprint` registriert und zusammengefasst.

Als zentrale Steuerungseinheit für den gesamten Programmablauf dient die Datei `manage.py`. Die Parameter dieser Datei werden im Folgenden kurz beschrieben

runserver Durch diesen Parameter wird der Server der Applikation gestartet.

rebuild_db Dies stößt den Prozess der Datenaggregation an. Streng genommen ist dies nicht Teil des Controllers. Aus Gründen der Einfachheit wurde dieser Programmablauf aber in die Steuerungseinheit integriert.

shell Durch diesen Parameter wird eine IPython-Shell gestartet. In diesem repl ist die Webapplikation mit all ihren Umgebungsvariablen initiiert. Diese Shell dient für das Debugging der Applikation.

list_routes Dieser Parameter listet alle Routen, die durch den Controller definiert sind auf der Kommandozeile auf.

6.2.3 Templates

Für das Erstellen der HTML Dateien wurde die Template-Engine Jinja2 verwendet. Diese ist gut in Flask integriert. Jinja2 erlaubt das Modularisieren von HTML Dateien, den Zugriff auf Variablen und übergebenen Daten sowie die Abarbeitung eingetragener Strukturen und Wahrheitswerte.

Die Templatestruktur wurde in dieser Applikation stark modularisiert. In Listing 17 ist der Einstieg in die Modulstruktur zu sehen.

```

1 {% extends '_base.html' %}
2
3 {% block map %}
4     {% include '_render_map.html' %}

```

```
5 {% endblock %}
```

Listing 17: webapp.templates.map.html

Die Datei `_base.html` ist für die Darstellung der Webseite verantwortlich. Diese wurde über Twitter Bootstrap realisiert. Als Vorlage diente dabei ein bereits ausgearbeitetes 2-Column Layout aus [Ng14]. Der Beispielcode wurde an die Anforderung der Applikation angepasst. Der Code wurde dabei in die logischen Bestandteile zerlegt und in die modulare Struktur der Webseite überführt.

6.2.4 Kartendarstellung

6.2.5 Zeichnen der Graphen

7 Testen

8 Demonstration und Auswertung

9 Anhang

Quellenverzeichnis

- [AGS⁺] Rew AuthorsRuss, Davis Glenn, Emmerson Steve, Davies Harvey, Hartnett Ed, Heimbigner Dennis, and Ward Fisher. Netcdf: Introduction and overview. <https://www.unidata.ucar.edu/software/netcdf/docs/index.html>.
- [Amb13] Scott W. Ambler. Mapping objects to relational databases: O/r mapping in detail. <http://agiledata.org/essays/mappingObjects.html>, Apr 2013. (Accessed on 02/20/2018).
- [Arg] Argo float data and metadata from global data assembly centre (argo gdac). <http://www.seanoe.org/data/00311/42182/>. (Accessed on 10/09/2017).
- [Arg02] Data management handbook, 2002.
- [Arg17a] Documentation - Argo Data Management. <http://www.argodatamgt.org/Documentation>, Dec 2017. [Online; accessed 12. Dec. 2017].
- [Arg17b] Argo data and how to get it. http://www.argo.ucsd.edu/Argo_data_and.html, Jun 2017. [Online; accessed 12. Dec. 2017].
- [Arg17c] Argo data selection - Argo Data Management. <http://www.argodatamgt.org/Access-to-data/Argo-data-selection>, Dec 2017. [Online; accessed 12. Dec. 2017].
- [Arg17d] Global Change Analysis. http://argo.ucsd.edu/Data_viewers.html, Apr 2017. [Online; accessed 12. Dec. 2017].
- [Bay02] Mike Bayer. zzzeek : Patterns implemented by sqlalchemy. <http://techspot.zzzeek.org/2012/02/07/patterns-implemented-by-sqlalchemy/>, Feb 2002. (Accessed on 02/20/2018).
- [CKT⁺15] Thierry Carval, Robert Keeley, Yasushi Takatsuki, Takashi Yoshida, Claudia Schmid, Roger Goldsmith, Annie Wong, Ann Thresher, Anh Tran, Stephen Loch, and Rebecca Mccreadie. Argo user manual, 2015.
- [Fla] Welcome to flask — flask documentation (0.12). <http://flask.pocoo.org/docs/0.12/>. (Accessed on 09/08/2017).
- [Fow02] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

- [GLLP04] S Guinehut, PY Le Traon, G Larnicol, and S Philipps. Combining argo and remote-sensing data to estimate the ocean three-dimensional temperature fields—a first approach based on simulated observations. *Journal of Marine Systems*, 46(1):85–98, 2004.
- [GSH15a] Thomas Gratier, Paul Spencer, and Erik Hazzard. *OpenLayers 3 Beginner s Guide*. Packt Publishing - ebooks Account, 2015.
- [GSH15b] Thomas Gratier, Paul Spencer, and Erik Hazzard. *OpenLayers 3 Beginner s Guide*. Packt Publishing - ebooks Account, 2015.
- [IBNW09] Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh. A classification of object-relational impedance mismatch. In *Advances in Databases, Knowledge, and Data Applications, 2009. DBKDA’09. First International Conference on*, pages 36–43. IEEE, 2009.
- [Mar16] Ignaszewski Mark. Description of the argo gdac file checks: Data format and consistency checks. 2.4, 2016.
- [Moo98] Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.
- [Mur17] Scott Murray. *Interactive Data Visualization for the Web: An Introduction to Designing with*. O’Reilly Media, Inc., 2017.
- [net] Python cdf4. <http://unidata.github.io/netcdf4-python/>.
- [Ng14] Jackie Ng. bootstrap-viewer-template. <https://github.com/jumpinjackie/bootstrap-viewer-template>, 2014.
- [num] The numpy ma module - numpy v1.13 manual. <https://docs.scipy.org/doc/numpy-1.13.0/reference/maskedarray.generic.html>.
- [Ope] Open layers - api. [Online; accessed 2017-05-28].
- [Phi10] Dusty Phillips. *Python 3 object oriented programming*. Packt Publishing Ltd, 2010.
- [Que] Query api — sqlalchemy 1.2 documentation. <http://docs.sqlalchemy.org/en/latest/orm/query.html>. (Accessed on 02/08/2018).
- [RD90] Russ Rew and Glenn Davis. Netcdf: an interface for scientific data access. *IEEE computer graphics and applications*, 10(4):76–82, 1990.

- [Ron11] Armin Ronacher. Opening the flask. <http://mitsuhiko.pocoo.org/flask-pycon-2011.pdf>, 2011.
- [Sch] Gustav Schwab. Die argonautensage. <http://gutenberg.spiegel.de/buch/sagen-des-klassischen-altertums-4962/12>.
- [SG] Megan Scanderbeg and John Gould. A beginners' guide to accessing argo data. http://www.argo.ucsd.edu/Argo_data_guide.pdf.