TECO Research Group

Marcel Köpke
Matthias Budde
Till Riedel

Implementierungsbericht

Version 1.0

Visualizing & Mining of Geospatial Sensorstreams with Apache Kafka

---

Jean Baumgarten
Thomas Frank
Oliver Liu
Patrick Ries
Erik Wessel

26. August 2018

Inhaltsverzeichnis

1 Einleitung

Dieser Implementierungsbericht dient als Übersicht der Implementierungsphase.

Die Implementierungsphase unseres Projekts wurde in zwei Hälften unterteilt, von denen die erste drei Wochen und die zweite vier Wochen dauerte. Bezieht man die zusätzliche Woche zur Klausurvorbereitung mit ein, summiert sich die Phasendauer auf acht Wochen.

Nachfolgend werden Abweichungen der Implementierung vom Entwurf analysiert, eine Übersicht zu Unit-Tests gegeben und auf die implementierten Muss- und Wunschkriterien eingegangen.

2 Änderungen am Entwurf

2.1 Bridge

- Um das Problem zu beheben, dass FROST in versendeten MQTT-Nachrichten nur `@iot.navigationLink`s für zusammenhängende Objekte angibt (statt einer Menge an `@iot.id`s), wurde eine neue Klasse `FrostIotIdConverter` erstellt, dessen Methoden genau diese Konvertierung bewerkstelligen.
- Das Format, in dem konvertierte MQTT-Nachrichten zu Kafka geschickt werden, wurde geändert (von `byte[]` zu einem Avro-Objekt). Dies hat folgende Auswirkungen auf dieses Modul:
  - Die `getSensorIdFromMessage`-Methode der Klasse `MqttMessageConverter` wurde entfernt.
  - Neun neue Klassen wurden hinzugefügt, die die zu versendenden Avro-Objekte repräsentieren.
  - Die Klasse `SchemaRegistryConnector` wurde entfernt, da die enthaltene Funktionalität nicht mehr benötigt wird.

2.2 Database

- Im Laufe der zweiten Implementierungsphase ist klar geworden, dass die Datenbank näher an dem Core arbeiten muss. Dadurch entfallen alle von `HTTPServlet` erbenden Klassen, da Zugriffe auf die Datenbank nun direkt von dem Core aus durchgeführt werden. Dennoch bleibt die Klasse `Facade` erhalten, um eine Implementierung der Servlets, sollte dies in der Zukunft nötig sein, ohne große Umstände zu ermöglichen.
- Alle Klassen, die für die Verwaltung von veralteten Daten zuständig sind, entfallen (also von `Maintainer` erbende Klassen und die Klasse `MaintenanceManager`). Memcached bietet die Möglichkeit, beim Hinzufügen eines Eintrags eine Zeit zu setzen, nach dem dieser Eintrag abläuft (d.h. gelöscht wird). Dies ist eine effizientere Lösung um alte Einträge zu entfernen als durch einen Maintainer.
- Da einzelne Datenwerte nun durch ein `ObservationData`-Objekt dargestellt werden, entfallen die Klassen `ClusterID` und `ZoomLevel`.

- Wegen obigem Grund wurde ebenfalls die Klasse `KafkaToStorageProcessor` in `ObservationDataToStorageProcessor` umbenannt. Diese bietet über die Fassade nun zwei Funktionen `add` und `get` für `ObservationData`-Objekte an.

2.3 Transfer

- Der Zugriff über das Servlet wurde entfernt und durch Kafka ersetzt
- Da sie nicht nötig waren, wurden die UIDS aus dem TransferManager (ehemals GraphDataTransferController) entfernt
- Die Arbeitsschritte der Übermittlung wurden noch stärker modular aufgebaut, sodass einzelne Komponenten bei Fehlern schnell überprüft und repariert werden können, ohne andere Teile zu beschädigen
- Weiterhin wurde die Darstellung durch Verwendung von Grafana optimiert

2.4 Core

- Die ConfigGUI wurde komplett verworfen weil unser Programm in einer Kommandozeile läuft und somit eine GUI überflüssig macht. Alle Einstellungen die man über die GUI hätte vornehmen können kann man nun über die .properties Dateien vornehmen. Zudem sollte man diese nicht während des laufenden Betriebs ändern.
- Hier wurde die Idee übernommen die Properties extern zu speichern nur wurden weitere Methoden hinzugefügt um den einzelnen Kafka Consumern, Kafka Producern und Kafka Streams sofort ihre benötigten Properties bereitzustellen.

2.4.1 Grid

- Aufgrund von extremen Komplikationen waren wir gezwungen diesen Teil neu zu implementieren
- Es wurde bei der Verwaltung der Karte aus zeitlichen Gründen auf Objekte und Zustände gesetzt statt den Ablauf mit Kafka aufzuteilen
- Die Verbindungen zu eng arbeitenden Komponenten wie der Datenbank und Grafana & Graphite wurden direkt gekoppelt. Es wurde bei der Kooperation der Komponenten auf Modularität geachtet, sodass durch entfernen von Einträgen im Code bestimmte Funktionen gestoppt werden können

2.4.2 Controller

- Im Core wurden im Grunde nur die ProcessStrategys beibehalten welche aber umbenannt wurden, weil es nicht mehr richtig dem Strategy Entwurfsmuster entspricht, weil man kann diese Prozesse als eigene Programme sehen und können auch unabhängig voneinander funktionieren. Das Interface wurde auch so implementiert wie es designt wurde. In der Nutzung stehen zurzeit 3 Prozesse:
  - Einer welcher die Observations und FeaturesOfInterest zusammenfügt umso damit besser arbeiten können.
  - Ein Export Prozess welcher alle Topics zusammenfügt, so dass die Export Methodik nicht alle Daten einzelne abfragen muss, sondern nur noch eine Anfrage in Kafka machen muss.
  - Ein Grid Prozess welcher die Grid Methodik ausführt und sie mit Observations füttert.

  Die Klasse TopologyBuilder wurde entfernt aus der Implementierung, weil diese Klasse nun ein Bestandteil vom StreamBuilder in Kafka 1.0.1 geworden und man sich darum nicht mehr selber kümmern muss. Der Controller wurde auch entfernt bzw. durch eine einfache Main Klasse ersetzt, weil das Programm immer läuft, wie das auch in der Kappa Architektur vorgesehen ist.

2.5 Import

- Durch das Wegfallen der Verwaltungs-GUI in die der Import eingebettet werden sollte, wurde eine neue Klasse mit dem Namen `ImportGUI` hinzugefügt. Diese dient dazu die GUI für den Import, welche in `DataImporter` enthalten ist, zu starten. Sie übernimmt die Aufgabe der Klasse mit der Main-Methode und kann, sollte eine Verwaltungs-GUI später noch entwickelt werden, einfach durch diese ersetzt werden.

- Die `FileExtension`-Klasse ist bei der Implementierung weggefallen, da es sich hierbei nur um einen einfachen String handelt (bspw. "csv").

- Erzeugen einer weiteren Implementierung der `FileReaderStrategy`: `DummyReaderStrategy`, welche keine Daten ausliest, sondern stattdessen zufällig generierte Daten an den Server sendet. Dies dient dem Test verschiedener Komponenten.

- Vorerst fällt die `NetCDFReaderStrategy` weg, da die Zeit nicht ausgereicht hat. Nach Möglichkeit soll diese während der Testphase noch nachimplementiert werden. Damit ist aber vorerst die Unterstützung für das NetCDF Format nicht gegeben.

- Da er nur einen statische Klasse enthält wurde der `FrostSender` zu einer Hilfsklasse umfunktioniert, sodass dieser nicht mehr als Parameter übergeben werden muss.
- Statt einem `FilePath` Objekt wird nun der `File` verwendet. Einige Parameter von Funktionen wurden verändert, entfernt oder hinzugefügt, bzw. in den Konstruktor verschoben.

2.6 Export

- Zusammenfügen der verschiedenen Servlets zu einem einzelnen. Das `ExportServlet` übernimmt alle Aufgaben, die mit dem Export zu tun haben. Daher wird ein weiterer Parameter bei der Anfrage aus dem Web nötig.
- `DownloadID` wurde entfernt, da es sich hierbei nur um einen String handelt, der genutzt wird um einen Export eindeutig zu kennzeichnen. Dieser String wird nun von der Webansicht erzeugt und ist für eine spezielle Anfrage eindeutig, sodass zwei Personen mit derselben Anfrage zur selben Export-Datei geleitet werden. Soll doppelte Arbeit vermeiden.
- `DownloadState` und `AlterableDownloadState` wurden angepasst, sodass nicht mehr nur die Zustände *Richtig* und *Falsch* für die Bereitschaft der Datei existieren, sondern ebenfalls einen Fehler angeben können, sowie die Info, dass ein derartiger Export nie angefragt wurde.
- Die `ExportProperties` können nun nur noch Cluster und nicht mehr Sensorlisten enthalten. Hängt mit der Änderung der Anfrage in dem Webinterface zusammen.
- `FileExtension` wurde durch einen String ersetzt, analog zum Import.
- Ebenso analog zum Import ist aus denselben Gründen die `NetCDFWriterStrategy` Klasse nicht dabei.
- Der `ExportStreamGenerator` wurde als Klasse entfernt, aber dessen Funktionalität in die Implementierung der `FileWriterStrategy` eingebaut. Ziel ist es, diese wieder dort auszubauen, aber statt der Rückgabe eines Streams, diesen für sich zu behalten und immer nur neue Werte zu pollen, sobald alles bisherige bearbeitet wurde.
- `FileExporter` verliert die Methode, die eine `DownloadID` erzeugt hätte, da diese Aufgabe nun vom Webinterface übernommen wird. Siehe Info zu `DownloadID`.
- Auch hier wurden anlog zum Import an einigen Stellen veränderte Parameter und Rückgabewerte verwendet, ohne die Logik der Abläufe zu ändern.

2.6.1 FrostStealer

- Sollte dazu dienen Testdaten von Sensorup zu erhalten.
- Ermöglicht es von einem bestimmten Server jegliche relevante Daten in dem im Projekt genutzten CSV-Standard zu exportieren. Beispieldateien mit etwa 25000 Observations wurden erstellt.

2.7 Webinterface

- Die Komponenten, aus denen sich das Webinterface / View zusammensetzt, wurden stark abstrahiert. Da für das Webinterface HTML, CSS und JavaScript verwendet wurden, ließ sich eine exakt gleiche und objektorientierte Klassenstruktur nicht vollständig implementieren.
- Es gibt konkrete Gridtypen, die das Adressieren und Darstellen von Clustern kapseln. Der Tile-/Clustertyp ist an den Gridtyp angebunden.

3 Muss- und Wunschkriterien

Kriterium	erfüllt	Beteiligte	Bemerkungen
MK1000	ja	Erik, Oliver, Patrick	
MK1010	ja	Erik, Oliver, Patrick	
MK1020	ja	Erik, Oliver	
MK1030	ja	alle	
MK1040	ja	Erik, Oliver, Patrick	
MK1050	ja	Erik, Jean, Patrick	
MK1060	nein		Nur skalarwertige Werte
MK1070	ja	Jean	Eingeschränkt auf Import
MK1080	ja	alle	
MK2000	ja	Thomas	
MK2010	ja	Thomas	
MK2020	ja	Thomas	
MK2030	ja	Erik, Thomas	
MK2040	ja	Jean, Thomas	
MK2050	ja	Erik, Thomas	
MK2060	ja	Erik, Thomas	
MK2070	ja	Erik, Thomas	
MK2080	ja	Erik, Thomas	
WK1000	ja	Oliver, Patrick	
WK1010	ja	alle	
WK1020	nein		Fehlerhafte Daten werden erst gar nicht gespeichert
WK1030	nein		
WK1040	nein		Jean: Import von CSV
WK2000	ja	Thomas	
WK2010	ja	Erik, Thomas	
WK2020	ja	Erik, Thomas	Statt an/ausschalten werden bestimmte ausgewählt
WK2030	ja	Thomas	
WK2040	ja	Thomas	
WK2050	nein	Erik, Thomas	Wird in Testphase behoben
WK2060	nein		
WK2070	ja	Thomas	Cluster sind dann transparent
WK2080	ja	Erik, Thomas	
WK2090	nein		
WK2100	ja	Thomas	

4 Unit-Tests

4.1 Database

- Die ObservationDataToStorageProcessor-Klasse ist gut getestet. Diese übernimmt im Moment alle Interaktionen mit der Datenbank.

4.2 Transfer

- Die wichtigen Komponenten wie Konsument, Sender und Serialisierer wurden ausgiebig getestet
- Alle weiteren Komponenten waren entweder in obigen Tests mit eingeschlossen, oder für Unit-Tests schlichtweg ungeeignet

4.3 Core

- Die zentralen Komponenten wie GeoPolygon und GeoGrid wurden auf wichtige Funktionen getestet
- Alle weiteren Komponenten waren entweder in obigen Tests mit eingeschlossen, für Unit-Tests schlichtweg ungeeignet, oder wurden aufgrund der geringen Relevanz zum Gesamtanteil und dem hohen Zeitdruck (vorerst) ausgelassen

4.4 Import

- FrostSenderTest: Prüft ob das Senden an den FROST-Server erwartungsgemäß ausgeführt wird. Test an pavos-master.

4.5 Webinterface

- Funktionen wurden getestet aber bisher ohne Unit-Tests. Javascript-Unit-Test Tool muss verwendet werden um auch Code-Überdeckung zu ermitteln.

5 Wöchentliche Arbeitsverteilung

Ursprünglich geplante Arbeitsaufteilung (alphabetisch nach Nachnamen sortiert):

Name	Aufgabe
Jean	Import und Export
Thomas	Webinterface
Oliver	Bridge und Datenbank
Patrick	Core
Erik	Transfer (Graphite und Grafana)

Nachfolgend genannte Wochennummern sind folgende Zeiträume:

Woche	entsprechender Zeitraum
1	02. Juli - 08. Juli
2	09. Juli - 15. Juli
3	16. Juli - 22. Juli
4	23. Juli - 29. Juli
5	30. Juli - 05. August
6	06. August - 12. August
7	13. August - 19. August
8	20. August - 26. August

Insgesamt ergibt sich folgende Anzahl an Codezeilen (LOC):

Name	LOC (Schätzung)
Jean	1400
Thomas	3500
Oliver	1000
Patrick	1100
Erik	5000

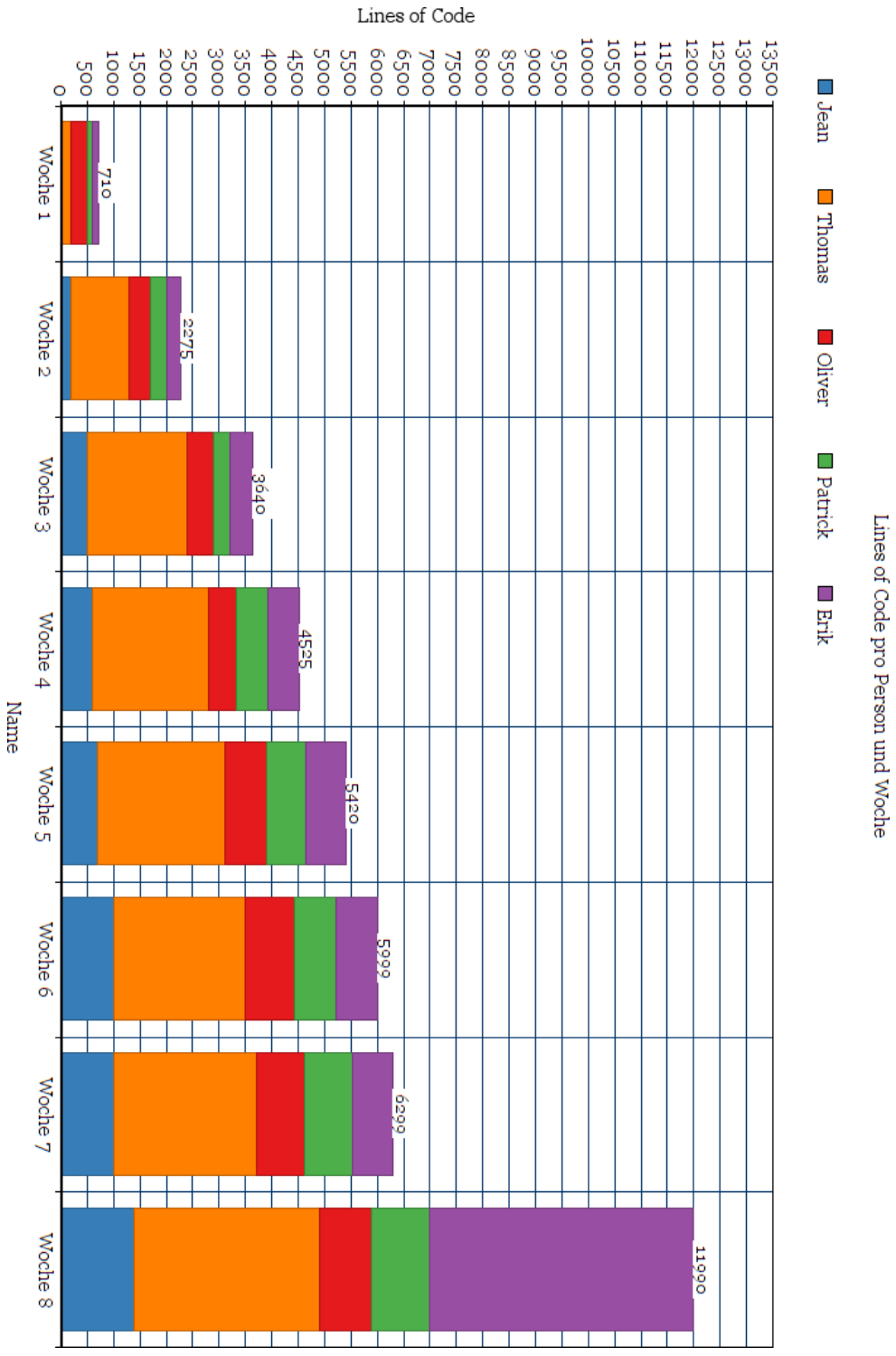

Abb. 5.1: Codezeilen pro Person pro Woche (Schätzung)

5.1 Jean

Woche 1 Einarbeitung ins Thema.

Woche 2 Testdaten erzeugen durch DummyReaderStrategy.

Woche 3 Implementierung der restliche Import-Klassen, ausgenommen anderer FileReaderStrategy.

Woche 4 Download Implementierung und Teile des Exportes.

Woche 5 Umbau der Exportservlets zu einem einzelnen Servlet.

Woche 6 FrostStealer und CSVReaderStrategy.

Woche 7 Abwesenheit durch Urlaub ohne Arbeitsgerät.

Woche 8 Der Rest ohne NetCDF Implementierung.

5.2 Thomas

Woche 1 Initialisierung des Codes.

Wochen 2 bis 5 Implementierung der visuellen Komponenten des Webinterfaces, Struktur und Styling mit Bootstrap, Karte, Tabelle, Buttons, Modals

Wochen 6 bis 8 Code-Umstrukturierung, vor allem Javascript Umstrukturierung, Implementierung von wichtigen Webinterface Funktionalitäten, Grid-Funktionalität, Listeners, Http-Requests, State für Favoritenspeicherung

5.3 Oliver

Ein Großteil der Arbeit zur Bridge fand bereits vor Beginn der Implementierungsphase statt, da die Funktionalität des Cores von einer funktionsfähigen Bridge abhängt. So war zu Beginn die Bridge bereits beschränkt einsatzbereit und konnte erfolgreich MQTT-Nachrichten als String an Kafka weiterleiten.

Woche 1 Es wurde mit Testdaten experimentiert um die korrekte Funktionsweise der Bridge zu testen. Fehler in der Bridge wurden behoben.

Wochen 2 bis 3 Einarbeitung in Avro-Schemas und Schema-Registries. Definieren von Avro-Schemas. Hierbei traten Schwierigkeiten wegen rekursiven Referenzen auf andere Objekte auf, die in der SensorThings API beschrieben sind. Behebung von Fehlern bzgl. der generierten Avro-Objekte. Umschreiben des KafkaProducers um Avro-Objekte an Kafka zu senden.

Woche 4 Klausurvorbereitung.

Woche 5 Fehlerbehebung in der Bridge, u.a. Konvertierung von iot.navigationLinks, einarbeiten in Memcached.

Woche 6 Klausurvorbereitung.

Wochen 7 bis 8 Implementierung der Klassen für die Datenbank, Dokumentation und Testfälle für Module Bridge und Datenbank.

5.4 Patrick

Woche 1 Einarbeitung in Kafka und Kafka Streams.

Woche 2 Erste Kafka-Anwendungen für das Projekt, Daten können nun aus Kafka ausgelesen werden. Hier gab es Probleme mit dem Serializer.

Woche 3 Klausurvorbereitung.

Woche 4 Ich habe mein erster Prozess zu schreiben, zum Zusammenfügen von dem Topic Observation und FeatureOfInterests. Dabei ist mir aufgefallen, dass wir im Projekt mit Kafka 1.0.1 arbeiten und das wissen welches ich mir vorher angeeignet hatte auf Version 0.11 bzw 0.10 basiert war. Das hat zur Folge dass sich doch viele kleine Dinge geändert haben, wo ich anfangs den Fehler dafür nicht gefunden habe. Somit für die einfache Zusammenfüge Prozess doch recht viel Zeit in Anspruch genommen hat und viele verschiedene ansätze versuchen musste, weil ich zudem mit der Kafka Streams API noch nicht ganz klargekommen bin.

Woche 5 Hier habe ich dann endlich mein zusammenfügt Prozess fertig bekommen und auch ausgiebig getestet. Es gab Schwierigkeiten mit dem Testen von Kafka Stream Applications. Zudem habe ich den PropertysManager eingebaut welcher sich um das beziehen der Properties der einzelne Kafka Stream/Consumer kümmert.

Woche 6 Hier wurde neue Prozesse entwickelt welche im Weiteren Verlaufes des Projektes nicht mehr genutzt werden, weil die Probleme anders gelöst wurden und zudem war ich hier auch wieder etwas abwesend wegen einer Klausur.

Woche 7 In der Zeit war ich in der Heimat und habe mich leider auch richtig erkältet wodurch ich leider nicht fähig war am Projekt weiter zu arbeiten.

Woche 8 Jetzt habe ich meine letzten Prozesse fertig gestellt, den Export Prozess, welche alle Topic zusammenfügt und der Grid Prozess, welcher die Grid Methodik zum Laufen bringt.

5.5 Erik

Woche 1 Initialisierung - Code des Entwurfs wurde zu GitHub hinzugefügt und Maven wurde aufgesetzt.

Wochen 2 bis 5 Transfer - Die Verbindung zu Graphite wurde aufgesetzt und nachträglich wurde Grafana noch ergänzt. Generell ist die Code-Struktur ähnlich zum Entwurf geblieben. Allerdings wurde die Struktur beim coden in weitere Sinnabschnitte unterteilt, sodass mehr Modularität gewährleistet ist. Nach dem ersten Erstellen wurde weiter optimiert.

Woche 6 Transfer - Informationen wurden gesammelt & Prüfungen geschrieben.

Woche 7 Transfer - Performance und Stabilität verbessern.

Woche 8 Core & Transfer - Da bis zum jetzigen Zeitpunkt der Core nicht funktioniert hat, habe ich mich mit Hochdruck damit befasst. Ich habe selbstständig den kompletten Grid und die Cluster entwickelt, sowie die Verbindung zum Webinterface und zu Graphite & Grafana etabliert. Da keinerlei funktionierende Strukturen für diese vorlagen, habe ich sämtliche Konstrukte neu entworfen. Vorschläge und Ideen habe ich versucht umzusetzen. Schlussendlich bietet der Grid nun eine einfache Schnittstelle, bei der man manuell im Code nur neue Einträge hinzufügen muss. Zeitliche Vorgänge und getaktete Abläufe wurden intern behandelt und vor dem Benutzer versteckt.