



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	4
2	Änderungen am Entwurf	5
2.1	Bridge	5
2.2	Database	5
2.3	Transfer	6
2.4	Core	6
2.4.1	Grid	6
2.5	Import	6
2.6	Export	7
3	Muss- und Wunschkriterien	9
4	Unit-Tests	10
4.1	Transfer	10
4.2	Core	10
4.3	Import	10
5	Wöchentliche Arbeitsverteilung	11
5.1	Jean	12
	Woche 1	12
	Woche 2	12
	Woche 3	12
	Woche 4	12
	Woche 5	12
	Woche 6	12
	Woche 7	12
	Woche 8	12
5.2	Thomas	12
	Woche 1	12
	Woche 2	12
	Woche 3	12
	Woche 4	12
	Woche 5	12
	Woche 6	12
	Woche 7	12
	Woche 8	12

5.3	Oliver	13
	Woche 1	13
	Woche 2	13
	Woche 3	13
	Woche 4	13
	Woche 5	13
	Woche 6	13
	Woche 7	13
	Woche 8	13
5.4	Patrick	13
	Woche 1	13
	Woche 2	13
	Woche 3	13
	Woche 4	13
	Woche 5	13
	Woche 6	13
	Woche 7	13
	Woche 8	13
5.5	Erik	14
	Woche 1	14
	Wochen 2 bis 5	14
	Woche 6	14
	Woche 7	14
	Woche 8	14

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.

In der ersten Hälfte wurde hauptsächlich...

In der zweiten Hälfte wurde schließlich...

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.navigationLinks` für zusammenhängende Objekte angibt (statt einer Menge an `@iot.ids`), 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` ererbenden 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` ererbende 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

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

2.5 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 ausreicht 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 eine 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 analog 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.

3 Muss- und Wunschkriterien

Nr. Kriterium	erfüllt	Beteiligte	Bemerkungen
MK1000	ja	Erik	
MK1010	ja	Erik	
MK1020	ja	Erik	
MK1030	ja	Erik	
MK1040	ja	Erik	
MK1050	ja	Erik	
MK1060	nein		
MK1070	nein		
MK1080	ja	Erik	
MK2000	nein		
MK2010	nein		
MK2020	nein		
MK2030	ja	Erik	
MK2040	nein		
MK2050	ja	Erik	
MK2060	nein		
MK2070	ja	Erik	
MK2080	ja	Erik	
WK1000	nein		
WK1010	nein		
WK1020	nein		
WK1030	nein		
WK1040	nein		
WK2000	nein		
WK2010	ja	Erik	
WK2020	ja	Erik	
WK2030	nein		
WK2040	nein		
WK2050	nein		
WK2060	nein		
WK2070	nein		
WK2080	ja	Erik	
WK2090	nein		
WK2100	nein		

4 Unit-Tests

4.1 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

4.2 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

4.3 Import

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

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)

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

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 Einarbeitung ins Thema.

Woche 2 Testdaten erzeugen durch DummyReaderStrategy.

Woche 3

Woche 4

Woche 5

Woche 6

Woche 7

Woche 8

5.3 Oliver

Woche 1 Einarbeitung ins Thema.

Woche 2 Testdaten erzeugen durch DummyReaderStrategy.

Woche 3

Woche 4

Woche 5

Woche 6

Woche 7

Woche 8

5.4 Patrick

Woche 1 Einarbeitung ins Thema.

Woche 2 Testdaten erzeugen durch DummyReaderStrategy.

Woche 3

Woche 4

Woche 5

Woche 6

Woche 7

Woche 8

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.