

Real-time visualization of analyzed industrial communication network traffic

Implementation Report

PSE Group

Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB

Advisor: M.Sc. Ankush Meshram

Version 1.0.0

Contents

1	Design	1
1.1	Introduction	1
1.2	Changes in the Design	1
1.2.1	User Interface changes for better aesthetics and convenience	1
1.2.2	User Interface changes for usability improvements	1
1.2.3	Refactoring for cleaner code and changes for convenience reasons	1
1.2.4	Changes due to clarified requirements	2
1.2.5	Changes due to oversights	2
1.2.6	Changes due to unexpected complexity	3
1.3	List of implemented must- and should-requirements	4
1.3.1	List of implemented must-requirements	4
1.3.2	List of implemented should-requirements	4
1.3.3	List of not implemented must-requirements	4
1.3.4	List of not implemented should-requirements	4
1.4	Timeline and Delays	4
1.5	Overview of unit tests	6

1 Design

1.1 Introduction

This implementation report covers changes in the design from the one described in the original design phase and describes the current state of the implementation by lists of completed and not completed requirements.

1.2 Changes in the Design

Changes are grouped together according to the reason for the change.

1.2.1 User Interface changes for better aesthetics and convenience

- New Login Page
Opted for a different graphical design than the one in the mockups due to aesthetic reasons. The functionality and behavior of the login page remains the same as in the design docs.
- Removal of the filter button inside diagram control containers
The filters now can be easily accessible inside the config modal of the diagram.

1.2.2 User Interface changes for usability improvements

- New text input box for WebSocket endpoint
Added the feature of choosing an arbitrary WebSocket endpoint by persisting the input with Browser Local Storage API, making the frontend application completely standalone and therefore largely simplified the deploy process of the entire DHSTTOS suite.

1.2.3 Refactoring for cleaner code and changes for convenience reasons

- Data formatting helper function on the frontend
Added a helper function **formatData = ({ groupName, x, y, rawData = [] }) : Object[]** which converts the raw data points the frontend receives from the server to structured data arrays which allows easy data passing into the diagram drawing routines.
- Add parameter
Added parameter DBname to **MongoConsumer(user, pass, dbName)** for creating a reference to pass onto the MongoClientMediator
- Refactoring
Add attribute **private KafkaConsumer<String, String> consumer** because other functions need to use the consumer
- Refactor: extract instance attribute
Add attribute **private MongoDBDatabase db** as a reference to the database all methods need to access.

- Convenience functions for different data types
Added variations of **addRecordToCollection(Record record, String collection)** that take a document or an list of documents or an array of record instead of a Record.
- Add convenience function
Added **getCollectionAsRecordsArrayList()** to DataProcessor.
- Refactor passing the current mediator object
Add parameter **MongoClientMediator** to **public static void ProcessData:processData(String collectionName, MongoClientMediator clientMediator)** so that **processData** can use it to write the processed data to the database. Remove attribute **ProcessData:MongoClientMediator client** which was used for this before.
- Add convenience function
Add method **public static void processData(ArrayList<String> collectionNames, MongoClientMediator clientMediator)** to process a list of collections (instead of calling **processData** for each collection.
- Add convenience function
Added method **public Document getNewAggregatorDocument(Date tstmp)** for easier handling of date values.
- Add convenience attributes
Add the variables Variables **private ArrayList<Map<String, Object>> connectionsMapList** and **private Document currentDocument** to the classes **FlowRatePerSecond** and **NumberOfConnectionsPerNode** to keep track of which document is being processed now and which connections happened within this second.
- Refactoring for cleaner code in protocol handling
Change the protocol parsing in class **ClientProtocolHandler** from a switch construct to using a private enum.

1.2.4 Changes due to clarified requirements

- Differing input formats for Date/Timestamp
Split class **PacketRecord** into **PacketRecordDesFromMongo** and **PacketRecordDesFromKafka** to handle different formats.

1.2.5 Changes due to oversights

These are changes and additions due to oversights and mistakes in the original design.

- added dbName to MongoClientMediator since we need to know from which DB we want to read/write collections.
- Unspecified return type
The return type of **public ArrayList<Document> processData(ArrayList<Record> records)** in **IAggregator** was unspecified in the Design document.

- Session handling
To handle session state, **Hub:login()**, **Hub:loginWithToken()**, and **Hub:logout()** were added. To keep track of client session state, the private attributes **Hub:sessions** and **Hub:loginTokens** were added.

1.2.6 Changes due to unexpected complexity

These changes can be attributed to lack of familiarity with the used components and libraries.

- Workaround for Kafka's API
Change **getAllTopics()** to **getAllTopicsPartitions()**: return a Collection of topic partitions essentially to force kafka to send all records from the start. It was complex to make kafka read all the topics from the beginning. Secondary aspect: convenient because it relegates topic creation to another method.
- Workaround for Kafka's API
Add method **ArrayList<String> getTopicsForProcessing()** because there are some topics in kafka which are for internal use, e.g. `__consumeroffsets`. This returns the topics we need to process.
- Exception handling
The constructor for class **MongoClientMediator** now throws a `LoginFailureException` instead of forwarding an unchecked exception.
- Converting between different APIs
Add method **mongoliteratorToStringArray(Mongoliterable)** because the hub expects an array but the mongodb returns a `Mongoliterable`.
- Handling the login happening in another websocket session than the main app
To deal with a restart of the websocket connection when changing from the login page to the main page, session handling was changed. Added the **LOGIN_TOKEN** request to the protocol and **Hub:loginWithToken**.
- Adapt to React and MobX
To adapt to the observer-driven architecture of React and MobX, store data from the server in datastructures **dataStore.rawData** and **dataStore.alarms** instead of returning it as return values of **getAvailableCollections()**, **getCollection()**, **getCollectionSize()**, **getRecordsInRange()** and **getRecordsInRangeSize()** in `wsutils.js`.

1.3 List of implemented must- and should-requirements

1.3.1 List of implemented must-requirements

FR100, FR110, FR200, FR300, FR400, FR500, FR700, FR710, FR720, FR1110, FR1300, FR1310
cancelled after Mike left: FR800

1.3.2 List of implemented should-requirements

- FR1332 filter to compute flow rate
 - this has instead been implemented in the backend which provides this as a new data stream
- FR1400

1.3.3 List of not implemented must-requirements

- FR600 dynamically change the selected/displayed components
- FR900 The amount of data can be limited via a slider [...] to which all diagrams must update to.
- FR910 Within the slider the user is able to scroll through the timeline and the diagrams need to react in real-time.
- FR1000 auto scroll
- FR1100 pick data points, hover
- FR1200 selecting data points
- FR1210 create new diagram from selected data
- FR1330

1.3.4 List of not implemented should-requirements

- FR1320 per-diagram filters

1.4 Timeline and Delays

There were delays in the implementation phase caused by

- clarification of requirements
- evaluation of graphics libraries
- familiarization with the used APIs and libraries

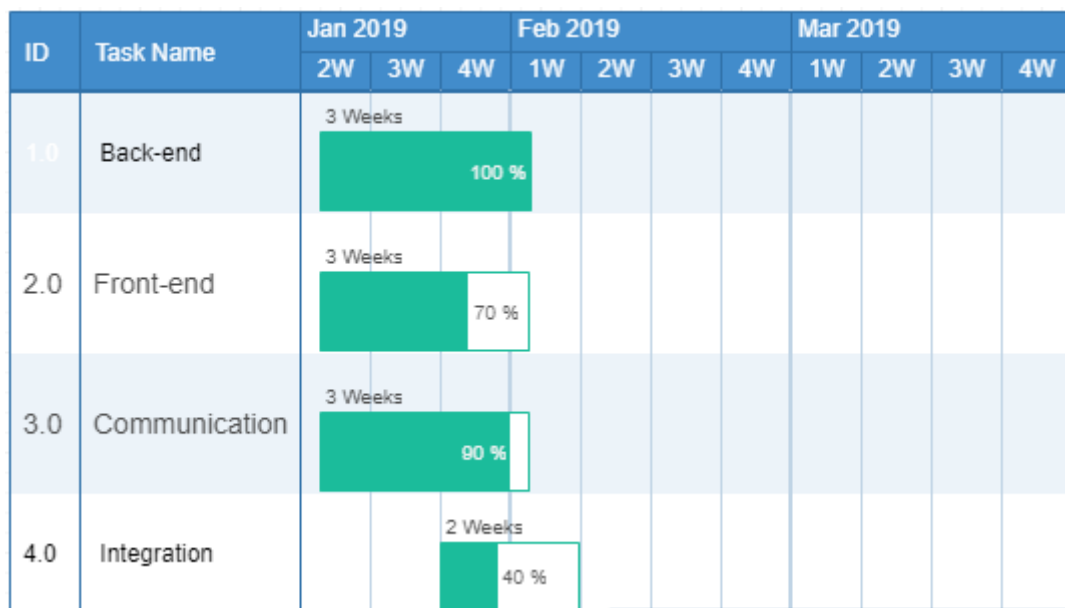


Figure 1: The timeline.

1.5 Overview of unit tests

Number of unit tests:

DataProcessor	2
Test 1 and 2 both test the functionality of trying to process missing data. In both cases the aggregators responsible should not fail and simply make sure nothing gets added into the Database.	
Hub	4
Test session handling methods.	
MockMongoDBUserSession	4
MongoClientMediator	5
Test1 and 2 and 3: test different credential errors, like missing credentials or credential without access to the necessary databases. All tests cases return a LoginFailureException.	
Test 4 and 5 test handling null reference objects to adding records into MongoDB. In this case then simply, nothing is added into the database.	
MongoConsumer	2
Test 1 : tests the case on failure to communicate with kafka, the consumer exists and reports this to the console but operation of all other parts of the program continue since failure of this does not affect working with data already stored in the database.	
Test2: tests case when the consumer does not have the right credentials to access the MongoDB, the Consumer should then fail to be created and call a LoginFailureException.	
TestClientProtocolHandler	2
FormatData.	
Testing the GUI would be possible however, this is both complex and time consuming. Due to time constraints minimal testing was done outside of basic functionality.	