**Interim Report**

**EE5500**

Name: Wojciech Lesnianski

Student number: 1644612

Electronic and Computer Engineering

School of Engineering and Design



Dr. Ali Mousavi

Friday, March 16, 2018

School of Engineering & Design

Electronic & Computer Engineering

MSc Distributed Computing Systems Engineering

Brunel University West London

Simulation and Performance Analysis

of a Distributed Position Correction

Scheme for Unmanned Aerial Vehicles

Dionysios Satikidis

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Declaration:** I have read and I understand the MSc dissertation

guidelines on plagiarism and cheating, and I certify that this

submission fully complies with these guidelines.

Abstract

Acknowledgements

Me for taking teh time to write all this stuff

List of Figures

[Figure 1-1: Waste-water treatment plant stages 2](#_Toc509208678)

[Figure 1-2: Waste-water treatment in Europe 4](#_Toc509208679)

[Figure 1-3: Water suppliers in the United Kingdom 6](#_Toc509208680)

[Figure 1-4: Big picture – System in context 10](#_Toc509208681)

[Figure 2-5: Top-Down design 16](#_Toc509208682)

[Figure 2-6: Spiralmodel combined with top-down approach 17](#_Toc509208683)

[Figure 2-7: Timeplan / Gantt chart 28](#_Toc509208684)

[Figure 3-8: Waste-water treatment plant stages 30](#_Toc509208685)

[Figure 3-9: Single window system example – big picture 41](#_Toc509208686)

[Figure 4-10: Whitebox diagram – harmonization system 51](#_Toc509208687)

[Figure 4-11: Blackbox diagram – harmonization system and its interfaces 53](#_Toc509208688)

[Figure 4-12: Data-schema relations and hierarchy 55](#_Toc509208689)

[Figure 4-13: Database schema 60](#_Toc509208690)

[Figure 4-14: Use case diagram 62](#_Toc509208691)

[Figure 4-15: Component diagram of the harmonization service 63](#_Toc509208692)

[Figure 4-16: Workflow diagram – table formatted data 68](#_Toc509208693)

[Figure 4-17: Example table 69](#_Toc509208694)

[Figure 4-18: Workflow diagram – tree formatted data 71](#_Toc509208695)

[Figure 4-19: Component diagram of the data provider 72](#_Toc509208696)

[Figure 5-20: Service tolerance to data-misinterpretation diagram 75](#_Toc509208697)

[Figure 5-21: Example data table 81](#_Toc509208698)

[Figure 5-22: Example data table – multiple indicators 81](#_Toc509208699)

[Figure 5-23: Example data table – vertical orientation 82](#_Toc509208700)

[Figure 5-24: Example of a table with two valid indicator names for one value column 86](#_Toc509208701)

[Figure 5-25: End of a table with 10000 entries containing indicator name, value and datetime 87](#_Toc509208702)

[Figure 5-26: The amount of entries in the QualityIndicator table before the test 88](#_Toc509208703)

[Figure 5-27: Request result (10.000 entries) 88](#_Toc509208704)

[Figure 5-28: Database after harmonization request 88](#_Toc509208705)

[Figure 5-29: Visual Studio 2017 performance profiler 89](#_Toc509208706)

[Figure 5-30: Harmonization result of 100000 entries-file 90](#_Toc509208707)

[Figure 5-31: Visual Studio 2017 performance profiler – 100000 entries 90](#_Toc509208708)

[Figure 5-32: Visual Studio 2017 performance profiler – 1000000 entries 91](#_Toc509208709)

[Figure 5-33: Diagram: Expected time vs actual result – time to harmonize 92](#_Toc509208710)

[Figure 5-34: Example postman call towards locally running web-service 94](#_Toc509208711)

[Figure 5-35: Example postman call for treatment step types on a water plant 94](#_Toc509208712)

[Figure 5-36: Example postman call for quality indicator types of a treatment step 95](#_Toc509208713)

[Figure 5-37: Example postman call for quality indicators of a treatment step 95](#_Toc509208714)

[Figure 5-38: Swagger file of the harmonization system 96](#_Toc509208715)

List of Tables

[Table 2-1: Risks, Effects, Classifications and Strategies 25](#_Toc509208716)

[Table 2-2: Project tasks in categories and their descriptions 27](#_Toc509208717)

[Table 4-3: Waste-water treatment plant in the context of this system 57](#_Toc509208718)

[Table 4-4: Treatment step metadata in the context of this system 57](#_Toc509208719)

[Table 4-5: Water quality indicator in the context of this system 58](#_Toc509208720)

[Table 4-6: Basic quality indicator types 58](#_Toc509208721)

[Table 4-7: Database schema description: Tables, description and dependencies 61](#_Toc509208722)

[Table 4-8: PullSources table definition 64](#_Toc509208723)

[Table 5-9: Measured harmonization process times 91](#_Toc509208724)

List of Abbreviations

|  |  |
| --- | --- |
| VIN | **V**ehicle **I**dentification **N**umber |
| SF | **S**auberes **F**ahren |
| C# | C-Sharp |
| UMTS | **U**niversal **M**obile **T**elecommunications **S**ystem |
| XML | **E**xtensible **M**arkup **L**anguage |
| Wifi | **Wi**reless **Fi**delity |
| OBD | **O**n-**B**oard-**D**iagnose |
| OSM | **O**pen **S**treet **M**ap |
| GUI | Grafische Benutzeroberfläche (**G**raphical **U**ser **I**nterface) |
| WPF | **W**indows **P**resentation **F**oundation |
| PNG | **P**ortable **N**etwork **G**raphics |

List of Therms

|  |  |
| --- | --- |
| VIN / Vehicle Identification Number | **Vehicle Identification Number**,  ist die international genormte, 17-stellige Nummer, mit der ein Kraftfahrzeugeindeutig identifizierbar ist. |
| Fahrzeugtyp | Fahrzeugtyp wird in diesem Projekt gleichgesetzt mit von der Informationen, welche aus der VIN gewonnen werden können. Dazu gehören der Hersteller, das Modell und die Motorisierung. |
| OSM / Open Street Map | Open Street Map ist ein frei nutzbares Projekt, welches Geodaten bereitstellt. |

Content

[1. Introduction 1](#_Toc508647053)

[1.1 Context of the Project 2](#_Toc508647054)

[1.2 Problem Description 7](#_Toc508647055)

[1.3 Aims and Objectives 9](#_Toc508647056)

[2. Methodology and Project Organisation 14](#_Toc508647057)

[2.1 Software Development Process 14](#_Toc508647058)

[2.2 Tools and Architectures 18](#_Toc508647059)

[2.3 Strengths and Risks 22](#_Toc508647060)

[2.4 Project Management 25](#_Toc508647061)

[3. Literature Review 29](#_Toc508647062)

[3.1 Water-Plants 29](#_Toc508647063)

[3.2 Data Harmonization 38](#_Toc508647064)

[3.3 Comparable Industries and Projects 44](#_Toc508647065)

[4. Design and Implementation 50](#_Toc508647066)

[4.1 System overview 50](#_Toc508647067)

[4.2 Data-Schema 53](#_Toc508647068)

[4.3 Data Base 59](#_Toc508647069)

[4.4 Harmonization Service 61](#_Toc508647070)

[4.5 Data Provider 72](#_Toc508647071)

[5. Experimental Results and Analysis 75](#_Toc508647072)

[5.1 Proving the concept 75](#_Toc508647073)

[5.2 Harmonization Strategies 75](#_Toc508647074)

[5.3 Was ist das 77](#_Toc508647075)

[6. Summary and Further Work 78](#_Toc508647076)

1. Introduction

Access to clean water is the most basic and fundamental type of the human infrastructure. The quality of life highly depends on the accessibility to clean water. We require water not only for drinking, but also for cooking, and washing. Additionally, various professions and commercial establishments, like farms or restaurants, could not exist without certain quality and quantity of water. The quantity of clean water in most cases, depends on collecting water and sewage from rivers and lakes, cleaning it in dedicated **water-plants** and thus bringing it to a specific quality standard, and then distributing it.

A software groundwork for acquisition, analysis and modelling of historical and real-time data of water-plants could become the first step to provide an infrastructure capable of “keeping track of the water on a national level” – its amount, quality and source, as well as making forecasts and statistical reports easier. The topic of this thesis is to design and create a prototype of a system to acquire, harmonize and provide waste-water treatment plants -related historical and real-time data.



Figure 1-1: Waste-water treatment plant stages

(Source: [http://www.water.org.uk/about-water-uk/wastewater 18.09.2017](http://www.water.org.uk/about-water-uk/wastewater%2018.09.2017) last accessed: 28.09.2017)

Even though for a common/public user, the most significant outcome is the one of step 6 – Final treatment[[1]](#footnote-1), which also indicates the quality of water available for public usage, the incoming and outgoing water of the other steps provide data for other kinds of interesting analyses, especially due to the fact, that each step deals with a specific kind of water quality related problem, meaning that all “possible to gather” data[[2]](#footnote-2) should also be gathered, harmonized and stored by the system, for further investigation.

* 1. Context of the Project

To understand how important waste water treatment is, one has to understand the dimensions of its effect. On average, a UK water-customer pays 1 pound a day to be able to enjoy high quality water. This money is highly needed and is used for the treatment of around 16 billion litres of wastewater, gathered in around 345 thousand kilometres of culvert and cleaned in about 9000 wastewater plants – every day, as well as developing the water infrastructure in the country. [3]

Every country has its own way of dealing with the regulations of the water treatment process. (<https://www.eea.europa.eu/data-and-maps/indicators/urban-waste-water-treatment/urban-waste-water-treatment-assessment-4>): Taking into consideration the 4 Steps:

* Collecting wastewater
* Primary treatment[[3]](#footnote-3)
* Secondary treatment
* Tertiary treatment

The trend in Europe over the past decades was to connect more and more population to waste water treatment plants. In northern countries 80% have already been reached in 1995, with over 70% of the water receiving tertiary treatment. In central Europe (this includes the UK) on average over 95% of the population enjoy treated water. The trend also shows increasing amount of tertiary treating among all countries:



Figure 1-2: Waste-water treatment in Europe

What UK is doing great, is the fact that 100% of the population is connected to at least one of the waste water treatment plants. Also, all of the collected water receives at least a secondary treatment (Data from 2015). What UK still lacks, is tertiary treatment which is close to 100%, like in other countries including Germany, Netherlands, Denmark and Austria. Again, the trend shows improvement over the past years, but it seems rather slow in the progress. The tertiary treatment usually significantly reduces nitrogen and phosphorus pollution and might not always be required, but is still recommended by the Urban Waste Water Treatment Directive.

On a big scale, the UK can compare its achievements to other countries to find out what causes their good and bad results. The comparison on a small scale includes comparing the single water-plants within the UK to see which treatments steps are lacking, where the water quality is better, the treatment more effective and efficient and why.

The UK-wide water supply regulations are set by the government and regulate the water treatment process of every water provider whose area is wholly or partially in the United Kingdom. The list of indicator parameters is long and contains minimum, maximum values and ranges within which values are allowed to lie. Only if all regulations apply, the water may be called drinking water. With all the regulations and monitoring organizations the quality of UKs water might seem assured – yet the process of doing so is very troublesome and laborious. Twelve big companies, responsible for water and sewerage, cover most of UKs water supply. Additionally, there are some water-only companies providing water for some of the remaining regions. [4] [5]



Figure 1-3: Water suppliers in the United Kingdom

(Source: https://www.water.org.uk/consumers/find-your-supplier last accessed: 28.09.2017)

**[a]**

**Water transfer and interconnection**

Depending on the location, the population and the climate, some regions of the UK have less available clean water than others. Those regions usually lie in the south and east. This is why a system for water moving was needed and build already in the 17th century (New River to transfer water from Hertfordshire to London). This transfer and interconnection system was since then optimized and expanded to provide water to regions in need, despite other water companies being “responsible” for this region[[4]](#footnote-4).

Water can be transferred treated or untreated using canals, pipes, aqueducts or rivers. Treated water is typically transported over buried pipes. Transferring the water is costly, especially when lowland reservoirs have to supply upland reservoirs and the water needs to be pumped instead of having the gravity doing most of the work. This is why water should not be transferred if not explicitly necessary. Even though water transfer – or “water trade” when referring to water transfer between companies, is costly, it is still in most cases cheaper in money and energy than water desalination, and thus promoted by the government and regulators in the UK.

* 1. Problem Description

The water quality is regulated UK-wide, yet the way the different companies ensure their quality and monitor their water treatment process is not unified. This makes comparison of data between companies and water-plants, as well as getting a global picture difficult. Reacting to lack of quality water in specific regions, or forecasting such a scenario, while still monitoring which of the remaining regions has enough “spare” quality water to help out the company in need would be a lot easier with a common information base. It would simplify the monitoring of local area changes caused by changes in the water and wastewater treatment regulations. To assure better forecasts or more meaningful reports, other information bases, like weather information might be taken into account – but using those external systems are not a topic in this part of the (data-gathering) system.

The advantages of a big dataset from various sources are obvious – especially in a case where the geographical location of sources also matter. Co-operating, comparing, planning, monitoring and analysing is a lot easier when all the data is stored at seemingly one place in a unified format.

Several questions need to be investigated upon before the approach of gathering the water information from various water-plants can be attempted[[5]](#footnote-5):

* Which data is available for the existing water-plants
  + **Which water-quality indicators**
  + How can the data be accessed
  + How can the data be legally used
  + What needs to be done to access the data
* Which data is mandatory to create a useful platform
  + Who are the stakeholders
  + Do all / most water-plants provide this data
  + **Which format is the data provided in**
  + **Which format is optimal for this system**
* What is the best way to handle this data in terms of
  + Storage
  + Security
  + **Processing**
* What is the best way of harmonizing the data in terms of
  + **Flexibility** (adding new sources and components / reacting to changes in source schema)
  + **Performance**
  + **Usability** (Target-Schema)
* What is the best way to provide an entry point for various data types
  + **Historical data**
  + **Actual data** (real-time / near real-time)
  + Non-water-data which could be useful anyway (i.e. for forecasts or to put more context on the stored data)
  1. Aims and Objectives

The following chapter describes the aims and objectives of the dissertation. It is separated into two parts. The first part describes the general objectives of the dissertation and the second describes the aims and objectives towards the required system, in form of requirements. The requirement form takes into account that the most fitting strategies and specific techniques have yet to be determined.

General objectives

The purpose of this dissertation is to investigate and design a knowledge and data engineering infrastructure for big amounts of water and wastewater treatment-process-specific data. This includes answering the questions asked in the chapter 1.2 – “Problem Description”, as well as designing an effective solution to harmonize data monitored within various water-plants based on the answers. Finally, a prototype of the designed system should be implemented and act as a proof of concept for the design. Simulated and historical data should be used to test the systems harmonization capabilities, as well as its flexibility in terms of expansion by further plants, treatment steps and quality indicators. The below picture shows the basic idea of the system integrated with the real world. The scope of this project lies within the Large Network Performance Collider, which represents the harmonization and storage layer.



Figure 1-4: Big picture – System in context

(Source: [8] Page 3)

Requirements

The systems functionality will be defined in the following requirements. To ensure readability and a better overview the requirements will be numbered as followed:

|  |  |
| --- | --- |
| **Top-Level Requirement:** | 10.000 |
| **Required Use Cases:** | 11.000 |
| **Requirements for the Use Cases:** | 12.XXX |
| **Requirements for technical functions:** | 13.XXX |
| **Requirements for the quality indicators:** | 14.XXX |
| **Requirements for the limitations of the solution:** | 15.XXX |

* + - 1. Functional Requirements

**Top-Level Requirement**

**Requirement 10000**

The developed system must **gather and accept** various, defined forms of waste-water treatment-plant data, **harmonize** the data into a common format, **validate** the harmonized data and **store** it. The system should **provide** an endpoint for the outside world to request the stored data.

**Requirements for the Use Cases**

**Requirement 11000**

The system should fulfil the following Use Cases:

* The user must be able to send data to the system in order to harmonize it. The administrator must be able to define endpoints and matadata of water-plants in order for the system to pull the data from the defined source and harmonize it. (Use Case: “**Harmonize Data**”)
* The system must provide endpoints for the users to access the harmonized data and process the requests. (Use Case: “**Provide Data**”)

**Requirements for specific Use Cases:**

* **Requirements for the Use Case “Harmonize Data”**

**Requirement 12110**

The system must provide an endpoint which accepts multiple formats commonly used for storing of data monitored by waste-water treatment-plants.

**Requirement 12120**

The system must have a possibility to configure endpoints and endpoint specific metadata and pull the data from the configured endpoints.

**Requirement 12130**

Incoming data (if in the correct format) must be transformed (harmonized) into a predefined format.

**Requirement 12140**

The harmonized data must be stored by the system.

**Requirement 12150**

The harmonization process must include data-validation.

* **Requirements for the Use Case “Provide Data”**

**Requirement 12210**

The system must provide endpoints to request available water-plants, treatment-steps and quality-indicators.

**Requirement 12220**

The system must provide an endpoint to request harmonized data.

**Requirements for technical functions**

**Requirement 13100**

Harmonization processes, which for some reason don’t end successfully must provide a meaningful error message to the harmonization requester.

**Requirement 13200**

A fitting simulation should be built to test the finished system.

* + - 1. Non-Functional requirements

**Requirements for the Quality Indicators**

**Requirement 14100**

Data consistency must be granted.

**Requirement 14200**

The harmonization process must be implemented efficiently and effectively.

**Requirement 14300**

The system must be able to handle multiple requests in parallel.

**Requirement 14400**

The system must be designed to be expandable during runtime[[6]](#footnote-6).

**Requirements for the limitations of the solution:**

**Requirement 15100**

The endpoints should use the REST technology and be defined accordingly to the REST guidelines.

**Requirement 15200**

The service must be deployable as a cloud service.

**Requirements 15300**

The storage must be able to handle big amounts of data efficiently.

1. Methodology and Project Organisation

This chapter will go into detail about the development method, tools and frameworks, which will be used to design and implement this project. It shows the combination of the methods and the project plan. Furthermore, this chapter will introduce to the time-plan describing the tasks of this project presented as a gnat chart.

* 1. Software Development Process

The system developed within this dissertation has a clear aim. There challenges and risks can be estimated, because the objective is clearly stated and there are comparable systems and projects already existing. Based on this knowledge, the system design will be done **top-down**. Since the system will consist of components, which can be implemented separately, the approach already follows the spirit of a top-down design, and the system as a whole will be designed one component after another.

Top-Down Design

**[I]**

The top-down design starts by creating an overview over the entire system. The details are neglected at first. This approach requires a good understanding of the system (opposing to a bottom-up design, where the system is being implemented without detailed understanding of every component, and the gain on information comes in during the development). The problem as a whole, and thus the system, is broken down into smaller sub-problems. This is why the approach is also called “Divide and Conquer strategy”. This breaking down continues, until the problem is small enough to be confidently solved on its own. While in big projects dividing the problems often results in breaking the project into sub-systems, where different programmers, or programmer teams can work separately on their own part of the system. The advantage of this approach in a one-man-project like this dissertation is the fact that subsystems become easier to estimate time and risk-wise the smaller they get. They can be prioritised or stripped down of less important features if the time wouldn’t be sufficient otherwise.

Figure 2-5: Top-Down design

**System**

Step 1

**Subsystem1**

**Subsystem2**

**Subsystem3**

Step 2

Step 3

Subsystem1-A

Subsystem2-A

Subsystem3-A

Subsystem1-A

Subsystem2-A

Subsystem3-A

Subsystem1-A

Subsystem2-A

Subsystem3-A

The advantage of dividing the system into subsystems is also the fact that each component can be tested individually if defined sufficiently. In most cases this will also assure that the subsystems become exchangeable – due to their modular character.

Spiral Model

Developing a system like this is special. The most stakeholders and contact persons knowing a lot of the subject will not have much software-technical knowledge. An iterative approach is of advantage in this case. Additionally it allows to detect risks and upcoming problems early on, which might lead to a change of the entire approach of the system.

A classical spiral model typically consists of four stages, in which the system is incrementally developed. In combination with the Top-Down design, the spiral model will help in developing each of the components during an iteration, planning the next iteration and thus the next component, and implementing the next component, as well as integrating it with the previous components.

**Evaluate alternatives, identify and resolve risks**

**Determination of objectives, alternatives, constraints**

**Plan next iteration**

**Develop and identify next product level, Integrate the developed system**

SubSystem1

SubSystem3

SubSystem4

**Prototypes**

**Project exit point axis**

**Project entry point axis**

Figure 2-6: Spiralmodel combined with top-down approach

SubSystem2

Each of the development phases not only includes work on the current subsystem, but might also include changes and adjustments in the previously developed systems to assure their integration in the current stand of the system with all its developed components.

* 1. Tools and Architectures

This chapter deals with the tools, frameworks and architectures, which will be used to develop this system. The core components of the system will be the database, the harmonization service and the data provider. It has to be made sure, that not only the components can fulfil the aims and objectives described in the previous chapters, but also that the various tools, frameworks and technologies are compatible with each other.

Visual Studio / C# .Net

The software will be developed in the programming language C#, developed by Microsoft. C# is an object oriented language based on the .Net framework. Its syntax is influenced by languages like Java and C++. C# is a language constantly developed by Microsoft and up to date on the state of art in terms of architectures, testability and support-tools and allows clean programming of back-ends, as well as front-ends. C# has lots of good documentation and can be used for building web-services for the most common cloud platforms like azure or amazon cloud. Another advantage of C# is, that even though the language itself is type-safe[[7]](#footnote-7), there is the possibility to work with the *dynamic* data-type, which usually counts as bad practice, but is very fitting for a software which desires a high tolerance for data formats. Those types allow for assigning of different types to one and the same variable during runtime. The risk of this are obvious: the developer has to keep track of what could be stored in the variable at all time to prevent runtime failures.

The most common IDE (Integrated Development Environment) for C# development is Visual Studio. Visual studio is a huge tool which provides a lot of possibilities for developers. Not only code can be developed within VS, but also databases and websites. Using additional frameworks (like Xamarin), Visual Studio even allows for development of Android and iOS applications. Summarized, C# in combination with Visual Studio (2017) provide for all the features which are required to develop this system with an adequate architecture.

Postman

Postman is a tool, which simplifies API development, by allowing the developer to create web calls, defining the call method, header, body and payload, as well as showing the call result. It also allows for easy passing of binary files.

Amazon Cloud

[J]

Amazon Cloud (Amazon Web Services or short „AWS“) provides a lot of services including databases and deployment tools. Services fully deployed in the (Amazon-) cloud can take advantage of its scalability and analysis infrastructure. Although the system under design has no matter of security in the current stage, future security requirements can be added without having to make big changes on the service itself. As for the storage, the AWS provides multiple options on how the data can be stored, including the object storage called “Amazon S3” useful for modern cloud native entity frameworks, a block storage called “Amazon Elastic Block Storage” which ensures data-loss protection through replications of the data, a File System called “Amazon Elastic File System”, which allows for intuitive storing and accessing files and many more. The features of AWS are by far sufficient to fulfil the requirements of this project and also provide tools to further improve and optimize the system once the aims and objectives of this dissertation are finished. This includes components like an API gateway which helps with the handling of security issues and load balancing.

Microsoft SQL Server / MSSQL Management Studio

The previously described Amazon Cloud allows for integration of the most common SQL databases. One of the options is the Microsoft SQL Server (short MSSQL). Like .Net and Visual Studio, MSSQL is also developed by Microsoft and thus offers good support and integration into the named framework and IDE. Aside of integration with all the used technologies, MSSQL also offers a fast relational database system which supports stored procedures as well as views. Both are of advantage when trying to build and integrate a flexible and testable system. Stored procedures are functions stored on the database, which can be integrated into the web-service and allow the execution of SQL queries, without the need to write actual SQL queries in the web-service itself. Views on the other hand, allow for the implementation of a lazy-loading mechanic, as well as for creation of types based directly on the database view structure. The lazy-loading mechanic allows to create so called *Queryable* objects, which act as a tunnel between the service and the database. The objects act like usual objects, but do not actually contain any other data, than an SQL query until the data is really needed. It then only retrieves the data which is actually needed or requested, instead of iterating through and retrieving the entire view data. This is especially useful for systems with high amounts of data and parallel processing.

JSON

One of the main components for this system is description of the data schema. This not only includes the description of the individual data, but also the format in which this data should be provided to the system, as well as the data format the system provides its data to the outside world.

JSON is a file/data format. It has become a very commonly used format for client-server communication and is replacing XML in this matter. It is supported by the modern programming languages and frameworks. JSON basically consists of object identifiers followed by the objects. Those objects might be further JSON types. The advantages of JSON when compared to other formats:

**Readability:**

JSON formatted objects are easy to read for a human, as its syntax focuses at removing the unnecessary parts and presents the data as an object similar to programming languages syntax.

Example of JSON a formatted object “Person”:

{

"**firstName**": "Jack",

"**lastName**": "Black",

"**age**": 29,

"**address**":

{

"**streetAddress**": "Ringelweg 35",

"**city**": "Esslingen",

"**state**": "GER",

"**postalCode**": "73730"

},

"**phoneNumber**":

[

{

"**type**": "home",

"**number**": "212 555-1212"

},

{

"**type**": "fax",

"**number**": "646 555-4567"

}

]

}

**Speed:**

JSON is a lightweight format, and thus faster than other formats like XML. It is written in JavaScript object notation and thus require less time to parse.

**Size:**

JSON does not add much overhead to the data when parsed, which is not only of advantage for the readability, but also size. Data size is especially critical when it comes to transferring it over the internet and in big amounts and thus JSON is a good choice for this purpose.

**[HIER EVTL NOCH MEHR VORTEILE VON JSON]**

* 1. Strengths and Risks

As mentioned in chapter 2.1 “Software Development Process”, one of the advantages of an iterative approach is the fact that progress can be frequently presented to make sure it is exactly what is required. Another strength of this approach, is the fact that system components can be prioritised. To provide a proof of concept one would usually need a simulation. Defining and building a simulator can have a variable complexity – from static values to a highly complex model of the real world system. Depending on the remaining time and priority, this and any other task can be adjusted correspondingly. Since only proven tools and frameworks are used to create this system, there is no speculation in the question, if the tools and frameworks are sufficient to fulfil this task.

There are several risks in the development process of this system. One of the biggest risks is the time to finish the whole project. The project requires a lot of research work on the subject of waste-water treatment process, as well as on the topic of data harmonization. Furthermore the creation of a corresponding relational database system, a fitting data-schema and the implementation of the harmonization process as well as the data provider is a time consuming task. Also as mentioned, some additional work has to be done to prove the concept entirely, since it will not have the access to actual waste-water treatment plants.

Another big risk is the fact that there is not much divergent example data from wastewater treatment plants. This not only makes testing of the system harder, but also leads to the possibility, that a better solution to the problem could be found if there was more data samples.

Additionally to the problems in the development phase of the project, there are also general problems. At the time of the implementation of this system, there is no possibility in accessing the waste-water treatment plants data any near real-time. There are only general reports produced by the waste-water treatment plant operators, most of the times once per year. The operators would not only have to make their data available per internet, but also provide this data in the format required by the system. This might even require the plants to upgrade their systems of control.

|  |  |  |  |
| --- | --- | --- | --- |
| Risk | Effect | Classification | Strategy |
| Time develop the system is low | The System will not be finished until the time runs out | Catastrophic | The time-plan has to be as precise as possible, also including buffer time and has to be strictly followed. |
| No way for trivial comparison of different data-formats | The harmonization requires a different strategy for every format-structure included into the system | Critical / Delays the project | Try to merge the different formats into one common format as early in the harmonization phase as possible. |
| Low amount of divergent sample data | The system cannot be sufficiently tested, some solution strategies cannot be developed without sufficient sample data. | Critical | Only solutions which lie within the possibilities, based on the boundaries of this project will be considered. |
| No access to real-time data | The system cannot be tested with actual water-treatment plants | Marginal / Delays the project | The aim of this project is only to create a proof of concept and not a working system – thus a water-plant simulator is enough to fulfil this task. |
| No access to water-plant data in general | The system, even if finished, would have no use in the current time. | Insignificant | It is not the aim of this project to create a working system, but only to provide a working proof of concept showing, that under different circumstances, this solution would work. |

Table 2-1: Risks, Effects, Classifications and Strategies

* 1. Project Management

Based on the spiral development strategy described in a previous chapter, this project will be developed one functioning component after another. Previous to that, the problem to solve has to be described in detail and separated into defined smaller problems. The following sub-chapter describes the tasks which need to be fulfilled in order to successfully end the project.

Project Tasks

|  |  |  |
| --- | --- | --- |
| Category | Task | Description |
| Analysis | Analysis of the Problem | Analyse why the problem exists, what possibilities and advantages it would bring to solve this problem and why it might be a difficulty to solve it. |
| Analysis of water-plants | Analyse how water-plants work, what the water represent, how the water-plants differ from each other and what their similarities are. |
| Analysis of data harmonization methods | Analyse what data harmonization is, how it is defined, how it is generally used and common problems. |
| Analysis of similar projects | Analyse projects having similar aims as this project. Which knowledge, techniques and methodologies of data harmonization can be used for this project? |
| Interim Report | Creation of the interim report | - |
| Planning | Defining the development model | The development process will be defined by one, or a combination of proven models. |
| Deciding on Tools and Frameworks | The Tools and Frameworks include the database, the cloud, the programming language as well as the IDE used in the project, as well as the data-format of the schema |
| Detemining Risks and Strengths | Determining risks and strengths of the project includes finding tasks which for a reason might lead to problems within the project, and find strategies to deal with them, as well as finding strengths in the planning decisions. |
| Design | Design of the System | Designing the overall system. Defining how the components should interact with each other and the outside world. |
| Design of the data-schema | Deciding on a common schema for the outsides world communication with the system. |
| Design of the database | This step includes the design of the tables and their relations. |
| Design of the harmonization service | Designing the harmonization service. This includes breaking down the service in components and defining each components output as well as defining the strategies for each |
| Design of the data provider | Designing a data provider in terms of access and data format of the output, as well as its access to the database. |
| Design of the simulator | Designing a simulator which can be used as real-time data provider as well as an endpoint to pull water-plant data from. |
| Implementation / Realisation | Setting up the data base | Setting up the database includes the creation of the tables, the views and the stored procedures, as well as making it accessible to the system. |
| Implementing the harmonization service | Implementation of the harmonization service is the most crucial task for the proof of concept. It includes the implementation of the harmonization service and integrating it with the database. |
| Implementing the data provider | Implementation of the data provider includes the integration with the database, as well as making the provider accessible by the outside world |
| Implementing the simulator | Implementation of a tool which acts like water-plant, in terms of data providing. |
| Conclusion | Research on the system behaviour | Investigating if the proof of concept was successful, defining further work which needs to be done to make the system work in the real-world. |

Table 2-2: Project tasks in categories and their descriptions

**Beschreibung der Abbildung**

Gantt Chart

Figure 2-: Timeplan / Gantt chart

1. Literature Review

Designing the software and creating a fitting data-schema, as well as making decisions about the project infrastructure requires a deeper understanding of the subject. For this purpose it should be examined in detail with all its components.

The first survey will examine water-plants related topics. It will explain how a water-plant work and what the most important steps of the water cleaning process are. Additionally it will investigate on the similarities and differences between single water-plants in terms of water-based data – which data is stored, how it is stored, which data can be accessed, how and by whom.

The second survey will investigate on the topic data harmonization. It will explore the common methods of harmonization and the theory behind them.

The third survey will investigate on similar projects and industries, which also harmonized data coming from different sources for a common use. The advantages and disadvantages of each approach will be taken into consideration to see, which approaches – or parts of approaches, are fitting for this kind of a project.

HIER ÜBER DIE WEITEREN SURVEYS SCHREIBEN WENN MENR ALS 3.1 3.2 3.3 BESCHLOSSEN WURDE!!! :D:D:D: ASDF WTF LOL OMG

* 1. Water-Plants

**[A]**

Water-plants (wastewater-plants or sewage-plants) are used to clean water – primary household, -/ but also industries and businesses sewage – for further use. This is accomplished by speeding up the natural process by which water is purified. Today it is mostly done in three steps which will be explained later.

Water was privatised in the UK in 1989, since then the quality of water and its availability increased, but the administration of each water-plant differed more than before.

Functionality

The picture below shows a life-cycle of water: 

Figure 3-8: Waste-water treatment plant stages

(Source: https://www.water.org.uk/about-water-uk/wastewater)

**[B]**

As mentioned before, there are several steps within which the water (already filtered of grit and large solids) gets treated during the cleaning process. Every step is controlled and monitored. Those steps[[8]](#footnote-8) are:

1. **Screening**

During the screening large objects such as rags or boards are dismissed, to prevent them from damaging the instruments of the further treatment

1. **Primary Treating**

Physical and chemical settlement of suspended solid waste which didn’t get removed before, as well as reduction of its biochemical oxygen demand.

This step should reduce:

* Biochemical oxygen demand by 20%-30%
* Suspended solids by 60%

1. **Secondary Treating**

This step involves biological treatment to break down and reduce residual organic matter. This treatment step must comply with the standards of the Directive.

1. **Tertiary Treating (Final Treatment)**

This treatment step depends upon the location. It can involve disinfection by violet light, nutrient removal or the removal of specific toxic substances.

Monitoring equipment and process

**[H]**

In the following chapter, the current state of the art in terms of wastewater treatment monitoring will be described.

The four major things when it comes to wastewater treatment monitoring are:

* Insight into the process
* Sensors (data providers)
* Fitting strategy
* Actors which implement the controller output

Even though the four mentioned blocks evolved over the past years, the claim on water quality has evolved too and with that, a more advanced treatment for the water is required. Especially the standards and regulations for the level of organic carbon, nitrogen and phosphorus kept getting stricter. For a lot of outdated waste-water treatment plants this fact meant that they were forced to implement some new monitoring technologies, as the only alternative would be increasing the reactor volume – which is not only less flexible, but also more costly.

* + - 1. Sensor classification

**Functional application**

Most often, sensors are used only for monitoring purposes. In this case, the reason behind the monitoring is significant. It might be to provide information to the operators of the water-plant controlling system, or to consultants, whose aim it is to optimize the process.

**Complexity**

Depending on their purpose, sensory can either be simple, reliable and easy to maintain or advanced and maintenance intensive.

* + - 1. Commonly monitored data

The following sub-chapter will provide a list of water-properties, which are commonly monitored, as well as some description on why the data is needed, who it is interesting for and/or how it is measured, for the respective property.

This list will be used to design the common schema in a later chapter.

The properties are separated into categories which indicate the step of the treatment process they are measured in:

**General:**

This section describes properties, which are not specific to a certain treatment step, but instead are measured on different stations, as they are of high importance overall.

**Temperature**

Measured with a thermistor. Important for anaerobic digesters. Not of importance for the public.

**Pressure**

Measured for alarm functions in aeration and anaerobic digesters. Not of importance for the general public.

**Liquid level**

Measured with internal electric switch, conductivity switches, pressure transducers, capacitance measurements and ultrasonic level detection. This data is not only interesting for the waste-water treatment plant system operators, for high-level alarms and emergency shut-offs, as well as low-level alarms and leak detections, but might also be interesting for the public to view the storage inventory.

**Flow of liquid/gas**

Based on the change in water level as a result of an obstacle in the flow path of the water. The efficiency of a wastewater treatment plant is a function which includes the flow rate of the water entering the water-plant.

**pH**

pH is a value important in all biological processes, but is especially valuable in anaerobic digestion and nitrification. It is measured by pH electrodes. Despite its importance in the control of biological processes, it is not recommended for process supervision and control of waste-water plants as it might be rather insensitive to indicate process changes.

**Conductivity**

Conductivity sensors are used to monitor influent composition changes. They also help with the control of chemical phosphorus removal.

**Biomass/suspended solids**

This might be the most important value in the waste water treatment process. It measures the suspended solids concentration (SS). The most common measuring techniques are optical, ultrasound and dielectric spectrometry. **MEHR SCHREIBEN**

**Anaerobic digestion**

This process is mineralizing organic material into gaseous products such as H2, CH4, CO2 and H2S. The measurement of intermediates and the final gaseous product properties is of high value, as it helps with the control of the entire process.

**Gaseous products**

This measurement gives insight about the gas composition.

**Calorimetry**

This measurement monitors the heat production and provides direct insight into the biological process as all biological activities are characterized by the production of heat.

**Volatile fatty acids (VFA)**

The VFA is the most important intermediate in the anaerobic digestion process. Not only does it help with the process control, as values out of range may lead to a process failure, the VFA concentrations also act as performance indicators. Due to the importance of this value and the lack in different implementations, the sensors in this area were developed a lot in the close past.

**Activated Sludge**

**Dissolved oxygen (DO)**

The sensors measuring the oxygen are most likely the most common sensors in the wastewater treatment plants, as it plays a key role during the activated sludge processes. The aeration, which takes place during this process can take as much as 40% of the overall waste-water plant running costs, and its optimization has therefor a very direct influence on the turnover.

**Respirometry**

Indicates the respiration rate of activated sludge, defined as the amount of oxygen per unit of volume and time that is consumed by the microorganisms within the activated sludge. It is of importance for the characterization of wastewater and the activated sludge kinetics.

**Biological oxygen demand (BOD)**

This is a measure that indicates the amount of dissolved oxygen required for the biochemical oxidation of the organic solutes within 5 days from the seeding of the test sample in a microbial system. This is not a typical real-time value, as it indicates the quality of wastewater and sludge from 5 days ago.

There are further measurement techniques, but they all require time and the less time they require the less representative their outcomes are.

**Chemical oxygen demand (COD)**

Another highly monitored variable within a water-plant. It determines a plants efficiency in terms of carbon removal.

**Total organic carbon (TOC)**

This measurement converts organic carbon into CO2 and measures this product in the evolving gas phase.

Difficulties

The problematic in the context of the system under design, with the focus on its usefulness within the real world, lies mainly in the data acquisition. First of all, the data in the UK is not only not available to the public, but also not available to any authorities as online provided real time data.

Sensors which analyze the water and provide information are not 100% reliable. They may provide wrong data, data-holes or provide data in an inconsistent frequency. Those sensors may differ from water-plant to water-plant and be placed on different places within a plant, which makes comparison between different water-plants difficult. The sensors might not provide all the data which is needed to do a representative comparison between different water-plants. Also the data formats may not only differ in form but also in type. While the one might use an XML schema, the other one might use JSON or even worse: a table based format like CSV or XLS/XLSX. While XML and JSON differ in their syntax, JSON and CSV or XLS/XLSX differ in their structure. A comparison between a tree-based data-structure like JSON and table-based data-structure like any format of Excel tables is not trivial and needs to be defined based on the task. This may lead to a development of two or more different harmonization strategies depending on the source data-format.

Treatment steps may differ from water-plant to water-plant, meaning that even if a sensor is placed on the same position of a primary treatment in two water-plants, the data may still differ a lot.

There is not much real data to develop the system. Real, water-plant-created data, is very useful when designing a harmonizing system, because it gives an idea about the different formats and possible deviations within the schemas. Additionally, having multiple sources gives statistical insight on which data is ‘usually’ available/tracked, and which is rather rare.

* 1. Data Harmonization

**[b] [D]**

Data harmonization’s aim is to create a single source of information based on multiple sources. The general problem during a harmonization process is, that the different base-sources differ the form in which their information is provided. To present a clean, harmonized information source, the data needs to be cleared of inaccurate and misleading entries. This means, that harmonization alone might not be enough for every process. To make sure the created information set is useful, it needs to undergo some additional processing.

Why Data Harmonization?

As already mentioned, harmonization is needed when dealing with different sources of information, but trying to get a common knowledge base. Sometimes the same type of information is stored in different types of data-formats. An example for this is the DateTime Format.

The DateTime might be stored in formats like:

* DD/MM/YY
* MM/DD/YY
* DD-MM-YYYY
* YYMMDD
* DD MONTH YYYY
* …

All the formats store the exact same information, but to extract this information further processing is needed. In some cases it might be enough to look at the data to find out which of the formats it is using, but in other cases (DD/MM/YY and MM/DD/YY) it is mandatory to know the format before extracting the needed information.

Single Window Harmonization

**[E]**

The **s**ingle **w**indow **h**armonization (SWH) is was defined by the United Nations Economic Commission for Europe (UNECE) as:

*“a system that allows all participants in trade and transport file requested information in only one place, in a standard format, in order to carry out import, export and transit operations.”[E]*

It was designed to simplify foreign trade operations, where control agencies require around 40 documents all together, with often repeated data. It includes:

* Simplifying or cutting procedures or documents
* Aligning them with national standards
* Automating documents and processes

The main idea of the SWH is to store information at a Single Authority (Single Window) with traders and control agencies located at different places, but connected information flow.

Even though the basic problem to be solved by the SWH is a different one, since it is coming from a different field, the solution of having a single authority gathering and providing the data to interested stakeholders is also of interest for this project, since it is basically the task. The below picture shows the basic idea:



Figure 3-9: Single window system example – big picture

[E Page 4]

[E]

Knowledge gathered from UNECEs attempt to create a single window harmonization solution includes following points:

High-level support by strong lead organisations is needed. Those organisations are too important for the system as a whole to not be included and thus have a deciding role in its success. It is important to have a clear vision of the single window system form the beginning, not only to plan the system, but also, since the system is about connecting different sources, to be able to describe the advantages and methodologies in detail and thus get them interested in joining. 90% are negotiations and 10% are technical work. Companies function very differently in terms of their processes and techniques and most of the time every company will want to have its own process as the standard. Agile development of the system is of advantage, as the involved stakeholders want to see progress and avoid misunderstandings. The techniques of this UNECEs approach will be discussed in the chapter 3.3.

Data Simplification

This step eliminates unnecessary and redundant data. It is usually happening before the harmonization takes place. This has the advantage, that in high amounts of data, not all the data needs to undergo further processing. The disadvantage to have it as first step is, that some data might be removed that is actually needed, but not recognized as such and would be only after the harmonization step. A good approach is to have a simplification take place once before and once after the harmonization took place. The first simplification in this approach aims at clearly unneeded data and leave out data classified as ‘unsure’ to handle it in the second iteration after the harmonization is done.

Data sources are likely to have additional data, which is only relevant for the specific source itself and not for inter-source-analyses. This data needs to be identified and ‘cleaned’ from the source dataset.

Example with two different sources A and B:

Target Schema:

|  |
| --- |
| DataA |
| DataB |
| DataC |

Source A:

|  |  |  |
| --- | --- | --- |
| DataA | Simplification | DataA |
| DataB | DataB |
| DataD | ~~DataD~~ |

Source B:

|  |  |  |
| --- | --- | --- |
| DataA | Simplification | DataA |
| DataB | DataB |

Simplification needs to know the target schema to function. It doesn’t add values which are missing when comparing source to target. It only removes the unnecessary.

Data Standardization

Data standardization is about processing of the dataset into a standard form through standard bodies. It is not said that a harmonized dataset matches the standard forms outside of the system. It might be harmonized but still useless to the outside world. This is why a standardization step might be needed to do further processing of the data, change its form or format, before publishing it. An example for standardization is converting the velocity from one of its many units like meters/s, km/h, knots… or temperature from Celsius, Fahrenheit or Kelvin into the one which is standard in the country/field of interest, since they all contain the same information in a different form.

* 1. Comparable Industries and Projects

To find the most fitting approach for the system, it is needed to take a look on projects and industries which already successfully developed such a system, or projects where such a system failed, to see what caused its success or failure, as well as understanding why specific approaches were more successful than others.

UNECE Project

As already mentioned in a previous chapter, in the year 2008 the UNECE attempted to build a harmonized single window system for international trading.

**[F]**

This project lead the UNECE, in cooperation with UNNEXT (United Nations Network of Experts for Paperless Trade in Asia and the Pacific) and the United Nations ESCAP (Economic and Social Commission for Asia and the Pacific) to create a report on how the data harmonization should be optimally approached. Since the UNECE project had a different context, only the parts of the report significant for this project will be investigated upon, as well as only the technique of harmonization and not the project planning.

One difference to this project is, that this project should also consider real time data, while the UNECE project defined a schema which is only complete with all elements filled out. This should not mean any problem, as there is the option to simply ignore missing data if it was not provided during the time period in which the water-plant data was created. The timestamp of each data entry in most cases will have no significance in the UNECE project, while a data entry without a timestamp is basically useless in this system.

The UML provides sufficient options to describe a Data Model as a Class Diagram with Property Terms[[9]](#footnote-9) and Object Classes[[10]](#footnote-10).

The following describes the five main steps taken to develop a data harmonization process, transformed into steps in the scope of this project.

**Step 1:**

Everything starts with the capture data requirements. This step is about collecting information about which data is interesting, as well as how this data is produced. This includes getting background and understanding of the information source work-flow. Understanding the work-flow helps in understanding the stakeholders needs, as well as the significance of specific data members.

**Step 2:**

Providing a detailed definition of the data elements within single information sources. This step is about setting the data definition, type, format and constraints of each information type for each information source. The outcome of this step should be a data dictionary corresponding to a specific source of information.

**Step 3 & 4:**

Analysing data elements across various information sources. This includes the organisation of data elements in a comparable manner so that it can be used for analyses. The desired outcome of this step is a data dictionary compilation as well as mapping to the desired data model.

**Step 5:**

This step is about the creation of reports and not in scope of this project.

FCTC Project

**[G]**

In the year 2008, the Conference of the Parties to the WHO (**W**orld **H**ealth **O**rganization) requested **F**ramework **C**onvention on **T**obacco **C**ontrol (FCTC) to compile a report on data collection measures. Within two expert meetings (2009 and 2010) the draft outlines were defined, for the further development of the report. It was supposed to be based on the most relevant international literature and other tobacco-related information sources. A review of existing data sources and data collection systems as well as an investigation on a possible data harmonization process was also a part of this meeting. The report created during this meeting gives an insight on possible problems which may come up during the creation of the system, as well as criteria to consider when deciding for a harmonization technique. It is especially interesting, as the tobacco control is probably one of the most researched areas in public health.

The international data collection systems of tobacco-related information were distinguished in two types: **population-based surveys** (primary data collection systems) and **policy monitoring surveys/systems** (secondary data collection systems).

**Population-based surveys**

The data collection systems, which were in use at the time of the creation of the FCTC report, varied in every thinkable aspect:

* Target Group (young / adult)
* Scale (global / regional)
* Time (point in time / repeated periodically)
* (Money-) Budget (cheap / expensive)
* Organizer
* Specificness (tobacco specific / number of health determinants including tobacco)

Additionally, many countries have they own health survey systems, which have no direct relation to the international system despite their information being similar.

The reporting system of the Convention was established by the conference of the Parties in 2006 and enforces every participating party to report on its implementation of the convention for the first two years after being included in the convention, as well as after a specific time in the future. The aim of the system was to collect data already available at the time of reporting, instead of having to implement the new population based survey system first. Only the parties without any population-based system were required to implement it. The convention secretariat provided **feedback on the assigned reports**. This feedback included missing mandatory information, not suitable formats within the report and inconsistencies within the answers on the report and the supporting documentations. This data is then provided through a web based database maintained by the convention secretariat for further reports.

Another source of information on tobacco is the impressive number of databases which provide tobacco-related information.

**Standardization and harmonization**

Even though the majority of surveys follow standard methodologies and use standardized questions and patterns, the form still differs from survey to survey and therefore requires at the very least some form of harmonization. Additionally, as already mentioned, the majority of surveys are non-tobacco specific ones and thus need to be “cleaned” of unnecessary (not tobacco related) data. Additionally, the surveys may differ based on the age, since the questions might be directed more in a specific direction for a specific target group. It has to be made sure, that data which has the same identifier across multiple sources has also **the same definition** across those sources. Comparing seemingly the same datatype, even though it means something completely different within the scope of different systems will cause inconsistencies and false conclusions. Another important knowledge won in this project is, that there is a big amount of **redundant, repeated and overlapping data** across the different surveys and approaches.

In the case of population-based surveys it is important to have a common information schema, methodology and data definition, to create comparable data sources. In the case of monitoring systems, harmonization of the existing data is the main challenge. This knowledge transferred to this project means that especially in the case of historical data and data provided by the existing systems, the main challenge is to find a way to harmonize it, while it is desirable to have every participant use a common, efficient and suiting data-schema. Accomplishing this desire is a hard challenge.

Standardization of data not only refers to processing of the acquired data, but also to the **data acquisition / creation** (such as sampling techniques), in order to create more meaningful data. This may require additional training (i.e. workshops) for the data collectors. Step-by-step instructions in how to implement or use the new system might be needed in order to promote it.

Even if some parties may not be able, or may not want to fully adapt the common approach, they can still undertake steps to help the system to gather their data in a better way, like adding additional data or adjusting some data definitions. Additionally, experts have warned from creating a global standardized approach in terms of survey content, as this would take away the variation between regions, where specific information might be of high value in some regions and help the parties to fulfil their reporting obligations, while it is of no use in others.

**Differences**

There is a lot of knowledge which can be won from the FCTC project, but some of it can’t be fully or not even partially applied to this project because the two projects differ too much in their aims and objectives. This project needs a way to deal with real-time data. It also needs a way of adapting data into the own schema, as historical data placed on some storage will most likely not be pre-processed by the provider.

1. Design and Implementation

The key parts of the system to design are the data-schema, the database / storage and the processing / harmonization software. All the mentioned parts can be designed (more or less) independently and implemented independently, if the interfaces are clearly defined.

Due to the fact, that there is no access to real-time data, and only limited sources of historical data, a data simulator would be of advance.

* 1. System overview

The below picture shows a white box diagram of the system, for a better understanding of its scope and relations:

Single Window System

Harmonization System

Data Provider

Water-Plant

#2

Water-Plant

#1

System #1

Data Storage

System #2

Unified Schema

Modified Schema

Water-Plant

Simulator

Historical Data

Simulated Data

Diagram Beschreibung

Figure 4-10: Whitebox diagram – harmonization system

The picture shows five data sources which can be split into two categories. Those two types will be referred to as **real-time data** (green and blue) and **report data** (yellow).

It is assumed, that real-time data is being pulled by the system frequently from the same (registered) source. If there is no new data, the system waits a defined time and then pulls the data again.

On the other hand, the report data is not pulled by the system, but rather provided through a dedicated API.

The unified schema is known to the harmonization system, and in best case the data from real-time sources and reports is coming in that format. It can happen, that the system receives data which is not in the unified schema, but a modified one. A modified schema will be defined as a schema that differs in any way from the unified schema. The harmonization system needs a way to deal with this kind of schemas. The report data will usually include a bigger amount of data in one session.

Real-time data will usually be of higher priority to process by the system, as the real-time character will fade with time passing.

Data processed by the harmonization system will be stored on a data-storage. This storage will not be public, but can still be accessed for reading by chosen 3rd party systems. It is possible, that there will be a data access service placed between the system and the data storage as a later step, to have a better management over the access rights, load balancing and most important of all – an predefined API.

Figure 4-11: Blackbox diagram – harmonization system and its interfaces

Water-plant data pulling API

Report Submission API Endpoint

Data Access Endpoint

Single Window System

Putting all the facts together, the picture shows the system as a black-box diagram with all the mandatory interfaces to the outside world.

The endpoints provided by the system will be described in detail in a later chapter. There is also a need for the Water-Plants to provide an endpoint for the system to access the data. This endpoint will also be described in a later chapter.

* 1. Data-Schema

The final schema can be seen in the **Appendix (A).**

The development of a common schema is one of the most important parts of this dissertation. It has to take into consideration, that the data-sources may vary a lot in their format, type of data and frequency, but also be specific enough to provide meaningful data for further analyses. Some values needed to determine a key-factor may not be available on a data source, or only as an interpolation. Some values may not be as current as other values, or update equally frequent. An important factor for the data-schema is the fact, that it has to be extendable, as well as still remain consistent and valid if some of the elements are deleted.

Data Categories

At first categories of data need to be established to ensure a better understanding and overview over their source, meaning and purpose. In this project those categories will be:

* **Wastewater treatment-plant metadata**
* **Treatment step metadata**
* **Water quality data**
  + - 1. Description

In the following those categories will be described in more detail:

**Wastewater treatment-plant metadata** (WWTPM)

This category consists of data specific for the corresponding water-plant. It includes data which helps to not only identify the source of data, but also boundaries which describe the location and further static[[11]](#footnote-11) metadata.

**Treatment step-type metadata** (TSM)

Measurements are done at various places of the waste-water treatment process. The value of a water quality indicator will vary depending on the place it is measured and thus will be of no value without a specification of the sensor position.

**Water quality indicator data** (WQID)

This category will consist of data described in a previous chapter (chapter 3.1.2.2). An entry not only requires a value, but also the timestamp and unit to be of full use for further analyses.

* + - 1. Relations

The following diagram shows the relational hierarchy of the three categories:

WQID \_1

WQID \_2

WQID \_3

**TSM\_1**

**TSM\_2**

WQID\_4

WQID\_5

**WWTPM**

WQID \_X

WQID \_Y

Figure 4-12: Data-schema relations and hierarchy

One water-plant may contain *n* TSMs, although at the moment the maximum number will be set to 7 as this is the number of defined treatment steptypes. Each step may contain *m* WQIDs, although the maximum number in this project will be set to 16 as this is the number of defined indicators (described in the next chapter).

Additionally to this hierarchy, there might also be some loose variable data, like amount of water within water containers. This data will be assigned directly to the water-plant without any direct relation to a specific treatment step.

Data-Schema content

The following data will be included in each of the categories. Note that the property names will be actually written as camel case to follow the defined JSON standards and conventions:

* + - 1. WWTPM

Description of how a WWTPM entity looks:

|  |  |  |
| --- | --- | --- |
| Property | Description | Required? |
| Name | This property represents the name of the waste-water treatment plant. The name has to be mandatory as it acts as the primary key, which has to track back a specific plant. | ✔ |
| Location | This property indicates the geographical location of the water plant. The location is of interest in terms boundary information like weather. Location also includes the actual country, as UK consists of 4 (England, Wales, Northern Ireland, Scotland), which might have some water regulations for their own. |  |
| Supplier | This property refers to the water-supplier owning or controlling the corresponding water-plant. Even though this property is not required, it is of high value. |  |

Table 4-3: Waste-water treatment plant in the context of this system

* + - 1. TSM

Description of how a TSM entity looks:

|  |  |  |
| --- | --- | --- |
| Property | Description | Required? |
| Name | This property represents the name of the treatment step. Treatment step might be one of the following values:   * Entry * Screening * Primary Treatment * Secondary Treatment * Tertiary Treatment * Sludge Treatment * Exit | ✔ |

Table 4-4: Treatment step metadata in the context of this system

* + - 1. WQID

**Properties**

Description of how a WQID entity looks:

|  |  |  |
| --- | --- | --- |
| Property | Description | Required? |
| Name | Name of the water quality indicator  (possible indicators will be described later) | ✔ |
| Timestamp | Timestamp of when the indicator was measured. It should include the Date and the time. | ✔ |
| Unit | The unit of the indicator. | (✔)[[12]](#footnote-12) |
| Value | Value of the indicator | ✔ |

Table 4-5: Water quality indicator in the context of this system

**Possible Entries**

The following table shows an overview of possible Water Quality Indicators, their units and known aliases. Known aliases are helpful when it comes to harmonization, since names can be mapped without having to make speculations or unnecessary calculations.

|  |  |  |
| --- | --- | --- |
| Name | Unit | Known Aliases |
| Temperature | C° (Celsius) | Temp |
| Pressure | Pa (Pascal) |  |
| Liquid level |  |  |
| Flow Rate (liquid / gas) |  | Flow; Liquid Flow; Gas Flow |
| pH | [no unit] |  |
| Conductivity | S/m |  |
| Biomass | mg/L | Suspended solids; Total Suspended Solids; TSS |
| Nitrate | mg/L | NO3 |
| Nitrogen Dioxide | mg/L | NO2 |
| Ammonium | mg/L | NH4 |
| Reduction Potential | mV | ORP |
| Calorimetry | C° |  |
| Volatile fatty acids | mg/L | VFA |
| Dissolved oxygen | mg/L | DO |
| Biological oxygen demand | mg/L | BOD |
| Total organic carbon | mg/L | TOC |

Table 4-6: Basic quality indicator types

Even though defined at this point, this list isn’t final or will ever be hard coded into the software. It has to have an extension possibility as measurements are developing as well. To keep this as generic as possible, it will require the data provider to also return a list of available indicators for a specific plant. This topic will be further described in a later chapter.

* 1. Data Base

The database will act as the platform for many comparisons – to recognize patterns, map names and find mistakes and therefore needs to be designed to be performant. Additionally, requests from the outside world will be performing selects on the same database, which makes its scalability even more important. Note that even though the names of tables will partially equal the names of the data-schema properties, they might not represent the exactly same thing and will thus be defined in the following chapter.

Data Base Schema

The following picture presents the database schema including the names of the tables, properties, unique primary keys and foreign keys:

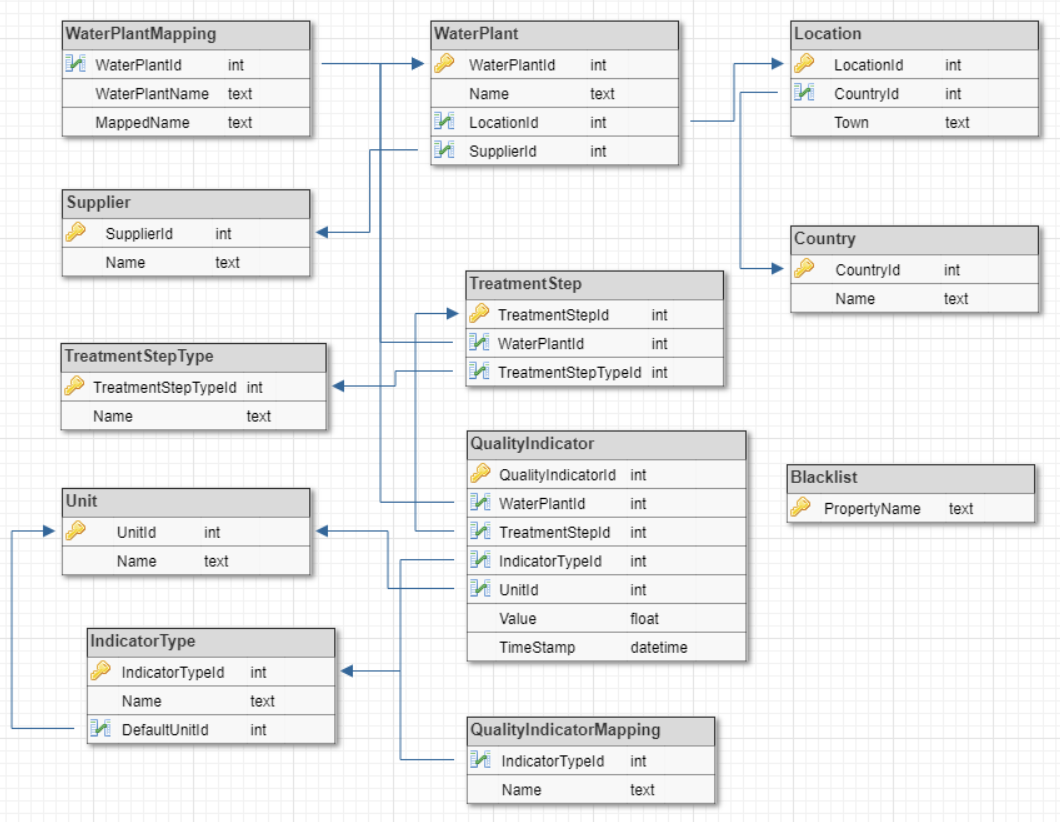


Figure 4-: Database schema

There are no m:n relations to assure better data integrity and subtypes are extracted to make selects as performant and extensible as possible.

Following is a short description of the tables in the database, not mentioning the types and their properties:

|  |  |  |  |
| --- | --- | --- | --- |
| Table | Description | | Dependencies |
| WaterPlant | | The WaterPlant table represents the physical water-plants. It includes the unique names of the various water-plants. | * Location * Supplier |
| WaterPlantMapping | | This table contains names which are confirmed to be mapped from the specific WaterPlant world to this system, but only apply to a specific plant. | * WaterPlantId |
| Location | | The location describes a graphical location. | * Country |
| Country | | This table represents a country. |  |
| Supplier | | Supplier represents a water supplier. |  |
| TreatmentStep | | TreatmentStep represents the water-treatment step within the treatment process. | * TreatmentStepType * WaterPlant |
| TreatmentStepType | | TreatmentStepType contains all the unique names of possible treatment steps. |  |
| QualityIndicator | | QualityIndicator refers to one of the values which describe the quality of water. A quality indicator can refer **either** a WaterPlant or a TreatmentStep. QualityIndicators assigned to WaterPlants are general indicators. | * Unit * IndicatorType * WaterPlant * TreatmentStep |
| Unit | | Unit refers to the unit directly corresponding one or more qualityIndicator types |  |
| IndicatorType | | The IndicatorType Table contains all possible qualityIndicator types which can be stored in the system. It also contains the default unit, which is set if no other is specified. | * QualityIndicatorMapping * Unit |
| QualityIndicatorMapping | | The QualityIndicatorMapping table consists of known aliases for QualityIndicator names, which are treaten equally. |  |
| BlackList | | Names of properties found on this map will be ignored / cleaned from the data during the harmonization process |  |

Table 4-7: Database schema description: Tables, description and dependencies

* 1. Harmonization Service

The core service of this system is the harmonization service. Its purpose is not only to process the data into a common schema, but also receive incoming data and pull data from defined sources, as well as storing the data in the corresponding tables in the database. The data is processed step by step. The earlier in the process different formats[[13]](#footnote-13) get combined together into a common format, the less work has to be done when extending the system by new formats. Investigations have shown, that the formats are too different to combine them before the actual harmonization step, and need separate treatment in the previous steps. The harmonization strategies of each format will be described in a later chapter.

Use Case Diagram

The use case diagram displays the functionality provided by the system, as well as the actors interacting with it:

**Waste Water**

**Treatment Plant**

**Historical Data**

**Owner**

**3rd Party System**

**System**

**User**

<<includes>>

<<extends>>

Figure 4-14: Use case diagram

Waste water treatment plants added to the system can provide data for harmonization. Anyone owning historical data in a certain format can enter it into the system. The stored data can be requested and read by other systems and system users.

Component Architecture

The following diagram displays an overview over the components of the harmonization service, and their relations.

DocumentListener

WaterPlantFetcher

PreSimplificator

Harmonizator

Simplificator

Standardizator

DataBaseAccessor

DataBase

Converter

Figure 4-15: Component diagram of the harmonization service

The service is separated into 7 components, which are described in the following:

* + - 1. WaterPlantFetcher

The *WaterPlantFetcher* component is responsible for pulling data from defined sources (Water-Plants). The data has to be under the configured path and preferably in the defined schema[[14]](#footnote-14). The sources are stored in a separate table with the following format:

|  |  |  |
| --- | --- | --- |
| Column Name | Type | Nullable |
| PullSourceId | Int |  |
| SourcePath | String(100) |  |
| WaterPlantId | Int | Yes |
| TreatmentStepTypeId | Int | Yes |
| QualityIndicatorTypeId | Int | Yes |
| DataType | String(10) |  |

Table 4-: PullSources table definition

SourcePath defines the path from which the data is pulled. It expects a directory with any number of files meant to be harmonized. DataType gives information about which of the allowed datatypes is pulled. If the data is stored as JSON WaterPlantId, TreatmentStepTypeId and QualityIndicatorTypeId can be set to null (provided, that they are defined in the JSON file). Files which got processed do not get deleted, but renamed to “\_filename”. This requires the client to have writing permissions on the directory, otherwise the same file might get processed over and over again. The fetched data is not further processed in this step, but passed over to the harmonization process.

* + - 1. DocumentListener

The *DocumentListener*, on the contrary to the *WaterPlantFetcher*, does not actively pull data from specific sources, but waits until data is provided by the outside world. For this matter, it provides an Endpoint which accepts documents. It does not further process the data itself. Data provided through this endpoint requires more information, as the sender is “unknown”[[15]](#footnote-15) to the system. The Rest API is described in the Appendix C.

* + - 1. Converter

The *Converter* component is used by both, the *WaterPlantFetcher* and the *DocumentListener*. Its purpose is to accept data and try to parse it into an object, which can then be further processed. If the *Converter* fails to parse the data, its processing ends at this step and the data is dismissed because this means it is not harmonizable by the system by any means.

* + - 1. PreSimplificator

The purpose of the *PreSimplificator* component is to minimize the data which will be further processed by removing the parts of it, which can already at this point be recognized and mapped to parameters in the blacklist. This step does not dismiss any further data or decide if the data will be further processed. Once its task is done, the data gets passed to the *Harmonizator* component. The *PreSimplificator* subscribes to the two system entry points and gets notified when new data is ready to start a harmonization process.

* + - 1. Harmonizator

The *Harmonizator* component is responsible for the actual harmonization of the data. It searches the object for familiar data-formats and structures, properties, dismisses incomplete data (with missing mandatory properties). Before doing so an attempt is made to “fix” the data. This attempt varies depending on the type of data. After finishing the harmonization process, the *Harmonizer* checks if the final data-object is complete (no missing mandatory properties), and if so, the object is passed on to the *Simplificator*. As already mentioned, accepted data-formats are too diverse to handle them the same way (table-based formats / tree-based formats…). This step is the first step in the chain which has the same output format for every incoming format.

* + - 1. Simplificator

The harmonized data is further reduced in this step by comparing the harmonized object to the blacklist.

* + - 1. Standardizator

The *Standardizator* represents the last step of the harmonization process and is responsible for adjusting the data to the conventions set in the system. The formats and structures are finally adjusted in this step before the *Standardizator* pushes the object into the database.

* + - 1. DataBaseAccessor

A general component used by many services. Its purpose is to act as an abstraction layer between the components and the database. (Not an exclusive part of the harmonization service). This component uses the entity framework to access views and stored procedures. It does not access database tables directly.

Accepted DataFormats

The two described entry points to the system – the *WaterPlantFetcher* and the *DocumentListener* – both expect a valid file in one of the following commonly used formats:

* JSON
* CSV
* XSD

It is important to mention that JSON is a tree-based format, while CSV and XSD both represent table-based formats. This fact requires a different processing strategy for both of the base formats up until the *Harmonizator* component.

Harmonization Workflow

The detailed functionality and workflow of each component depends upon the format of provided data. While data coming in in the predefined JSON format does not need any mapping or further processing, XLS/XLSX and CSV data, as well as different kinds of JSON files require conversion.

The following chapters describe methods which help to process different kinds of formats.

* + - 1. CSV / XLS/XLSX Parsing

CSV (Comma Separated Values) and XLS/XLSX (Excel Spreadsheet) are both commonly used file-formats for table-formatted data storage. To create a parsing method which is as generic as possible, yet still usable it is needed to investigate on example data coming from water-plants, as well as define what criteria needs to be fulfilled to make a file usable.

**Common Table Structure:**

A table needs to at least have timestamps, indicator values and indicator names to be of use. If no unit is detected or found, the default unit is used (data is then validated by comparing to the average values and standard deviations of the same datatype). If there are more value columns detected than indicator names, the data is dismissed, as this provides risk of data misinterpretation by assigning the wrong indicator name to the wrong row. Boundaries like step or water-plant metadata will be required as URI or configuration parameters in the first step, for this base-format.

**Harmonization Strategy workflow:**

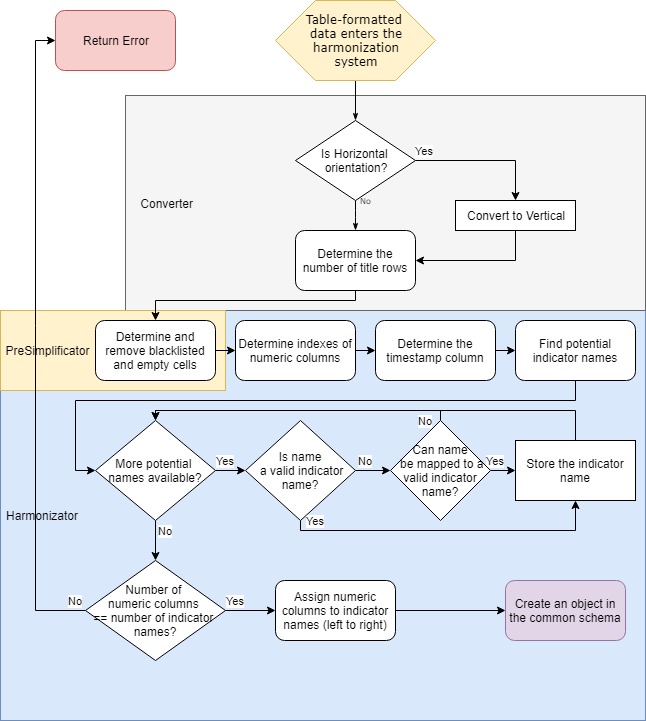


Figure 4-16: Workflow diagram – table formatted data

The last step of the workflow is also the last step of the *harmonizator*. It creates a valid object in the common water-plant schema. Further processing is equal tree and table based data and consists of cleaning and storing the data.

**Problematics:**

* Finding the name of the indicator after identifying a value column (title columns might not always have the same cell size per indicator and have differently placed names)
* Finding redundant columns (indicator can be provided in multiple units)
* Identifying the correct timestamp format (might not always be possible – best effort principle)

**Example of a Table:**



Figure 4-17: Example table

**Process of entire harmonization on example table:**

1. Table is converted into an object containing *title rows* (containing everything of colour) and *payload columns* (everything else)*.* (*Converter*)
2. Object is cleaned of payload columns which cannot be interpreted by the system (i.e. columns only containing Z and M) (*PreSimplificator*)
3. The data is matched into the predefined JSON schema object by trying to map payload values with their indicator names and units in the title. (*Harmonizer*)
4. The now harmonized data is cleaned of properties which are not tracked by the system (i.e. H2O) (*Simplificator*)
5. Adjusting formats / units and storying in the database (*Standardizer*)
   * + 1. JSON Parsing

Even though in best case the incoming data is already in the predefined schema, a usual case is that the data is indeed in a JSON format, but the structure differs from the predefined one. Forthis matter, the previously mentioned *dynamic* data-type will be of high importance. One of the biggest strengths of a tree-based format like JSON is its expandability. There is no limit to the children nodes and nesting levels. This requires for a recursive solution. Additionally, the datatypes of the properties can reach from simple types like strings or numeri’s to complex types like further JSON objects. The core of the harmonization process will be the cast of a JSON object to a dictionary of string and dynamic pairs, where the string represents the property name and dynamic the actual object. Mapping of properties is of high importance, to interpret the object correctly.

**Harmonization Strategy Workflow:**

The following flow-chart shows the harmonization strategy in the case of a JSON formatted object entering the system. The current data type and conversions will be mentioned in the chart, as the dynamic type requires the developer to keep track of the type at all times.

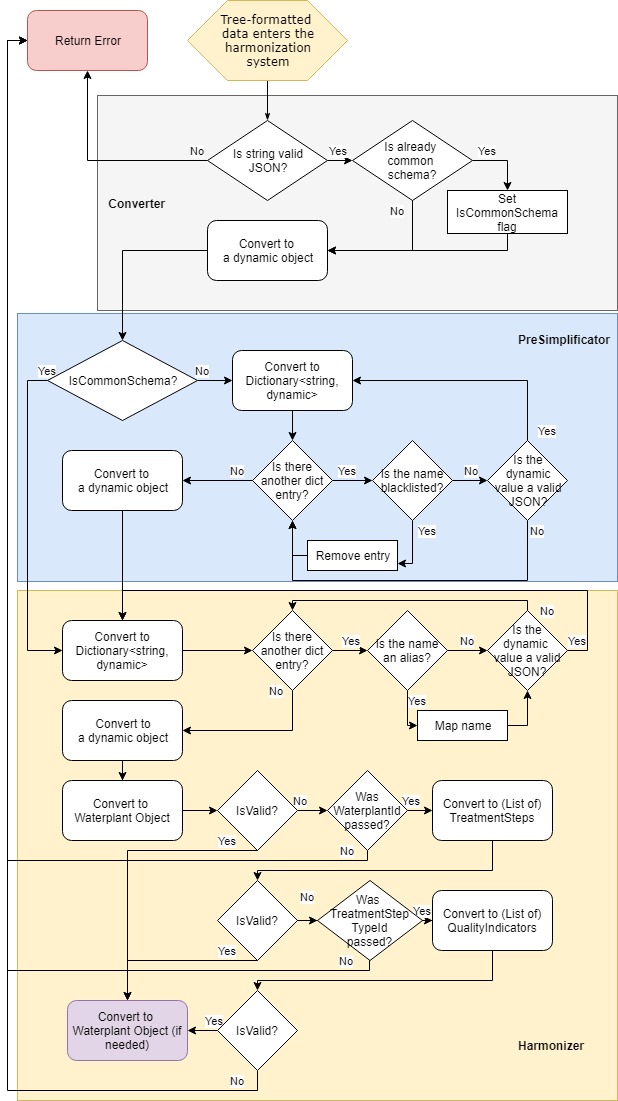
****

Figure 4-18: Workflow diagram – tree formatted data

The workflow of tree-based harmonization is more complex than the one of the table-based strategy. It requires two recursive sub-workflows (which are hard to display on a workflow chart). Additionally the number of passed URI parameters is of high importance for the casting options. If the user i.e. has passed the water plant id, it is an option to convert the payload to only the treatment steps, and retrieve the water plant metadata from the database based on the id. This of course requires the id to be valid.

* 1. Data Provider

The purpose of the data provider is to act as an abstraction layer between the data storage of the system and the outside world. The data provider accepts data requests through a HTTP listener and provides the requested data.

Component Architecture

RequestListener

DataBaseAccessor

DataBase

BusinessLogic

Figure 4-19: Component diagram of the data provider

* + - 1. RequestListener

The *RequestListener* provides an API (described in a later chapter) to access the data stored in the system. The requests converted and forwarded to the *BusinessLogic* component

* + - 1. BusinessLogic

The *BusinessLogic* is responsible for the processing of the requests. It is also responsible for “combined database calls”[[16]](#footnote-16).

API

The REST API is described in Appendix B. It follows the common REST rules and provides 4 endpoints:

* GetWaterPlants
* GetTreatmentStepTypes
* GetQualityIndicatorTypes
* GetData

The structure leads to flexibility and extendibility, as new entries can be added without having to adjust the clients or the system. The usual workflow in the API is:

1. Request WaterPlants > determine desired **waterPlantId**
2. Request TreatmentStepTypes for the desired waterPlantId > determine desired **treatmentStepTypeId**
3. Request QualityIndicatorTypes for the desired treatmentStepTypeId > determine desired **qualityIndicatorTypeId**
4. Use all 3 ids to request actual data through a dedicated endpoint

The data can be filtered according to the API description. General indicators can be requested by leaving out step 2 and directly requesting QualityIndicatorTypes on the waterplant.

1. Experimental Results and Analysis

The following chapter presents the results and conclusions of the research, the design and the implementation of the system, as well as a prove for a working prototype.

* 1. Harmonization Strategies

Generally spoken, the more different formats the system accepts and harmonizes, the better. This is only true if the harmonization is complete and correct. Trying out different harmonization strategies for each of the formats has shown, that the more format-tolerant the system is, the more misinterpretations take place and therefore the more incorrect data gets store in the database.

The following diagram shows experiences won from the design and implementation phases of this system:

Figure 5-20: Service tolerance to data-misinterpretation diagram

The diagram shows 4 levels of a systems format-tolerance. The names are chosen by the author of this dissertation.

A **Non-Tolerant** system is a system, which only accepts one predefined **data-format**. This system (if implemented correctly) will not have any misinterpretations (wrongly assigned data, not recognized fields, misinterpreted values). This results in a robust, but very inflexible system.

A **Partially-Tolerant** system accepts multiple formats, but their **data-structure** has to be strongly defined and complied by the user. If the format entering the system does not match the structure definition, the data is dismissed. This leads to a more flexible system than a system using the first strategy, but is harder to implement and a little more prone to errors.

A **Very-Tolerant** system accepts data in a format which is similar enough to the defined format and its structure, to be interpreted by the system. This system more flexible than a Partially-Tolerant system but also leads to a lot more misinterpretations and errors and thus to inconsistencies in the stored data, which limits the usefulness of the system.

A system which accepts **everything[[17]](#footnote-17)**, and tries to interpret it the best possible way will require the most design and implementation time, but will also have the most inconsistent data.

In conclusion this leads to the trade-off between flexibility and data-consistency, but there is a save grade of flexibility which can be implemented into the system without the danger of making the data inconsistent, so the Partially-Tolerant system with its defined structure seems to be the best choice.

* 1. Proving the concept

The following chapter focuses on the implemented software, its description and end-to-end tests. The main objective of the dissertation was to create a software prototype of a system to gather historical and real-time data from waste water treatment plants, harmonize it and provide it for other systems and prove that this system, under certain circumstances, would work in the real world. The user feedback in the case of harmonization requests is given as reply with the error message, or with a 200 OK http code in case of a successful process.

Expandability

The system is extendable during runtime. This characteristics is granted due to the fact that the waterplants, treatment step types and water quality indicators are read from the database for every request. Extending the database dataset also means extending acceptable and harmonizable data.

To prove the system is expandable in the entire chain, the following JSON object will be harmonized:

**Request**:

|  |
| --- |
| {      **"name"**:"TestWaterPlant",    **"treatmentSteps"**:[         {            **"qualityIndicators"**:[               {                  **"name"**:"TestQualityIndicator",                **"timestamp"**:"2018-03-15T16:40:55",                **"unit"**:"mg/L",                **"value"**:123             }          ],          **"name"**:"TestTreatmentStepType"       }    ] } |

The output of the request-result, when the three names are not present in the database:

|  |
| --- |
| {  "**Message**": "An error has occurred.",  "**ExceptionMessage**": "Data could not be harmonized. Reason: Could not retreive the wastewater treatment plant id for the name [TestWaterPlant]",  "**ExceptionType**": "System.Web.HttpException"  } |

After adding “*TestWaterPlant*” to the WaterPlant table:

|  |
| --- |
| {  "**Message**": "An error has occurred.",  "**ExceptionMessage**": "Data could not be harmonized. Reason: Could not retreive the treatment type id for the name [TestTreatmentStepType]",  "**ExceptionType**": "System.Web.HttpException"  } |

After adding “*TestTreatmentStepType*” to TreatmentTypes table:

|  |
| --- |
| {  "**Message**": "An error has occurred.",  "**ExceptionMessage**": "Data could not be harmonized. Reason: Could not retreive the indicatorTypeId for the indicatorTypeName [TestQualityIndicator]",  "**ExceptionType**": "System.Web.HttpException"  } |

After adding “*TestQualityIndicator*” to QualityIndicatorTypes table:

|  |
| --- |
| **Status: 200 OK**  {  "**name**": "TestWaterPlant",  "**treatmentSteps**": [  {  "**qualityIndicators**": [  {  "name": "TestQualityIndicator",  "timestamp": "2018-03-15T16:40:55",  "unit": "mg/L",  "value": 123  }  ],  "**name**": "TestTreatmentStepType"  }  ]  } |

This response represents the object added to the service storage. Note, that this is an entirely new treatment step and an entirely new quality indicator type and not mapping.

Mapping

Mapping is a big topic in the harmonization process. The system has to deal with aliases[[18]](#footnote-18) on plant-basis as well as on global basis, meaning that aliases for all incoming data can be added, as well as aliases for data coming from specific plants.

As a follow up to the previous example, the following JSON object will be harmonized:

|  |
| --- |
| {      **"name"**:"TestWaterPlant",    **"treatmentSteps"**:[         {            **"qualityIndicators"**:[               {                  **"name"**:"TestQualityIndicatorALIAS",                **"timestamp"**:"2018-03-15T16:40:55",                **"unit"**:"mg/L",                **"value"**:123             }          ],          **"name"**:"TestTreatmentStepType"       }    ] } |

Request result:

|  |
| --- |
| {  "**Message**": "An error has occurred.",  "**ExceptionMessage**": "Data could not be harmonized. Reason: Could not retreive the indicatorTypeId for the indicatorTypeName [TestQualityIndicatorALIAS]",  "**ExceptionType**": "System.Web.HttpException"  } |

After adding the entry “*TestQualityIndicatorALIAS*” to the QualityIndicatorMapping table, and referencing the *TestQualityIndicator* type:

|  |
| --- |
| **Status: 200 OK**  {  "**name**": "TestWaterPlant",  "**treatmentSteps**": [  {  "**qualityIndicators**": [  {  "name": "TestQualityIndicator",  "timestamp": "2018-03-15T16:40:55",  "unit": "mg/L",  "value": 123  }  ],  "**name**": "TestTreatmentStepType"  }  ]  } |

Note that this time the name of the quality indicator changed – this means “*TestQualityIndicator*” and “*TestQualityIndicatorALIAS*” will both be stored referencing the same quality indicator type.

Format Tolerance

The system defines 3 data formats as valid formats. Those formats are CSV, XLS/XLSX and JSON. CSV and XLS/XLSX are handled in a different way than JSON, due the differences in their structure. The following example shows how different formats get harmonized.

* + - 1. XLS/XLSX / CSV

First an XLS file will be harmonized. The harmonization URI containing waterPlantId and treatmentStepTypeId is used for that matter.

The example table looks following:

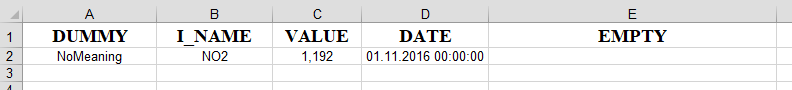


Figure 5-: Example data table

|  |
| --- |
| **Status: 200 OK**  {  "**name**": "TestWaterPlant",  "**treatmentSteps**": [  {  "**qualityIndicators**": [  {  "**name**": "Nitrogen Dioxide",  "**timestamp**": "2016-11-01T00:00:00",  "**unit**": "mg/L",  "**value**": 1.19213  }  ],  "**name**": "TestTreatmentStepType"  }  ]  } |

As the result implies, the name NO2 got mapped to the quality indicator type “*Nitrogen Dioxide*”. Also the screenshot rounded the value, while the service stored the exact value in the database. Empty and dummy columns got cleaned or ignored.

The next step is to harmonize a table with multiple different indicators:

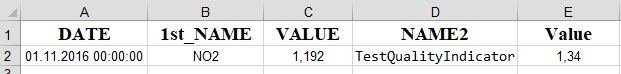


Figure 5-22: Example data table – multiple indicators

The names of the columns are purposely called “*1st\_NAME”* and “*NAME2* “to display, that mapping isn’t done based on title alone, and the value columns are found anyway:

|  |
| --- |
| **Status: 200 OK**  {  "**name**": "TestWaterPlant",  "**treatmentSteps**": [  {  "**qualityIndicators**": [  {  "**name**": "Nitrogen Dioxide",  "**timestamp**": "2016-11-01T00:00:00",  "**unit**": "mg/L",  "**value**": 1.19213  },  {  "**name**": "TestQualityIndicator",  "**timestamp**": "2016-11-01T00:00:00",  "**unit**": "mV",  "**value**": 1.337  }  ],  "**name**": "TestTreatmentStepType"  }  ]  } |

The values are stored accordingly to the request. Next, a vertical table will be harmonized:

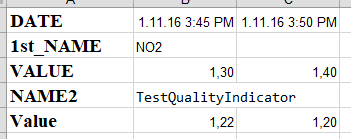


Figure 5-23: Example data table – vertical orientation

In this case, the values of an identifier are placed horizontally in the table instead of one above the other. Following is the outcome:

|  |
| --- |
| **Status: 200 OK**  {  "**name**": "TestWaterPlant",  "**treatmentSteps**": [  {  "qualityIndicators": [  {  "**name**": "Nitrogen Dioxide",  "**timestamp**": "2016-11-01T15:45:00",  "**unit**": "mg/L",  "**value**": 1.3  },  {  "**name**": "Nitrogen Dioxide",  "**timestamp**": "2016-11-01T15:50:00",  "**unit**": "mg/L",  "**value**": 1.4  },  {  "**name**": "TestQualityIndicator",  "**timestamp**": "2016-11-01T15:45:00",  "**unit**": "mV",  "**value**": 1.22  },  {  "**name**": "TestQualityIndicator",  "**timestamp**": "2016-11-01T15:50:00",  "**unit**": "mV",  "**value**": 1.2  }  ],  "**name**": "TestTreatmentStepType"  }  ]  } |

All values are mapped correctly to their indicator types, despite the different orientation of the source table.

* + - 1. JSON

Following examples show different JSON formats being harmonized. At first a request is sent using the harmonization URI containing a waterPlantId.

Request:

|  |
| --- |
| {  "**qualityIndicators**":[  {  "**name**":"TestQualityIndicatorALIAS",  "**timestamp**":"2018-03-15T16:40:55",  "**unit**":"mg/L",  "**value**":123  }  ],  "**name**":"TestTreatmentStepType"  } |

The step does not contain any information about waterplants. Additionally, there is an alias for one of the quality indicators. The outcome is following:

|  |
| --- |
| **Status: 200 OK**  {  "**name**": "TestWaterPlant",  "**treatmentSteps**": [  {  "**qualityIndicators**": [  {  "**name**": "TestQualityIndicator",  "**timestamp**": "2018-03-15T16:40:55",  "**unit**": "mg/L",  "**value**": 123  }  ],  "**name**": "TestTreatmentStepType"  }  ]  } |

The system recognized that only treatmentsteps were passed, and mapped the quality indicator alias to the correct name.

The next request contains only a quality indicator without any information on the step type or waterplant within the body. The identifiers of both are present in the URI. Request:

|  |
| --- |
| {  "**name**": "TestQualityIndicator",  "**timestamp**": "2018-03-15T16:40:30",  "**unit**": "mg/L",  "**value**": 123  } |

Corresponding answer:

|  |
| --- |
| **Status: 200 OK**  {  "**name**": "TestWaterPlant",  "**treatmentSteps**": [  {  "**qualityIndicators**": [  {  "**name**": "TestQualityIndicator",  "**timestamp**": "2018-03-15T16:40:30",  "**unit**": "mg/L",  "**value**": 123  }  ],  "**name**": "TestTreatmentStepType"  }  ]  } |

The quality indicator got mapped into the right plant and right step.

Robustness & Feedback

Robustness and user feedback are described in the same chapter for a simple reason – if the harmonization was successful and the data got stored in the system, the only feedback the user requires is the http code 200 OK. Returning the created object, like in the previous chapters, is good for control, especially in the development phase, but not necessary – since no further user actions are required. On the other side, it is very important to give the user feedback on why the harmonization of his data failed, so he can “fix” his data structure to comply the system. The system must be able at any time to deal with “wrong” data, without having impact on the stability.

The following examples show how the system reacts to data, which it cannot harmonize:

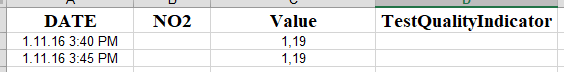


Figure 5-24: Example of a table with two valid indicator names for one value column

The figure shows a table with two valid indicator names (NO2 and TestQualityIndicator) but only one value column. The chance of misinterpretation is high, as without knowledge on how to interpret the table both indicators are valid options. Following is the output of a harmonization attempt:

|  |
| --- |
| {  "**Message**": "An error has occurred.",  "**ExceptionMessage**": "Data could not be harmonized. Reason: The number of numeric columns doesn't match the number of indicator names",  "**ExceptionType**": "System.Web.HttpException"  } |

The exception message is modified into a meaningful user message – implying, that there is either a column with values missing, or too many specified quality indicator names.

Some more examples of errors:

|  |
| --- |
| {  "**Message**": "An error has occurred.",  "**ExceptionMessage**": "Data could not be harmonized. Reason: The passed object didnt have a supported format",  "**ExceptionType**": "System.Web.HttpException"  } |

|  |
| --- |
| {  "**Message**": "An error has occurred.",  "**ExceptionMessage**": "Data could not be harmonized. Reason: Could not retreive the wastewater treatment plant name for the id [10]",  "**ExceptionType**": "System.Web.HttpException"  } |

|  |
| --- |
| {  "**Message**": "An error has occurred.",  "**ExceptionMessage**": "Data could not be harmonized. Reason: This fileFormat is not supported on this endpoint. Please use: [base-uri]/harmonize/fileFormat/{fileFormat}/waterPlant/{waterPlant}/treatmentStepType/{treatmentStepType}",  "**ExceptionType**": "System.Web.HttpException"  } |

|  |
| --- |
| {  "**Message**": "An error has occurred.",  "**ExceptionMessage**": "Data could not be harmonized. Reason: no datetime was found",  "**ExceptionType**": "System.Web.HttpException"  } |

The messages are meaningful to the point, that the person looking at it, can imagine what went wrong without looking at the request itself.

Big amount of data

Historical data will usually come in bigger chunks. For that reason, the system must be able to harmonize files with a higher amount of data. The following examples show, how the system handles such files. The database is cleaned of indicators previous to every test. Each of the tables will contain only the indicator name, its value and the timestamp, as well as a title row.

1. **Table with 10.000 entries**

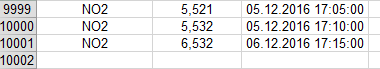


Figure 5-25: End of a table with 10000 entries containing indicator name, value and datetime

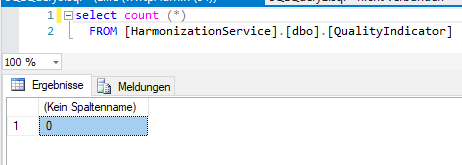


Figure 5-26: The amount of entries in the QualityIndicator table before the test

Result:

The below picture shows the time it took to harmonize the dataset and its size and that the request was successful.

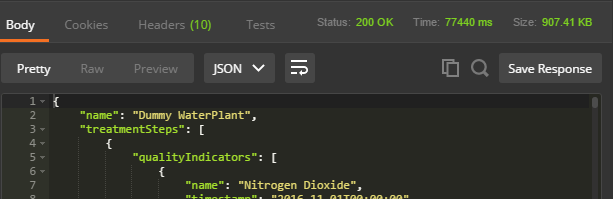


Figure 5-27: Request result (10.000 entries)

The below figure shows that all the entries are actually stored in the systems previously empty database.

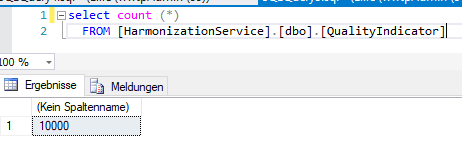


Figure 5-28: Database after harmonization request

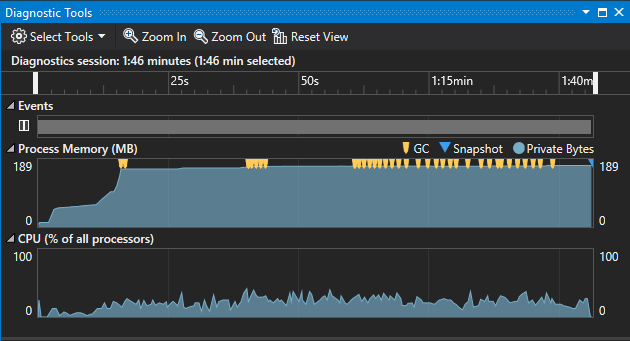


Figure 5-29: Visual Studio 2017 performance profiler

The CPU usage during the process does not go over 50% at any point. The memory has its first peak after the program is done loading dependencies, starting the background pull service and starting the web listeners. The second arrow shows the start of the harmonization process. It can be seen, that the memory usage barely goes up during that process until the end of harmonization at the 3rd arrow.

1. **Table with 100.000 entries**

The following picture shows the result of the harmonization:

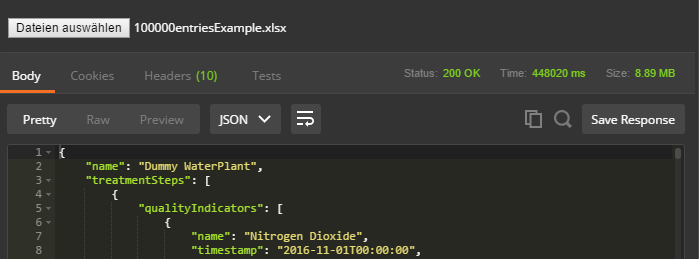


Figure 5-30: Harmonization result of 100000 entries-file

Multiplying the amount of data by 10 had the following impact on the performance:

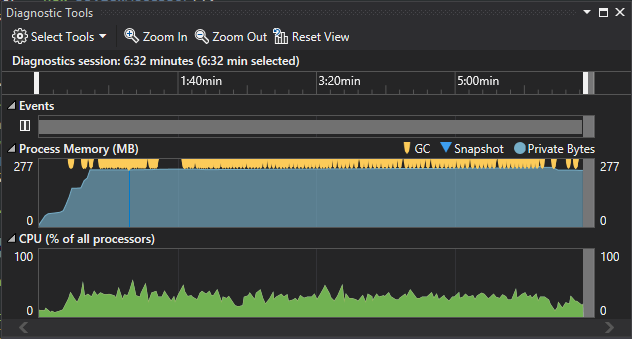


Figure 5-31: Visual Studio 2017 performance profiler – 100000 entries

The memory usage went up significantly to around 250mb. CPU usage does pick at around 50%

1. **Table with 1.000.000 entries**

Last includes a file containing 1.000.000 entries.

Result:

The request was timed out, so there is no postman result for this case. The harmonization process required around 140min to finish.

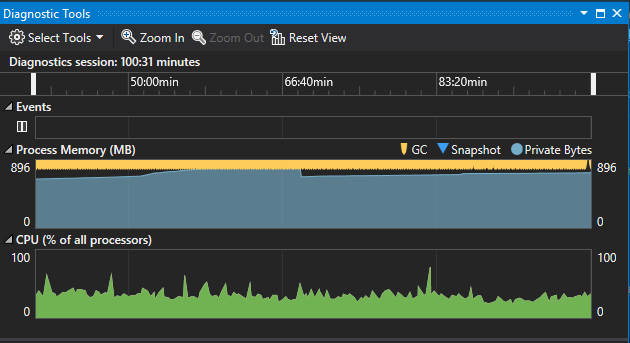


Figure 5-32: Visual Studio 2017 performance profiler – 1000000 entries

The ram requirement peaked around 900mb.

**Conclusion:**

The system was able to harmonize all test files. The times required for an entire harmonization in minutes:

|  |  |
| --- | --- |
| Entries | Time |
| 10.000 | 1,29min |
| 100.000 | 7,46min |
| 1.000.000 | 140min |

Table 5-: Measured harmonization process times

Based on the time it takes to harmonize 10.000 entries and the factor 10, the following diagram shows the expected time it should take to harmonize 100.000 and 1.000.000 entries:

Figure 5-33: Diagram: Expected time vs actual result – time to harmonize

The time it takes to harmonize a dataset with 100.000 entries is a lot lower than the calculated expectation. This is mainly due to the fact, that for 10.000 entries, the initialization of the mapping list, the blacklist and other events which take place only once during a harmonization plays a significant role. The more entries there are, the less significant this time becomes – as can be seen in the 100.000 entries set. The set with 1.000.000 entries takes longer than expected, even though the previous set actually was a lot faster. This is because of the fact that the system checks if the value is already stored in the system. The check requires to know if there is another entry for a specific treatment step type, on a specific water plant for a specific quality indicator for a specific datetime. If there is, the entry is dismissed. The more entries there are in the system, which match many of those conditions, the longer it takes to perform this check.

The required RAM is going up significantly for bigger files. The data is stored in the memory, because it is processed a lot. Storing it on the hard drive would make the time it takes to harmonize a dataset (especially the bigger ones) significantly higher.

Providing harmonized data

Based on the data acquired during previous tests, the system is now capable of providing data to the users. The test will follow the defined generic call chain containing a request for available:

* Water plants
* Treatment step types
* Quality indicator types
* Data

The following picture shows an example call to the web-service running locally:

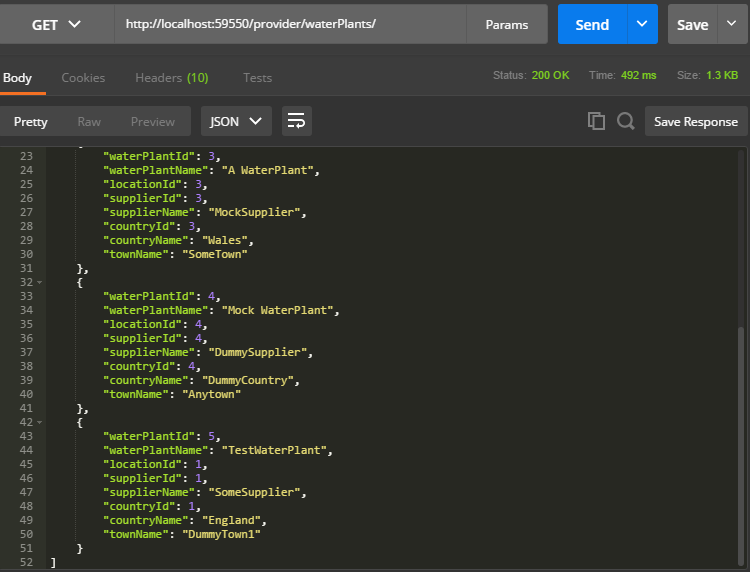


Figure 5-34: Example postman call towards locally running web-service

It can be seen, that the response contains a waterPlantId followed by a lot of metadata, which is not required to retrieve specific quality indicator data. Thanks to the relations in the database, this call requires only 492ms, as the bigger tables are only referencing the smaller ones instead of containing their values.

The next call goes towards a specific water plant with the id 5:

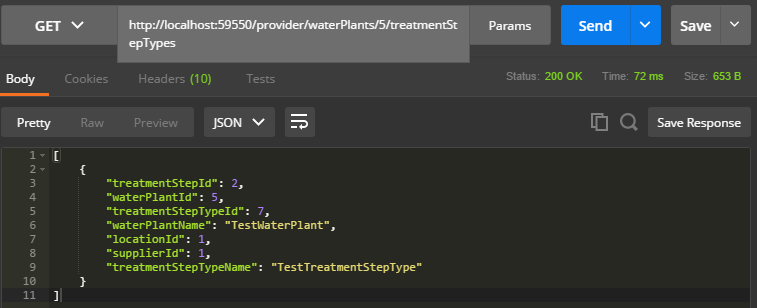


Figure 5-35: Example postman call for treatment step types on a water plant

The call takes only 72ms. It provides all treatment step types, which contain values on a specific water plant. Having the id of the water plant and the id of the treatment step type, the available quality indicator types can be requested:

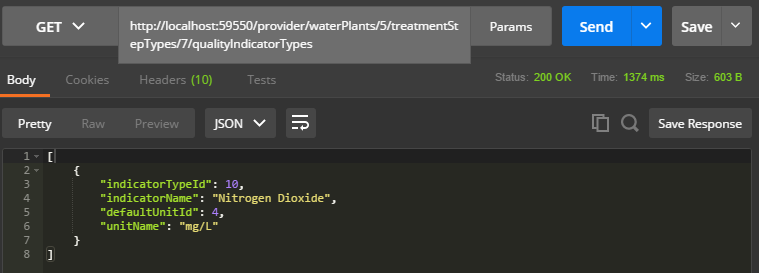


Figure 5-36: Example postman call for quality indicator types of a treatment step

The request of actual values is the longest and most expensive call, and thus can be filtered by date:

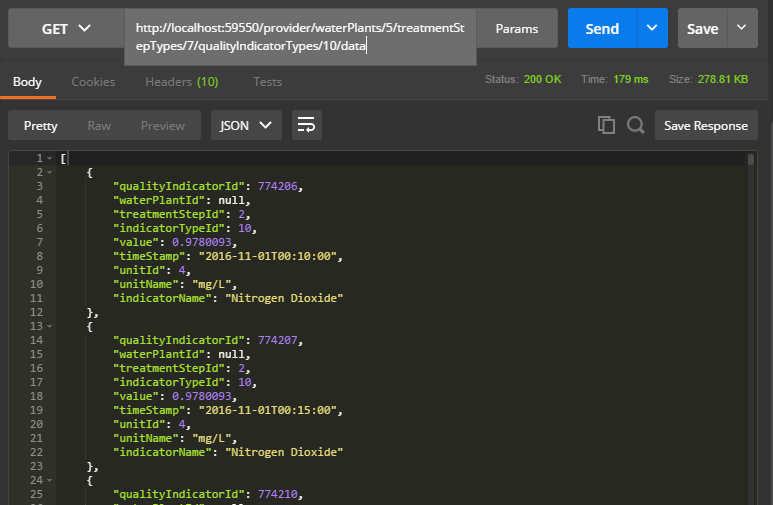


Figure 5-37: Example postman call for quality indicators of a treatment step

The user cannot tell if the data has its origin in a CSV, a XLS or a JSON file. It is coming out in a defined format which can be easily used for further processing and analyses due to the knowledge of its format. For the matter of format and available endpoints, the service describes itself with the help of a swagger file to the outside world. The swagger description can be seen by adding a /swagger/docs/[version number] to the end of the base URI:

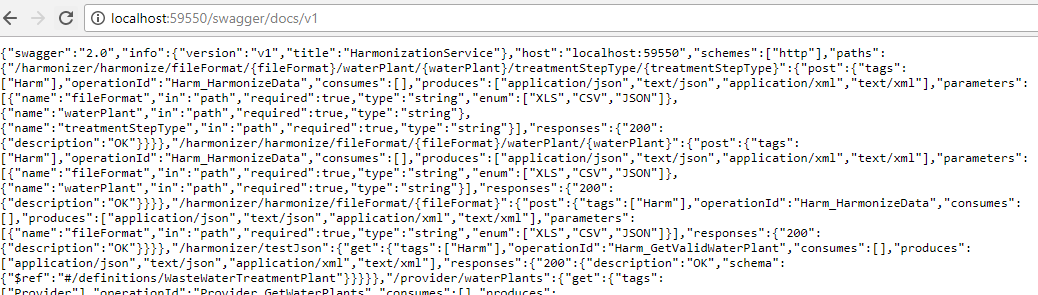


Figure 5-38: Swagger file of the harmonization system

The swagger file contains not only the formats of accepted files and expected result types, but also the description of all available endpoints.

1. Summary and Further Work

The following chapter summarizes the work done in this dissertation, the won knowledge of the project development and the problematics which occurred during the design and implementation phase. Furthermore it describes what is still to be done until this system can work in the real world, what the restrictions and problematics are and what it takes to solve them.

The first step after describing the topic, was to investigate upon waste water treatment plants, to gather knowledge about their functionality and current state in terms of technology. Since water plant treatment is done in defined steps, it was decided that the best way to make information comparable between different plants would be to assign the data to specific treatment steps.

The progress in winning knowledge about the functionality of a waste water treatment plant was paired with a parallel research about harmonization methods and projects. The UNECE project defined the (name-giving for this dissertation) method of a single-window harmonization system, which stored all harmonized data within the system, giving the users the possibility to access it, without having to access multiple different sources. Its advantage over a system that harmonizes data of defined sources at request is not only, that the harmonized data is already available at the time it is requested, but also additional metadata which can be stored for the sources, enabling load balancing, like harmonizing bigger amounts of data when there are not many requests on the service (i.e. at night). The investigations on similar projects have shown, that it is of high importance to keep the defined schema expandable and as generic as possible. This is not only important due to the fact, that there are too many data suppliers to check the compatibility of the schema with all of them, but mainly due to the fact, that the system is supposed to work in longer terms and the water industry develops. Adding new water quality defining criteria is only a matter of time.

Furthermore, research on the topic of harmonization in general has shown, that the best practice of harmonizing information is to have several well defined steps. Those steps ensure the completeness and integration of the harmonized data and consist of: simplification, harmonization and standardization. In addition to that, also a pre-simplification step was implemented in the system, to further increase its performance. The final system was implemented like pipe, where the data was processed through all of the steps, and the final outcome was stored in the database.

* 1. Problems and trade-offs

During the planning and implementation phase of the project, many problems occurred and at some points decision had to be made, which not only provided advantages for the system. The following chapter summarizes the problems faced during the dissertation.

The objective of the dissertation was not to build a system which harmonizes data from waste water treatment plants in the United Kingdom, but only to build a system which would be capable of doing so. Due to this fact, it may seem like missing access to real water-treatment plants should not be a problem. In reality the problem was not the missing data, but the missing information on the formats and data structures the data gets stored in. Available data was provided by water-treatment plants from Italy and only included two different structures, which then lead to the structure definition which the service is using. Not only the internal structure of files, but also the data types were needed to create a useful system. For that matter, the system was created with the most common data storage formats, also including formats in which the example data was provided. The missing information on file formats has led to a system design, which accepts multiple different file formats. The formats were either tree or table based and could not be processed equally. This has led to a split in the harmonization process which required the user to pass the data format of his provided data. Up until the point of the final simplification, both data types required a different processing strategy. This also means, that every other base format added to the system requires a moderate amount of new code.

The next problem focuses on the usability. The created system was not implemented with the focus on ease of administration, but functionality. This has led to several processes being only able to be performed on the database directly at the present time. This includes things like:

* Adding new water plants
* Adding new treatment step types
* Adding new quality indicator types
* Adding new locations
* Adding new units
* Adding new sources for pulling of data
* Cleaning and modifying data in the database

A general problem was the time to finish the dissertation. Even though the project was divided in tasks and every task was planned, the implementation and integration on the amazon cloud took longer than expected. The work with dynamic types has led to many implementation mistakes and runtime errors. This required some lower priority tasks to be dismissed or only partially done. There was no simulator implemented. Instead of a simulator, the program “postman” was used to send requests to the system. Due to the fact, that the pulling is done from a specific path and the expected data is in file-form, there was no need for the implementation of a water-plant simulator for the testing of data-pulling.

* 1. Further work

There are two categories of work which should still be done to make this system work in the real world. The first category includes improvements and modifications on the system, which can be done at the present time and the second category includes work that needs to be done by the industry / water plant administration and adjustments on the system after those changes took place.

The current system lacks any kind of security and gives out redundant data which is useless to the common user. This includes data like database identifiers irrelevant for user requests. Every user can add data to any available water-plant. This security issue requires some kind of user-management in the system, allowing specific users to insert data for specific water-plants. Additionally the system requires some sort of logging, so the system administrator can also monitor the processes on the system. Even though different requests are handled in parallel on the system, all data within a request is processed successively. The code requires some refactoring to improve the CPU usage. Additionally, the already mentioned additional endpoints or user interface for management of the database would be highly recommended.

To integrate the system into the real world, first of all, it would require the waste-water treatment plant administrators to provide their water quality information. The format of the information would have to be one of the formats supported by the system. Additionally, to efficiently use the data, the system requires a 3rd party system, which uses the endpoints of this system to evaluate and analyse the stored data.

* 1. Conclusion

The prototype of the system was finished in time. It is based on research towards harmonization, combined with a data schema based on waste water plant functionality. The concept is proven with example data from Italian water-plants, as well as self-created data. It is clarified, what needs to be fulfilled to make the system usable in the real world, with real waste water treatment plants. The implementation of the system took longer than it was estimated during the planning phase. The reason behind the delay was mostly the work with dynamic types, as well as problems with amazon cloud integration.

The current solution can be used to harmonize data of commonly used formats. It can be deployed on amazon cloud or on a local computer. It provides user feedback, as well as maintainable code.

Literaturverzeichnis

|  |  |
| --- | --- |
| **Bücher:** | |
| [BUN09] | Hans Joachim Bungartz, et al. „Modellbildung und Simulation“, Springer, 2009 |
| [GOL12] | Joachim Goll „Methoden des Software Engineering“, Springer Vieweg, 2012; ISBN: 978-3-8348-2433-2 |
| [MEY14] | Bertrand Meyer, „Agile! – The Good, the Hype and the Ugly“, Springer, 2014; ISBN: 978-3-319-05154-3 |
| [LOR00] | Edward Lorenz „The Chaos Avant-grande: Memories oft he Early Days of Chaos Theory“, Kapitel 7 „The Butterfly Effect“, 2000; ISBN: 978-981-02-4404-0 |
| **Internetquellen:** | |
| [TUD] | <http://ls4-www.cs.tu-dortmund.de> (Letzter Zugriff: 04.01.16) |
| [AGI1] | <http://agilemanifesto.org/iso/de/principles.html> (Letzter Zugriff: 04.01.16) |
| [AGI2] | <http://agilemanifesto.org/iso/de> (Letzter Zugriff: 04.01.16) |
| [TUD1] | <http://ls4-www.cs.tu-dortmund.de/download/LehreMaterialien/MAO2011/Kap7.pdf> (Letzter Zugriff: 04.01.16) |
| [SWI] | http://sumo.dlr.de/wiki (Letzter Zugriff: 04.01.16) |
| [SWI1] | <http://sumo.dlr.de/wiki/SUMO_User_Documentation> (Letzter Zugriff: 04.01.16) |
| [SWI2] | <http://sumo.dlr.de/wiki/Theory/Traffic_Simulations> (Letzter Zugriff: 04.01.16) |
| [SWI3] | <http://sumo.dlr.de/wiki/Networks/Building_Networks_from_own_XML-descriptions> (Letzter Zugriff: 04.01.16) |
| [SWI4] | <http://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes> (Letzter Zugriff 04.01.16) |
| [SWI5] | <http://www.sumo.dlr.de/wiki/Simulation/Output> (Letzter Zugriff: 04.01.16) |
| [CUN04] | <http://www.artima.com/intv/simplestP.html> – Ein Gespräch zwischen Bill Venners und Ward Cunningham; „A Conversation with Ward Cunningham, Part V“ (Letzter Zugriff: 04.01.16) |
| [ZAV] | www2.research.att.com/~pamela/faq.html – Quelle angegeben in [MEY14]. (Letzter Zugriff: nicht erreichbar) |
| **Weitere Arbeiten und Projekte:** | |
| [C2X] | Matthias Schulze, Timo Kosch, Ilse Kulp, Thomas Benz, Andrea Tomatis, Ilja Radusch, Gerhard Noecker, Luisa Andreone, Tanja Kessel, Carola Klessen; „Pre-Drive C2X“ – Deliverable D0.3 Final Report; 30.06. 2010 |
| [EMMA] | Sven Lahde, Michael Doering, Wolf-Bastian Pöttner, Moritz Rosin, Gerrit Lammert und Lars Wolf; „Environmental Monitoring in/of Metropolitan Areas“ |
| [GAW] | Christian Gawron, „Simulation-Based Traffic Assignment – Computing User Equilibria in Large Street Networks“, 1998; <http://www.sumo.dlr.de/pdf/GawronDiss.pdf> (Letzter Zugriff: 04.01.16) |

Appendix A

**JSON Schema**

The following presents the JSON schema defined in the chapter 4.2, including the parameter properties:

"**title**": "WaterTreatmentPlant",

"**type**": "object",

"**properties**": {

"**name**": {

"**required**": true,

"**type**": "string"

},

"**supplier**": {

"**type**": "string"

},

"**location**": {

"**type**": [

"string",

"null"

]

},

"**treatmentSteps**": {

"**type**": [

"array",

"null"

],

"**items**": {

"type": [

"object",

"null"

],

"**properties**": {

"**qualityIndicators**": {

"required": true,

"type": "array",

"items": {

"type": [

"object",

"null"

],

"**properties**": {

"**name**": {

"required": true,

"type": "string"

},

"**timestamp**": {

"required": true,

"type": "string"

},

"**unit**": {

"type": [

"string",

"null"

]

},

"**value**": {

"required": true,

"type": "number"

}

}

}

},

"**name**": {

"required": true,

"type": "string"

}

}

}

},

"**generalIndicators**": {

"**type**": [

"array",

"null"

],

"**items**": {

"type": [

"object",

"null"

],

"**properties**": {

"**name**": {

"required": true,

"type": "string"

},

"**timestamp**": {

"required": true,

"type": "string"

},

"**unit**": {

"type": [

"string",

"null"

]

},

"**value**": {

"required": true,

"type": "number"

}

}

}

}

}

Appendix B

**API Definition - DataProvider**

**GetWaterPlants**

**Uri:** {*baseUri*}/waterPlant

**Method:** GET

**Body**: empty

**Returns**: List<WaterPlant>

**GetTreatmentStepTypes**

**Uri:** {*baseUri*}/waterPlant/{*waterPlantId*}/treatmentStepType

**Method:** GET

**Body**: empty

**Returns**: List<TreatmentStepTypes>

**GetQualityIndicatorTypes**

**Uri:**

For step-based indicators:

{baseUri}/waterPlant/{*waterPlantId*}/treatmentStep/{*treatmentStepTypeId*}/indicator

For general indicators:

{baseUri}/waterPlant/{*waterPlantId*}/indicator

**Method:** GET

**Body**: empty

**Returns**: List<QualityIndicatorTypes>

**GetData**

**Uri:** {*GetQualityIndicatorsBaseUri*}/{indicatorId}

**Method:** GET

**Optional Uri Parameters:** startDateTime : DateTime, endDateTime : DateTime

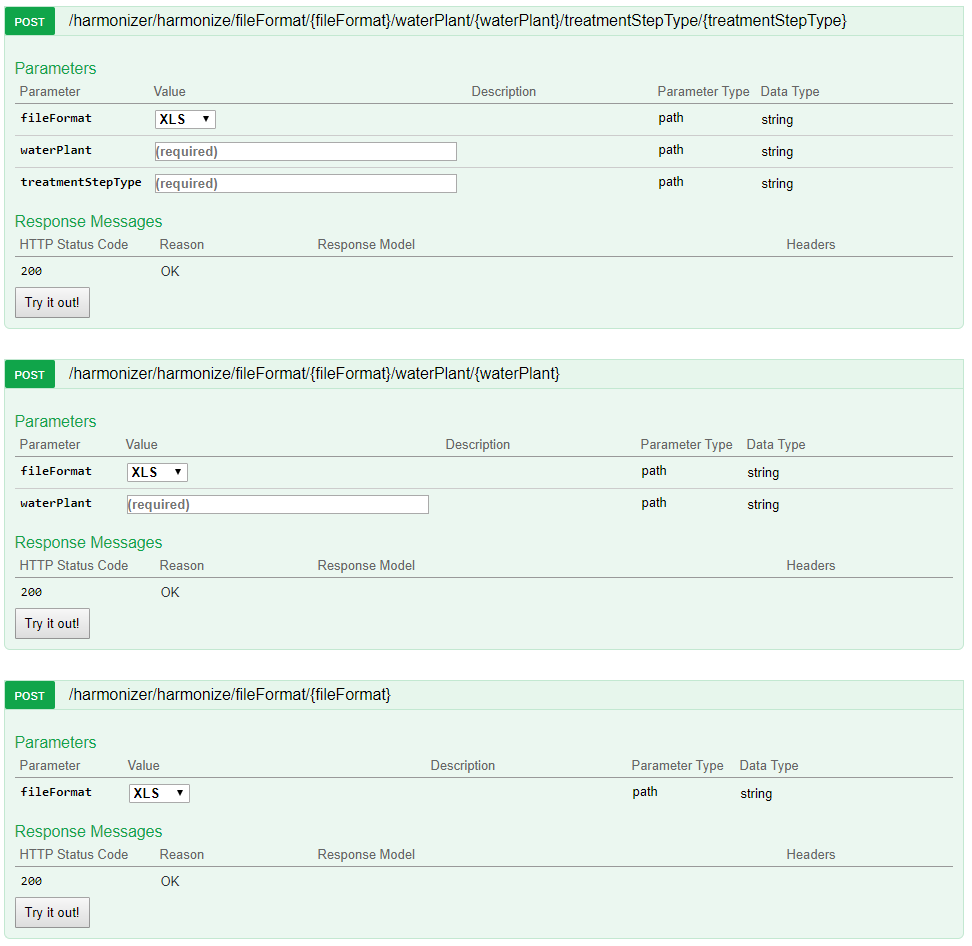
**Body:** empty

**Returns:** List<QualityIndicators>

Appendix C

**API Definition - DocumentListener**

**SetData**

**Description**: This endpoint expects a call with the file format within the URI. A file format might either be XLS/XLSX, CSV or JSON. It is required for easier handling in the further process. XLS/XLSX and CSV files may only be send through the first API endpoint, requiring the user to define the waterPlantId and the treatmentStepTypeId. JSON formatted data can be sent through any of the three endpoints, but if one of the parameters is missing (waterPlantId or treatmentStepTypeId), the service expects it to be defined within the provided data.

1. The final treatment not always is the 3rd treatment. If there are only two main treatment steps the secondary treatment becomes the final treatment [↑](#footnote-ref-1)
2. As long as this data is related to water quality [↑](#footnote-ref-2)
3. The 3 treatment stems will be explained later in detail [↑](#footnote-ref-3)
4. „Intra-Company transfer“ [↑](#footnote-ref-4)
5. The **fat** **marked questions** are the most significant ones for this dissertation, answering the other ones is optional for the prototype of the system [↑](#footnote-ref-5)
6. The software does not have to be recompiled to expand the system by new quality indicators, treatment steps or water plants [↑](#footnote-ref-6)
7. The type of a variable has to be specified before assigning a value to it and cannot be changed during runtime. [↑](#footnote-ref-7)
8. Not the same numbers as in the picture above [↑](#footnote-ref-8)
9. Describes a piece of information used to describe an object, such as “Name” [↑](#footnote-ref-9)
10. Describes a set of Property Terms, such as “Person” [↑](#footnote-ref-10)
11. Static in this case means – it is not changing very often. A change in this data might mean the historical data of this source might not be comparable to the current data. [↑](#footnote-ref-11)
12. The unit is in the most cases the same for the same type of indicator. It is good to have it passed, since this prevents errors, but storing the values with a “default” unit will be preferred to dismissing the values in total. [↑](#footnote-ref-12)
13. Not trivially comparable formats like tree-based format and table-based format [↑](#footnote-ref-13)
14. The common schema defined in a previous chapter [↑](#footnote-ref-14)
15. The sender cannot be tracked back through the configuration [↑](#footnote-ref-15)
16. Where one database call is not enough to generate the requested datastructure due to dependencies or formats. [↑](#footnote-ref-16)
17. Format and structure - wise [↑](#footnote-ref-17)
18. A name defining the exactly same thing as a different name [↑](#footnote-ref-18)