RWTH AACHEN UNIVERSITY
Chair of Information Systems
Prof. Dr. Stefan Decker

Bachelor Thesis

A META-Data Description for a Distributed Analytics Platform

Laurenz Neumann
Matr.-No.: 377539
Study Program: Bachelor Computer Science
August 27, 2020

Supervisors:	Prof. Dr. Stefan Decker Chair of Information Systems RWTH Aachen University
	Prof. Dr. Markus Strohmaier Chair of Computational Social Science and Humanities RWTH Aachen University
Advisors:	Dr. Oya Deniz Beyan, Sascha Welten, Yeliz Ucer Lehrstuhl Informatik 5 RWTH Aachen University

Contents

List of Figures

Abstract

In recent years, Data Analysis became an important tool in various domains because of rising amounts of data produced every day. Especially in the medical domain, data analysis already archives promising results, improving the healthcare system and therefore the well-being of humans. It can help to deal with diseases and can enable the discovery of new knowledge from the huge amount of medical data. However, centralising data for analysis becomes a persistent problem due to its possibly sensitive nature, privacy considerations or data protection regulations. To circumvent the data sharing problem, *Distributed Analysis (DA)* is an appropriate tool. An application of it to the healthcare domain is the so-called *Personal Health Train (PHT).* It is an architecture with different components enabling DA for researchers and healthcare facilities. But, the PHT currently lacks transparency and is a black-box for its users. To tackle this problem, a metadata schema for the PHT is proposed. We conduct interviews with domain experts, to find out what information should be included in the schema. It describes the two main components of the PHT, the *Trains* and the *Stations* with various attributes, based on the requirements stated in the interviews. Multiple existing ontologies and metadata schemas are reused for the schema. As a proof of concept, a dashboard is implemented, visualising selected parts of the schema. The schema is evaluated with the help of questionnaires, which are sent to members of the PHT research community. Furthermore, we examine how much the schema complies with the FAIR principles. The result of the evaluation is that the schema succeeds in increasing the transparency of the PHT infrastructure while being compliant with most of the FAIR principles.

1 Introduction

As the amount of data which is daily created did become almost unimaginably high, data analysis evolved into a vital tool [2].
In recent years, especially analysis of *Big Data*, denoting data sets so large that computer assistance is needed to operate on them, did become a highly optimistic research area, producing promising results [3, 4, 5, 6]. Especially in the life science and medical domain, data analysis has a huge potential to directly enhance the well-being of humans by improving the quality and efficiency of healthcare [5, 7, 6, 8]. Examples include increasing the effectiveness of treatments, enhancing medical workflows or detecting abuse in healthcare [9]. Analysis of data can help reduce the rising medical expenses in many industrial nations, by supporting the shift to *proactive healthcare*, predicting and preventing diseases before they can occur [10]. Big Data analysis can also enable personalized medicine, in which the treatments are tailored to the patient, based on phenotypic information and can help to share medical information with patients, who are more incorporated in the treatment process [5, 11]. From a scientific viewpoint, Big Data analysis can be used to uncover new findings in medicine by uncovering knowledge in the already existing huge amounts of medical data, collected by hospitals and other healthcare facilities [5]. One exemplary application uses data analysis for tracking and understanding the spread of diseases, e.g. the recent COVID-19 epidemic [12, 13].
However, due to the sensitive nature of medical data, privacy becomes a crucial matter, which must not be neglected. Privacy and the question of data ownership are key elements in the medical treatment process for many patients [14]. Unwanted leakage of medical information of patients can lead to serious problems and is highly ethically problematic [7, 15]. For example, the disclosing of psychological information can lead to discrimination and embarrassment for the victim of the data leak [16].
But as many data analysis processes include sharing medical data of patients across institutional barriers, many problems arise. Transferring such data over the internet is also seen as a security concern, as is storing medical data on an internet-accessible cloud storage [17, 18]. Another hindering factor in data sharing can be healthcare workers fearing lawsuits, for example, if past mistakes in treatment would be exposed through the data analysis [17].
Sometimes sharing medical sharing information of patients is even prohibited by federal laws [17]. For example, individual health data is considered under the *European General Data Protection Regulation (GDPR)*[1] to the group of especially protected information [19]. But not only ethical and legal problems hinder data sharing. An

[1] `https://gdpr-info.eu/`

additional problem is that medical data is very heterogeneous and large which results in large amounts of data that needs to be moved if it is collected for analysis [17]. Therefore, it would be necessary to have an accurate mechanism for access control on the data and giving a solution for the described challenges [20].
An approach to tackle these problems is *Distributed Analytics* (*DA*). In DA, the algorithm is sent to the data, instead of the data to the algorithm. Only the results of the decentralized execution of these algorithms are sent back [21]. This preserves the data sovereignty of the data owner, ensuring that the data is only used for valid and approved reasons because the data is not given away at any time. Furthermore, it can potentially reduce the amount of communication that needs to be done, before an analysis can happen, since instead of the large data sets, only the often smaller algorithms and results need to be transmitted [22]. It can also reduce computational cost [22].
An approach following the concept of DA for the healthcare sector is the *Personal Health Train* (*PHT*) [23]. It is an architecture with the goal of enabling DA for the healthcare domain. For this, it abstracts analysing tasks in so-called *Trains*, visiting multiple *Stations* with medical data. The approach has already proved its worth several times in practice [24, 25, 26].
This shows that the PHT architecture is capable of advancing data analysis by eliminating privacy related issues and preserving data sovereignty. Additionally, it solves legal problems of sharing medical data [23]. Therefore, it has the potential to empower data sharing in the medical domain and advancing data analysis. However, the architecture, due to its highly distributed design and various components, currently represents a black box for its users, hindering the efficiency and transparency [23]. Because of lacking information about the different components, it may be hard to understand the analysis process performed with the PHT infrastructure [23].
For example, once an analysing task is started, a user of the PHT architecture has no information about an execution except if it is finished or not. Furthermore, there is no way to register the distributed components, such as Trains and Stations, of the PHT in some sort of repository, hindering the reusability and findability of them. This may compromise the goal of the PHT ecosystem to make data more accessible, because the different sources of data may not be found. [23]
Therefore, the goal of this thesis is to propose a solution to this problem and to enable better privacy preserving data-analysis with the PHT.
A way of tackling such missing transparency and lack of information is by providing a metadata schema, which describes the different components, by enriching them with additional information [27, 28, 29].
Metadata is data about other digital assets or other data. It is widely used in different applications to describe given entities, to provide rich information about them and to make them findable. Examples include schemas for many digital formats of images or videos and various software repositories [30, 28]. Another example is DataCite, which solved the problem of poor quality of documentation for scientific data and their heterogeneous storage with the help of a metadata schema [31].

Such a metadata schema for the PHT architecture would enable the users to better understand the architecture and to have a better overview of the processes and the architecture in general. This would also increase the efficiency of users, because the information needed is easier accessible through a standardized representation and vocabulary.

1.1 Objectives

Since there is no metadata schema for any DA architecture, such as the PHT, at the moment, the goal of this thesis is to model a meaningful metadata description for the architecture to increase its transparency and to enable the users to gain more information about the different components of the PHT infrastructure.
The metadata should be able to convey information about the different components to the users of the PHT and enable them to understand an analysis process performed with it. Overall, this should make the PHT infrastructure more compliant with the *FAIR principles* from [32]. Therefore, it should increase the reusability of the components of it and also increase the reproducibility of analysis processes. Furthermore, the metadata description should make the components findable and therefore the data more accessible.
For this, the schema has to include both static and dynamic metadata.
However, the metadata should not contain too much information, which would make it too large and therefore hinder sharing and making the schema confusing through its size. Therefore, a good trade-off between the size and expressiveness has to be found. Additionally, the usefulness and understandability of the schema should be demonstrated with a visualization of exemplary metadata created with the schema as a proof of concept.

Based on these objectives the following research question can be formulated:

- **What information is important for the users of a Distributed Analysis architecture?**
- **How can a metadata description for a distributed analytics architecture look like?**
- **How can such metadata be displayed for a user?**

1.2 Findings & Contributions

The contribution of this thesis requirements analysis about information which is important for users of a DA architecture. The requirements are obtained as User Stories with the help of interviews, conducted with domain experts as interviewees. Based on these requirements, a metadata schema describing the two main PHT components,

i.e. the Train and the Station is developed. After this, the schema is then modelled in *RDFS* and *SHACL*.
The metadata schema is visualized with different graphs in a dashboard, created with the help of the visual data analytics platform Grafana [33].
It is evaluated with the help of a questionnaire, inquiring the opinion of domain experts on the proposed schema.
This leads to the conclusion that the proposed metadata schema is a good way to increase the transparency of the PHT infrastructure and that it describes the entirety of the components of the PHT and is useful for its users.
Furthermore, the schema is validated against the FAIR principles. The schema fulfils all of them and can, therefore, be seen as FAIR.

1.3 Overview

The structure of the rest of this thesis is given hereinafter.
In Chapter 2, related work in the domains of Distributed Analysis, metadata schemas and ontologies is presented. After that, the planned process for creating the schema is introduced in Chapter 3. The implementation of the metadata schema and the dashboard is discussed in Chapter 4. In Chapter 5 the quality of the created schema is evaluated. Finally, in the last Chapter, a conclusion is given and future work is presented.

2 Related Work

This chapter is about previous work done related to this thesis. First, the method of DA will be outlined, followed by two state-of-the-art approaches for DA. Then, the *Industrial Data Space Information Model (IDS)*, as an inspiration for this thesis, is presented. Finally, existing *metadata schemas* and *ontologies* are introduced.

2.1 Distributed Analytics

Many common approaches in data analysis are often performed on a central and prior collected data set [22]. However, in real-life scenarios, the data of interest may be stored in a highly distributed manner, for example on thousands of smartphones [34]. This may lead to problems when collecting the data, such as the need for a very large central storage system and a high bandwidth connection to the clients [22]. Further, when the centralized data set is analysed, high computational power is needed [22].
Additionally, the collection of data is questionable when it is sensitive. Since privacy is an essential part of medical ethics, this is especially true for the healthcare sector [35].
The leakage of personal medical information can harm the patient-doctor relationship or even the patient directly [7, 15, 17]. Furthermore, there is rising privacy awareness in society, also affecting the healthcare sector, resulting in increased hesitation against sharing personal information [23]. These factors prevent data sharing and result in a large amount of medical data scientifically unused [5].
Such challenges necessitate another way of accessing medical data for analysis. One proposed solution is DA. It enables the analysis of data stored in distributed data sources without the need for transferring the data to a central location [21]. The only things which are communicated are the analytical tasks and the results after the execution of them. With this approach, various advantages arise.
When the data does not leave the client, the owner of it can decide at any given time what happens with it [23]. Another advantage is the smaller amount of communication needed and reduced computational cost. The transmission of the used algorithm and the subsequent collection of the result is often less information that needs to be moved from one device to another than the whole data [22]. The same holds for the computation: Arising computational costs of a distributed execution are often less than in the case of an analysis done with the combined data set [22].
In the following two popular approaches for DA, *Federated Learning* (*FL*) and the PHT, will be presented.

2.1.1 Federated Learning

One paradigm which follows the concept of DA is FL.
It was proposed by McGraham et al. in 2017 [34]. FL denotes the training of a statistical model with multiple collaborating clients, each with computational power and own data.
This is done by first training local models on each client with only its data. These local models, containing information of the local data, are then transmitted to a central authority. This authority creates a global model by adequately combining the local models. An example of combining the local models to a global model is given in [34], where the global model's parameters are the weighted sum of the parameters of all local models.
An extension is to use multiple training rounds to enhance the quality of the global model. For this, the authority sends the global model back to each client, who further fits the model to its data, creating a new local model. After that, this local model is sent back to the server.
The method through which the local models are trained and the global model is created can vary [36]. With FL it is possible to train models on data distributed on multiple clients, which preserves the privacy and data sovereignty of those clients [34, 36].
Recent work in FL even tries to further enhance privacy and security [36].
FL algorithms can be usefully applied to domains in which highly distributed data is common, such as word prediction on smartphones, or Internet-Of-the-Things devices [37]. But applications in healthcare are also possible, such as analysis of data from wearable devices or segmentation of brain tumour on images [38, 39].

Federated Learning performs the analysis of data in parallel, however, there are other ways to tackle the distributed analysis problem such as the PHT, which is an approach for DA in a successive manner.

2.1.2 Personal Health Train

Another way to train a model is by performing training on each data set in successive.
This approach is denoted *Institutional Incremental Learning* [39].
A realization of this paradigm is the PHT architecture [23].
Its goal is to enable data analysis in the medical domain while preserving patient privacy and data sovereignty [23]. Additionally, it tries to circumvent the legal barriers of data sharing and achieves this by enabling data analysis without moving data beyond the area of influence of the owner [23]. For data analysis conducted with the PHT, two components are essential, the *Stations* and the *Trains*. Stations are all entities which provide data. This can be for example a hospital which wants to provide patient data for scientific research. The data is made locally accessible by providing a standardized interface to it.
The trains encapsulate analytical tasks that should interact with the data sets stored in the stations. They are directly executed within the station and interact there with

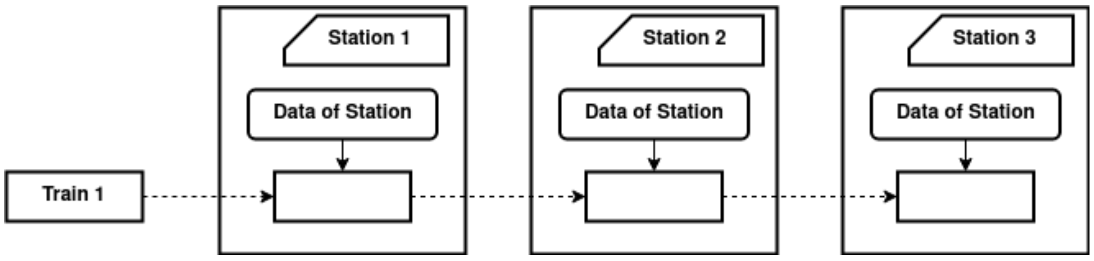


Figure 2.1: Visualization of an analysis task conducted with the PHT infrastructure. Utilized is the data from 3 stations in this case. The train starts at the first station and locally interacts with the data. After that, it visits the next station and analysis its data. After visiting the third station, this execution is finished and the train is sent back, containing the results.

locally available data. To aggregate data from multiple stations, the trains can visit one station after another, combining the results from each one. This process is also visualized in Figure 2.1.
A ‘visit’ at a station usually proceeds as follows: The station downloads the train from a central repository, executes it locally and uploads the train back to the said repository.
In the PHT infrastructure Trains should be made reusable. For this, each train has a unique identifier and should be stored in an accessible repository, so that researchers can use existing trains. The stations should be registered centrally, to enable researchers to easily find stations providing data they need for their research [23].

The PHT architecture is already used in scientific research. Shi et al. developed and validated a model for predicting the survival chances of lung cancer patients [24]. Another application was the training of a bayesian network for predicting shortness of breath in patients who had radiotherapy [25]. Chang et al. examined the dependency between Diabetes Mellitus Type 2 and healthcare costs [26].

2.2 Industrial Data Space

The IDS is an initiative to establish an environment for data sharing in the industry. Its goal is to create a domain-independent and distributed data-sharing platform which preserves the data sovereignty of the data owner. Its purpose is to enable companies to capitalize on their data and to consume data from other companies to increase productivity and innovation [40].
Different components make up this platform:
Trough *Connectors* participants can offer data to other parties. Not only data but also so-called *Applications*, programs that consume data for a specific purpose can be exchanged. *Brokers* serve as a metadata repository which stores information about existing data sets offered to the platform. To manage the process of data exchange and

for billing these transactions, *Clearing Houses* exist. *Identity Providers* handles information about the identities of the participating parties and validates them if needed. The IDS provides an *Information Model* in the form of an ontology with a semantic description of the different components and the data offered by connectors. It is domain-independent and does not lay out any domain related vocabularies.
The *Information Model* lays out three views. The *Conceptual Representation* denotes the most general view. Its purpose is to provide a human-readable and easy to understand overview of the Information Model, aimed at the public. The *Declarative Representation* uses technologies such as *OWL* and *RDFS* (see Section 2.3) to describe the *Information Model* into a machine-readable format. This representation acts as a well-defined standard and as a metadata schema for the IDS.
The third view is the *Programmatic Representation.* It contains an implementation of the Information Model into common programming languages. It is targeted at software engineers, to enable them to use the IDS in their infrastructure and to ensure that the Declarative Representation is followed.
To increase the understandability of the architecture, the Information Model is divided into 8 parts, called *facets* in the IDS.

- *Resource*: Includes basic descriptions of everything that is exchanged in the IDS, such as Data or Data analysis tasks.
- *Data*: Further describes data
- *Application*: Further describes analysis tasks that can be exchanged through the IDS
- *Infrastructure*: Describes components in the IDS which are utilized for the exchange of data and analysis task.
- *Participant*: Includes descriptions of the different users of the IDS.
- *Regulation*: Describes all regulations in the IDS, such as rules for data access
- *Interaction*: Describes the dynamic mechanisms of data access, consuming and other interactions between the users of the IDS.
- *Maintenance*: Describes dynamic information about maintenance processes performed in the IDS, such as monitoring of other process or management of infrastructure.

This different facets also echo in the Information Model [27].
In it, the different components of the IDS are described as *OWL* classes. They are populated with different attributes, describing various information about them. For complex information regarding the components, additional classes were proposed, such as *Representation*, which describes the structure of a data resource in the IDS. Another term describing such information about an architecture is called metadata [27].

2.3 Ontology & Metadata

Metadata provides information about other data. A simple example in daily life is the metadata of music files, often containing descriptive information, such as the interpreter and the corresponding album. So-called metadata schemas constrain the form of such information for a specific concept. They define the information that needs to be provided and often the format which the metadata should have.
An important part of metadata schemas are vocabularies, providing fixed terms for specific concepts to eliminate the ambiguity of natural language.

One way to define such vocabularies are ontologies. Ontologies describe entities of one or more domains and the relationship between these entities. A part of this concept are *classes*, which describe types of entities appearing in the domain. *Individuals* can be instantiated from these classes. *Attributes* describe the relationships between individuals of two classes or literals connected to an individual of a class [41]. Ontologies can reuse concepts of other ontologies, resulting in a hierarchical structure between them. An example of metadata described in the form of an ontology is the already presented Industrial Data Space Information Model (Section 2.2).

There are various formats for describing metadata schemas and ontologies. Simple metadata schemas can be even described by a text in natural language or a form of a spreadsheet. More structured is the description with *RDFS* or *XML Schema*, allowing to describe the different entities of the metadata schema in terms of their attributes and relationships [42, 43]. XML Schema allows it to constrain the described schema in terms of cardinality and data values. To achieve the same in RDF, the Shape Constraint Language (SHACL) can be used [44]. When the metadata schema has the form of an ontology, the Web Ontology Language (OWL) is a common tool for modelling [41].

Several metadata schemas for various applications were proposed. Therefore, we will afterwards present existing schemas which are suitable to the topic of this thesis.

DataCite is an organization providing identifiers for digital objects. It was created with the motivation to enhance accessibility and also reusability of scientific research data [45]. The motivating problem was that research data is, unlike research publications, not stored in a way that it can be referenced persistently. For this, DataCite acts as a registry for digital objects, storing metadata about them and providing *Digital Object Identifiers*, with which one can refer to those objects.
The Metadata schema of DataCite has the function of supporting the identification and discoverability of the registered digital objects. It consists of 19 attributes, some of them with additional sub-attributes for refined information. They are divided into three levels of obligation, namely *Mandatory*, *Recommended* and *Optional*. They cover various information, mainly important for reference, such as `Creator`, `Contributor`, `Date` or `Version`.

The Mandatory attributes are needed to register a digital object in the DataCite repository [31].

Schema.org is an initiative to develop metadata schemas for providing machine-readable information on documents, such as web pages or email. It provides such schemas for all kinds of domains, for example for persons, books or locations. The classes are structured hierarchically, with `Thing` as the parent class of all other ones. The goal of such metadata is to make websites and their content more discoverable for search engines [46].
It was created with the motivation to replace the prior existing different vocabularies for embedding semantic information, which led to an inconsistent usage of those.
It is commonly used on several websites. Guha et al. estimated in 2016, that roughly 12 million web pages use the schema.org metadata specification [47].
Since the Schema.org schema is limited to domain-independent information and does not cover domain-specific knowledge, it supports extensions to its specification. This can happen either in the form of *hosted extensions* or *external extensions. Hosted extensions* are extensions, strongly integrated to schema.org. Such a hosted extension is, for example, health-lifesci, which provides additional schemas for describing medical terms [48].
External extensions are new metadata schemas, which use schema.org as a foundation for the development [47].

CodeMeta is an initiative for creating a basic, unified metadata schema for software, compatible with the metadata descriptions from existing software repositories and other projects, such as DataCite. The motivation was to enhance the preservation, discovery, reuse and attribution of research software. The proposed schema encompasses information such as the author, the license and basic technical information like memory or processor requirements of software. It is built upon a part of *Schema.org* and expands it with additional terms [28].

The Software Ontology (SWO) is an ontology describing various concepts of scientific software [49]. The author's motivation for developing such an ontology was to improve the reproducibility of scientific experiments and workflows. In the development process, user stories and competency questions were successfully used for identifying the requirements for the ontology. It was intended for use in the biomedical domain, however, a large part of the ontology is domain agnostic.
In the ontology, software is described with various attributes, such as interfaces of the software, version, licences, used programming languages and used algorithms. Additionally, the data used by software, either as in- our output, can be described.
Furthermore, information about a development process and individuals participating in those can be expressed in this ontology [49].

Keet et al. presented with the *Data Mining OPtimization Ontology (DMOP)* an ontology for the domain of data mining and data mining optimization [50].

The authors were motivated by the fact that there were existing ontologies for the domain of data mining, but these would treat models as black-boxes and would not be powerful enough to represent the inner workflows of them. The goal of the ontology is to be powerful enough to support *semantic meta-mining*, the process of applying machine learning to metadata of analysis workflows to improve and extract knowledge about them.
The ontology consists of two parts. A *TBox*, which denotes the part of the ontology which lays out the different classes and their relationships. And a knowledge base, called *ABox* asserting facts in the ontology domain, such as knowledge about existing algorithms. The core element of the DMOP ontology is the `DM-Process` class. It denotes a complete process of data mining, containing test- and datasets, a task specification, and a relation to a hypothesis as an output.
A task specification is expressed by the class `DM-Task` having attributes specifying the in- and output of such a task. A task is a description of the objective of an action. There are several subclasses of `DM-Task`, such as `DataProcessingTask`, again with further subclasses, to specify what kind of task it is.
The `DM-Algorithm` class denotes different algorithms in the data mining domain and are connected via attributes to those `DM-Tasks`, which they try to solve. Similar to `DM-Task`, subclasses allow a finer classification in terms of the type of the algorithm. Algorithms are further described with characteristics, such as `HandlingOfCategoricalFeatures`, which denotes whether an algorithm can work with discrete features in a data set.
Data is described as `DataTables`, whose features are each described with the `Feature` class. Data and the features can, as algorithms, be further described with the help of characteristics, such as `NumberOfInstances` or `FeatureMaximumValue`. The authors identified several possible applications, such as the usage of DMOP in a Discovery Assistant for data mining workflows or in experiment databases for representing the different parts of the data mining experiments [50].

An ontology for describing data sets is the *Data Catalog Vocabulary (DCAT)* [51]. Its goal is to enable the description of data and data services of any given format in RDF. The main components are `Data Catalog`, containing metadata descriptions about various `Data Sets` and `Data Services`.
In DCAT a data set is a conceptual description of data, where a concrete representation of a Data Set, with which one can interact, is described with the class `Data Serialization`. `Data Services` describe APIs through which interaction with the data can happen. Describing other interaction patterns with data than APIs is not in the scope of DCAT. For describing the semantic content of a dataset, it relies on the *Simple Knowledge Organisation System (SKOS)*, by having an attribute connecting Data Sets to Concepts in SKOS which provides a vocabulary for describing knowledge in the form of Concepts in RDF.

2.4 FAIR Guideline Principles

Strongly related to metadata are the so-called *FAIR* principles, proposed by Wilkinson et al. in 2015 [32]. FAIR is an acronym for *Findable*, *Accessible*, *Interoperable*, *Reusable.*, which are increasingly adopted in the scientific community [52]. These denote the desired qualities of modern scientific data and metadata of scientific data. The authors created them as guidelines for creating, storing and using data after they have identified the increasing meaning of it in modern research.
The FAIR principles state the following necessities for data and metadata:
For being Findable, data should have a globally unique identifier, it should be described with meaningful metadata, which also has a unique identifier and it should be clearly connected to the data. Furthermore, data and metadata should be registered in a way that they can be found through search.
To fulfil the requirements for being Accessible, data and metadata should be accessible through their identifier, with an open protocol, which supports authentication and authorization. Also, the lifespan of the metadata should not be connected to the one of the data it describes, therefore metadata should be available even if the original data is deleted.
Interoperable demands that data and metadata use an adequate language for knowledge representation, including vocabularies and that they refer to other metadata if necessary.
The last of the qualities, Reusable, necessitates that the data and metadata have a clear license and accessible provenance information. Additionally, they should contain enough descriptions to reuse them and they should conform to relevant standards.

The PHT architecture is currently a black box for its user, disclosing information such as the train status, technical information about stations and information about the different possessors of the components. The reason for this is the highly distributed nature of the PHT [23]. Figure 2.2 depicts this. This complicates the usage of the PHT infrastructure and prevents users from having an overview of the architecture. A metadata schema for the PHT infrastructure could tackle these problems. But while many metadata schemas were proposed in the past, a schema describing a distributed analysis architecture such as the PHT architecture is still missing. Therefore the subject of this thesis is to model such a schema, based on the presented Related Work. The IDS, which is also a metadata model for a data analysis architecture, will be consulted as an inspiration. Since the past practice in ontology creation is to reuse existing ontologies, the presented approaches will be considered when the schema is created. The presented FAIR principles will be used in the evaluation part as a quality guideline for the proposed schema.

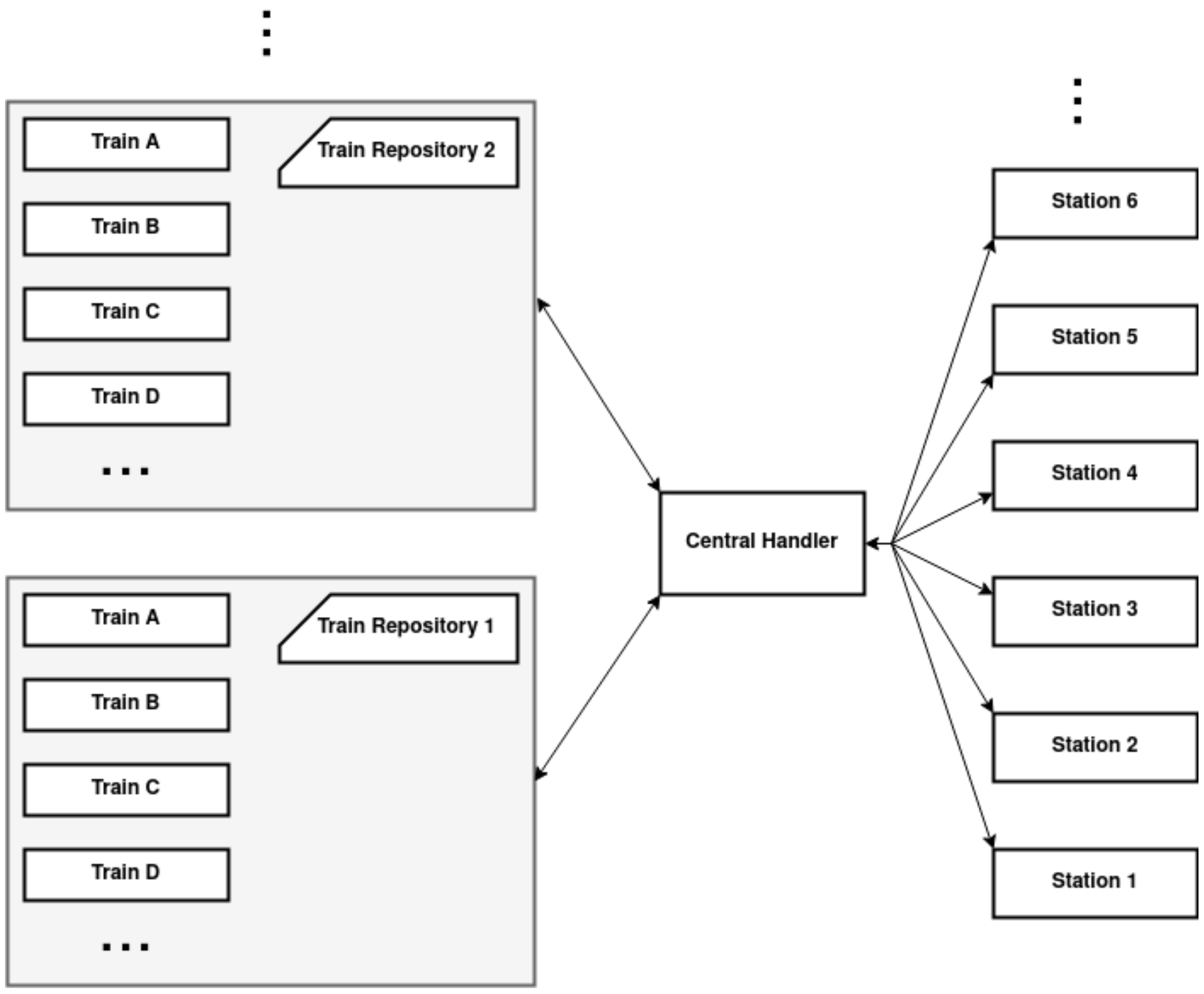


Figure 2.2: A visualization of the components of the PHT. On the left side, two train repositories containing multiple trains are displayed. On the right, multiple stations are shown. Connected are those through a Central Handler. Note that the number of stations, train repositories and trains are not capped. Rather, they can be scaled, which leads to an even more confusing architecture. For example, the number of stations is not fixed but can change at any given time. The same holds for the trains. This is not transparent, since an overview, such as this one, is not available for the users.

3 Concept

For the process of developing a suitable metadata schema for the PHT architecture, we propose a workflow consisting of multiple steps. The required information, which the schema should cover, are identified with the creation of user stories, gathered through multiple interviews. After that, the user stories are transformed into structured information by stating attributes, consisting of an identifier, a data type and a cardinality, for each item of metadata required by the user stories. To enable machine readability and to represent the schema in a standardized form, these attributes are expressed in RDFS and SHACL.

3.1 Problem Description

The metadata schema has the purpose of increasing the transparency of the PHT infrastructure. It is a highly distributed architecture. In the PHT, the amount of Stations and Trains is not capped. Rather, the complexity rises with every new participant who contributes new Stations and Trains. An example of this is given in Figure 2.2. Users of the PHT currently only have information about their own components, because there is not mechanism for obtaining information about other participants Stations and Trains. Therefore, the PHT currently is a black box for its users.
This hinders efficiency since the other components, such as Stations which hold data of interest, can not be easily utilized.
Responsible for this problem is lacking metadata about the different components. Therefore, the first important step is to identify what kind of information is obscure to the users of the PHT.

3.2 User Stories

To achieve this, so-called user stories, an approach for requirements analysis originating in the domain of software engineering, are used [53]. A User Story is a short statement, made by a user, expressing the expectations in that system in a compact shape. In the case of software engineering, Lucassen Garm et al. examined the usefulness of user stories and concluded that they have a high capability of identifying requirements [54].
A user story contains three types of information.
The *role* of the user in the system, what the user requires and the benefit of having this information [55]. In the context of the metadata schema, the user stories have the purpose of identifying which kind of information increases the transparency and

efficiency of the PHT architecture for the participating users.

An exemplary structure of such a User Story is given by Mike Cohn in [53]:

As a <role>, I want to <story> so that <benefit>.

The used structure of the user stories is similar to this, but altered to suit the context of a metadata schema:

As a <role of the user in the PHT>, I want to have information on <characteristic of a component/concept> so that <benefit of having that information>.

These user stories serve two purposes in the process of creating the metadata schema: First, the user stories ensure that the scope of information in the schema is justified and suitable. Second, the user stories can act as a tool for evaluation, by assessing how good the schema covers the desire for metadata expressed by them.
This makes the user stories in the process of creating the metadata, combined with their clear and short structure them a suitable tool for collecting the requirements by the users. However, before the user stories can be utilized in the process of creating the schema, they need to be acquired from the user base of the PHT.

To achieve this, conducting interviews is the method of choice. The interviewees are researchers working in the domain of DA and being familiar with the PHT. As domain experts they can justify the requirements they state and give the schema valid foundation. The user stories are gathered during the interviews in two ways.
The interviewees are asked what kind of information they would like to have in the metadata schema. The answers are directly transformed into user stories, expressing the desire for the information stated by the interviewee.
Second, the interviewees are asked for an exemplary data analysis workflow, by using the PHT. Additional user stories can be inferred, by examining the given exemplary workflows with respect to the information needed to perform those.

After the user stories collection, the contained information needs to be extracted. For achieving this, the information is transformed from natural language into a more structured representation.

3.3 Metadata Attributes

The collected user stories contain much valuable information about the requirements of the users concerning the metadata schema. However, to create this metadata schema, the required metadata stated in the user stories needs to be formulated in a structured representation.

The first step of this transformation is to group the user stories regarding their content. This improves the ordering of the user stories, helps discovering similar requirements and structures the metadata schema overall.
The first differentiation one can make in the metadata is between information addressing Trains and Stations, since these are the two main components of the PHT, both playing an essential role in the architecture.
Also, these two components are different in their nature. The Train acts as an encapsulation of one or more analysis algorithms, representing an executable and dynamic component. The Station acting as a data provider for the architecture, is more static in nature.
Each of these two sets can be further divided into subsets addressing different scopes of information.

The Train metadata can be split up into three parts:

- *Business Information*: This subset contains all metadata with a non-technical, organisational purpose. A trivial example would be information about the author of a Train

- *Technical Information*: This subset includes all metadata, which provides information about the technical nature of a Train. This includes all static information needed to use the Train for analysing data in the PHT. An example of such metadata would be the algorithm the Train utilizes or type of data.

- *Dynamic Execution Information*: This subset includes all the metadata which is created during the execution of a Train. An example residing in this subset would be information, at which Station a Train is at a given moment.

The Station metadata can also be split up into three parts:

- *Business Information*: The criteria for metadata in this subset is equivalent to Business Information of a Train.

- *Runtime Environment Information*: This subset includes all metadata about the computational environment a Station provides. An example would be information about the capabilities the Station has in terms of computation power.

- *Data Information*: This subset contains metadata about the data a Station provides for data analysis. Metadata belonging to this subset gives insights about the data and should help researchers using the PHT to decide whether data is suitable for analysis. An example from this subset would be the size of a dataset of the Station.

With this proposed classification of metadata, each user story can be assigned to one metadata type resulting in a more structured representation.

However, in this form, the user stories still only describe the information in natural language, which might ambiguous. Additionally, multiple user stories may express the same information, because they are extracted from multiple different interviewees, with overlapping requirements.
To tackle this problem, each user story is assigned one or more attributes, describing the metadata required by it. Furthermore, this enables to eliminate redundancies, because there will not be two attributes describing the same metadata. It is rather easy to check for each user story, whether there is already a suitable attribute for the stated requirements.

Each attribute is named with a simple identifier and a data type. For this, it is distinguished between primitive data types, enumerations and complex data types. Primitive data types denote simple literals such as a string or an integer.
Enumerations, often abbreviated *enums*, are a set of predefined values, from which the attribute can take a value.
Complex data types have the form of metadata schemas themselves. This leads to a metadata schema constructed from smaller schemas as building blocks. An advantage of this modular approach is that the complex data types could be reused in future work.
In addition to the data type and the identifier of the attributes, providing a reasonable cardinality is important. While some attributes may be mandatory for the schema, because they denote obligatory concepts, others may be optional since the information they convey are useful but not essential.
For this, the DataCite metadata schema (see Section 2.3) is a suitable example, since the authors differentiate whether an attribute is needed or not, by stating for each attribute whether it is mandatory, recommended or optional. Additionally, specified for each attribute is a minimum and a maximum cardinality.
Exemplary, the `identifier` attribute is mandatory and has a cardinality of exactly 1. An indication of a `version` is optional, with a minimum cardinality of 0 and a maximum cardinality of 1 [31]. A similar approach is used for the proposed metadata schema.
Each variable is provided with a minimum and maximum cardinality of occurrence i.e. how many different attributes with this identifier are allowed in the metadata. An explicit specification whether an attribute is mandatory, recommended or optional is omitted since this is implicitly indicated by the cardinality.

In the process of defining attributes based on the user stories, it is useful to consider and reuse existing metadata schemas and ontologies. Existing metadata schemas are incorporated by reusing attributes of these schemas.
The identifier of the attributes in these schemas are often well established and therefore widely understood. Furthermore, we achieve a degree of interoperability, because

the identifier of attributes with the same semantic is consistent across schemas.
We choose DataCite as one of the schemas reused. When reusing the mandatory terms of the DataCite metadata schema (see Section 2.3), we can achieve two goals: First, we can ensure that the naming and the datatype of the attributes are well-considered. Second, we can reuse DataCite in the way that the object which is described by the new approached metadata schema could be registered in DataCite, which would be another additional benefit.
The same holds for ontologies. They can be included in our metadata schema by using chosen classes as data types for attributes. Since ontologies act as a method to unambiguously denote concepts in different domains, it is ensured that even complex concepts are understood across different schemas.
Therefore, it is feasible to use ontologies which are well established in the domain. Because they are often developed and maintained by domain experts, it is ensured that the classes of the ontologies describe the concepts in it properly. Furthermore, an effort such as defining different shapes of a concept is already done and therefore can speed up the development of the new schema.

Because of these advantages, we reuse multiple existing metadata schemas and ontologies in our schema. The following were identified as suitable to describe concepts in the metadata (see Section 2.3):

- DataCite is considered for metadata belonging to the group of Business Information of both Train and Station since it is widely used and the attributes defined by it are well established. As discussed in Section 2.3, it contains several attributes needed for referencing to a digital object.
- Additionally, the FOAF ontology will be used in Business Information, for describing social entities like the owner of a Train in the metadata schema.
- The DMOP ontology and the SWO are suitable for describing Technical Information of the Train and for the Data Information of the Station. Both describe technical aspects. However, DMOP is more detailed in terms of the classes and attributes.
- The DCAT vocabulary is also considered for the Data Information.

Nevertheless, not all attributes proposed for our schema can be reused from existing schemas and ontologies. The reason for this is that our schema, as a metadata schema for a DA architecture, is a novel approach. Therefore, some important concepts in such an architecture are not covered by any existing proposed metadata schema.
For these requirements, attributes with meaningful identifiers and suitable data types have to be created.

The structured list of attributes, which was gathered through this process step, allows us to reduce the statements to a more structured form and to eliminate any redundancies since only one attribute is used for each metadata item.

While a structured list of attributes can already be considered as a definition of the metadata schema, this form is not standardised and not processable for computers, which may hinder sharing the schema. Therefore the schema needs to be transformed from a simple list of attributes into such a format, which is described in the next section.

3.4 Modelling the Metadata

A description in a standardised and machine-readable format allows to share the metadata with other researchers in a more clear way. Furthermore, it makes it easier to use the schema in software. Since ontologies are reused, the classes of the ontologies can be integrated directly, yielding a more interoperable schema.

A variety of technologies are suitable for structuring the schema in such a way.
As stated in Section 2.3, one could use OWL and create the schema in the form of an ontology. The IDS presented in Chapter 2 shows how this can be accomplished. However, OWL provides functionalities that are not used for the PHT metadata schema, such as the support for complex logical expressions. Instead, RDFS and SHACL (see Section 2.3) are used for modelling the proposed schema. RDFS allows defining schemas for RDF documents, by defining classes and attributes. An example of this can be seen in Figure 3.2. A class called `Cat` is created and given an attribute describing the age. Inheritance of classes is also possible, the `Cat` class is a subclass of `Mammal`.
But, RDFS is not able to force RDF documents to follow a specific template. The example in Figure 3.2 models a schema, but it is still possible to define an instance of the `Cat` class which varies from the defined schema. It is also not possible in RDFS to constrain the values of literal. In the example, one could set the age of a cat to values such as -1 or 100. For this reason, SHACL is additionally utilized.
In SHACL constraints can be given for a schema, in the form of so-called *shapes*. With these shapes, additional properties, which the schema has to fulfil, can be defined. Examples are the cardinality of an attribute or restrictions on the values of the attribute. It is also possible to constrain the values to a specific class.
Figure 3.2 gives an example of SHACL. It lays out constraints for the schema defined in Figure 3.1. The `age` attribute is limited to a value between 0 and 30. By setting the cardinality of the `age` attribute to exactly 1, the attribute is made mandatory.

In the proposed metadata schema, RDFS classes are used to create complex data types and enums. Attributes are used for primitive data types and to connect the schema to the classes of the complex ones.

For attributes covering primitive data types, the value of the attribute is restricted to specific patterns with SHACL. Furthermore, enums are defined by constraining the value of the attribute to a given set.

Since these RDFS and SHACL definitions are both themselves expressed in RDF, a proper serialization language is needed. While various of those serialization languages exist, we think that *TURTLE* is the most appropriate for this kind of task, because it provides a well readable syntax, important for sharing the modelled schema. The examples in Figure 3.1 and Figure 3.2 are expressed in TURTLE.

Overall, this well-defined metadata structure will have multiple purposes: Most importantly, the RDFS and SHACL definitions act as a format which can be shared with other researches, standardising the exact metadata schema in a human readable and processable way. For this purpose, sufficient comments on the components of the schema are provided. Furthermore, the validation functionality of SHACL can be used to check whether a given metadata document follows the proposed metadata schema. While sharing metadata in RDF yields high interoperability between systems, it is not fully accessible, because users not experienced with such technologies will struggle to extract usable information from it. Therefore, the schema does not provide increased transparency to all participants of the PHT, but rather only to those who are technically versed. Representing the Metadata in a visual and well processed way is an approach to tackle this problem and to provide transparency regardless of the user's knowledge about technologies such as RDF and TURTLE. A proof of concept of such a visualization is proposed in the next section.

```
 1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
 2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
 3  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
 4
 5  @prefix exp: <http://www.example.org/catSchema#> .
 6
 7  exp:Mammal rdf:type rdfs:Class .
 8
 9  exp:Cat rdf:type rdfs:Class .
10  exp:Cat rdfs:subClassOf exp:Mammal .
11  exp:Cat rdfs:comment "An exemplary class describing a cat as a
        subclass of the Mammal class" .
12
13  exp:age rdf:type rdfs:Property .
14  exp:age rdfs:range xsd:integer .
15  exp:age rdfs:comment "An examplary attribute describing the age
        of the cat" .
```

Figure 3.1: Example for an RDFS document describing animals. The class `Cat` is a subclass of the class `Mammal` (line 9,10) and describes a cat. A cat can have an age denoted by the attribute `age` (line 13-15). The classes and the attribute are defined in the exemplary domain `www.example.org`.

```
 1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
 2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
 3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
 4 @prefix sh: <http://www.w3.org/ns/shacl#> .
 5 @prefix dash: <http://datashapes.org/dash#>
 6 @prefix dcterms: <http://purl.org/dc/terms/> .
 7
 8 @prefix exp: <http://www.example.org/catSchema#>
 9
10 exp:CatShape rdf:type sh:NodeShape .
11 exp:CatShape sh:targetClass exp:Cat .
12 exp:CatShape sh:property _:exampleProperty .
13 _:exampleProperty sh:path exp:age .
14 _:exampleProperty sh:minValue 0 .
15 _:exampleProperty sh:maxValue 38 .
16 _:exampleProperty sh:minCount 1 .
17 _:exampleProperty sh:maxCount 1 .
18 _:exampleProperty rdfs:comment "This NodeShape constrains the
      values of age between 0 and 30" .
```

Figure 3.2: Example for a SHACL document defining constraints on the `exp:cat` class defined in Figure 3.1. The property `age` is constrained with an unnamed *Property Shape*. In line 13, the attribute, which the shape should constrain, is set to the `age` attribute. It has to be in the interval $[0, 38]$ (line 14,15) since we assumed that the highest possible age of a cat is 38. The cardinality of the attribute is set to exactly 1 (line 16,17).

3.5 Visualization of the Metadata

For achieving a suitable visualization, a Dashboard is developed, as a proof of concept to increase the transparency of the PHT architecture by processing and displaying the metadata defined in the schema.

The term Dashboard denotes a graphical interface which displays a variety of data, by putting it into graphs of various forms. It achieves this by visualizing relevant parts of the data in a meaningful way [56].
An example of the use of Dashboards are the ones created in the COVID-19 pandemic by various organizations, such as the World Health Organisation [1]. They were successfully used to communicate different statistics about the pandemic, such as the infection rate of each country, to a public audience.
This is an example, that Dashboards can ease the access to big sets of data. In our case, the Dashboard visualizes different metadata from our schema.

To reduce the effort of creating the dashboard, a software called Grafana is utilized. Grafana is a partly open source visual analytics platform. It supports various data sources and is developed by GrafanaLabs [33].
It has the advantage that visualization for other data than the PHT metadata can be added if necessary. The platform allows the user to customize dashboards by choosing from different visualizations, called panels, and arrange them as wished by drag-and-drop. The whole configuration of the dashboard can happen through the front-end and requires minor technical knowledge.

The target audience for the Dashboard are users who execute Trains for data analysis. Therefore, metadata of the Runtime Information set is mostly displayed. To achieve this, the dashboard consumes metadata from an *Application Programming Interface (API)* and displays the metadata with appropriate visualizations. For this, suitable graphs are chosen for the information in the metadata.

[1] `https://covid19.who.int/`

4 Realization

The workflow for the metadata schema creation, as proposed in Chapter 3, is implemented in multiple steps. First of all, the interviews were conducted. The information stated by the interviews is extracted into the user stories. We add for each user story attributes, describing the metadata required by the story. The proposed transformation of these attributes into a machine-readable part is then done using SHACL and *RDF*. Finally, we implement the Dashboard on the Grafana platform, providing a visualization of the modelled metadata.

4.1 User Stories

The interviews were performed with four experts from different scientific domains. They were conducted with the help of a video conference software. This had the advantage that after the approval of the interviewees, the interviews could be recorded. Further, the recordings simplified the extraction of the requirements stated in the interviews.
Each interview was planned with a duration of one hour, which was met in all cases. During the interviews, we asked various questions. The first question inquired addressed the motivation of the domain experts to use the PHT architecture in general. The goal of this question is to get a conception of the overall use processes in a better way. Furthermore, we tried to get a deeper understanding of what parts of the architecture are especially important for the different users of the PHT.
Also, the conception of the possibilities and the purpose of the PHT infrastructure may differ between the interviewees, which is important to fully understand the requirements they state. After that, we asked experts on the proposed partition of the metadata (Section 3.2).
The remaining question aimed at the requirements of the experts on the metadata schema. For each of the overall six proposed metadata groups, we inquired what metadata the experts require in the schema to achieve increased transparency.
Some example questions can be found in the appendix.

After conducting the interviews, the information was extracted from the records of the interviews. Regarding the partition of the metadata, all four experts agreed with it, meaning that this partition could be maintained for the metadata.

Also, we collected various requirements on the metadata schema. These were then transformed into user stories according to our described concept. This resulted in overall 89 user stories, including duplicates.

Examples of the extracted user stories include the following, the complete list can be found in the Appendix (see Appendix A).

- *As a Train User, I want to have Information about the dispatching pattern of a Train so that I can decide if I want to reuse it*
- *As a Train Dispatcher/Data Scientist, I want to have a contact person for each Station in the Metadata, so that I can contact him/her if issues arise*
- *As a Train User, I want to have a general description of the computing capabilities of a Station, so that I can decide if the Station will be able to run my Train*
- *As a Data Scientist, I want rich information about what kind of data is available at Stations, so that I can easily find Stations that I need*
- *As a Station Owner, I would like to access Information about past executions of the Train so that I know what the purpose of the Train, which deals with my data, is*

Some user stories were omitted for the process of creating the schema because they stated requirements which were not in the scope of the metadata schema of this thesis.

Through the user stories, the lack of transparency and the concrete reasons for it became visible. From these the following, overall goals of the schema became clear.

- It should enable users to decide whether to reuse a Train or not. It should also help users to decide whether they want to visit a specific Station. For this, the metadata required by the experts were information to increase the overall transparency.
- It should enable schema users to find suitable Stations for their Trains with respect to technical information and needed data.
- It should enable to obtain telemetric information about a Train during its execution.

We combined the user stories into a single document before they are transformed into a structured form, as proposed in Section 3.3, with the help of a tabular representation.

4.2 Metadata Table

After we have collected the user stories, we assign each to one of the metadata groups we proposed in Section 3.3. This yields overall six sets of user stories. For each one of these sets, a table is created, in which the user stories are incorporated and populated with appropriate attributes, describing the metadata required. These tables act as a mapping from user stories to attributes and give a first form of a metadata schema.

An example can be seen in Figure 4.1. These tables can be found in the Appendix (see Appendix B).

The attributes used in this process consist, as proposed, of an identifier, a data type and a cardinality. The cardinality is expressed in the form of a range, with minimum and maximum values or `n` when the maximum is unbound (e.g. "1..n", "0..1"). We choose the identifier of each attribute in a way that it provides a brief indication about the semantics of it.
The chosen notation of the attributes is: `identifier: datatype (cardinality)`
For the data types, existing metadata schemas and ontologies are reused when it is reasonable (see Chapter 2).
Figure 4.2 shows an overview which ontologies are utilized in which context. If attributes express more complex information, we create additional schemas as a datatype, called *subschemas* hereinafter. An example of this is the `event` attribute. It refers to an event happening through an execution of a Train. Since such an event is more complex than a primitive data type could express, an additional schema `ExecutionEvent` was proposed, describing such an event. Not that subschemas can have additional subschemas themselves.

As a Train User, I want to know who the owner is so that I can decide if I will use that Station	`owner: FOAF:Agent (1..n)`

Figure 4.1: An exemplary user story with corresponding attribute, taken from the complete table in the appendix. Here the user requested information about the owner of a Station. This information is modelled in the attribute owner with `FOAF:Agent` as the data type. The notation with *FOAF:* as a prefix, denotes that the datatype is a class from the FOAF ontology, in this case the `Agent` class.

The layout of the tables already indicates a structure for the schema. The main schema for the Train and the Station holds all business related information. The technical and dynamic execution information on the Train side plus the runtime environment and data information on the Station side are stored each in an additional subschema, to which the main schema references.
In the following sections, some attributes, proposed for the metadata schema, are discussed.

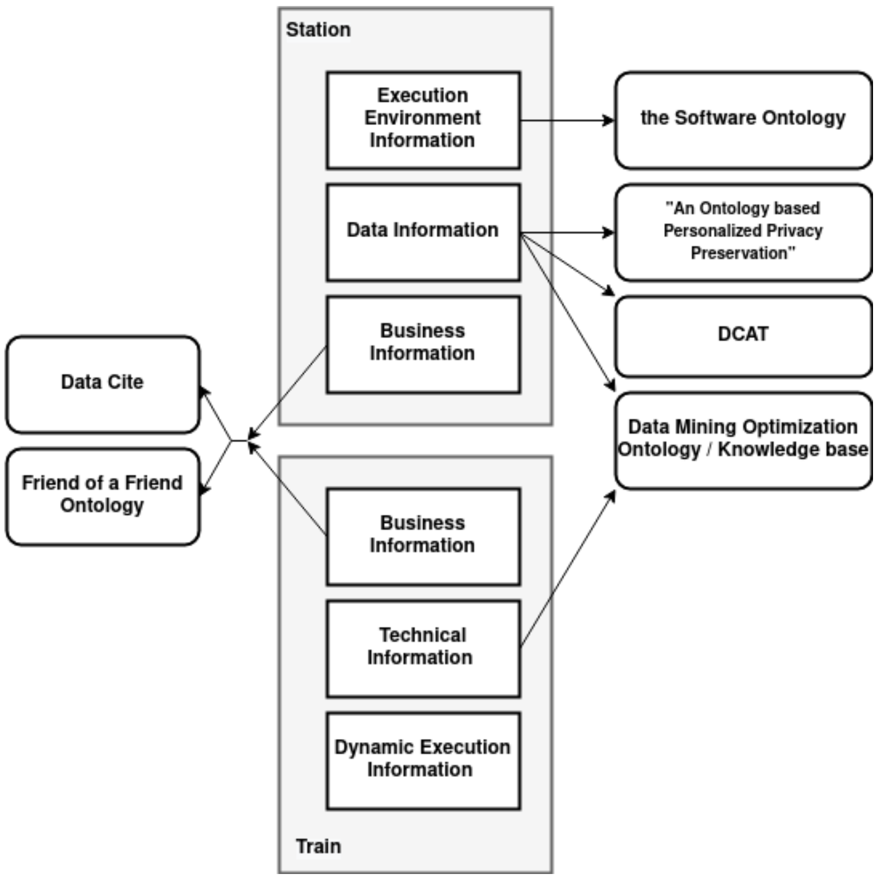

Station
Execution Environment Information
Data Information
Business Information
Business Information
Technical Information
Dynamic Execution Information
Train
the Software Ontology
"An Ontology based Personalized Privacy Preservation"
DCAT
Data Mining Optimization Ontology / Knowledge base
Data Cite
Friend of a Friend Ontology

Figure 4.2: The reused ontologies/metadata schemas in relation to the proposed groups of metadata. If an ontology is connected to a group of metadata, it means that this ontology/metadata schema was reused in that group to denote concepts. DataCite, denoting general information and FOAF, describing information about social entities, are used for business information. The SWO is used to describe the license for a model. DMOP is used to describe technical information for the Train and information about data for the Station. The ontology presented by Can et al. in [1] is used to denote the grade of anonymization of a data set.

4.3 Train Metadata

The structure of the metadata schema for the Train is displayed in Figure 4.3. For the Business Information, multiple attributes are proposed. One essential information in this group of metadata, desired from all experts, is information about the author. We expressed this information with the attribute `creator`, which has the `Agent` class of the FOAF ontology as the datatype. This class addresses social concepts in general such as a person or a company. Therefore, the author of the Train can also be any of these entities, which is reasonable since both a single researcher or a company can create such a Train. This is reused from DCAT [51], where creator / owner of a dataset is also expressed with this class. The cardinality of the `owner` was set to a minimum of 1 since information about the creator is essential if one wants to trust a Train. Also identified as important is the possibility to add a description to the Train to express details about the Train. This is realized in the form of a `description` attribute with string as data type.

When modelling the metadata for the business information of the Train, all attributes which are mandatory in the DataCite metadata schema, are reused and also made mandatory. This achieves that the Trains could be registered in DataCite and also ensures that enough basic information is available for each Train.
In contrast to the Business Information, we describe the Technical Information of a Train in a subschema.

This subschema is called `Model`. During the interviews, it was stated that the Train itself and its different executable parts should be distinctly described since for example a Train can have multiple different executable parts. Because of this, a Train can have multiple models.

As a way to denote the required computational power of a `Model`, stating an estimation of the minimum required flops is identified as reasonable. Being able to make a statement of the minimum required computational performance is important so that users reusing the Train can decide whether a Station can execute it in an appropriate time.
To mirror the development of the performance of processors and to prevent unreasonable big numbers in the schema, *Gigaflops* is chosen as the unit.
For stating information on the characteristics of the `Model`, the class `AlgorithmCharacteristic` of the DMOP ontology is reused. The several subclasses of this class can be used to denote various properties of an algorithm. Being able to state such characteristics is important so that it can be verified if a model works with a specific set of data. An example of a characteristic available through DMOP is `ToleranceToNoise` with `ToleratesNoise` and `ToleratesNoNoise` as possible values. Additionally, we utilize DMOP for stating which algorithm is used in the model. DMOP contains extensive domain knowledge about data mining, including various typically used algorithms. The attribute `usedAlgorithm` of the `Model` refers to these instances.

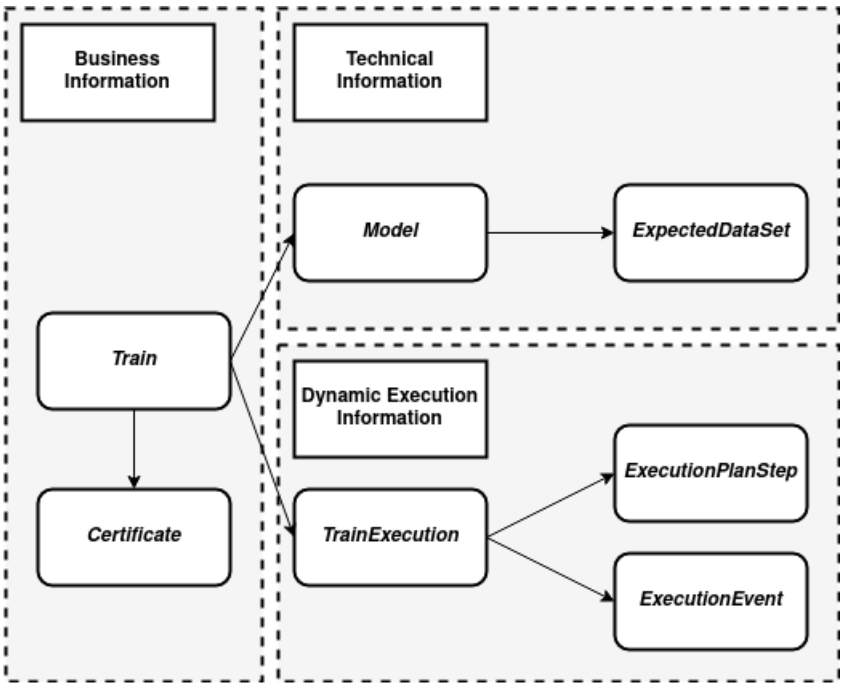


Figure 4.3: Visualization of the Train schema and its subschemas. The Train schema itself contains Business Information. Part of the Business Information is the `Certificate` subschema. The `Model` subschema contains Technical Information about the Train. The kind of data on which a `Model` can operate is described with the subschema `ExpectedDataSet`. `TrainExecution` contains information about a single execution of the Train. Important for the execution are the two subschemas `ExecutionPlanStep` and `ExecutionEvent`. `ExecutionPlanStep` contains all information about a step in the planned route of a Train. This includes which Stations should be visited and in what order. Information about the actual execution is then described with multiple instances of `ExecutionEvent`. This can be for example the CPU consumption of the Train at a given time or an event describing that a Train started running at a Station

Another important information, required in the user stories, is a description of data the Train consumes. This is given by stating the protocol with which the data should be accessed and by providing a description of the structure and the semantic content of the data. The protocol through which the data set can be accessed is described with the help of an *URL (Universal Resource Locator)*.
An URL is a string identifying a digital resource.
It typically consists of an *URL-Schema*, the location of the resource and optional information about username and password for accessing the resource. The URL-Schema denotes the protocol through which the resource can be accessed [57].
However, the location of the data source is different between Stations. Therefore, the important part for indicating what kind of data the `Model` consumes is the URL-Schema at the beginning of the URL, such as `file://` or `smb://`, identifying the used access protocol.
This attribute indicating the expected URL-Schema can also be matched to the URLs in the `DataSet` subschema, describing a data set of a Station (see Section 4.3), to determine whether a Train works with a given data set.
However, an URL does not give information about the exact structure of the data. If for example, the URL-Schema is *file://*, we only know that the Train interacts with a file, but not with what file format.
Therefore, the `Model` has an attribute referring to one of two subschemas: `ExpectedTableDataSet` and `ExpectedFileDataSet`. `ExpectedTableDataSet` describes desired properties of a dataset in a tabular format, while `ExpectedFileDataSet` states requirements at a dataset consisting of one or more files. These subschemas are similar to the subschemas describing data sets of Stations, discussed in the next section.

The last of the metadata groups, Dynamic Execution Information is also modelled with a subschema to which the main Train schema refers. This subschema `TrainExecution` holds all information about a single execution of a Train.

The typical process of such an execution can be shown with the state chart in Figure 4.4, in which all the different states and the possible transitions between them are visualized.
Before a Train can be used, it needs to be created and stored in a repository as a base image. From this, an instance of the Train can be created which then can be executed. When the first Station of the planned route starts downloading the Train, the state changes to *Train transmission*. This state is added because it is important to indicate that the Train cannot start at a given moment because it is still transmitting to the Station and this process can, depending on the size of the Train and the bandwidth of the internet connection of the Station, last for some time. When a Station has finished downloading the Train, the train has the *Idle at station* state, where it waits on actions taken by the station. This can be either refusing to execute the Train by rejecting it or can starting the execution. If the Train is rejected, its state changes to *Station error* and it is pushed back to the central repository. When the station executes the Train, it has the *Running* state. The execution can either be finished with an error or

successfully. If it was successful and there are Stations left, the state returns to *Train idle* until the next Station starts downloading the Train. If the execution is finished at the last Station, the Train state becomes *Finished Train State* and the lifecycle of the instantiated Train ends.

As shown in the state chart in Figure 4.4, it is differentiated in the metadata schema between a Train as a base image for execution (denoted also as *Train Class*) and concrete, executed instances of that Train. This is mirrored in the metadata schema. The Train schema itself and the `Model` subschema describe the Train Class, while an executed instance is described through the `TrainExecution` subschema.
The two core parts of this subschema are first, a description for the planned route of the Train and second, a description of the process of the actual execution.

The planned route is described through stating an ordering of the Stations which should be visited during the execution. For this, the subschema `ExecutionPlanStep` was proposed. It holds an attribute for identifying the Station which should be visited. Furthermore, each step is assigned to an index, so that the order, in which the Stations should be visited, can be indicated. While an execution plan can be modelled with this, information about the succession of the actual execution are still missing.
Additionally, it was stated by the user stories that detailed information about each execution of a Train are required. Therefore, information about running executions and past execution need to be included in the schema.
The proposed solution for this is an event log for each execution. When the events are updated during an execution, it not only serves as a way to model detailed information about past executions but can also convey telemetric information about a Train to its creator.
Each of the events has a mandatory attribute `timestamp` describing exactly the time the event occurred. This is important to reproduce the order of the events and can further be used to determine the state of the Train. Different possible events are proposed.
One example is `StartedRunningStationEvent`, which describes when a Train starts to run at a specific Station. Another example is `CPUUsageReportEventShape`, describing the CPU consumption of the Train at a given moment. The whole list of events can be found in the appendix. (C.2)

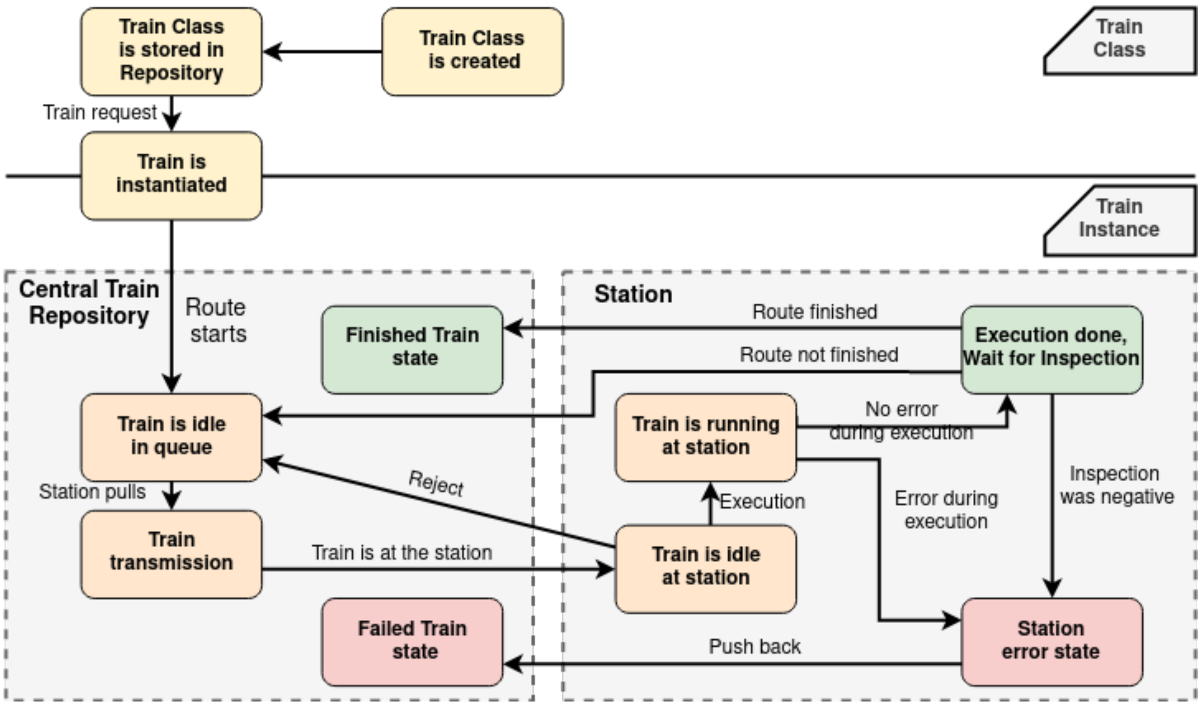


Figure 4.4: Overview of the different states of a Train during its execution. The visualization is divided by a horizontal line, differentiating between the static part and the dynamic part of the Train lifecycle. A Train state changes from static to dynamic, if it is initiated from the base image stored in the repository. Therefore, the dynamic lifecycle and static lifecycle of Trains are actually isolated from each other: A Train can be instantiated multiple times, with each instance having its own dynamic lifecycle. The dynamic part is also divided into a Station and a repository side. Red states symbolize an error during an execution, green states success. The lifecycle of a Train always begins with its creation and upload to a repository where it is stored.

4.4 Station Metadata

Figure 4.5 depicts the structure of the Station metadata and which subschemas addresses which group of metadata.
The business information of the Station is similar to the business information of the Train.
Instead of an attribute `creator` similar to the `creator` attribute of the Train, the Station has a mandatory attribute describing the owner of the Station. However, the datatype is the same as in the Train schema, the *Agent* class of the FOAF ontology. Additional to the creator, an attribute `responsibleForStation` refers to an `Agent` which can be contacted for questions or support, to have a contact person to quickly refer to if any problems with the Station arise, as this was explicitly stated in the user stories. An also desired information in the user stories was the geographical location of the Station, to increase the overall transparency and to prevent legal problems, e.g. when the Station is in another country. For this, an attribute containing the approximate longitude and latitude coordinates were added. This is reused from the DataCite metadata schema and is identified as a simple and unambiguous way to denote the location.

For metadata about the computational environment of a station, we proposed an additional subschema, called `ComputationalEnvironment`.
Since a Station can have multiple computational environments on which the Train can be executed and needs minimal one to be part of the PHT architecture, the cardinality of this attribute has a minimum of 1 and no maximum.
The computational power of the execution environment of the Station is measured in an estimation of the flops, similar to the required computational power of Trains.
To denote which programming languages are supported by a ComputationalEnvironment, the subclasses defined by the Software Ontology of the `ProgrammingLanguage` class are reused. This attribute is only important if the Station runs the software native on a given operating system. To indicate whether the Station supports the execution of containers the attribute `supportsOCI`, with a boolean datatype is added. *OCI* is the Open Container Initiative, aiming at providing a unified interface for the execution of containers. Executing the models of Trains encapsulated in a container has the advantage that every dependency of the model can be included inside the container.

Besides the computational environment, information about the data, which a Station provides, are also needed. This includes semantic information about the content of data sets, which is stated in the user stories as a requirement. For this, the subschema `DataSet` is proposed.
The subschema includes an attribute referring to the `Concept` class of the Simple Knowledge Organisation System (SKOS). SKOS is a specification, based on RDF, for creating controlled vocabularies and taxonomies. With this attribute, it can be referred to existing, already defined vocabularies to describe the semantic content of the

data set. This approach is reused from DCAT.

To describe the structure of a dataset we distinguish between two general types of data: *Tabular Data*, datasets which have the form of a table, such as a database and *File Data*, where the data is in the form of files, such as a collection of images. These two types of data sets are modelled with subschemas, called `TabularDataSet` and `FileDataSet`, inheriting the attributes from the DataSet schema. Having information about the structure of a data set is important to determine whether a data set is suitable for a Train.
The structure of a `TabularDataSet` is given by describing each attribute of the table. For this, there is the additional subschema `TabularDataAttribute`. This schema also describes the topic of the attribute with the help of SKOS, by referring to Concepts. An `TabularDataAttribute` additionally has an attribute indicating the data type of this tabular attribute, by referencing to the `Datatype` class from DMOP. This indicates exactly the kind data type, for example `boolean`

A `FileDataSet` adds to the `DataSet` schema an attribute, describing the type of the file. This attribute has the datatype string and contains a valid *Internet Media Type* (or MIME type), describing the format of the file. *Internet Media Types* were proposed to describe the format of messages transmitted over the internet by the W3C [58]. An example of such a file type is *image/jpeg*, describing the *jpeg* image format.

To state how strong a dataset is anonymized is the goal of the attribute `usedDifferentialPrivacy`. For this, the differential privacy methods described in the ontology by Can et al. in [1] are reused as possible shapes for the attribute.

After modelling this schema with the help of the tables, corresponding SHACL and RDFS definitions are created to make this schema machine readable and to create a validation tool for metadata.

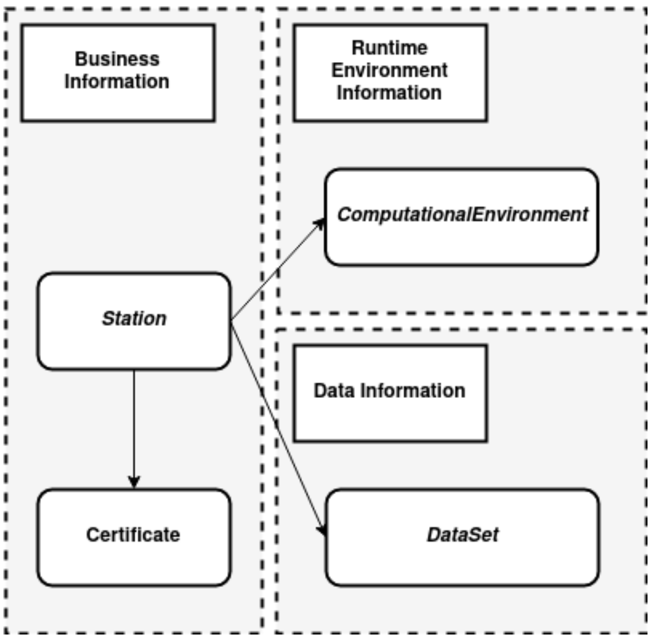


Figure 4.5: Visualization of the Station schema and the corresponding subschemas. Business Information is conveyed in the Station schema itself. Metadata about the computational environments a Station provides is contained in the `ComputationalEnvironment` schema. `DataSet` describes a data set which the Station holds.

4.5 Metadata Model Definition

As stated in Section 3.3, for the definition of the schema in a machine readable format, RDFS and SHACL are utilized. For this, three files containing the definition are created, one file for the RDFS definition of the schemas and subschemas. Further modelled is the schema with files containing the SHACL definitions. One file defines the constraints for the Train, the other for the Station.
For the Train and Station schemas and for each of the subschemas, an RDFS class is created. The attributes are created as RDFS attributes. After that, each is constrained with the help of SHACL. If an attribute with a restricted cardinality was proposed, it was embedded into SHACL with the `sh:minimumCount` and attributes. An excerpt from the definition of an attribute can be seen in Figure 4.6.

Attributes with Enums were realized by constraining the range of the attributes to a specified set with the `sh:in` attribute. An example of a definition of an enum can be also seen in Figure 4.6.
Names for the attributes are defined with `sh:name` and a brief explanation of the attribute, if necessary, is provided with `sh:description` For each schema, the SHACL attribute `sh:closed` was set to true, meaning that metadata documents defined following that SHACL definition must not add new attributes.
This ensures that each metadata document follows exactly the definition and does not add new attributes which can lead to confusion when sharing the metadata.
With these three documents, the metadata schema is readable for machines. Therefore, it can be used for example to validate metadata schemas. The content of the two SHACL files can be found in the Appendix C.
Validating a metadata document can happen with multiple tools, such as *pySHACL* [1] or the *SHACL Playground* [2]. A result of such a SHACL validation, in that case of a metadata document with an error, can be seen in Figure 4.7.

For the Dashboard, *RDF* documents describing metadata as modelled in the schema are visualized with the help of a Grafana Dashboard (see Section 3.5)

[1] `https://github.com/RDFLib/pySHACL`
[2] `https://shacl.org/playground/`

```
sh:property [
        sh:path pht:dispatchingPattern ;
        sh:name "Dispatching Pattern" ;
        sh:in (
                pht:trainDispatchingPatternParallel
                pht:trainDispatchingPatternLinear
                pht:TrainDispatchingPatternMixed
        ) ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
] ;
sh:property [
        sh:path pht:public ;
        sh:name "Is execution public" ;
        sh:datatype xsd:boolean ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
] ;
```

Figure 4.6: Two examples from the SHACL definition of the Train. *sh:path* denotes the attribute which is constrained. The upper property definition is an enumeration, *sh:in* constrains the possible value on one of the list. The lower definition is an attribute with a primitive data type, *boolean*. The cardinality is constrained on exactly one.

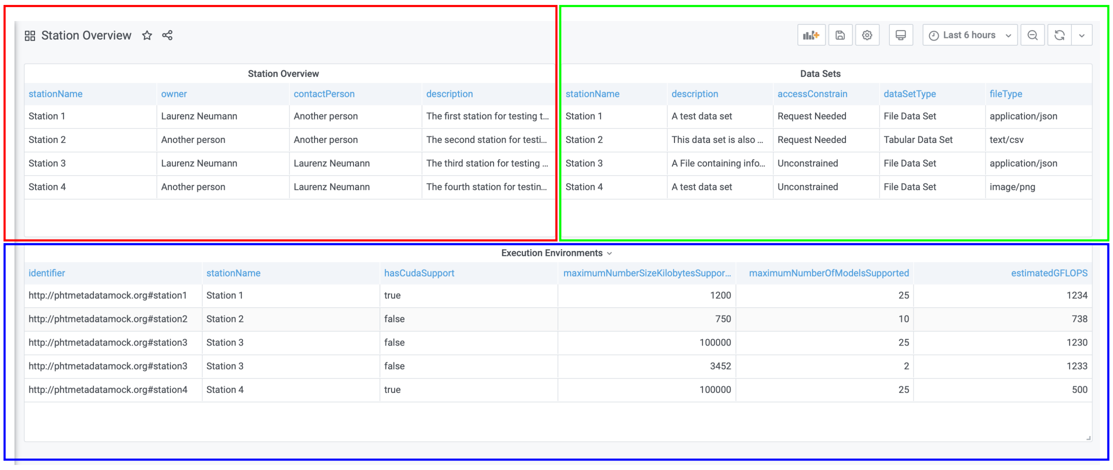
```
[
        a sh:ValidationResult ;
        sh:resultSeverity sh:Violation ;
        sh:sourceConstraintComponent sh:
           MinCountConstraintComponent ;
        sh:sourceShape _:125 ;
        sh:focusNode <http://phtmetadatamock.org#train1>
        sh:resultPath pht:creator ;
        sh:resultMessage "Less than 1 values" ;
]
```

Figure 4.7: An exemplary result of a validation of metadata with the help of SHACL. This validation was done with the SHACL Playground. The result of the validation is in RDF, the serialization is TURTLE in this case. A Train was described with the help of the metadata schema, but the attribute pht:creator, mandatory in our schema, is missing. *sh:resultPath* denotes the path of the missing attribute, *sh:focusNode* the subject (here the Train) which have the attribute missing. *sh:resultMessage* gives an explanation about what triggered this error.

4.6 Dashboard

As stated in Chapter 3, we use Grafana for the creation of the Dashboard.
The core components of Grafana are the already mentioned *panels*, visualizing the data. It already includes several of such *panels*, such as bar graphs or tables.
The data which should be visualized is obtained from so-called *data sources.* The function of the *data sources* is to add support for different data endpoints, such as databases, and transform obtained data into Grafana's representation of it, so called *data frames.* This general representation functions as middleware between *data sources* and *panels*, ensuring the interoperability of all components [59].
Both new panels and new data sources can be added with the help of *plugins*, which is what we do to enable support for the PHT metadata.
The visualization of the metadata is a proof of concept that the metadata can increase the transparency of the PHT infrastructure for the users, by making the information, which the schema provides, accessible in a visual, centralised way.

To achieve this, a data source was added to Grafana, accessing RDF data, serialized in *JSON-LD*, and converting the metadata about Trains and Stations in reasonable *DataFrames.*
JSON-LD is a serialization format for *RDF* data with the goal to combine RDF with *JSON (JavaScript Object Notation)*, which is a widely used format for data exchange. It expresses RDF triples in *JSON* and is therefore fully compatible with normal *JSON* parsers. This is done by querying an *HTTP API*, which provides the metadata in the JSON-LD format.
However, since the Dashboard is a proof of concept and there is no server-side software for the PHT infrastructure at the moment supporting the proposed metadata schema, a mock service for the API was created.
We done by creating a simple HTTP server, implemented in *node.js* and *express.js.* This mock serves the RDF documents and simulates the execution of a Train across four Stations.
Figure 4.9 shows the implemented Dashboard and provides a brief explanation about the different components of it. Such a Dashboard could be used by the different users of the PHT, to consume the metadata about the different components.



Station Overview

stationName	owner	contactPerson	description
Station 1	Laurenz Neumann	Another person	The first station for testing t...
Station 2	Another person	Another person	The second station for testi...
Station 3	Laurenz Neumann	Laurenz Neumann	The third station for testing ...
Station 4	Another person	Laurenz Neumann	The fourth station for testin...

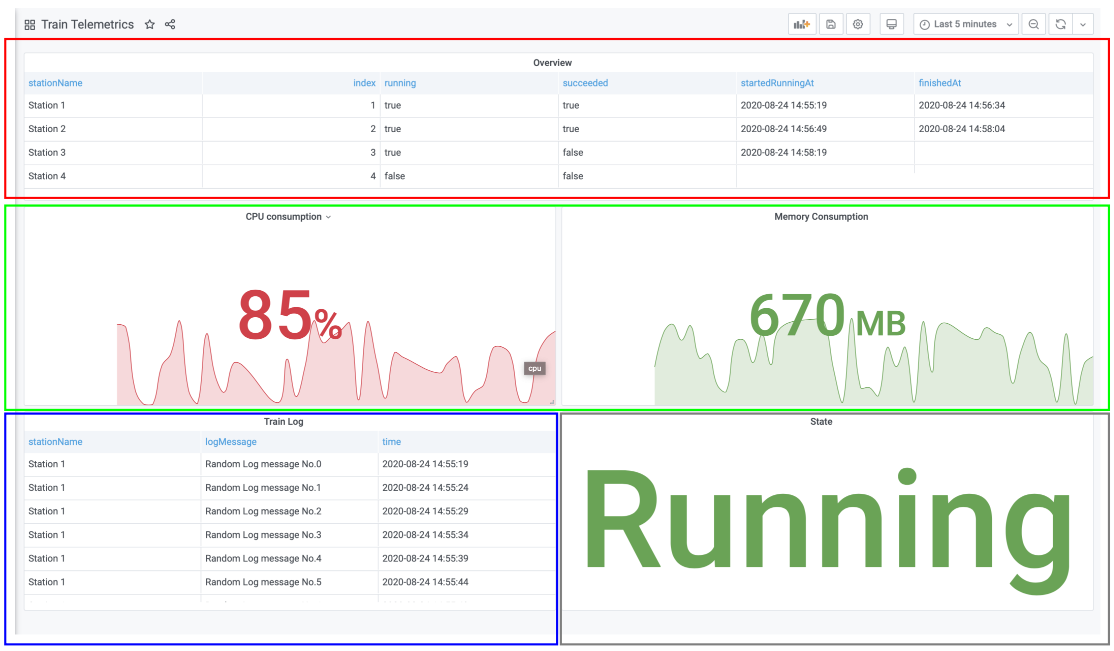
Data Sets

stationName	description	accessConstrain	dataSetType	fileType
Station 1	A test data set	Request Needed	File Data Set	application/json
Station 2	This data set is also ...	Request Needed	Tabular Data Set	text/csv
Station 3	A File containing info...	Unconstrained	File Data Set	application/json
Station 4	A test data set	Unconstrained	File Data Set	image/png

Execution Environments

identifier	stationName	hasCudaSupport	maximumNumberSizeKilobytesSuppor...	maximumNumberOfModelsSupported	estimatedGFLOPS
http://phtmetadatamock.org#station1	Station 1	true	1200	25	1234
http://phtmetadatamock.org#station2	Station 2	false	750	10	738
http://phtmetadatamock.org#station3	Station 3	false	100000	25	1230
http://phtmetadatamock.org#station3	Station 3	false	3452	2	1233
http://phtmetadatamock.org#station4	Station 4	true	100000	25	500

Figure 4.8: A screenshot of the part of the implemented Dashboard, which displays information about all stations, its execution environments and its data sets. In the panel marked by the red box, an overview of the Stations is given. For each Station, its owner, the responsible person and a description is displayed. The panel, marked by the green box, conveys basic information about all data sets, provided by stations. In the panel, marked by the blue box, metadata about the different executions environments are given. For each environment, its computational power, the maximal number of concurrent supported models and the maximal allowed model size is displayed. It is further indicated which environments supports hardware accelerated computations through CUDA.



Overview

stationName	index	running	succeeded	startedRunningAt	finishedAt
Station 1	1	true	true	2020-08-24 14:55:19	2020-08-24 14:56:34
Station 2	2	true	true	2020-08-24 14:56:49	2020-08-24 14:58:04
Station 3	3	true	false	2020-08-24 14:58:19	
Station 4	4	false	false		

Train Log

stationName	logMessage	time
Station 1	Random Log message No.0	2020-08-24 14:55:19
Station 1	Random Log message No.1	2020-08-24 14:55:24
Station 1	Random Log message No.2	2020-08-24 14:55:29
Station 1	Random Log message No.3	2020-08-24 14:55:34
Station 1	Random Log message No.4	2020-08-24 14:55:39
Station 1	Random Log message No.5	2020-08-24 14:55:44

Figure 4.9: A screenshot of the part of the dashboard which conveys telemetric information about the execution of a Train. In the part marked by the red box, the route of the Train is displayed. For each Station on the route, it is indicated whether the Train already visited it and if so, the exact data when the execution of the Train started and stopped are also shown. The two panels in the middle, marked by the green box show the CPU and memory consumption of the Train. The panel marked by the blue box displays the log messages of the train. The panel marked by the grey box shows the current state, in this case Running.

5 Evaluation

The evaluation of the metadata schema is divided into two parts. First, we conduct an evaluation based on the opinion of domain experts on the schema. Second, the metadata schema is evaluated with regard to the *FAIR* principles as a quality guideline for data sharing.

5.1 Questionnaire Setup

We create a questionnaire to evaluate the proposed metadata schema with the help of domain experts. It contains a brief introduction about the schema, the proposed schema itself and evaluation questions for it.
We evaluated the schema with regard to two qualities. The first quality is completeness, that is whether the schema completely describes the components of the PHT architecture or if information, seen as important by the experts, is missing. The second quality is usefulness, where we evaluate the comprehensibility and expressiveness of the schema. This includes whether the schema describes the components clearly and unambiguously.
The questions asked are the following.

Completeness:

- *Are additional components important for you that need to be addressed in the metadata schema? (in addition to Train and Station)*
- *Would you include other existing ontologies / metadata schemas into it?*
- *Do you think that the schema is helpful for identifying Stations suitable for a given analytical task?*
- *Would metadata, defined from the schema increase the transparency of the components of the PHT to you?*

Usefulness:

- *Does the proposed partition of the metadata (Business Information, ...) makes sense to you?*
- *Does the schema contain in your opinion unnecessary attributes / address unnecessary concepts?*

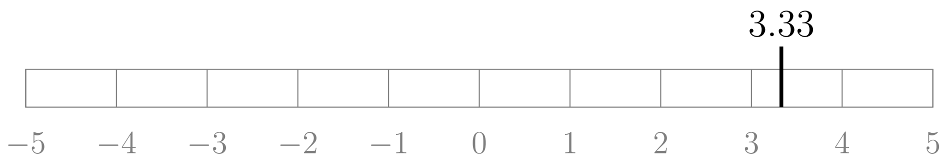
- *Are the proposed subschemas comprehensible?*
- *Is it clear what kind of information the different subschemas address?*
- *Would you be able to define metadata, following the schema, for your Station/your algorithm?*

All questions had a comment section. For 7 of these questions, the respondent could additionally mark on a scale from -5 to 5 how strong they would agree or disagree. The questionnaire was sent to 8 people, including the interviewees of the requirements analysis with the user stories.

5.2 Questionnaire Results

After receiving the filled out questionnaires, we averaged the results of the 7 questions, which could be answered on the given scale. The answers to the questions regarding the completeness of the schema are displayed in Figure 5.1. In Figure 5.2, the answers to the questions regarding the usefulness of the schema are shown. Furthermore, we examined the comments of the domain experts and tried to identify common feedback mentioned.

Question A: *Do you think that the schema is helpful for identifying Stations suitable for a given analytical task?*



Question B: *Would metadata, defined from the schema increase the transparency of the components of the PHT to you?*

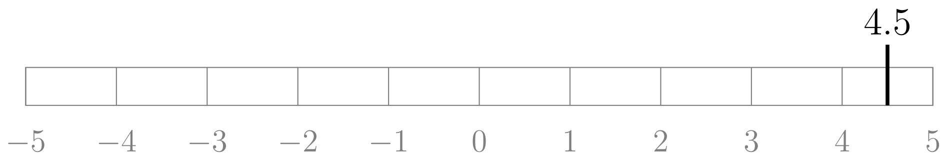
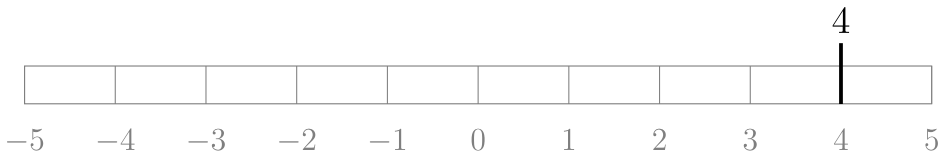
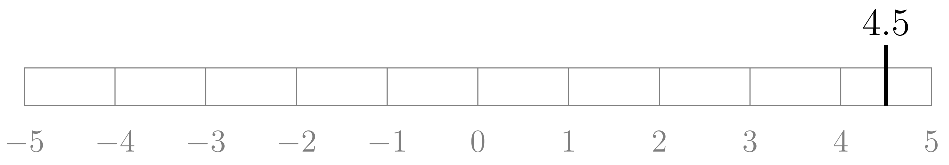


Figure 5.1: The answers to the question regarding the completeness of the schema. This includes only the questions, which could be answered on a given scale. Shown is the arithmetic average of all answered questionnaires.

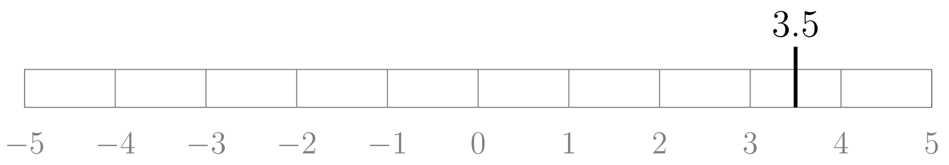
Question C: *Does the proposed partition of the metadata (Business Information, ...) makes sense to you?*



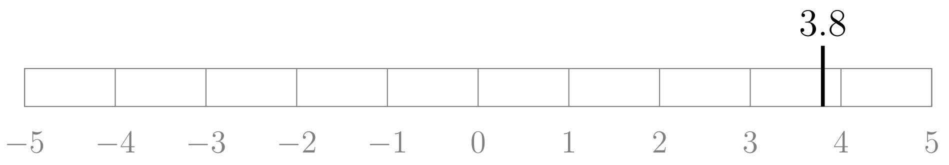
Question D: *Does the schema contain in your opinion unnecessary attributes / address unnecessary concepts?* (The higher, the less unnecessary attributes)



Question E: *Are the proposed subschemas comprehensible?*



Question F: *Is it clear what kind of information the different subschemas address?*



Question G: *Would you be able to define metadata, following the schema, for your Station/your algorithm?*

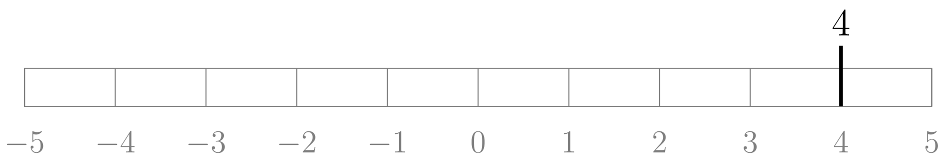


Figure 5.2: The answers to the question regarding the usefulness of the schema. This includes only the questions, which could be answered on a given scale. Shown is the arithmetic average of all answered questionnaires.

5.3 Discussion

The feedback collected from the questionnaires was overall positive. As seen in the answers to question **C** in Figure 5.2, many experts stated that they agree with the overall proposed six groups of metadata. As described in Section 3.3, we divided the metadata for each the Train and the Station into 3 groups.
These groups are Business Information, Technical Information and Dynamic Execution Information for the Train and Business Information, Runtime Environment Information and Data Information for the Station. This coincides with the conducted interviews, where multiple domain experts also state their agreement with these groups. Therefore, we can hold on to this division of metadata and can state that they clearly identify the different kind of information present in the PHT.
First, we discuss the completeness of the schema. Some optional information are given that would fit into the schema, such as attributes stating support for computational platforms. Further, it is suggested by some experts, to incorporate more ontologies into the schema, especially from the medical domain, since our approach of describing the data sets do not lay out unique defined medical vocabularies. Our approach is to denote the content of data sets with the help of SKOS.
It is possible to denote detailed descriptions of the datasets. However, SKOS, as a framework for creating controlled vocabularies, does not lay out any vocabularies itself. Therefore our schema does not dictate what vocabulary to use for describing the data. This may lead to the usage of different vocabularies for the same concepts across different instances of the PHT. Such inconsistency in vocabularies for the data set is hindering for the interoperability of Stations and Trains across different PHT instances.
But overall our approach is seen as sufficient by many experts. The incorporation of such medical ontologies is desirable for the already stated reasons but requires extensive knowledge about the medical domain and is therefore included in the Future Work.

Controversial in the feedback is, whether the schema is suitable to sufficiently describe the structure of the data sets, especially whether the two proposed types of data sets, FileDataSet and TabularDataSet cover all possible shapes of data. This is confirmed by the feedback to question **A** in Figure 5.1, which is not overall positive.
It is necessary to deliberate about the trade-off between adding support for rarely occurring data structures and the therefore increased complexity of the schema.
Data structures, which can not be described through a tabular representation or which are not in the form of simple files, such as text or image, are not very common in our opinion.
However, there are still examples, such as an RDF graph who do not just populate instances with attributes of the same kind but rather represent more complex data. To ultimately decide on the amount of required expressiveness, more research in the domain of medical data is needed.
But since the results are on an average positive, it can be assessed that the schema describes a data set sufficiently for the most application.

The need for more legal information regarding the usage of data and the possibility to lay out constraints regarding confidentiality is also stated. A part of this are more detailed information about the different roles of persons in the creation and publication process of a Train. The proposed Train schema has an attribute *creator*. Some experts see this not as sufficient to mirror the different Train stakeholders and their duties. This was not stated in the User Stories and was therefore not considered in the creation process of the metadata. Still, we see this as a valid point and useful information, therefore, it will be included in the Future Work.
Another aspect of this is the lacking of well-defined roles for users in our schema, which was criticised by some experts. This problem is also recognized by us but was not part of the intended scope of this thesis. Nevertheless it is necessary for utilizing the schema in real-world environment and will be a part of Future Work on the schema.

As a second quality we discuss the usefulness of the schema. This part of the evaluation is positively evaluated, which is reflected in the feedback to question **G**, shown in Figure 5.2.
There is an overall consistent opinion that the schema does not lay out redundant attributes. The answers to question **D** in Figure 5.2 confirms this. Stated as problematic in the feedback is the understandability of some components of the schema, since some domain experts state that they do not know to which architectural components of the PHT they correspond to. It was therefore unclear, what the purpose of some parts of the schema was. It is confirmed by the feedback to the answers **E** and **F** in Figure 5.2 which results are not constant positive.
But it became also apparent through the questionnaire, that, while the idea of the high-level structure of the PHT architecture is uniform, there are different conceptions of its detailed composition. An example of such different conceptions is that it is unclear what exactly a Train characterizes and whether it only consumes the data or if it can manipulate it some cases. The positive feedback of other experts on the understandability indicates, that the problem stated above is probably rooted in such different conceptions of details of the PHT.
When the schema is used in real-world applications, this may result in a not intended usage of some parts. This could lead to misunderstandings between participants of the PHT. Therefore, a detailed description of our conception of the architecture would be helpful, which currently is missing in the schema.

Nevertheless, even though some suggestions for improvement, many experts agreed that the metadata schema sufficiently fulfils its task of increasing the transparency of the different components of the metadata. The feedback to question **B** displayed in Figure 5.1 confirms this. An additional indication for this is the successfully implemented dashboard, which acted as a proof of concept for making metadata about the PHT more accessible. Based on the feedback, this especially holds for the Train part of the schema

This leads to the conclusion that the schema is overall useful for its purpose.

5.4 FAIR

In this section, we evaluate the metadata schema with the FAIR guidelines [32]. For a better overview, we structured the evaluation according to the different FAIR principles in Table 5.1

FAIR Principle	**Evaluation**
F1. Metadata are assigned a globally unique and persistent identifier.	This principle is fulfilled by the metadata schema. The metadata describing an object can be identified through its URI, since it is modelled in RDF. However, it has to be ensured by the user that the URI of the metadata object is unique.
F2. Data are described with rich metadata (defined by R1 below).	The principle is fulfilled by the schema, because the schema itself is the metadata describing the PHT. Therefore the PHT architecture itself also fulfils this FAIR principle with the help of the proposed schema.
F3. Metadata clearly and explicitly include the identifier of the data it describes.	This principle is fulfilled with the identifier attribute of both the Station and the Train.
F4. Metadata are registered or indexed in a searchable resource.	This is not in the scope of the metadata schema and has to be fulfilled at a later point when the schema is deployed and used. However, the metadata schema directly supports the searchability by using RDF with well-defined terms and by reusing existing Metadata schemas. Querying RDF can, for example, be tackled with SPARQL, a query language for RDF terms,

A1. Metadata are retrievable by their identifier using a standardized communication protocol.	This has to be fulfilled at a later point and is not applicable to the metadata schema alone. But since the schema utilizes RDF, it already gives a foundation for fulfilling this principle, since solutions for serving RDF fulfilling this principle exist, such as serving it over HTTP in the JSON-LD format, as done in the proof of concept for the dashboard.
A2. Metadata are accessible, even when the data are no longer available.	This has to be fulfilled at a later point and is not applicable to the metadata schema. However, since the metadata schema contains attributes for versioning, it helps fulfilling it.
I1. Metadata use a formal, accessible, shared, and broadly applicable language for knowledge representation.	This is fulfilled by the metadata schema. The schema is described in RDFS and SHACL, with the utilization of existing vocabularies and new proposed terms. This is formal since it is unambiguously defined. That it is accessible and shared has to be ensured at a later point when it is applied. However, through the good readability of TURTLE, the serialization language used, it is easily shareable. Furthermore, the possibility of SHACL to validate metadata definition, as described in Section 4.4, helps ensure consistency throughout the metadata. RDF and SHACL are both standards of the W3C and used in many different applications and are therefore broadly applicable. The reused metadata schemas, such as DataCite and the reused ontologies such as DMOP are also applicable in further domains.
I2. Metadata use vocabularies that follow FAIR principles	Because the PHT fulfils this with the help of the proposed schema, we can say that the schema itself also fulfils this. The vocabulary of the metadata schema is FAIR, as it is shown in this table. Therefore, the PHT is described with such a FAIR vocabulary.

I3. Metadata include qualified references to other (meta)data.	This is fulfilled by the metadata schema, since it refers to other metadata schemas / ontologies, such as SKOS.
R1. Metadata are richly described with a plurality of accurate and relevant attributes. R1.1 Metadata are released with a clear and accessible data usage license. R1.2 Metadata are associated with detailed provenance. R1.3 Metadata meet domain-relevant community standards.	The evaluation with the questionnaires as described in Section 5.2, suggested that the schema fulfils the principle R1. R1.1 has to be fulfilled at a later point, when the metadata schema is published. The schema fulfils R1.2, because the provenance, as in who created the Train/Station, can be detailed described with the FOAF. R1.3 is in some parts fulfilled with the reuse of existing schemas, in other parts it can not be fulfilled because the proposed schema is the first in its domain and therefore there are no community standards.

Table 5.1: The evaluation with respect to the FAIR principles. The FAIR principles are located in the left column and the evaluation for each principle in the corresponding right column. The formulation of the FAIR principles is adopted from [32].

Overall it can be stated that the proposed metadata schema fulfils those FAIR principles to which it can be applied. Furthermore, the PHT does become more FAIR, when the proposed metadata schema is used.

6 Conclusion

The goal of the thesis was to create a metadata schema for the PHT architecture. This was motivated by the black-box characteristic of the highly distributed PHT architecture faced its users. The purpose of the schema was to increase transparency and to improve the accessibility of information about the different architectural components for said users. To achieve this goal we conducted a multi-step process for designing and constructing the schema.
As discussed, we analysed the requirements for such a metadata schema by collecting user stories from experts in the metadata and DA domain. Based on them, we identified six sets of information in which the metadata can be partitioned. First, we divided the metadata into a train and a station part. On the train side, we proposed Business Information, Technical Information and Dynamic Execution Information as a suitable partition. On the station side, we divided the metadata into the groups Business Information, Technical Information and Data Information.
We created the metadata schema by proposing attributes which expressed the information required in the user stories. Each attribute consists of an identifier, a cardinality and a suitable datatype. During this, we reused existing metadata schemas and ontologies by incorporating parts of those into the schema. To enable machine readability of the schema and to define it in a standardized format, we modelled the schema in RDF with the help of SHACL and RDFS. As a proof of concept that the metadata increases the transparency, we implemented a dashboard, consuming the metadata from a mock server, processing and visualizing it.
To evaluate the work done, we created a questionnaire and distributed it to members of the metadata and DA research community. The feedback included some critique and improvement suggestions for the part of the schema which describes the data of Stations. Further, some experts criticized the understandability of the schema. The overall evaluation was positive and indicated that the schema sufficiently describes the components of the PHT architecture. As an additional evaluation, we checked whether the metadata schema fulfils the FAIR principles. The result was that the metadata schema accords to those FAIR principles, which can be applied to it. An additional finding was that the PHT architecture itself becomes more FAIR when the schema is utilized.
Based on the evaluation, we conclude that the proposed metadata schema fulfils the goal of increasing the transparency of the PHT architecture. We can expect, that through the schema the users have an increased overview and therefore an increased efficiency. We can also state, that the PHT becomes more FAIR, because the schema increases the Findability, Accessibility, Interoperability and Reusability of the components.

Furthermore, the proposed metadata schema is a novel approach in the DA domain and can, therefore, act as a foundation for future developments.
However, while the metadata schema was evaluated as sufficient to describe the PHT architecture, there are still things to improve and to add to the schema.
We can further state that the proposed process, through which the metadata schema was created, is suitable.

6.1 Future Work

Additional work can be done in the description of the data at stations. It has to be reviewed, whether expanding the schema yields an additional value when describing data sets. For this, it would be reasonable to examine what are the most common data structures, e.g. in the medical domain, and evaluate how the schema performs with describing those.

Since the PHT ecosystem is applied to medical use cases, the schema can also be improved by adding more ontologies, especially from the medical data domain. Currently, the semantic part of the data sets are described by referring to vocabularies based on SKOS. However, concrete vocabularies for describing medical data are not mandated through the schema. Describing data sets with unambiguously medical ontologies would further increase the interoperability, especially across different instances of the PHT architecture. Therefore, research needs to be done whether suitable ontologies exist and if so which one is reasonable to use.

Even if the metadata provides an accurate description of the trains and the stations, it cannot ensure that given information is true. Participants of the architecture with malicious intent could provide false descriptions about their components. For example, one could provide an description of past executions of a train that never happened, to pretend that the train is trustworthy. A mechanism for ensuring the validity of given provenance information in the metadata would be therefore desirable and more research in this domain would be worthwhile for the metadata schema. Such a mechanism could, for example, be cryptographic.

The next step after developing the metadata schema is the real-world application of it. For this, implementing a metadata repository is reasonable. This repository should be able to store metadata created from the schema. Additionally, it should provide interfaces to interact with the data, such as *SPARQL*, a query language for RDF [60], and should also automatically update the dynamic execution information for trains, to ensure that the metadata about the different components is always up-to-date. Further applications, such as an automatic matching of trains and stations or a user-friendly search interface for the different components are also feasible.
However, at the moment, the metadata schema does not include any description of access control to the metadata. In a real-world application of the schema, it would

be reasonable to constrain the access to metadata based on user roles. This could ensure, that sensitive information, such as the location of data sets on the station, is not disclosed to unauthorized users. Therefore, embedding such different roles into the schema and denote for each attribute which roles are allowed access it, maybe necessary before implementing the schema into a repository.

Bibliography

[1] O. Can. and B. Usenmez., "An ontology based personalized privacy preservation," in *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 2: KEOD,*, pp. 500–507, SciTePress, 2019.

[2] T. J. Barnes, "Big data, little history," *Dialogues in Human Geography*, vol. 3, no. 3, pp. 297–302, 2013.

[3] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 1, pp. 97–107, 2013.

[4] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile networks and applications*, vol. 19, no. 2, pp. 171–209, 2014.

[5] T. B. Murdoch and A. S. Detsky, "The Inevitable Application of Big Data to Health Care," *The Journal of the American Medical Association*, vol. 309, pp. 1351–1352, 04 2013.

[6] N. Mehta and A. Pandit, "Concurrence of big data analytics and healthcare: A systematic review," *International journal of medical informatics*, vol. 114, pp. 57–65, 2018.

[7] W. N. Price and I. G. Cohen, "Privacy in the age of medical big data," *Nature medicine*, vol. 25, no. 1, pp. 37–43, 2019.

[8] V. Palanisamy and R. Thirunavukarasu, "Implications of big data analytics in developing healthcare frameworks–a review," *Journal of King Saud University-Computer and Information Sciences*, vol. 31, no. 4, pp. 415–425, 2019.

[9] H. C. Koh, G. Tan, *et al.*, "Data mining applications in healthcare," *Journal of healthcare information management*, vol. 19, no. 2, p. 65, 2011.

[10] H. K. Patil and R. Seshadri, "Big data security and privacy issues in healthcare," in *2014 IEEE international congress on big data*, pp. 762–765, 2014.

[11] D. Wegener, S. Rossi, F. Buffa, M. Delorenzi, and S. Rüping, "Towards an environment for data mining based analysis processes in bioinformatics and personalized medicine," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 2, no. 1, pp. 29–44, 2013.

[12] A. J. Kucharski, T. W. Russell, C. Diamond, Y. Liu, J. Edmunds, S. Funk, R. M. Eggo, F. Sun, M. Jit, J. D. Munday, *et al.*, "Early dynamics of transmission and control of covid-19: a mathematical modelling study," *The lancet infectious diseases*, 2020.

[13] W. Raghupathi and V. Raghupathi, "Big data analytics in healthcare: promise and potential," *Health information science and systems*, vol. 2, no. 1, p. 3, 2014.

[14] N. Serenko and L. Fan, "Patients' perceptions of privacy and their outcomes in healthcare," *International Journal of Behavioural and Healthcare Research*, vol. 4, no. 2, pp. 101–122, 2013.

[15] C.-A. Azencott, "Machine learning and genomics: precision medicine versus patient privacy," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 376, no. 2128, p. 20170350, 2018.

[16] P. S. Appelbaum, "Privacy in psychiatric treatment: threats and responses," *American Journal of Psychiatry*, vol. 159, no. 11, pp. 1809–1818, 2002.

[17] K. J. Cios and G. W. Moore, "Uniqueness of medical data mining," *Artificial intelligence in medicine*, vol. 26, no. 1-2, pp. 1–24, 2002.

[18] S. Banerjee, T. Hemphill, and P. Longstreet, "Wearable devices and healthcare: Data sharing and privacy," *The Information Society*, vol. 34, pp. 1–9, 12 2017.

[19] dejure.org Rechtsinformationssysteme GmbH. `https://dejure.org/gesetze/DSGVO/9.html`.

[20] M. Meingast, T. Roosta, and S. Sastry, "Security and privacy issues with health care information technology," in *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5453–5458, 2006.

[21] D. Peteiro-Barral and al., "A survey of methods for distributed machine learning," *Progress in Artificial Intelligence*, vol. 2, no. 1, pp. 1–11, 2013.

[22] G. Tsoumakas and I. Vlahavas, "Distributed data mining," in *Database technologies: Concepts, methodologies, tools, and applications*, pp. 157–164, IGI Global, 2009.

[23] O. Beyan et al, "Distributed analytics on sensitive medical data: The personal health train," *Data Intelligence*, vol. 2, no. 1-2, pp. 96–107, 2020.

[24] Z. Shi, I. Zhovannik, A. Traverso, F. J. Dankers, T. M. Deist, P. Kalendralis, R. Monshouwer, J. Bussink, R. Fijten, H. J. Aerts, *et al.*, "Distributed radiomics as a signature validation study using the personal health train infrastructure," *Scientific data*, vol. 6, no. 1, pp. 1–8, 2019.

[25] A. Jochems, T. M. Deist, J. Van Soest, M. Eble, P. Bulens, P. Coucke, W. Dries, P. Lambin, and A. Dekker, "Distributed learning: developing a predictive model based on data from multiple hospitals without data leaving the hospital–a real life proof of concept," *Radiotherapy and Oncology*, vol. 121, no. 3, pp. 459–467, 2016.

[26] C. Sun, L. Ippel, J. Van Soest, B. Wouters, A. Malic, O. Adekunle, B. van den Berg, O. Mussmann, A. Koster, C. van der Kallen, *et al.*, "A privacy-preserving infrastructure for analyzing personal health data in a vertically partitioned scenario.," in *MedInfo*, pp. 373–377, 2019.

[27] C. Mader, J. Pullmann, N. Petersen, S. Lohmann, C. Lange-Bever, Fraunhofer IAIS/EIS, Fraunhofer FIT, Fraunhofer IAIS/EIS, Fraunhofer IAIS/EIS, and Fraunhofer IAIS/EIS, "Industrial data space information model." `http://ids.semantic-interoperability.org`, 2018. Accessed: 06.04.2020.

[28] "The codemeta project." `https://codemeta.github.io/`.

[29] E. Duval, W. Hodgins, S. Sutton, and S. L. Weibel, "Metadata principles and practicalities," *D-lib Magazine*, vol. 8, no. 4, pp. 1082–9873, 2002.

[30] "Exchangeable image file format for digital still cameras:exif version 2.32," 2019.

[31] D. M. W. Group *et al.*, "Datacite metadata schema documentation for the publication and citation of research data," *DOI: https://doi. org/10.5438/0012 Version*, vol. 4, 2016.

[32] M. D. Wilkinson et al, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, 2016.

[33] GrafanaLabs, "Grafana features." `https://grafana.com/grafana/`.

[34] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.

[35] A. Allen, "Privacy and Medicine," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, winter 2016 ed., 2016.

[36] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[37] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[38] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "Fedhealth: A federated transfer learning framework for wearable healthcare," *IEEE Intelligent Systems*, 2020.

[39] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas, "Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation," in *International MICCAI Brainlesion Workshop*, pp. 92–104, 2018.

[40] I. D. S. Association, "About the international data space." `https://www.internationaldataspaces.org/our-approach/`.

[41] D. L. McGuinness, F. Van Harmelen, *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[42] D. Brickley, R. V. Guha, and B. McBride, "Rdf schema 1.1," *W3C recommendation*, vol. 25, pp. 2004–2014, 2014.

[43] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, W. W. W. Consortium, *et al.*, "Xml schema part 1: Structures," *W3C recommendation, May*, vol. 21, 2001.

[44] H. Knublauch and D. Kontokostas, "Shapes constraint language (shacl)," *W3C Candidate Recommendation*, vol. 11, no. 8, 2017.

[45] DataCite, "Datacite's value." `https://datacite.org/value.html`.

[46] P. F. Patel-Schneider, "Analyzing schema. org," in *International Semantic Web Conference*, pp. 261–276, Springer, 2014.

[47] R. V. Guha, D. Brickley, and S. Macbeth, "Schema. org: evolution of structured data on the web," *Communications of the ACM*, vol. 59, no. 2, pp. 44–51, 2016.

[48] "Schema.org section: health-lifesci." `https://schema.org/docs/health-lifesci.home.html`.

[49] J. Malone et al, "The software ontology (swo): a resource for reproducibility in biomedical data analysis, curation and digital preservation," *Journal of biomedical semantics*, vol. 5, no. 1, p. 25, 2014.

[50] C. M. Keet, A. Ławrynowicz, C. d'Amato, A. Kalousis, P. Nguyen, R. Palma, R. Stevens, and M. Hilario, "The data mining optimization ontology," *Journal of web semantics*, vol. 32, pp. 43–53, 2015.

[51] F. Maali, J. Erickson, and P. Archer, "Data catalog vocabulary (dcat)," *W3c recommendation*, vol. 16, 2014.

[52] B. Mons, C. Neylon, J. Velterop, M. Dumontier, L. O. B. da Silva Santos, and M. D. Wilkinson, "Cloudy, increasingly fair; revisiting the fair data guiding principles for the european open science cloud," *Information Services & Use*, vol. 37, no. 1, pp. 49–56, 2017.

[53] M. Cohn, *User stories applied: For agile software development.* Addison-Wesley Professional, 2004.

[54] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper, "The use and effectiveness of user stories in practice," in *International working conference on requirements engineering: Foundation for software quality*, pp. 205–222, 2016.

[55] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper, "Improving agile requirements: the quality user story framework and tool," *Requirements Engineering*, vol. 21, no. 3, pp. 383–403, 2016.

[56] M. Derntl, S. Erdtmann, and R. Klamma, "An embeddable dashboard for widget-based visual analytics on scientific communities," in *Proceedings of the 12th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '12, (New York, NY, USA), Association for Computing Machinery, 2012.

[57] T. Berners-Lee, R. Fielding, L. Masinter, *et al.*, "Uniform resource identifiers (uri): Generic syntax," 1998.

[58] T. Bray, "Internet media type registration, consistency of use," *World Wide Web Consortium. http://www. w3. org/2001/tag/2002/0129-mime*, 2004.

[59] GrafanaLabs, "Grafana api reference." `https://grafana.com/docs/grafana/latest/packages_api/data/`.

[60] E. PrudHommeaux, "Sparql query language for rdf," *http://www.w3.org/TR/rdf-sparql-query/*, 2008.

A User Stories

These is the list of user stories that we collected through the interviews with domain experts. There are overall 89 user stories.

- As a Train User, I want to know who the owner is, so that I can decide if I will use that station
- As a Train User, I want to know how to access the data in a station, so that I can ensure that my Train will work with this station
- As a Train User, I want to know what the policies are for accessing data in a station, so that I can ensure that I meet those policies
- As a Train User, I want to have descriptions of the data sets (maybe multiple) in a station, so that I can decide if the data is useful for me
- As a Train User, I want to differ in the Metadata between Train and underlying model, so that I can properly reuse models
- As a Train User, I want to have information on the requirements and licences of each models, so that I can meet all requirements, because models can have different requirements and licenses
- As a Train User, I want to have information on Trains about the used models, so that I can understand that Train
- As a Train User, I want to have information on the owner of the train and the models, so that I can distinguish between these too
- As a Train Operator, I want to have information about the current station of a train, so that I can assure that the Train works correctly
- As a Train Operator, I want to have information after the execution of the Train, if the execution at all stations was successful, so that I know that the Train worked correctly
- As a Train Operator, I want to know after the execution if the train did follow the route, so that I know that the Train worked correctly
- As a Train Operator, I want to have information about the memory, cpu and storage consumption and the execution duration of the train, so that I can assess the performance of my train and know how I can improve my Train
- As a Train Operator, I want to have an overview on what data my train accessed at which point in time, so that I can check if my train worked correctly and how it performed
- As a Train User, I want enough information on a Train execution to reproduce the results, so that I can reproduce results and check them
- As a Train User, I want information about the structure of the data a Station provides, so that I can decide if I can use that Station

- As a Station Owner, I want to decide how depth the description of my data is, so that I can preserve privacy of the Data
- As a Train User, I want that the policies of the stations are open accessible, so that I can decide which Station my Train will visit
- As a Train User, I want to have information about the computational environment a station provides, so that I can decide if the Station will be capable of running my model
- As a Train User, I want to have information about how do access the data in a station, so that I can ensure that my Train will be able to access the data
- As a Train User, I want to have information about what kind of technologies the station supports, so that I can ensure that the Station will be capable of running my model
- As a Train User, I want to have information about the maximal model size a station supports, so that I can ensure that the Station will be capable of running my model
- As a Train User, I want to have information about how many models can be run simultaneously at a station, so that I can ensure that the Station will be capable of running my model
- As a Train User, I want to have a description of the data a station provides, so that I can decide if that data is useful for me
- As a Train User, I want to have information if the data is anonymized and if yes with what kind of technology, so that I can consider that in my analysis
- As a Station Owner, I want to lay out conditions how the data must be anonymized by the trains, so that I can ensure that the privacy is protected
- As a Station Owner, I want to know when a train will visit my station, so that I can make preparations if necessary
- As a Station Owner, I want to know the route of each Train which visits my station, so that I know what goal of the Train is
- As a Station Owner, I want to know how the train did perform before it comes to my station, so that I can estimate how the train will perform at my station
- As a general User of the PHT, I want a good versioning system of my trains, so that I can reproduce results
- As a general User of the PHT, I want that data access is time stamped, so that I can reproduce results or can understand why I get different results because of changing data
- As a Train User, I want to know what the purpose of the Train is, so that information does not get lost over time
- As a general User of the PHT, I want to have mandatory information about the purpose of each execution of a train, so that this information is for all participating parties in an analysing task accessible
- As a Train User, I want to trace the location of my train at any given time, so that I am able to understand my trains execution
- As a Train User, I want to lay out information how the train should react if there is failure, so that I don't have to intervene the execution

- As a Train User, I want to have information about the quality of data at a station in the form of ranges, so that I can decide if I want to use that data
- As a Train User, I want to have information about the quality of the output data of a train, so that I can decide if I want to use that train
- As a Train User, I want to have information about the pre processing steps a train performs, so that I know what a Train does
- As a overall PHT User, I want that the Metadata helps to match trains to suitable stations, so that I can perform that faster
- As a Train Operator, I want that the execution history of my train is not visible to everybody, so that my intentions and privacy are kept secret
- As a Station Owner, I would like to access Information about past executions of the train so that I know what the purpose of the train, which deals with my data, is
- As a Train Operator, I want to decide how many telemetric data the train send to me so that no data is send which is not needed and the telemetric data is suitable for the train
- As a Train Operator, I want to have Information which stations my train visited and if there were problems at any station, and if yes, also what kind of problems
- As a Train User, I want to have Information about the dispatching pattern about a train so that I can decide if I want to reuse it
- As a Train User, I want to have detailed Information about the Workflow the train incorporates, so that I can decide if I want to reuse it and so that I can understand that train better
- As a Train User, I want to have Information with what kind of data the train interacts so that I can search for trains suitable for the data that I have access to
- As a Train User, I want to have Information about the kind of output the train generates and if it generates an output at all, so that I can decide if I want to reuse it
- As a Train User, I want to know what the purpose of the Train is, so that I can decide if I want to reuse it
- As a Train User, I want to know what kind of Interaction Pattern the Train Uses so that I can check if it works with the Stations I want it to visit
- As a Train User, I want to know who the author of the Train is, so that i can decided whether i can trust the Train
- As a Train User, I want to know if the Train is certificated, so that i can decided whether i can trust the Train
- As a Train User, I want to know who the owner of a station is and under which licence that Station can be accessed so that i can decide if I will let the Train visit that station
- As a Train User, I want to have a general description of the computing capabilites of a station, so that I can decide if the station will be able to run my train
- As a Train User, I want to have Information about the computational mechanisms a station provides, so that I can decide if the station will be able to run my train or if I need to tweak my train

- As a Train User, I want to have Information about the geographical location of a Station so that I know if I have to consider national regulations
- As a Train User, I want to have Information about the general purpose of the station so that I can know if the Station is useful for me
- As a Train User, I want to have Information about whether some agreements have to be made before I can use that Station so that I can do this
- As a Train User, I want to have Information about the size of the data a station provides, so that I can decide if the Station is useful for me
- As a Train User, I want to have Information about the Format of the data a station provides, so that I can decide if my train will be able to access the data or if I need to tweak my train
- As a Train User, I want to have a semantic description of the data, so that I can understand if I can use the data
- As a Train User, I want to have Information about the provenance of Data, so that I can assess the quality and usefulness of the data
- As a Train User, I want to have Information if I am entitled to access data at a Station and if yes, under what constraints
- As a Train Dispatcher, I want to have different Metadata schemes for static “blueprints” of train and concrete instances, because of the different key aspects in the needed Metadata
- As a general User of the PHT, I want to differ between a train and the underlying program/algorithm so that the program can be easily reused across different trains
- As a Data Scientist, I want to have Information on the Computational Environment of a station, so that i can identify which stations are suitable for the execution of a programm
- As a Data Scientist, I want to have Information on the system architecture of the station, so that i can better develop suitable programs/algorithms
- As a Data Scientist, I want to have a list of available systems at a station, so that i can better develop suitable programs/algorithms and can decide which station are suitable for my programs
- As a Station Owner, I want only give information on the performance capabilities of my station in a categorical manner,so that I avoid giving away too much information about my infrastructure
- As a Train Dispatcher/Data Scientist, I want to have a contact person for each Station in the Metadata, so that I can contact him/her if issues arise
- As a Data Scientist, I want to have a possibility to be informed if new data is available at a station, so that I always know which station hold information relevant to my research
- As a Data Scientist, I want to have detailed information addressing the data format of the data at each station so that I know how I can access this data with my program
- As a Data Scientist, I want to have a record of memory usage and run time, so that I can examine the performance of my train and tweak the train if necessary

- As a Data Scientist, I want to have access to live telemetric data, so that I can inspect how the execution of my train makes progress
- As a Data Scientist,I want to have enough detailed Information on the Trains to decide if I can reuse the train, so that I can reuse Trains and do not need to re-implement existing algorithm
- As a general participant of the PHT, I want information about the author of a train, so that there is increased transparency on the train
- As a general participant of the PHT, I want information about the creation date of a train, so that there is increased transparency on the train
- As a general participant of the PHT, I want information about the algorithm that a train uses, so that i can trust the train and understand the train better
- As a general participant of the PHT, I want an open accessible, detailed record of each execution of a train, so that I have a way of assessing different trains
- As a general participant of the PHT, I want to have open accessible performance metrics for each train so that I can assess the performance of trains
- As a Data Scientist, I want that the trains are detached of the stations and can be used on arbitrary (compatible) station, so that I can reuse trains
- As a Data Scientist, I want that everyone can execute any train in a Repo, so that I can reuse trains
- As a Data Scientist, I want rich information about what kind of data is available at stations, so that I can easily find stations that I need
- As a station owner, I want to have information about what trains are at the station at the moment, so that there is increased transparency and checkability over the station
- As a station owner, I want to have information about what trains are planning to visit my station, so that I have better control and overview over the station
- As a station owner, I want to have information about what happens with his data in that moment, so that I have better control and overview over the station He would like to distinguish between train and underlying programs to...
- As a PHT user, I want to have information on the current position of my train so that there is a increased traceability my train and to enable me to take action if the execution is not continuing
- As a PHT user, I want to have information in general which help completing the analysis task, so that I can reduce the time an analysing task needs
- As a Data Scientist, I would like to have the possibility to lay out constraints that an aggregating station is only visited if a minimum of parallel stations could be successful visited, so that i can ensure that enough data was processed and the train execution is worthwhile

B User Stories Tables

Prefixes

FOAF:	The Friend Of a Friend Ontology. `http://xmlns.com/foaf/spec/`
DMOP:	The Data Mining Optimization Ontology. `http://www.e-lico.eu/ontologies/dmo/DMOP/DMKB.owl`
SWO:	The Software Ontology. `http://theswo.sourceforge.net/`
SKOS:	The Simple Knowledge Organization System. `http://www.w3.org/2004/02/skos/core#`

B.1 Train - Business

User Stories	**Attributes**
• As a Train User, I want to have information on the owner of the train and the models, so that I can distinguish between these two • As a Train User, I want to know who the author of the Train is, so that i can decided whether i can trust the Train • As a general participant of the PHT, I want information about the author of a train, so that there is increased transparency on the train	`creator: FOAF:Agent (1..n)`

• As a general User of the PHT, I want a good versioning system of my trains, so that I can reproduce results	`version: String (1)`

• As a Train User, I want to know what the purpose of the Train is, so that information does not get lost over time
• As a Train User, I want to know what the purpose of the Train is, so that I can decide if I want to reuse it
	`description: String (1..n)`
	`title: String (1)`

• As a Train User, I want to know if the Train is certificated, so that i can decided whether i can trust the Train	`Certificate: pht:Certificate (1)`
	`Certificate.data: String (1)`
	`Certificate.issuer: String (1)`
	`Certificate.begins: Date (1)`
	`Certificate.expires: Date (1)`

• As a Train Dispatcher, I want to have different Metadata schemes for static “blueprints” of train and concrete instances, because of the different key aspects in the needed Metadata	In the Metadata Schema it will be differentiated between a Train (static) and a Train execution (dynamic)
• As a general participant of the PHT, I want information about the creation date of a train, so that there is increased transparency on the train	`published: Date (1)`

• As a Data Scientist, I want to have a identification mechanism for the train	`id: URL (1)`

B.2 Train - Technical

User Stories	Attributes
• As a Train User, I want to differ in the Metadata between Train and underlying model, so that I can properly reuse models • As a general User of the PHT, I want to differ between a train and the underlying program/algorithm so that the program can be easily reused across different trains	`model: Model (1..n)`
• As a general User of the PHT, I want a good versioning system of my trains, so that I can reproduce results	`version: String (1)`
• As a Train User, I want to have information on the requirements and licences of each models, so that I can meet all requirements, because models can have different requirements and licenses	`Model.license: SWO:license (0..1)`
• As a Train User, I want to have information on Trains about the used models, so that I can understand that Train	`Model.description: String (1..n)`

• As a Train User, I want to have information about the quality of the output data of a train, so that I can decide if I want to use that train • As a Train User, I want to have Information about the kind of output the train generates and if it generates an output at all, so that I can decide if I want to reuse it	`Model.characteristic: DMOP:AlgorithmCharacteristic (0..n)`
• As a Train User, I want to have information about the pre processing steps a train performs, so that I know what a Train does	`Model.preProcessingAlgorithm: DMOP:Algorithm (0..n)`

• As a Train User, I want to have Information about the dispatching pattern about a train so that I can decide if I want to reuse it	`dispatchingPattern: TrainDispatchingPattern (0..1)`
	
	`TrainDispatchingPattern` as a class, with individuals for different dispatching patterns:
	`TrainDispatchingPatternParallel`
	`TrainDispatchingPatternLinear`
	`TrainDispatchingPatternMixed`

• As a Train User, I want to have detailed Information about the Workflow the train incorporates, so that I can decide if I want to reuse it and so that I can understand that train better
• As a general participant of the PHT, I want information about the algorithm that a train uses, so that i can trust the train and understand the train better
	`Model.algorithm: DMOP:Algorithm (1..n)`
	`Model.dataInteractionRead: boolean (0...1)`
	`Model.dataInteractionWrite: boolean (0...1)`
	`Model.dataInteractionDelete: boolean (0...1)`

• As a Train User, I want to have Information with what kind of data the train interacts so that I can search for trains suitable for the data that I have access to • As a Train User, I want to know what kind of Interaction Pattern the Train Uses so that I can check if it works with the Stations I want it to visit	`Model.expectedDataSet: pht:ExpectedDataSet (0..n)`
• As a general participant of the PHT, I want to have open accessible performance metrics for each train so that I can assess the performance of trains	`Model.minimumEstimatedGFLOPS: float (0..n)`

B.3 Train - Dynamic execution

User Stories	Attributes
• As a general User of the PHT, I want to have mandatory information about the purpose of each execution of a train, so that this information is for all participating parties in an analysing task accessible	`TrainExecution.description: String (1..n)` `TrainExecution.creator: FOAF:Agent (1)` `TrainExecution.id: URL (1)`
• As a Train Operator, I want to decide how many telemetric data the train send to me so that no data is send which is not needed and the telemetric data is suitable for the train	`TrainExecution.enableEvent: ExecutionEvent(0..n)`

• As a Train Operator, I want to have information after the execution of the Train, if the execution at all stations was successful, so that I know that the Train worked correctly
• As a Train Operator, I want to know after the execution if the train did follow the route, so that I know that the Train worked correctly
• As a Train Operator, I want to have Information which stations my train visited and if there were problems at any station, and if yes, also what kind of problems
• As a Station Owner, I want to know how the train did perform before it comes to my station, so that I can estimate how the train will perform at my station
• As a Station Owner, I would like to access Information about past executions of the train so that I know what the purpose of the train, which deals with my data, is
• As a Train User, I want to trace the location of my train at any given time, so that I am able to understand my trains execution
• As a PHT user, I want to have information on the current position of my train so that there is a increased traceability my train and to enable me to take action if the execution is not continuing
	`TrainExecution.state: StateEnum (1)`
	`TrainExecution.events: ExecutionEvent (0..n)`

• As a Train User, I want to have a record of memory usage and run time, so that I can examine the performance of my train and tweak the train if necessary • As a Train User, I want to have access to live telemetric data, so that I can inspect how the execution of my train makes progress • As a general participant of the PHT, I want an open accessible, detailed record of each execution of a train, so that I have a way of assessing different trains • As a Train Operator, I want to know after the execution if the train did follow the route, so that I know that the Train worked correctly	(Continuation of the last row)
• As a Station Owner, I want to know when a train will visit my station, so that I can make preparations if necessary • As a Station Owner, I want to know the route of each Train which visits my station, so that I know what goal of the Train is • As a station owner, I want to have information about what trains are planning to visit my station, so that I have better control and overview over the station	`TrainExecution.plannedRoute: ExecutionPlanStep(1..n)`

• As a Train user, I want that the execution history of my train is not visible to everybody, so that my intentions and privacy are kept secret	`TrainExecution.public: boolean (1)`

B.4 Station - Business

User Stories	**Attributes**
• As a Train User, I want to know who the owner is, so that I can decide if I will use that station • As a Train User, I want to know who the owner of a station is and under which licence that Station can be accessed so that i can decide if I will let the Train visit that station	`creator: FOAF:Agent (1..n)`
• As a Train Dispatcher/Data Scientist, I want to have a contact person for each Station in the Metadata, so that I can contact him/her if issues arise	`responsibleForStation: FOAF:Agent (1)`

• As a Train User, I want to have Information about the geographical location of a Station so that I know if I have to consider national regulations	`longitude: float (1..n)`
	`latitude: float (1..n)`

• As a Train User, I want to have Information about the general purpose of the station so that I can know if the Station is useful for me	`description: String (1)`

B.5 Station - Runtime Environment

User Stories	**Attributes**

• As a Train User, I want to have information about the computational environment a station provides, so that I can decide if the Station will be capable of running my model
• As a Train User, I want to have Information about the computational mechanisms a station provides, so that I can decide if the station will be able to run my train or if I need to tweak my train
• As a Data Scientist, I want to have Information on the Computational Environment of a station, so that i can identify which stations are suitable for the execution of a programm
• As a Train User, I want to have a general description of the computing capabilities of a station, so that I can decide if the station will be able to run my train
• As a Station Owner, I want only give information on the performance capabilities of my station in a categorical manner,so that I avoid giving away too much information about my infrastructure
	`ComputationalEnvironment.estimated-GFLOPS: Float (0...n)`
	`ComputationalEnvironment.hasCUDA-Support:  boolean (0..n)`

• As a Train User, I want to have information about what kind of technologies the station supports, so that I can ensure that the Station will be capable of running my model • As a Data Scientist, I want to have a list of available systems at a station, so that i can better develop suitable programs/algorithms and can decide which station are suitable for my programs	`ComputationalEnvironment .programmingLanguageSupport: SWO:ProgrammingLanguage (0..n)`
• As a Train User, I want to have information about how many models can be run simultaneously at a station, so that I can ensure that the Station will be capable of running my model	`ComputationalEnvironment .maximumNumberOfModelsSupported: Integer (0..1)`
• As a Train User, I want to have information about the maximal model size a station supports, so that I can ensure that the Station will be capable of running my model	`ComputationalEnvironment .maximumModelSizeKilobytesSupported: Integer (0..1)`

B.6 Station - Data Information

User Stories	Attributes

• As a Train User, I want to know how to access the data in a station, so that I can ensure that my Train will work with this station
• As a Train User, I want to have information about how do access the data in a station, so that I can ensure that my Train will be able to access the data
• As a Train User, I want to have Information about the Format of the data a station provides, so that I can decide if my train will be able to access the data or if I need to tweak my train
• As a Data Scientist, I want to have detailed information addressing the data format of the data at each station so that I know how I can access this data with my program
• As a Train User, I want information about the structure of the data a Station provides, so that I can decide if I can use that Station
• As a Train User, I want to have a description of the data a station provides, so that I can decide if that data is useful for me
	`DataSet.accessURL: String (0..n)`
	
	`TabularDataSet.dataFormat: DMOP:DataFormat (0..1)`
	`TabularDataSet.attribute: TabularDataSetAttribute (0..n)`
	`TabularDataAttribute.dataType: DMOP:DataType (0..1)`
	`TabularDataAttribute.key: String (0..1)`
	`TabularDataAttribute.description: String (0..n)`
	
	`FileDataSet.fileType: String (0..1)`

• As a Train User, I want to know what the policies are for accessing data in a station, so that I can ensure that I meet those policies
• As a Train User, I want to have Information about whether some agreements have to be made before I can use that Station so that I can do this
	`DataSet.accessConstrain: AccessConstrainEnum (1)`
	`DataSet.license: SWO:license (0..1)`

• As a Train User, I want to have descriptions of the data sets (maybe multiple) in a station, so that I can decide if the data is useful for me
• As a Train User, I want to have a semantic description of the data, so that I can understand if I can use the data
	`DataSet.description: String (0..1)`
	`DataSet.theme: SKOS:Concept (0..n)`

• As a Train User, I want to have information if the data is anonymized and if yes with what kind of technology, so that I can consider that in my analysis	`DataSet.anomyzationMethodUsed: DifferentrialPrivacyMethod (0..n)`

• As a Train User, I want to have information about the quality of data at a station in the form of ranges, so that I can decide if I want to use that data • As a Train User, I want to have Information about the size of the data a station provides, so that I can decide if the Station is useful for me	`DataSet.charateristic: DMOP:DataSetCharacteristic (0..n)`
• As a Train User, I want that the policies of the stations are open accessible, so that I can decide which Station my Train will visit • As a Train User, I want to have Information if I am entitled to access data at a Station and if yes, under what constraints	`DataSet.right:  String (0..n)`

C Metadata Schema

Afterwards, the SHACL definitions are displayed. The RDFS file is not listed, since it is more a technical necessity and do not lay out any more information than the SHACL files.

C.1 Station SHACL definition

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix dmop: <http://www.e-lico.eu/ontologies/dmo/DMOP/DMKB.owl
    > .

#The prefix for PHT specific data; this should be changed later
    on, because i dont know who owns the domain
@prefix pht: <http://www.personalhealthtrainmetadata.org/#> .

pht:StationShape
    rdf:type sh:NodeShape ;
    sh:targetClass pht:Station ;
    sh:description "A station providing data for the PHT" ;
    sh:property [
        sh:path pht:creator ;
        sh:name "Train Creator" ;
        sh:class foaf:Agent ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:responsibleForStation ;
        sh:name "Responsible for the station" ;
        sh:class foaf:Agent ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:certificate ;
        sh:name "Certification" ;
        sh:description "One or more certificates providing
            additional trust in the station" ;
        sh:class pht:Certificate ;
    ] ;
    sh:property [
        sh:path pht:description ;
        sh:name "Station Description" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:title ;
        sh:name "Station Title" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:right ;
        sh:name "Station Rights" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:longitude ;
        sh:name "The Longitude of the coordinate of the station"
             ;
        sh:datatype xsd:float ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:latitude ;
        sh:name "The Latitude of the coordinate of the station"
            ;
        sh:datatype xsd:float ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:computationalEnvironment ;
        sh:name "Computational Environment" ;
        sh:class pht:ComputationalEnvironment;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:dataSet ;
        sh:name "Data Set" ;
        sh:or (
            [
                sh:class pht:TabularDataSet ;
            ]
            [
                sh:class pht:FileDataSet ;
            ]
        ) ;
        sh:minCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

# ExecutionEnvironment

pht:ExecutionEnvironmentShape
    rdf:type sh:NodeShape ;
    sh:targetClass pht:ExecutionEnvironment ;
    sh:description "An execution environment provided by a
       station" ;
    sh:property [
        sh:path pht:estimatedGFLOPS ;
        sh:name "The computation power of the environment in
           GLFOPS (estimation)" ;
        sh:datatype xsd:float ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:supportsOCI ;
        sh:name "States whether the environment allows the
           execution of OCI container" ;
        sh:datatype xsd:boolean ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:programmingLanguageSupport ;
        sh:description "States which programming languages are
           supported by the environment; the programming
           languages are expressed with the subclasses of
           programming languages from the software ontology" ;
        sh:class <http://purl.obolibrary.org/obo/IAO_0000025>;
        sh:name "Supported Programming languages" ;
    ] ;
    sh:property [
        sh:path pht:hasCUDASupport ;
        sh:name "States whether the environment allows the
           execution CUDA acclerated software" ;
        sh:datatype xsd:boolean ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:maximumNumberOfModelsSupported ;
        sh:name "States the maximum number of supported models"
           ;
        sh:datatype xsd:integer ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:maximumModelSizeKilobytesSupported ;
        sh:name "The maximum size of a model supported" ;
        sh:datatype xsd:integer ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

# DataSet

pht:DataSetShape
    rdf:type sh:NodeShape ;
    sh:description "A data set provided by a station" ;
    sh:property [
        sh:path pht:identifierToStation ;
        sh:name "Identifies the data set" ;
        sh:maxCount 1 ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:description ;
        sh:name "Station Description" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:theme ;
        sh:name "Theme of the dataset" ;
        sh:class skos:Concept ;
    ] ;
    sh:property [
        sh:path pht:license ;
        sh:name "License" ;
        # SWO has a weird naming scheme:
        sh:class <http://www.ebi.ac.uk/swo/SWO_0000002>;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:right ;
        sh:name "Legal information about the station" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:accessURL ;
        sh:name "URL through which the data can be accessed" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:accessConstrain ;
        sh:name "Access constrain information" ;
        sh:in ( pht:accessConstrainRequestNeeded pht:
           accessConstrainUnconstrained pht:
           accessConstrainNoAccess ) ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:usedDifferentialPrivacy ;
        sh:name "Differential Privacy Method" ;
        sh:description "This property indicates the differential
            privacy method used for anonymizing the data set" ;
        sh:or (
                [
                    sh:class pht:DifferentialPrivacyKAnonymity
                ]
                [
                    sh:class pht:DifferentialPrivacyLDiversity
                ]
                [
                    sh:class pht:DifferentialPrivacyTCloseness
                ]
                [
                    sh:class pht:
                       DifferentialPrivacyDifferentialPrivacy
                ]
        ) ;
    ] ;
    sh:property [
        sh:path pht:characteristic ;
        sh:name "Data Set characteristic" ;
        sh:class dmop:DataSetCharacteristics ;
    ] ;
    sh:ignoredProperties ( rdf:type ) .

pht:DifferentialPrivacyKAnonymityShape
    rdf:type sh:NodeShape ;
    sh:targetClass pht:DifferentialPrivacyKAnonymity ;
    sh:property [
        sh:path pht:parameter ;
        sh:name "Required parameter" ;
        sh:datatype xsd:float ;
        sh:minCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

```

pht:DifferentialPrivacyLDiversityShape
    rdf:type sh:NodeShape ;
    sh:targetClass pht:DifferentialPrivacyLDiversity ;
    sh:property [
        sh:path pht:parameter ;
        sh:name "Required parameter" ;
        sh:datatype xsd:float ;
        sh:minCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:DifferentialPrivacyTClosenessShape
    rdf:type sh:NodeShape ;
    sh:targetClass pht:DifferentialPrivacyTCloseness ;
    sh:property [
        sh:path pht:parameter ;
        sh:name "Required parameter" ;
        sh:datatype xsd:float ;
        sh:minCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:DifferentialPrivacyDifferentialPrivacyShape
    rdf:type sh:NodeShape ;
    sh:targetClass pht:DifferentialPrivacyDifferentialPrivacy ;
    sh:property [
        sh:path pht:parameter ;
        sh:name "Required parameter" ;
        sh:datatype xsd:float ;
        sh:minCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

# TabularDataSet

pht:TabularDataSetShape
    rdf:type sh:NodeShape ;
    sh:targetClass pht:TabularDataSet ;
    sh:and ( pht:DataSet ) ;
    sh:description "A Data set in a tabular form" ;
    sh:property [
        sh:path pht:dataFormat ;
        sh:name "Further specifies the data format" ;
        sh:class dmop:DataFormat ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:attribute ;
        sh:name "Attribute of the table" ;
        sh:class pht:TabularDataSetAttribute ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:fileType ;
        sh:name "File type" ;
        sh:description "File type if stored in file, expressed
            with a MIME type expression" ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:TabularDataSetAttributeShape
    rdf:type sh:NodeShape ;
    sh:targetClass pht:TabularDataSetAttribute ;
    sh:description "An attribute/column of a tabular data set" ;
    sh:property [
        sh:path pht:dataType ;
        sh:name "specifies the data type" ;
        sh:class dmop:DataType ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:key ;
        sh:name "Access key" ;
        sh:description "The key to access the attribute/column (
            e.g.column name)" ;
        sh:datatype xsd:string;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:description ;
        sh:name "Description" ;
        sh:datatype xsd:string;
    ] ;
    sh:property [
        sh:path pht:theme ;
        sh:name "Theme of the attribute" ;
        sh:class skos:Concept ;
    ] ;
    #sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

# FileDataSet

```

```
pht:FileDataSetShape
    rdf:type sh:NodeShape ;
    sh:and ( pht:DataSet );
    sh:targetClass pht:FileDataSet ;
    sh:description "A Data set in a tabular form" ;
    sh:property [
        sh:path pht:fileType ;
        sh:name "File type" ;
        sh:datatype xsd:string;
        sh:maxCount 1 ;
    ] ;
    #sh:closed true ;
    sh:ignoredProperties ( rdf:type  ) .
```

C.2 Train SHACL definition

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix dash: <http://datashapes.org/dash#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dmop: <http://www.e-lico.eu/ontologies/dmo/DMOP/DMKB.owl
    > .

#The prefix for PHT specific data; this should be changed later
    on, because i dont know who owns the domain
@prefix pht: <http://www.personalhealthtrainmetadata.org/#> .

pht:TrainShape
    a sh:NodeShape ;
    sh:targetClass pht:Train ;
    rdfs:comment "A Train as a static blueprint in the PHT." ;
    sh:property [
        sh:path pht:creator ;
        sh:name "Train creator" ;
        sh:class foaf:Agent ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:publisher ;
        sh:name "Train publisher" ;
        sh:class foaf:Agent ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:identifier ;
        sh:name "Train identifier" ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:source ;
        sh:name "Train source" ;
    ] ;
    sh:property [
        sh:path pht:description ;
        sh:name "Train description" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:title ;
        sh:name "Train title" ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:version ;
        sh:name "Version" ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:published ;
        sh:name "Publication date" ;
        sh:datatype xsd:date ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:certificate ;
        sh:name "Certificate" ;
        sh:description "One or more certificates providing
           additional trust in the train" ;
        sh:class pht:Certificate ;
    ] ;
    sh:property [
        sh:path pht:execution ;
        sh:name "Train Execution" ;
        sh:description "An execution of the train" ;
        sh:class pht:TrainExecution ;
        sh:minCount 0 ;
    ] ;
    sh:property [
        sh:path pht:model ;
        sh:name "Model" ;
        sh:description "A model containing a programm executed
           on the station" ;
        sh:class pht:Model ;
        sh:minCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

# Certificate

pht:CertificateShape
    a sh:NodeShape ;
    sh:targetClass pht:Certificate ;
    rdfs:comment "A certificate object for a PHT-Train" ;
    sh:property [
        sh:path pht:certificateData ;
        sh:name "Certificate Data" ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:certificateIssuer ;
        sh:name "Certificate Issuer" ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:certificateBegins ;
        sh:name "Beginning date of certificate validity" ;
        sh:datatype xsd:date ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:certificateEnd ;
        sh:name "Ending date of certificate validity" ;
        sh:datatype xsd:date ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

# Model

pht:ModelShape
    a sh:NodeShape ;
    sh:targetClass pht:Model ;
    sh:property [
        sh:path pht:identifier ;
        sh:name "Model identifier" ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:creator ;
        sh:name "Model Creator" ;
        sh:class foaf:Agent ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:description ;
        sh:name "Model rights" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:description ;
        sh:name "Model description" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:size ;
        sh:name "Model size" ;
        sh:datatype xsd:integer ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:license ;
        sh:name "License" ;
        # SWO has a weird naming scheme:
        sh:class <http://www.ebi.ac.uk/swo/SWO_0000002>;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:characteristic ;
        sh:name "Model characteristics" ;
        sh:class dmop:AlgorithmCharacteristic ;

    ] ;
    sh:property [
        sh:path pht:preProcessingAlgorithm ;
        sh:name "Preprocessing Algorithms" ;
        sh:description "In model used pre processing algorithms"
             ;
        sh:class dmop:Algorithm ;

    ] ;
    sh:property [
        sh:path pht:algorithm ;
        sh:name "In model used algorithms" ;
        sh:class dmop:Algorithm ;

    ] ;
    sh:property [
        sh:path pht:dataInteractionRead ;
        sh:name "Model Data Interaction: Read" ;
        sh:datatype xsd:boolean ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:dataInteractionWrite ;
        sh:name "Model Data Interaction: Write" ;
        sh:datatype xsd:boolean ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:dataInteractionDelete ;
        sh:name "Model Data Interaction: Delete" ;
        sh:datatype xsd:boolean ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:usedAccessProtocol ;
        sh:name "Model used access Protocol" ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:expectedDataSet ;
        sh:name "Model Expected Data Set" ;
        sh:or (
            [
                sh:class pht:ExpectedTabularDataSet ;
            ]
            [
                sh:class pht:ExpectedFileDataSet ;
            ]
        )
    ] ;
    sh:property [
        sh:path pht:minimumEstimatedGFLOPS ;
        sh:name "Model minimum needed estimated GFLOPS" ;
        sh:datatype xsd:float ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:needCUDASupport ;
        sh:name "Model need CUDA support" ;
        sh:datatype xsd:boolean ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

# TrainExecution

```

```
pht:TrainExecutionShape
    a sh:NodeShape ;
    sh:targetClass pht:TrainExecution ;
    sh:property [
        sh:path pht:identifier ;
        sh:name "Execution identifier" ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:trainId ;
        sh:name "Train identifier" ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:description ;
        sh:name "Execution description" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:creator ;
        sh:name "Execution Creator" ;
        sh:class foaf:Agent ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:enableEvent ;
        sh:name "Enabled Events for this execution" ;
        sh:in ( pht:StartedRunningStationEvent pht:
           StartedTransmissionToStationEvent pht:
           FinishedTransmissionToStationEvent pht:
           MemoryUsageReportEvent pht:CpuUsageReportEvent pht:
           StationErrorEvent pht:StationRejectedEvent pht:
           StationLogEvent) ;
        sh:qualifiedValueShapesDisjoint true ;
    ] ;
    sh:property [
        sh:path pht:state ;
        sh:name "Current state of the train" ;
        sh:in ( pht:trainStateIdle pht:trainStateTransmission
           pht:trainStateWaiting pht:trainStateRunning pht:
           trainStateFailed pht:trainStateFinished) ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:event ;
        sh:name "List of events of the execution" ;
        sh:or (
                [
                    sh:class pht:StartedRunningStationEvent ;
                ]
                [
                    sh:class pht:StartedTransmissionToStationEvent ;
                ]
                [
                    sh:class pht:FinishedTransmissionToStationEvent ;
                ]
                [
                    sh:class pht:MemoryUsageReportEvent ;
                ]
                [
                    sh:class pht:CpuUsageReportEvent ;
                ]
                [
                    sh:class pht:StationErrorEvent ;
                ]
                [
                    sh:class pht:StationRejectedEvent ;
                ]
                [
                    sh:class pht:StationLogEvent ;
                ]
                [
                    sh:class pht:FinishedTransmissionEvent ;
                ]
                [
                    sh:class pht:FinishedRunningAtStationEvent ;
                ]
        ) ;
    ] ;
    sh:property [
        sh:path pht:plannedRouteStep ;
        sh:name "Steps of the planned route" ;
        sh:class pht:ExecutionPlanStep ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path pht:dispatchingPattern ;
        sh:name "Dispatching Pattern" ;
        sh:in ( pht:trainDispatchingPatternParallel
                pht:trainDispatchingPatternLinear
                pht:TrainDispatchingPatternMixed ) ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:public ;
        sh:name "Is execution public" ;
        sh:datatype xsd:boolean ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:ExecutionPlanStepShape
    a sh:NodeShape ;
    sh:targetClass pht:ExecutionPlanStep ;
    sh:property [
        sh:path pht:station ;
        sh:name "Station of the step" ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:description ;
        sh:name "Step description" ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pht:stepNumber ;
        sh:name "Index of the plan step" ;
        sh:datatype xsd:integer ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:dataSet ;
        sh:name "DataSet of the station that should be used" ;
    ] ;
    sh:property [
        sh:path pht:model ;
        sh:name "Model that should be used in this plan step" ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

# Events

pht:ExecutionEventShape
    a sh:NodeShape ;
    sh:targetClass pht:ExecutionEvent ;
    sh:property [
        sh:path pht:timestamp ;
        sh:name "The timestamp when the event happend" ;
        sh:datatype xsd:datetime ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:ignoredProperties ( rdf:type ) .

pht:TrainInstantiatedEventShape
    a sh:NodeShape ;
    sh:targetClass pht:TrainIstantiatedExecutionEvent ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:StartedRunningEventShape
    a sh:NodeShape ;
    sh:targetClass pht:StartedRunningEvent ;
    sh:property [
        sh:path pht:station ;
        sh:name "The station on which the execution started" ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:StartedTransmissionEventShape
    a sh:NodeShape ;
    sh:targetClass pht:StartedTransmissionEvent ;
    sh:property [
        sh:path pht:station ;
        sh:name "The station to which the transmission started"
            ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:FinishedTransmissionEventShape
    a sh:NodeShape ;
    sh:targetClass pht:FinishedTransmissionEvent ;
    sh:property [
        sh:path pht:station ;
        sh:name "The station to which the transmission finished"
            ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

```


pht:MemoryUsageReportEventShape
    a sh:NodeShape ;
    sh:targetClass pht:MemoryUsageReportEvent ;
    sh:property [
        sh:path pht:station ;
        sh:name "The station at which the train runs" ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:value ;
        sh:name "The memory usage in Megabyte" ;
        sh:datatype xsd:float ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:CPUUsageReportEventShape
    a sh:NodeShape ;
    sh:targetClass pht:CPUUsageReportEvent ;
    sh:property [
        sh:path pht:station ;
        sh:name "The station at which the train runs" ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:value ;
        sh:name "The CPU usage in percent" ;
        sh:datatype xsd:integer ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:StationErrorEventShape
    a sh:NodeShape ;
    sh:targetClass pht:StationErrorEvent ;
    sh:property [
        sh:path pht:station ;
        sh:name "The station at which the train did run" ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:message ;
        sh:name "The error message" ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:StationRejectedEventShape
    a sh:NodeShape ;
    sh:targetClass pht:StationRejectedEvent ;
    sh:property [
        sh:path pht:station ;
        sh:name "The station in question" ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:message ;
        sh:name "The reason for rejection" ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:FinishedRunningAtStationEventShape
    a sh:NodeShape ;
    sh:targetClass pht:FinishedRunningAtStationEvent ;
    sh:property [
        sh:path pht:station ;
        sh:name "The station in question" ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:success ;
        sh:name "Success" ;
        sh:description "Did the execution had success?" ;
        sh:datatype xsd:boolean ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

```

```
pht:StationLogEventShape
    a sh:NodeShape ;
    sh:targetClass pht:StationLogEvent ;
    sh:property [
        sh:path pht:station ;
        sh:name "The station at which the train runs" ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:message ;
        sh:name "The log message" ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

# Data

pht:ExpectedTabularDataSetShape
    a sh:NodeShape ;
    sh:targetClass pht:ExpectedTabularDataSet ;
    sh:property [
        sh:path pht:dataFormat ;
        sh:name "Specifying the Data Format further" ;
        sh:class dmop:DataFormat ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path pht:dataFormat ;
        sh:name "Specifying the Data Format further" ;
        sh:class pht:TabularDataSetAttribute ;
        sh:minCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .

pht:ExpectedFileDataSetShape
    a sh:NodeShape ;
    sh:targetClass pht:ExpectedFileDataSet ;
    sh:property [
        sh:path pht:dataFormat ;
        sh:name "Specifying the file format" ;
        sh:datatype xsd:string ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```