A data model for linking testbed and field test data

Christopher Sauer[1]*, Benjamin Schleich[1], Sandro Wartzack[1]

[1] Lehrstuhl für Konstruktionstechnik (KTmfk), Friedrich-Alexander-Universität Erlangen-Nürnberg

** Corresponding author:*
Christopher Sauer
Lehrstuhl für Konstruktionstechnik, FAU Erlangen-Nürnberg
Martensstraße 9
91058 Erlangen
Telefon: +49 9131/8227985
Mail: sauer@mfk.fau.de

Abstract

With the help of data-driven methods such as machine learning, the development of the current product generation can be supported and improved through the early use of data from previous products and product generations. For example, machine learning can be used to predict later product behaviour in field tests from testbed data. This can significantly shorten the development time and save expensive field tests. To implement this data provision for the development processes, uniform data models enable the use of data-driven methods and are of central importance. This paper presents a data model using the example of a testbed for electric vehicle transmissions. Here, potentials for a later data-driven prediction of the product behaviour in the field test for the optimisation of the existing development are shown.

Keywords

Graph databases, machine learning, digital engineering

1. Motivation

With the help of data-driven methods such as machine learning, the development of the current product generation can be supported and improved through the early use of data from previous products and product generations. For example, machine learning can be used to predict later product behaviour in field tests from testbed data. This can significantly shorten the development time and save expensive field tests. In this context, digital engineering is trying to provide methods to enable holistic horizontal and vertical data usage across the entire product development process [1]. To implement this data provision for the development processes, uniform data models enable the use of data-driven methods and are of central importance [2]. This paper presents a possible data model using the example of a testbed for electric vehicle transmissions. Here, potentials for a later data-driven prediction of the product behaviour in the field test for the optimisation of the existing development are shown.

2. Research Problem and Question

Unstructured data exists in different authoring tools and formats. For example, the various Computer-Aided-X, X can stand for Design, Engineering, Manufacturing and more (CAx), generator systems usually store data in their own file format. These are often not directly usable for product development and above all cannot be used for data-driven methods such as the prediction of product behaviour through machine learning. Data models are required so that the data contained can nevertheless be accessed and a connection for data-driven methods is possible. This leads to the following research question, among others: How can existing data sources be structured to provide data for the prediction of product behaviour so that these are made available to product development for the prediction of subsequent product behaviour?

3. Related work

3.1. Relational and Graph Databases

Most common management systems for product data handle data hierarchically and statically and in a project-based manner [3]. Data is stored project-wise and very similar to how one would store CAD data since CAD models show the same hierarchical kind of buildup. Therefore, most product data management happens via relational database management systems (RDBMS) such as Microsoft or Oracle [4]. Inside RDBMS the data is stored in tables and the linkage between different tables happens via so-called keys. Besides accessing the data via queries using the Structured Query Language (SQL), data can be manipulated via so-called transactions. Transactions inside a relational database need to provide four key elements.

- **A**tomicity – a transaction is only completed in full or not.
- **C**onsistency – after the transaction, the previously defined consistency terms for the database need to be fulfilled.
- **I**solation – simultaneous transactions lead to the same results as single transactions.
- **D**urability – Databases states can only be changed via transactions.

Those four key elements define the term ACID for databases. However, these key elements of RDBMS lead to some limitations [5] stated below.

- Inadequate representation of arbitrary data – this means the strict convention only to store tabular data, which is not feasible for every type of usage.
- Semantical overdescription – exemplifies that a relational database must store instances and relations inside different tables. This leads to unnecessary overhead, especially for heavily intertwined data.
- Weak support for recursive queries – queries that need to access data inside the databases recursively take a very long time for this action.
- Law of homogeneity – the data must always be and stay in the form of the table it is stored in. Every datum added to a table also needs to hold all categories defined by the table.

These limitations show that RDBMS might not be feasible for usage with arbitrary and heavily heterogenic data as it could be the case for machine learning tasks and the usage for building machine learning models such as image recognition tasks, etc. Graph databases however provide the following advantages [6]: Data is modelled naturally; this means the easier formulation of instances and relationships via the usage of graphs and graph structure to mark relationships and objects. This can be very useful, especially for inexperienced product developers as they do not need sophisticated product data management tools and can prototype their data model on a whiteboard. Moreover, when querying the whole data, the structure can be exposed to the user, so that one can easily see existing relations and other objects which interact with the one selected. Lastly, graph theory a subarea of mathematics can be used to analyze the data stored inside graph databases, for example, to search for the shortest connection between two objects.

3.2. Data-driven Methods

Machine learning (ML) is a data-driven method that allows application inside the product development process [4, 1]. As mentioned above, ML can for example be used to predict later product behaviour in field tests from testbed data. This can significantly shorten the development time and save expensive field tests. The most common tasks for ML can be grouped into classification and regression tasks. For both tasks, a variety of different mathematical prediction models exists. For this contribution, we want to highlight two of them, which can be applied to regression problems. The focus lies on regression problems because we later want to show the prediction of resulting product behaviour from product features defined by product developers. Machine learning can be divided into three categories, as shown in the following figure 1: Supervised learning, unsupervised learning, and reinforcement learning [7].

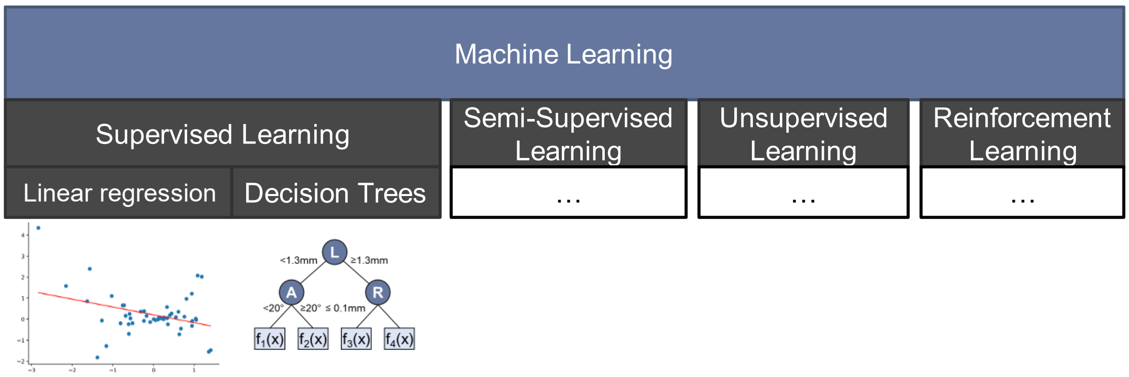


Figure 1: Machine Learning (ML) and its categories, with two Supervised Learning methods highlighted.

Using ML inside product development focuses mainly on supervised learning, to be more specific regression tasks. For these tasks, one can start from data that describes a given input or feature set, so for example the product features defined by product developers and attributed labels such as resulting product characteristics. ML mainly relies for regressional supervised learning on numerical data in the form of n-dimensional arrays to first train and later predict via ML models. One ML basic model we want to highlight is a simple linear regression model with a varying polynomial degree. This model can learn very basic functional connections between the above-specified input and output data. Moreover, one more sophisticated model can be decision tree regressors (DTR). DTRs are piecewise-defined tree models consisting of many simple linear or polynomial functions as their leaves [8]. DTRs can also be called ensemble models.

4. Methods and Use Case

First, we analyzed possible data sources in testbed and field test development and their respective processes. The data sources were then subdivided according to their specific occurrence of the data, this means the creation of three main categories.

- **System elements –** these are the elements of the testbed system, e.g. parts or assemblies of the testbed.
- **Knowledge elements –** elements that can be in relation to the system elements and hold or store knowledge such as simulation models and more.
- **Field test elements –** these are the elements that can represent data and datasets from field tests.

To give more detail for the possible elements we start with the system elements. System elements store information about the manufacturer, the parts and assemblies of the testbed, dimensions and build volumes and structural information about the whole testbed. For example, the hierarchical structure of the testbed, just like in a CAD assembly.

Knowledge elements can store data in the form of simulation input and output files (simulation results). Moreover, trained ML models can also be stored inside knowledge elements. For example, these models can predict testbed behaviour. Furthermore, knowledge elements can store data from the testbed and structural or metadata about the knowledge elements.

Finally, field test elements can hold field data, which can be data or datasets from a carried-out field test. This data can also be streamed from the field via Internet-of-Things technologies [9].

The following table 1 below holds a detailed overview of the assignments, types, and feasible database systems for the different types of data. The linkage between knowledge elements from testbed data and field test data is key in training our ML models for later prediction of product behaviour from field tests. When using a graph database this linkage can happen arbitrarily and does not have to follow the law of homogeneity of RDBMS.

Table 1: Assignments, types, and feasible database systems of different data sources

Data source	Type (Data type)	Feasible database system
System element: Manufacturer	Text (String)	Relational
System element: Parts and Assemblies	Proprietary CAD data	Graph
System element: Dimensions	Numerical (float)	Relational
System element: Build volume	Numerical (float) / CAD data	Relational / Graph
System element: Structural information	Tabular (Object)	Graph
Knowledge element: Simulation data	Proprietary Simulation tool data	Graph
Knowledge element: (Trained) Machine Learning models	JSON / h5 exports	Graph
Knowledge element: Testbed data	Numerical (float)	Relational
Field test element: Field test data	Numerical (float)	Relational

As stated in table 1, most of the data sources mentioned above are numerical data which can be represented via tables or arrays. Both are suitable for using relational database management systems. However, the heterogeneity and the variety of the different data sources needs a more flexible approach to data modelling. So the law of homogeneity for RDBMS stands in the way of implementing a RDBMS only approach. However, existing RDBMS databases can be added as knowledge or field test elements inside the graph database. For our contribution, we want to start from a greenfield approach.

One further point is that graph databases can also store arbitrary data or files inside their nodes such as CAD models and proprietary simulation data. Putting it all together, figure 2 shows the general concept for the data model that emerged from the preliminary considerations. One important aspect of the data model is the ML models which can link field tests and knowledge elements. This enables the training and later the prediction of field behaviour based on the testbed data (M inside figure 2).

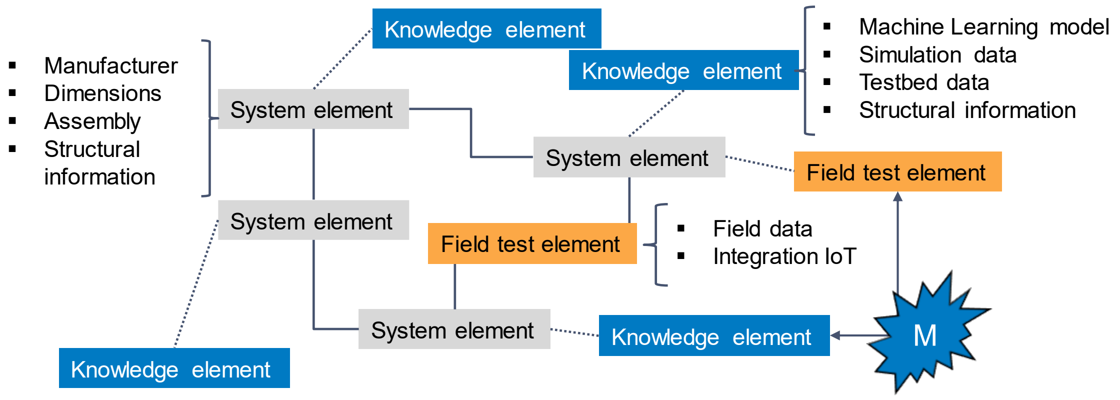


Figure 2: Concept of a data model for a testbed and connected ML models (M)

Starting from figure 2 and with the theoretical considerations in mind the three main categories were populated with exemplary objects in the following way: Starting with the mechanical structure the transmission, the mechanics, the electronics and the measurement and sensors were implemented as nodes of the graph database (see figure 3 a)). Then the knowledge elements such as requirements, properties, measurement data, measurement principles and measurements were implemented as more nodes (see figure 3 b)). Lastly, the field test elements were implemented as nodes such as rounds per minute (RPM), vibrations, or temperature data (see figure 3 c)). Figure 3 puts the use case implementation data into perspective.

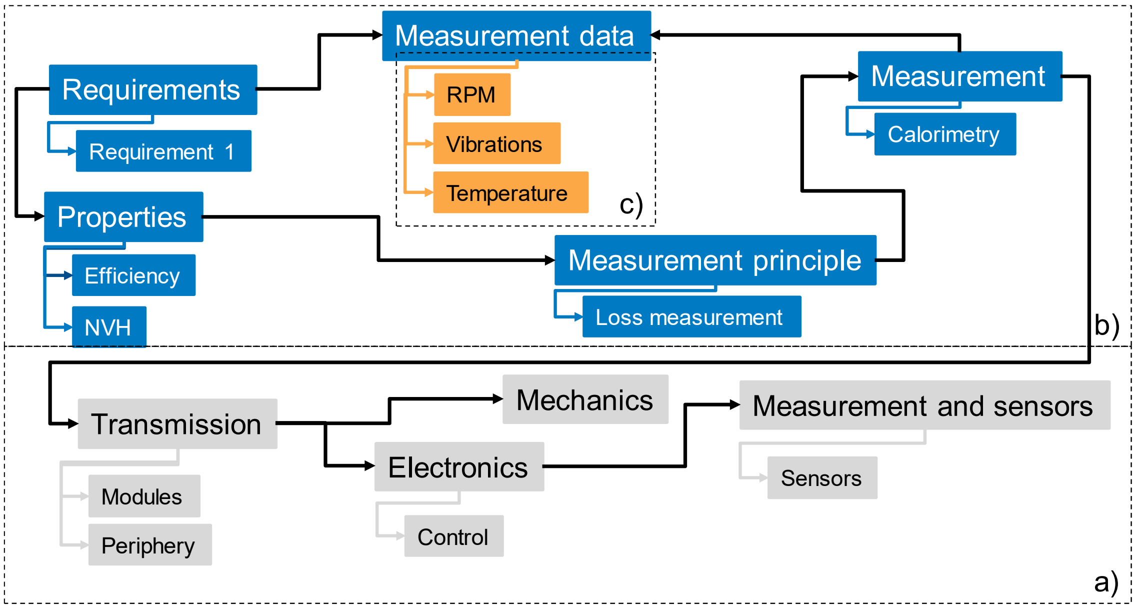


Figure 3: The populated graph database with use case data

For implementing graph databases multiple solutions exist, one can for example use the very popular graph database tool called Neo4J [10]. It is worth noting that when using Neo4J deeper knowledge of the Cypher language is required. Via using Cypher the data inside the graph database is stored and queried. For our use case, we chose a simpler implementation of a graph database in the programming language Python.

When accessing a node or an object that is in the graph database, in most cases a Javascript Object Notation (JSON) file is presented to the user. JSON is a modern and widely used data format and very similar to extensible markup language files (XML) [11]. JSON files have two advantages, which make them very useful for our use case. On the one hand, it is human and machine-readable and on the other hand, it allows simple serialization and storage of Python arrays. As Python is the most preferred programming language for scientific computing and machine learning this comes in handy [12]. By using JSON as data format it is possible to store arbitrary Python arrays and matrices inside a single file with additional information such as presented in the following listing 1. So for example one can directly interchange Python array data by using JSON files.

Listing 1: Exemplary JSON file

```
{
        " name": "rpm",
        " node": "     MeasurementValueEntity",
        " id": 14,
        " measurement_value_list": [
                "1256",
                "1376",
                "1410",
                …
        ],
        " measurement_value_number": 2,
        " measurement_value_name": "Drehzahl"
}
```

Listing 1 also shows the key-value like the structure of the JSON file. One can see that the file is human readable and not in a binary format. For our example use case we want to access the property stored within the key “measurement_value_list”, as these are our exemplary measurement values from the testbed. This needs to be done to predict later product behaviour from the list of measurement values. For the use case, a simple four-step process was implemented in Python using the open-source machine learning library scikit-learn [13]. The process is depicted in the following figure 4.

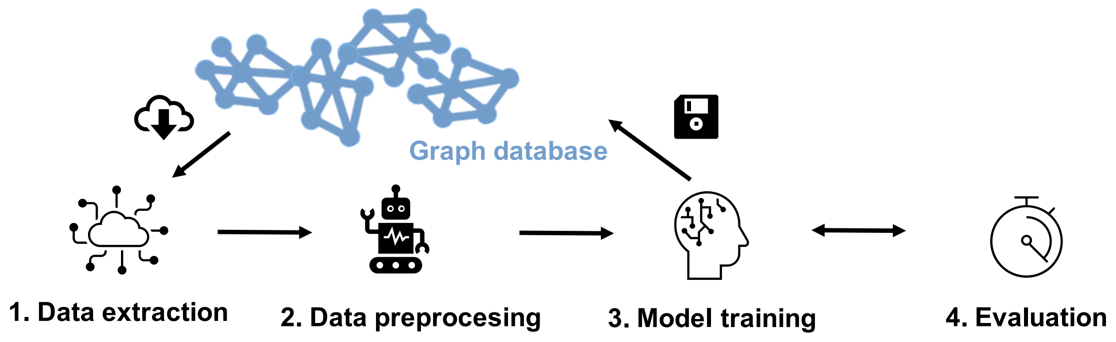
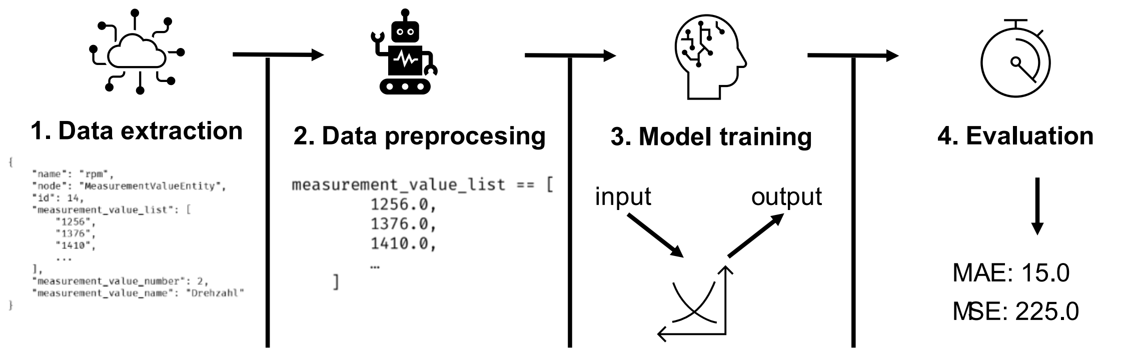


Figure 4: Exemplary process

Our exemplary process starts with the data extraction from the graph database, in this case, we use simple graph database implementation based on Python. The data extraction happens via Python scripts directly from the graph database. The result of the first step is some sort of JSON file, as the one depicted above in listing 1. The extracted JSON file is then processed in step two, the data preprocessing step, in this step the desired data from keys inside the JSON files are extracted. As stated above we want to extract the array stored inside the key “measurement_value_list”. The array stored inside the key is then converted to the necessary Python representation. So the result for step two is then a Python object or Python array which can be used for model training and evaluation. In step three the array or object is then fed to the model training step. In this step, the models mentioned in section two get trained on the “measurement_value_list”-Array. For our use case, we used the scikit-learn library to enable model training and later save the trained model to disk and store it inside the graph database again via a Python script. The trained model then gets carried over to step four, in this last step the model is then evaluated according to quality measures. In our case, we mainly used two error measures, the mean absolute error (MAE) and the mean squared error (MSE). What the Machine Learning model is then able to predict is for example resulting RPMs on the testbed during a test run. The whole data flow inside our exemplary process steps is depicted in figure 5.



```
{
    "name": "rpm",
    "node": "MeasurementValueEntity",
    "id": 14,
    "measurement_value_list": [
        "1256",
        "1376",
        "1410",
        ...
    ],
    "measurement_value_number": 2,
    "measurement_value_name": "Drehzahl"
}
```

```
measurement_value_list == [
        1256.0,
        1376.0,
        1410.0,
        …
    ]
```

Figure 5: Data flow inside the exemplary process steps

This is only the first step for the application of Machine Learning, as we want to extend the prediction not only on testbed or field test data, but we also want to try to predict the field behaviour of our exemplary product based on the testbed behaviour. For this case, we want to access knowledge elements as well as field test elements from our graph database. This is exemplified by the model (M) inside figure 2. For this case, the data extraction needs to be extended to implement a two-way approach. One way is the extraction of knowledge elements and data from the testbed, the second way being the extraction of field data from field test elements inside the graph database.

5. Results and Discussion

For the use case in the test stand, system elements of the respective testbed were identified as possible data sources; these are primarily the physical elements of each testbed. In addition, knowledge elements were identified that can be added to the system elements. These can be, for example, testbed data. Field test elements complete the trio considered. These can contain, for example, field test data. Later, an integration of Internet-of-Things data sources is also conceivable for this [9]. It was stated that a human and machine-readable format such as JSON is used to store arbitrary data inside the graph database, as it is the basic format for the data inside most graph databases. The graph databases can then be accessed from the exemplary process and the data stored inside can be used to train ML models, which can predict field test or testbed behaviour, such as resulting RPMs. Using error measures such as MAE or RMSE can then support model quality analysis.

The article uses this implementation to describe the procedure and the connection to a data-driven method. In addition, the semantic linking of data within the graph database maps the relationships and links between the individual elements contained [14]. For product development, the uniform data model, and the access to all available data result in early optimisation potentials. In this way, the first step towards a possible "Design for Data-driven Methods" approach can also be recognised, to use all data at an early stage in the development of new products and, for example, to draw on them for the prediction of the resulting product behaviour.

So to answer the question stated in section 2: “How can existing data sources be structured to provide data for the prediction of product behaviour so that these are made available to product development for the prediction of subsequent product behaviour?” the data sources can be accessed via the graph database which holds the necessary data from the different sources and by using the exemplary use case process we can use the data to build and train models to predict product behaviour and more.

One main discussion point of the presented solution is the need for the implementation of the graph database. Small and medium-sized enterprises (SMEs) often lack the manpower to focus on data-driven methods and solutions using them to improve their products. If SMEs want to focus on this area sufficient manpower needs to be established on the other hand product data management systems (PDM) can be implemented in the enterprise via the help of engineering service providers. PDM systems can be a first step towards implementing data-driven methods, as they store product data and can be accessed from tailored applications inside the enterprise.

The second point is data acquisition and availability, to dive into data-driven methods there simply needs to be data. This is one major concern for every enterprise thinking about data-driven methods. As the presented data model is flexible to storing datasets and streaming data from IoT solutions this can be one viable strategy. Implementing IoT sensors and measurements to track the field test data and testbeds in the specific SME. To extend this idea even further there is a possible approach to exploit use phase data with the help of graph databases [15].

On the implementation side, one might ask, why we need graph databases for this data model. It is precisely because of the strong heterogeneity that the approaches from the field of relational databases are rather unsuitable for later use in data-driven methods. Mostly because of the law of homogeneity and the strict laws and semantical overdescription RDBMS are not feasible for this use case.

6. Outlook

To further extend the concept and ideas presented in this contribution we want to focus on three main areas in the future. On the data side, we want to extend the basic models with more use case data and objects, to store more information inside the presented graph database. On the methods side, we want to focus on adding more data-driven methods to the possible use cases, such as unsupervised learning tasks and more supervised learning models. The third area of focus is the potential use cases for data-driven methods along the product development process. In this contribution we only showed one very basic approach to implementing and training a ML model for prediction, this needs to be extended.

On the organisational side, the presented methods need to be put together to give especially SMEs a valuable starting point for their dive into data-driven methods and solutions for their ML problems. This should remove barriers to entry for the SMEs, however, the necessary manpower needs to be distributed for the application of data-driven methods.

Acknowledgements

This research work is part of “FORCuDE@BEV - Bavarian research association for customized digital engineering for bavarian SME's“ and is funded by the “Bayerische Forschungsstiftung (BFS)”. The authors are responsible for the content of this publication. Special thanks are directed to the Bayerische Forschungsstiftung (BFS) for financial support of the whole research project. Furthermore, N. Sprogies from FZG at TUM Munich, Germany is thanked for the fruitful discussions and input about testbeds and their specific requirements.

References

[1] Gerschütz, Benjamin; Sauer, Christopher; Kormann, Andreas; et al.: Towards Customized Digital Engineering: Herausforderungen und Potentiale bei der Anpassung von Digital Engineering Methoden für den Produktentwicklungsprozess. In: SSP 2021. Stuttgart, 2021

[2] Maack, Stefan; Bertovic, Marija; Radtke, Martin; et al.: Deutsche Normungsroadmap künstliche Intelligenz. In: opus4.kobv.de (2020)

[3] Conrad, Jan; Deubel, Till; Köhler, Christian; Wanke, Sören; Weber, Christian: Comparison of Knowledge Representation in PDM and by Sematic Networks. In: Proceedings of the 16th International Conference on Engineering Design (ICED) 2007, Paris. Design Society (2007). https://www.designsociety.org/publication/25624/

[4] Vajna, Sandor; Weber, Christian; Zeman, Klaus; Hehenberger, Peter; Gerhard, Detlef; Wartzack, Sandro: CAx für Ingenieure. 3. Auflage. Berlin/Heidelberg: Springer-Verlag (2018). https://doi.org/10.1007/978-3-662-54624-6

[5] Wiese, Lena: “Advanced Data Management”. Berlin/Boston: Walter de Gruyter GmbH. (2015) ISBN: 978-3110441406

[6] Angles, Renzo; Gutierrez, Claudio: Survey of graph database models. In: ACM Computing Surveys (CSUR), 40 (1) (2008). https://doi.org/10.1145/1322432.1322433

[7] Witten, Ian; Frank, Eibe; Hall, Mark; Pal, Christopher. Data Mining: Practical machine learning tools and techniques. (2016). ISBN: 9780128043578.

[8] Breiman, Lucas; Friedman, Julian; Olshen, Richard; Stone, Clark. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.

[9] Van Quyet, Nguyen; Thi Xuan Lac, Bui; Van Hau, Nguyen: An Efficient Graph Modeling Approach For Storing And Analyzing Heterogeneous IOT Data. In: JST. Vol. 27 (2020), p. 2127.

[10] Fernandes, Diogo; Bernardino, Jorge: “Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB” In: *DATA*. (2018) S. 373-380.

[11] Peng, Dunlu; Cao, Lidong; Xu, Wenjie. Using JSON for data exchanging in web service applications. Journal of Computational Information Systems, 2011, 7. Jg., Nr. 16, S. 5883-5890.

[12] Raschka, Sebastian; Patterson, Joshua; Corey, Nolet “Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence.” Information, vol. 11, no. 4, 4 Apr. 2020, p. 193, (2020) https://doi.org/10.3390/info11040193

[13] Pedregosa, Fabian; Varoquaux, Gaël; Gramfort, Alexandre; Michel, Vincent; Thirion, Bertrand; Grisel, Olivier; Blondel, Mathieu; Prettenhofer, Peter; et al.: Scikit-learn: Machine Learning in Python. In: Journal of Machine Learning Research vol. 12 (2011), Nr. 85, pp. 2825–2830

[14] Pokorný, Jaroslav: Graph Databases: Their Power and Limitations. In: Computer Information Systems and Industrial Management (2015), pp. 58–69 — ISBN 9783319243689

[15] Hollauer, Christoph; Shalumov, Boris; Wilberg, Julian; Omer, Mayada: Graph Databases For Exploiting Use Phase Data In Product-Service-System Development: A Methodology To Support Implementation. In: Proceedings of the DESIGN 2018 15th International Design Conference (2018) — ISBN 9789537738594. https://doi.org/10.21278/idc.2018.0399.