A Model Management Platform for Industry 4.0 – Enabling Management of Machine Learning Models in Manufacturing Environments

Christian Weber[1(✉)], Pascal Hirmer[2], and Peter Reimann[1,2]

[1] Graduate School of Advanced Manufacturing Engineering, University of Stuttgart, Stuttgart, Germany
{christian.weber,peter.reimann}@gsame.uni-stuttgart.de

[2] Institute for Parallel and Distributed Systems, University of Stuttgart, Stuttgart, Germany
pascal.hirmer@ipvs.uni-stuttgart.de

Abstract. Industry 4.0 use cases such as predictive maintenance and product quality control make it necessary to create, use and maintain a multitude of different machine learning models. In this setting, model management systems help to organize models. However, concepts for model management systems currently focus on data scientists, but do not support non-expert users such as domain experts and business analysts. Thus, it is difficult for them to reuse existing models for their use cases. In this paper, we address these challenges and present an architecture, a metadata schema and a corresponding model management platform.

Keywords: Model management · Machine learning · Metadata management · Industry 4.0

1 Introduction

Through the emergence of cyber-physical systems and initiatives, such as Industry 4.0 [8] and the Industrial Internet [13], manufacturing environments are equipped with a variety of sensors that produce large amounts of data. Analyzing these data enhances various use cases in manufacturing. These use cases include predictive maintenance, product quality control, root cause analysis of quality problems, and the optimization of manufacturing processes [6,16]. In the analysis process, data scientists experiment with different algorithms, frameworks and tools for machine learning (ML). The result of such data analyses are several promising candidates for the final machine learning model. By estimating the generalization error and conducting A/B tests, the most suitable model is selected and used for inference or prediction [3,9].

In the process of creating and maintaining a ML solution, a high amount of diverse models is created even for a single use case. The reason for this is the variety and multitude of manufacturing processes, machines and product variants that require individual models. This is further reinforced by the fact that companies implement many different uses cases that multiply the amount of models being created. Due to the high dynamics of the manufacturing processes, such models tend to become stale very quickly, e.g., because of concept drifts in data [1,4,17]. For example, a machine learning model that predicts the wear of a tool based on multiple sensors may lose quality because patterns in data change, e.g., through changes in air pressure or damage to a sensor. Thus, stale models need to be continuously replaced by new models. Since a large number of new models are created every day and others become stale at the same time, the management of these models turns into a great challenge. Hence, for all stakeholders, it becomes a tedious task to keep an overview of all models in the manufacturing environment. This prevents effective reuse of machine learning models and may result in costly reengineering tasks. So, data scientists often recreate models for each new use case, although they could possibly reuse an existing model that has been created for a similar use case.

In order to cope with these issues, we introduce a Model Management Platform (MMP) for I4.0 that enables to store, index, and retrieve machine learning models in a manufacturing context. Our platform enables central access to all involved models and provides domain experts with a clear overview of different versions of models and their current status, i.e., whether they are currently planned, experimental, in use, maintained or retired. A rich interface to query context data facilitates the effective discovery of models for similar use cases and ML tasks. Furthermore, the platform provides reporting functionalities for domain experts to keep the overview on all models within the company.

Related work does not cover these issues to a full extend, i.e., they do not consider the manufacturing or business context in the lifecycle of a machine learning model. Furthermore, to the best of our knowledge, current solutions do not provide status tracking to machine learning models.

The remainder of this paper is structured as follows: In Sect. 2, we provide an exemplary model management use case which guided the design of our platform and also serves as base for our later case study. In Sect. 3, we describe foundations and discuss related work in the scope of model management and related platforms. Section 4 describes our main contribution: the Model Management Platform (MMP), including its metadata schema and architecture. Section 5 discusses the implementation of the platform and presents the features of the platform by a case study. Finally, Sect. 6 concludes this paper and lists possible future work.

2 Exemplary Use Case and Requirements

We provide an exemplary use case in which a platform for model management becomes crucial. The use case was derived by conducting interviews with our industrial partners from the manufacturing sector.

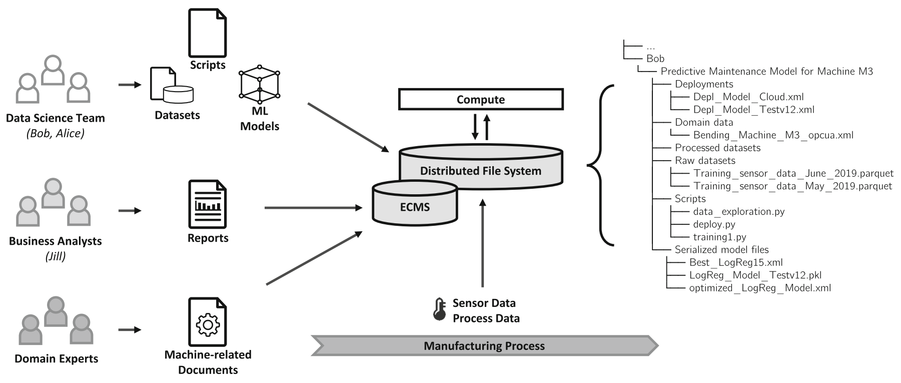

Fig. 1. Example showing the current IT infrastructure of the model management use case of the steel plate manufacturer. Each data scientist owns a ML project folder on the distributed file system with his or her files. Business Analysts and Domain Experts manage their files in an Enterprise Content Management System (ECMS).

2.1 Use Case Environment

We consider that a medium-sized manufacturer produces tailor-made steel plates for customers all over the world. In order to serve the market in the best possible way, the steel plate manufacturer owns several plants in different countries. The products are of good quality, but in the past there have often been problems in meeting delivery times. Completing orders took longer than planned because machines suddenly failed due to wear on machine parts. Therefore, based on the experience of production employees, the manufacturer introduced fixed maintenance intervals to replace machine parts preventively. Nevertheless, there were still occasional machine downtimes, as machine parts were worn out faster than planned. Furthermore, the maintenance personnel noticed that some machine components showed only slight signs of wear and were replaced much too early. In order to better determine the maintenance intervals individually for each machine component, the steel plate manufacturer introduced a ML solution which can make predictions based on historical wear data for the machine components. At the moment, the data of machines from different locations are collected centrally. This includes time-series data, such as temperature, humidity, and vibration data, as well as high-resolution image data from cameras.

2.2 Use Case for Model Management

We imagine the data scientists Bob and Alice, who are located in different departments. Together with other data scientists, they form a decentralized data science team. In addition, Jill is a business analyst whose task is to keep an overview on ML projects in the company. Bob, Alice, and Jill share a common IT infrastructure (see Fig. 1). A distributed file system is part of this IT infrastructure to store datasets, scripts, and model files of ML projects.

Bob wants to create a machine learning model that predicts downtimes for a specific machine type. After receiving data about the machine, he performs several machine learning experiments by writing a Python script and trying out several algorithms for logistic regression. Each time the script is run, it creates a new version of the model. Some of the model versions provide good evaluation results and meet the target thresholds of evaluation metrics. Therefore, he marks them as candidates for the final model. Through A/B testing, he selects the best performing model and deploys it to a ML serving system. The deployed model may however become stale at some point of time, because the data from machines is non-stationary and concept drifts occur. Bob retrains and redeploys the model from time to time to keep it up-to-date, which results in a new model version. Each time he trains or retrains a model, he stores the raw data, his script, and model files on the distributed file system, thereby using his individual folder structure and file names as shown in Fig. 1. Exemplary names for model files include *LogReg_Model_Testv12*, *optimized_LogReg_Model*, *Depl_Model_Testv12*.

Later, Alice begins to develop a logistic regression model for another machine that includes the same sensor types as Bob's machine. Unfortunately, she knows nothing about Bob's experiments. After performing a short search on the distributed file system, she cannot find any related models. She is confused by the names of the model files and wonders to which machines which model belongs. Therefore, she decides to develop the model from scratch and spends a lot of time on it. By chance she meets Bob, who tells her that he has developed a model for a similar type of machine. Bob shows Alice where he saved his model files. After examining his files and doing A/B tests on the models, Alice finds that Bob's final model is more accurate than any of the models she created on her own. She uses Bob's model and puts it into production. She copies Bob's files into her folder on the distributed file system.

Some time later, Jill needs to create a report for the chief digital officer. The report should contain an overview of currently used models, the involved departments and business processes. She talks to Alice and Bob to acquire the necessary information. With this information, Jill creates a report using a spreadsheet. Jill, however, complains about the slow reporting process. This is because Bob and Alice must manually examine the model serving system and cannot do so immediately due to other tasks.

2.3 Required Functions of a Model Management Platform

Overall, the steel plate manufacturer's analytics team has several problems establishing an efficient way of model management. For these reasons, the manufacturer's data science department wants to establish a model management platform to cope with the depicted issues. In the following, we discuss the issues of the use case and identify required functions a model management should offer.

Although the analytics team stores data sets, scripts and model files in a customized folder structure on a distributed file system, it is difficult to keep track on which model is the most recent one and which files relate to it. This can even lead to re-deploying model versions which have even been retired. Here, a function to *versionize models*, according datasets, and scripts would be beneficial.

In addition, information about assets, such as machines and other manufacturing equipment for which the models are built, is hidden in an ECMS where domain experts store their documents. Therefore, models are isolated from these documents and it is hard to discover related models used for similar assets and for corresponding ML tasks. These management problems let data scientists (re-)develop models, even if they exist already. Thus, a function to add *manufacturing and business context* to models is required. Afterwards, the platform can use the added context to provide a better discovery of models and to provide views for effective model reporting. For example, users may search for models that are appropriate for a certain business process or machine. In addition they may also want to have an overview on models that are used within different departments of the company.

Further difficulties exist to distinguish between models. That is, experimental versions of the model are stored along with model candidates and the final version in the same folder. Beyond managing the files themselves, it is also unclear which model is currently in use or stale. This requires a function to distinguish these different *states of ML models.*

In sum, required model management functions include a) *model versioning*, b) *providing context*, and c) *tracking the status* of models across their lifecycle.

3 Related Work and Challenges

Our work is based on the process model for managing machine learning models by Weber et al. [15]. This process is depicted in Fig. 2. It shows all the lifecycle steps a machine learning model passes through and that must be supported by model management. In the following, we discuss the steps in the process model and show how the derived model management functions a), b) and c) (see Sect. 2.3) can provide support for each process step. Afterwards, we discuss related work on model management platforms and possible tool support.

In the first step, new use cases serve as initiators to plan models that can help, e.g., to improve manufacturing processes. Planning includes, e.g., specifying the involved data sources, describing specifics of the use case, defining acceptance criteria, or documenting expected results. Different stakeholders are required that bring in domain-specific or technical expertise. In this step, function b) providing context to models would help to identify and re-use existing models for similar use cases. After that, if the model passes a feasibility check, the model is created and evaluated by data scientists and programmers (step 2). In order to document results of experiments and to store different model files, function a) model versioning is required. In case the evaluation of the model shows promising results, the model can be deployed into its target environment (step 3). Otherwise, e.g., because the data quality is poor, the model needs to be replanned or even retired (step 6). After deployment, the model is used, e.g., to make continuous predictions (step 4a). In parallel, it is constantly monitored

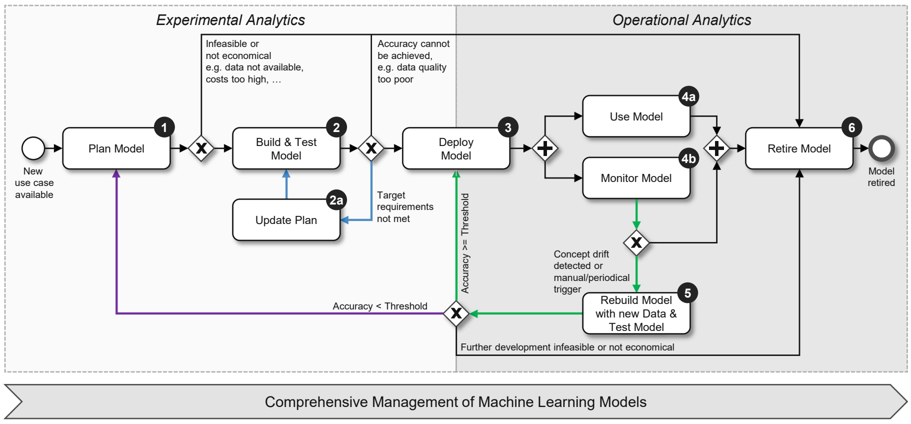
Experimental Analytics
Operational Analytics
Infeasible or not economical e.g. data not available, costs too high, …
Accuracy cannot be achieved, e.g. data quality too poor
New use case available
Plan Model
Build & Test Model
Update Plan
Target requirements not met
Deploy Model
Use Model
Monitor Model
Retire Model
Model retired
Accuracy >= Threshold
Accuracy < Threshold
Concept drift detected or manual/periodical trigger
Rebuild Model with new Data & Test Model
Further development infeasible or not economical
Comprehensive Management of Machine Learning Models

Fig. 2. Process of the model lifecycle [15] that is covered by the platform. Blue lines: experimental loop, green lines: update loop, purple line: upgrade loop. (Color figure online)

to recognize concept drifts (step 4b). In case of a concept drift, the model needs to be rebuilt with new data and tested once again before it can be redeployed (step 5). Here, function a) is also needed because this step is similar to step 2 in that different versions of a model may be created. Once a model is not useful or not needed anymore, it is retired (step 6). Function c) tracking the status of models is required throughout the whole model lifecycle to check the status of a model in each step, e.g., if it is already in production or just a candidate for future application. This prevents users from selecting models that are outdated or do not conform to their requirements.

In order to evaluate solutions for implementing the functions proposed in Sect. 2.3, we also conducted an evaluation of different tools for managing models in machine learning projects. For function a) model versioning, data scientists can rely on versioning tools that provide a repository such as Git LFS [2] or more specialized tools like DVC [11] for bigger model files. Additionally, for storing metadata about how the model files were produced, data scientists can use experiment management tools. Examples for such tools are ModelDB [14] and MLflow [18], which is available both in the Microsoft Azure platform and as a standalone application. These tools store settings and metrics about training runs, and they track the history of experiments along with their evaluation results. This makes it possible for data scientists to reproduce experiments and share their results with others. Furthermore, the tools can be used for saving metadata about experiments, as well as materialization and versioning of model files produced in step 2) and 5) of the ML lifecycle. MLflow provides a model registry which allows to add states to a model version, *staging* and *production*. Multiple model versions belong to a model that has a unique name. However, we argue that the status information should be kept by the model entity as well because status information about other lifecycle steps is required. A platform that addresses the whole lifecycle would need additional states for each lifecycle step, e.g. planned (step 1), experimental (step 2), staging (step 3), production (step 4a, 4b), stale (step 4a, 4b), maintenance (step 5) and retired (step 6). Thus, c) tracking the status of models across their lifecycle is partially supported. Since these tools are intended to support data scientists rather than domain experts and business analysts, requirement b) providing context information to models is also an open issue that we want to support with the platform.

Based on the required functions resulting from the model management use case as well as our discussion of related work and tool support, we identified open challenges that must be addressed by the platform. These challenges can be summarized as follows:

① **Tracking the status of models across their lifecycle** –
According to the process model for the model lifecycle, a model passes several steps. In each step, metadata is generated that can be used to further describe the model. However, current tools and related work do not focus on tracking metadata across all lifecycle steps to describe the whole lifecycle. The following states require support by a model registry component and an appropriate metadata schema: planned (step 1), experimental (step 2), staging (step 3), production (step 4a, 4b), stale (step 4a, 4b), maintenance (step 5) and retired (step 6).

② **Adding context to models for discovery/reporting** –
Currently, there is no related work on adding context to machine learning models to enable functions for search and discovery that are tailored to domain experts. In order to discover the most suitable model for a given use case, it is essential that users are provided with context information. However, this information is kept in files that are often located in Content Management Systems. These files may contain semantical information about manufacturing assets that conforms to standards for Industry 4.0 or the Industrial Internet. Due to broader use of these standards in the future, they need to be supported by different platforms such as model management platforms. To cope with these formats in manufacturing, a platform requires concepts to extract, store, and link information about manufacturing assets to a machine learning model. This would allow data scientists to search for a model version that matches a specific machine or sensor type. Moreover, through adding business context, business analysts can be supported with model reporting functionalities which reduces manual efforts to collect the required information from different stakeholders.

4 Model Management Platform for Industry 4.0

Our model management platform (MMP) builds on the process model of the model lifecycle by supporting users in each step. In this section, we present a metadata schema, as well as the MMP's architecture and its components. We follow up with details about the implementation of the components.

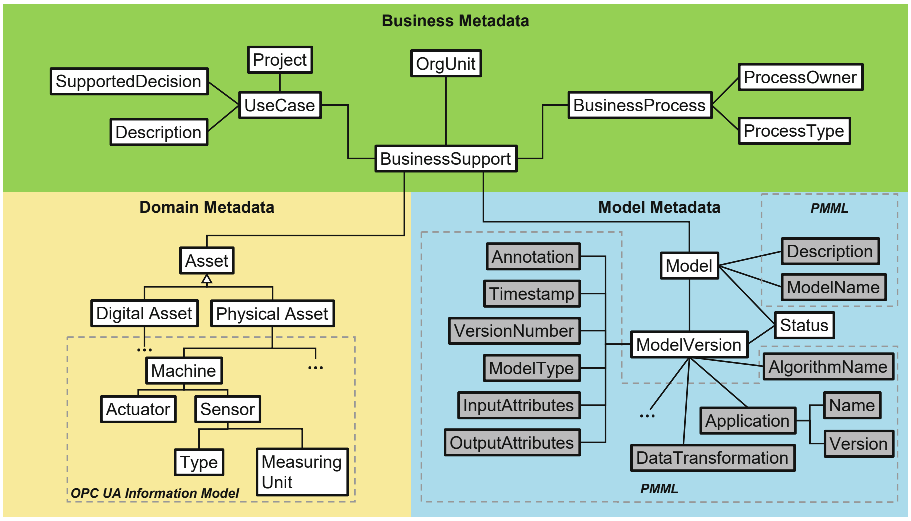

Fig. 3. Extract from the MMP metamodel, showing the combination of the model, business, and domain metadata package.

4.1 Metadata Schema

For storing models with according metadata for the different steps of the lifecycle process, we provide a metadata schema as depicted in Fig. 3.

The model metadata are provided via a model metadata package. In order to provide structure to massive amounts of ML models, the lifecycle of machine learning models is represented by the *model entity*. A model entity is comprised of several *model versions* that are produced within steps 2) and 5) of the ML lifecycle process. It contains a model name, a description and the status of a model. Thus, the metamodel supports challenge ①.

A model version contains descriptive attributes, e.g., the version number, model type, annotation, a creation timestamp, and the input and output parameters. Furthermore, a model version contains lineage metadata that describe the process to generate the model version, e.g., metadata about experiments – the algorithm used, data transformations used for preprocessing the input attributes and quality metrics. These metadata can be parsed from model files by a model metadata extractor component (attributes are color-coded in grey).

In contrast, the domain metadata package describes the manufacturing context. Different *assets*, e.g., *machines*, stations and manufacturing equipment that provide input data to models can be linked to the model's lifecycle. In order to support linking models to as much manufacturing entities as possible, we rely on the concept of Gröger et al. [5] which stores links between models and manufacturing entities in a data warehouse. However, this concept does not provide means to link models to semantical information contained in files that are not part of the datawarehouse, e.g. XML-based OPC UA files that describe machines. We hence support parsing these files, extracting their content, storing it, and linking it to a model. Currently, we support parsing semantical descriptions of machines. By providing the machine's *actuator*, as well as *sensors* with their *type* and *measuring unit*, we can add context to models, e.g., to search for models for a certain machine and/or sensor type. Thus, we extend the concept of Gröger et al. by providing further details on machine entities.

We further define a business metadata package to support reporting for models. It supports linking *business processes*, *organizational units*, and *use cases* of *projects* to the model lifecycle entity. A use case contains the supported decision for which the model should provide support and a description. Examples for decisions are e.g. root cause analysis for production failures or the prediction of maintenance schedules. The description then explains more domain specific details about the problem, e.g., about quality issues, downtimes, the production process, production materials, and machines. This enables to provide visualizations such as dashboards and overviews on models used within different departments of a company.

By combining the business, domain and model metadata package via a *BusinessSupport* entity, the metamodel supports adding context to models and facilitates model discovery and reporting according to challenge ②.

4.2 Architecture

Our architecture (see Fig. 4) consists of a frontend component, a backend component, and data storages for metadata and model files. The frontend component offers a graphical user interface to manipulate and query the metadata and model store. In the backend component, corresponding functions are provided to manage models and according metadata based on user input. The detailed components are presented in the following. Thereby, we point out how these components offer the required functions discussed in Sect. 2.3 and especially how they cope with the challenges identified in Sect. 3.

Frontend Components. The frontend provides functions for model management and model reporting. It communicates with the backend components through a REST API. Model management functionalities include model upload, model comparison, model deployment and model discovery. Model reporting comprises a model landscape visualization which provides a holistic overview of models used in different organizational units and business processes for business analysts. Furthermore, a model dashboard provides multiple KPIs, e.g., about the amount of current models in use and added models over the past months.

Model Metadata Store. The model metadata store contains metadata that are generated in each step of the lifecycle process. These are organized according to the metadata model we provided in Sect. 4.1. The model metadata store enables functions for model discovery, model comparison, and model reporting

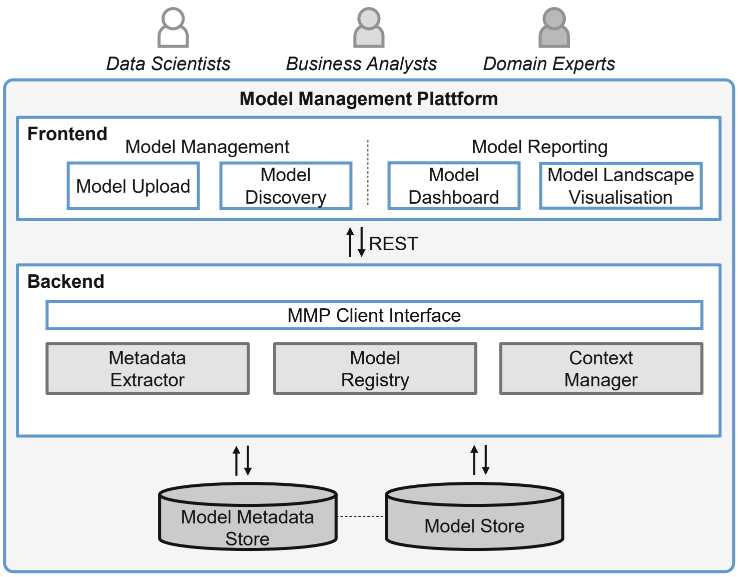

Fig. 4. MMP Architecture showing the backend and frontend components connected via REST interfaces

to store and retrieve information about models. For example, the stored metadata allows users to perform semantic queries to discover related models for a particular use case. By materializing metadata in the model metadata store, the platform can provide faster access than parsing model files at run-time for each user request.

Model Store. The model store is used to store machine learning models that should be materialized for future use. It is closely tied to the model metadata store, because metadata of models need to be linked to materialized model files. Additionally to different serialization formats for ML models, the model store contains information models of machines.

Model Metadata Extractor. The metadata extractor is a functional component that extracts metadata from model files by parsing their content. The model metadata extractor's functionality is of utmost importance to handle the indexing of all the model versions which are created in steps 2) and 5) of the model lifecycle process. By using the metadata, users can search for models with, e.g., specific input attributes or model types (e.g. Regression, Tree Model, Neural Network). The extracted metadata are inserted into the model metadata store and a link is set to the corresponding model file in the model store. For experimental models that should not be materialized, the user can decide that the extractor should only store the available metadata.

Due to the plenty amount of available tools and libraries for machine learning, data scientists also use different formats to persist their model files. A commonly used format is the *Predictive Modeling Markup Language (PMML)*. It is an open standard for an interchange format that is supported by a variety of tools, frameworks, and ML serving systems. As depicted in Fig. 3, grey attributes represent attributes that can be extracted from PMML files. PMML supports a variety of metadata and is therefore a suitable model format for storing models. The format is XML-based and allows to store the model and its metadata in one file. Hence, it can be parsed using common XML parsing libraries. However, there also a lot of other formats, e.g., *scikit-learn* models use *pickle*[1] as serialization format, *RDS*[2] is used for *R*, and *HDF5*[3] for *Tensorflow*. Often, these formats are binary formats that are just readable by the according libraries. For these model formats, the metadata extractor uses the *jpmml*[4] library to convert them into the PMML format. Thus, the metadata extractor also supports users who want to use their custom model formats. The models are kept in their converted and original model file format for users that want to deploy them quickly to different ML serving systems.

Model Registry. The model registry keeps track on model files and the actual status of a model. Thus, the component addresses challenge ①. The status is set for both, model version and model entity. The status of the model entity depends on the collective status information of its underlying model versions. For example, if a model contains multiple model versions with status "experimental" and another model version which is in production, the status of the model entity is set to "production". The same applies when the actual model version becomes stale, then the overall status of the model is "stale". Users can set the status information manually or by inserting specific REST-calls in their training scripts to collect it automatically. For example, if the user rebuilds a model, a setModelStatus-Call can be inserted right before the training routine to set it to maintenance. After the training routine and the deployment, another call sets the status to "production" again. Old model versions can then be set to retired.

Context Manager. The context manager is a component to realize function b) of the use case and to provide a solution to challenge ②. It provides interlinking between models, manufacturing entities, and business entities. In order to support model discovery for domain-related queries, we link machine learning models to assets in the manufacturing environment, e.g., machines. For that purpose, we rely on information models [12] that semantically describe machines. For example, the information model can describe device information, process

[1] Pickle: https://docs.python.org/3/library/pickle.html.
[2] RDS: https://www.rdocumentation.org/packages/base/versions/3.6.1/topics/readRDS.
[3] HDF5: https://www.hdfgroup.org/solutions/hdf5/.
[4] jpmml: https://github.com/jpmml.

variables or machine capabilities. Information models are part of the Open Platform Communications Unified Architecture (OPC UA) [10]. We consider OPC UA information models as suitable because they can be modeled according to companion specifications that are uniform across different machine manufacturers. Therefore, a machine learning model for a certain type of machine can be re-used across machines of the same type from different machine manufacturers. The linking of machine learning models to information models enhances the functions for model discovery. In order to also provide functions for model reporting to business users, it is possible to attach an enterprise architecture repository that contains information about business processes and organizational units. By interlinking these with the model metadata, we can provide powerful visualizations for business analysts such as the model landscape view (see Fig. 5). Such views are common in the scientific field of enterprise architectures and provide a holistic overview [7]. The model Landscape view shows business processes on top, departments on the left and corresponding models highlighted in green color. The view provides two benefits. First, business analysts can get an overview on all the models within the company. Second, they can also identify spots with missing models in the model landscape.

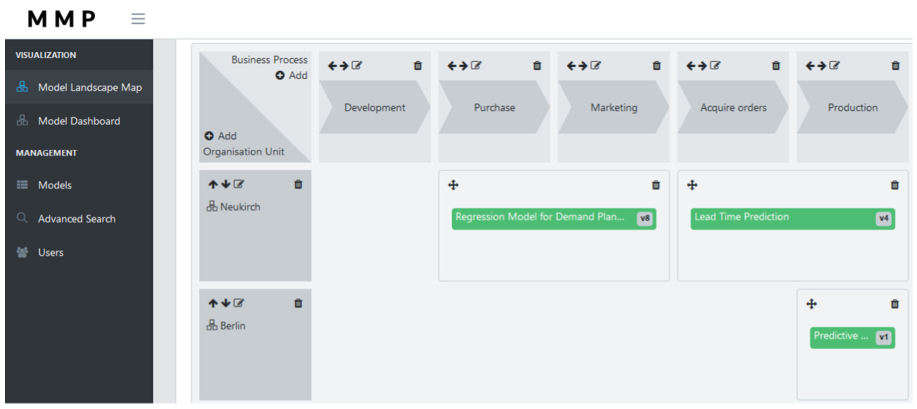

Fig. 5. The MMP Frontend – showing the model landscape of machine learning models in the manufacturing environment. Models (green) are listed with a name and a version number. A grid organizes the models via business processes on top and according organizational units on the left. (Color figure online)

5 Case Study

In this section, we assess how the platform addresses the main challenges ① **Tracking the status of models across their lifecycle**, and ② **Adding context to models for discovery/reporting**. For the assessment, we use the steel plate manufacturer's use case for model management.

The steelplate manufacturer develops a new type of machine that must be supported by a machine learning model for predictive maintenance. In the plan model step, Bob creates a new project on the MMP and adds a label and description to it. For documenting purposes, he also adds the business process "Machine Maintenance" and the department "Machine Maintenance Services" that should be supported by the model. After that, domain experts provide a dataset with historical records and an OPC UA information model of the machine to Bob. With the supported data, he conducts multiple machine learning experiments in the build & test model step. He screens the resulting models and selects the best performing model from a list of candidates. After uploading the model to the platform, the metadata extractor stores the model in the model store and then extracts its metadata and saves it to the model metadata store. Bob assigns the status "experimental" to the models created. In order to support model discovery for future experiments, he also uploads the OPC UA information model to the platform. The information model is parsed and its semantic information is linked to the machine learning model and stored in the model metadata store. He deploys the model into production and sets its status to "production". From time to time, he updates it on detected concept drifts (model status: "stale"). Models that are currently updated are set to status "maintenance" through entries in the training pipeline that interface with the MMP. When a new model version is deployed, the old version is set to status "retired" and the new version to status "production".

Some time later, Alice creates a new project in the MMP for the model she wants to develop. The status "planned" is assigned to the model. She does not know about Bob, but she has access to the model management platform. Therefore, she searches for similar models in the model store that satisfy her use case requirements. For instance, she searches for models that belong to a certain machine and sensor type. She discovers Bob's model as a possible candidate. With this information, she can also get more context, e.g., about the machine in which the sensor is built in and she also retrieves information about Bob, who built the model. She finds out that the model was developed for the same type of machine and that it already satisfies her requirements. Therefore, she copies it into the newly created project. Through the MMP's support, Alice saves time by finding and re-using existing models. The MMP stores a link between the copied version and the original model version of Bob to keep track of the history for later analysis.

Again, some time later, Jill wants to create the report about all models that are used for predictive maintenance in the company. With the MMP, she now can generate reports faster and provide them to the chief digital officer. She searches the repository for models with the status "production" and the term "predictive maintenance" in the use case description. By using the visualization function of the platform, she can display all the models with corresponding business processes and departments, e.g., in a model landscape view (see Fig. 5). This relieves Alice and Bob from the burden of creating such visualizations manually.

Right now, the status experimental can be automatically tracked via REST calls to the MMP which are inserted into the training scripts by data scientists. Candidate models need to be explicitely flagged on the platform, because it is a manual selection process. The same applies to the final model selected. When data scientists create a script to put the model into production, REST calls in this script put the model into status production. Typically such scripts contain a training routine that is also used to retrain a model when a model becomes stale. When the script is executed, the current model version is set to maintenance via a REST call. Stale models that are replaced by new ones are set to retired automatically. If a model is not used anymore data scientists set it to retired.

To sum up, the proposed platform and its function address both challenges introduced in Sect. 3: ① Tracking the status of models across their lifecycle, and ② adding context for model reporting and discovery. This constitutes an advancement to the state of the art, because existing model management platforms to do not offer such dedicated means. More precisely, our platform goes beyond existing platforms in that it provides status information across the whole lifecycle of machine learning models. Other platforms provide this information just for the steps *build and test model* and *use model* according to the process model we provide in Sect. 3. Furthermore, our platform supports adding context to machine learning models to enable functions for search and discovery that are tailored to domain experts and business analysts. Thus, it is the first platform that provides such functionalities for model management.

6 Conclusions and Future Work

Providing structure and context to machine learning models is a core requirement in ML projects. Machine learning models that are well organized facilitate the effective re-use of artifacts and collaboration between data scientists and business analysts. In this paper, we focus on the challenge of adding context to models to enable model discovery and reporting as well as tracking the status of models across their lifecycle. To address these challenges, we propose a model management platform. The model metadata is then extended to include additional context, namely, manufacturing and business metadata for Industry 4.0 use cases. The platform integrates with existing ML environments for training and model serving systems through standardized interfaces.

Our future work will focus on the collection of metadata from operational pipelines and lineage tracking across different steps in the life cycle of machine learning models. In addition, we are going to support more industrial standards for adding context and to develop more views to enhance user experience.

References

1. Breck, E., Cai, S., Nielsen, E., Salib, M., Sculley, D.: The ML test score: a rubric for ML production readiness and technical debt reduction. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 1123–1132. IEEE (2017)

2. Carlson, B.M., Schneider, L., Schuberth, S., et al.: Git large file storage. https://git-lfs.github.com/
3. Ding, J., Tarokh, V., Yang, Y.: Model selection techniques: an overview. IEEE Sig. Process. Mag. **35**(6), 16–34 (2018)
4. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. **46**(4), 44:1–44:37 (2014)
5. Gröger, C., Schwarz, H., Mitschang, B.: The deep data warehouse: link-based integration and enrichment of warehouse data and unstructured content. In: 2014 IEEE 18th International Enterprise Distributed Object Computing Conference, pp. 210–217. IEEE (2014)
6. Gröger, C., Schwarz, H., Mitschang, B.: The manufacturing knowledge repository - consolidating knowledge to enable holistic process knowledge management in manufacturing. In: ICEIS (2014)
7. Kirchner, A., Scheurer, S., Weber, C., Wiechmann, A.: Architektur eines Cockpits zur interaktiven Analyse von Enterprise Architectures auf Basis von Viewpoints. In: Porada, L. (ed.) Informatiktage 2014, pp. 139–142. GI-Edition, Gesellschaft für Informatik, Bonn (2014)
8. Knafla, F., Loewen, U., et al.: Implementation strategy Industrie 4.0: report on the results of the Industrie 4.0 platform. http://www.zvei.org/Publikationen/Implementation-Strategy-Industrie-40-ENG.pdf
9. Kumar, A., McCann, R., Naughton, J., Patel, J.M.: Model selection management systems: the next frontier of advanced analytics. ACM SIGMOD Rec. **44**(4), 17–22 (2015)
10. Mahnke, W., Leitner, S.H., Damm, M.: OPC Unified Architecture, 1st edn. Springer, Berlin (2009). https://doi.org/10.1007/978-3-540-68899-0
11. N.N.: Data version control. https://dvc.org/
12. OPC Unified Architecture: Part 5 : Information Model (2017)
13. Sisinni, E., Saifullah, A., Han, S., Jennehag, U., Gidlund, M.: Industrial internet of things: challenges, opportunities, and directions. IEEE Trans. Ind. Inf. **14**(11), 4724–4734 (2018)
14. Vartak, M., Madden, S.: MODELDB: opportunities and challenges in managing machine learning models. IEEE Data Eng. Bull. **41**, 16–25 (2018)
15. Weber, C., Hirmer, P., Reimann, P., Schwarz., H.: A new process model for the comprehensive management of machine learning models. In: Proceedings of the 21st International Conference on Enterprise Information Systems , ICEIS, vol. 1, pp. 415–422. INSTICC, SciTePress (2019)
16. Weber, C., Wieland, M., Reimann, P.: Konzepte zur Datenverarbeitung in Referenzarchitekturen für Industrie 4.0: Konsequenzen bei der Umsetzung einer IT-Architektur. Datenbank-Spektrum **18**(1), 39–50 (2018)
17. Wuest, T., Weimer, D., Irgens, C., Thoben, K.D.: Machine learning in manufacturing: advantages, challenges, and applications. Prod. Manuf. Res. **4**(1), 23–45 (2016)
18. Zaharia, M., Chen, A., et al.: Accelerating the machine learning lifecycle with MLflow. IEEE Data Eng. Bull. **41**(4), 39–45 (2018)