

The Next Evolution of MDE: A Seamless Integration of Machine Learning into Domain Modeling

Thomas Hartmann*, Assaad Moawad†, Francois Fouquet*, and Yves Le Traon*

*Interdisciplinary Center for Security Reliability and Trust (SnT), University of Luxembourg. email:{first.last}@uni.lu

† DataThings S.A.R.L. email:{first.last}@datathings.com

Abstract—Machine learning algorithms are designed to resolve unknown behaviours by extracting commonalities over massive datasets. Unfortunately, learning such global behaviours can be inaccurate and slow for systems composed of heterogeneous elements, which behave very differently, for instance as it is the case for cyber-physical systems and Internet of Things applications. Instead, to make smart decisions, such systems have to continuously refine the behaviour on a per-element basis and compose these small learning units together. However, combining and composing learned behaviours from different elements is challenging and requires domain knowledge. Therefore, there is a need to structure and combine the learned behaviours and domain knowledge together in a flexible way. In this paper we propose to weave machine learning into domain modeling. More specifically, we suggest to decompose machine learning into reusable, chainable, and independently computable small learning units, which we refer to as micro learning units. These micro learning units are modeled together with and at the same level as the domain data. We show, based on a smart grid case study, that our approach can be significantly more accurate than learning a global behaviour while the performance is fast enough to be used for live learning.

Index Terms—Domain modeling, Live learning, Model-driven engineering, Meta modeling, Cyber-physical systems, Smart grids

I. INTRODUCTION

In order to meet future needs, software systems need to become increasingly intelligent. A prominent example are cyber-physical systems (CPSs) and Internet of Things (IoT) applications, where smart objects are able to autonomously react to a wide range of different situations, in order to minimize human intervention [34]. Advances in software, embedded systems, sensors, and networking technologies have led to a new generation of systems with highly integrated computational and physical capabilities, which nowadays are playing an important role in controlling critical infrastructures, like the power grid. Such systems face many predictable situations for which behaviour can be already defined at design time of the system. In order to react to critical overload situations, for example, the maximum allowed load for customers can be restricted. This is called *known domain knowledge*. In addition, intelligent systems have to face events that are unpredictable at design time. For instance, the electric consumption of a house depends on the number of persons living there, their activities, weather conditions, used devices, and so forth. Despite such behaviour is unpredictable

at design time, it is identifiable and a hypothesis about it can be already formulated and solved later by observing past situations, once data becomes available. Sutcliffe *et al.*, [43] suggest to call this *known unknown*.

To make smart decisions, intelligent systems have to continuously refine behaviour that is known at design time with what can be learned only from live data to solve known unknowns.

a) *Coarse-grained vs. fine-grained learning.*

We distinguish two different learning granularities, coarse-grained and fine-grained. Coarse-grained learning means extracting commonalities over massive datasets in order to resolve unknown behaviours.

Fine-grained learning, on the other hand, means instead of searching for commonalities over the whole dataset, to apply learning algorithms only on specific elements of the dataset. To decide which parts of the dataset should be taken into consideration for which learning algorithm usually requires domain knowledge, *e.g.*, structured in form of domain models.

Nonetheless, nowadays the most common usage of machine learning algorithms is to resolve unknown behaviours by extracting commonalities over massive datasets. Peter Norvig describes machine learning and artificial intelligence as “*getting a computer to do the right thing when you don’t know what that might be*” [37]. Learning algorithms can infer behavioural models based on past situations, which represent the learned common behaviour. However, in cases where datasets are composed of independent and heterogenous entities, which behave very differently, finding one coarse-grained common behaviour can be difficult or even inappropriate. This applies particularly for the domain of CPSs and IoT. For example, considering the electrical grid, the consumption of a factory follows a very different pattern than the consumption of an apartment. Searching for a coarse-grained, common behaviour across all of these entities (the whole or at least large parts of the dataset) is not helpful. Coarse-grained learning alone, which is based on the “*law of large numbers*”, can be inaccurate for systems which are composed of heterogenous elements which behave very differently. In addition, in case of data changes, the whole learning process needs to be fully recomputed, which often requires a lot of time.

Instead, following a *divide and conquer* strategy, learning on

finer granularities can be considerably more efficient for such problems [48], [13]. This principle is, for example, also used in text sentiment [29], where a segmentation by the domain of words can help to reduce complexity. Similarly, multi-granular representations [49] have been applied to solve hierarchical or micro-array-based [11] learning problems. Aggregating small learning units [39] has also been successfully used to build probabilistic prediction models [8]. In accordance to the pedagogical concept [27], we refer to small fine-grained learning units as “*micro learning*”. We believe that micro learning is appropriate to solve the various known unknown behavioural models in systems which are composed of heterogenous elements which behave very diverse and can be significantly more accurate than coarse-grained learning approaches.

b) Modeling ML vs. domain modeling with ML.:

Applying micro learning on systems, such as the electric grid, can potentially lead to many fine-grained learning units. Furthermore, they must be synchronised and composed to express more complex behavioural models. Therefore, an appropriate structure to model learning units and their relationships to domain knowledge is required. Frameworks like TensorFlow [1], GraphLab [32] or Infer.NET [4] also divide machine learning tasks into reusable pieces, structured with a model. They propose a higher level abstraction to model the learning flow itself by structuring various reusable and generic learning subtasks. These approaches focus solely on modeling the learning flow without any relation to the domain model. As a consequence, domain data and its structure is expressed in different models than learning tasks, using different languages and tools and leads to a separation of domain data, knowledge, known unknowns, and associated learning methods. This requires a complex mapping between learning units and domain data. A similar conclusion has been drawn by Vierhauser *et al.*, [44] for monitoring system of systems.

To address this complexity, in this paper we propose to weave micro machine learning seamlessly into data modeling. Specifically, our approach aims at:

- Decomposing and structuring complex learning tasks with reusable, chainable, and independently computable micro learning units to achieve a higher accuracy compared to coarse-grained learning.
- Seamlessly integrating behavioural models which are known at design time, behavioural models that need to be learned at runtime, and domain models in a single model expressed with one modeling language using the same modeling concepts.
- Automating the mapping between the mathematical representation expected by a specific machine learning algorithm and the domain representation [4] and independently updating micro learning units to be fast enough to be used for live learning.

We take advantage of the modeled relationships between domain data and behavioural models (learned or known at design time), which implicitly define a fine-grained mapping

of learning units and domain data. This is a natural extension of basic model-driven engineering approaches.

We implemented and integrated our approach into the open-source framework GreyCat¹. GreyCat is an extension and the successor of the Kevoree Modeling Framework KMF [14]². Like EMF [5], KMF is a modeling framework and code generation toolset for building object-oriented applications based on structured data models. It has been specifically designed for the requirements of CPSs and IoT.

c) Motivating case study.:

Let us consider a concrete use case. We are working together with Creos Luxembourg, the main electrical grid operator in Luxembourg, on a smart grid project. A major challenge in this project is to monitor and profile various data, *e.g.*, consumption data, in order to be able to detect anomalies and predict potential problems, like electric overload, before they actually happen. The important smart grid entities for the context of this paper are *smart meters* and *concentrators*. Smart meters are installed at customers houses and continuously measure electric consumption and regularly report these values to concentrators, where the data is processed. To which concentrator a meter sends its data depends on various conditions, *e.g.*, distance or signal strength and changes frequently over time [19].

For various tasks, like electric load prediction or detection of suspicious consumption values, customers’ consumption data need to be profiled independently and in real time. This is challenging due to performance requirements but also and mainly due to the large number of profiles, which need to be synchronized for every new value. To model such scenarios, we need to express a relation from a machine learning profiler to the consumption of a customer. Since the connections from smart meters to concentrators vary over time, a concentrator profiler depends on the profiles of the currently connected meters. Coarse-grained, in this context, means profiling on the concentrator level, while fine-grained means profiling on a smart meter level and then combining the profiles of the smart meters connected to one concentrator together. Profiling on a concentrator level is often needed to evaluate the electric load situation for a specific geographical region of the grid and many operational decisions are based on this. One coarse-grained profiler at the concentrator level will not take real-time connection changes and their implications in predicting the electric load into account. Coarse-grained profiling alone can be very inaccurate in such cases.

Another example where micro learning and composing complex learning from smaller units can be significantly more accurate than coarse-grained learning are recommender systems. In such systems, coarse-grained learning is to recommend to the users of the same category or user groups, the same products. Fine-grained learning create one micro learning unit per user and/or per product. Again, using only coarse-grained profiles for customers and products can be very

¹<http://greycat.ai/>

²<http://modeling.kevoree.org/>

inaccurate, or generic. In case of recommender systems, micro learning can be even combined with coarse-grained learning by using the coarse-grained learning in cases where the user's fine grained learning does not have enough information to recommend accurately.

The bottom line is that micro learning units and combining them to larger learning tasks are especially useful for systems which are composed of multiple independent entities which behave very differently. CPSs and IoT systems are domains where these characteristics apply specifically.

We evaluate our approach on a concrete smart grid case study and show that:

- Micro machine learning for such scenarios can be more accurate than coarse-grained learning.
- The performance is fast enough to be used for live learning.

d) *Remainder of this paper.:*

The remainder of this paper is as follows. Section II introduces the necessary background. Section III presents our model-based micro machine learning approach. We discuss the meta model definition used in our approach and present a modeling language to seamlessly model machine learning and domain data. In Section IV we evaluate our approach on a smart grid case study, followed by a discussion in Section V. The related work is discussed in Section VI. A conclusion and future work is presented in Section VII.

II. BACKGROUND

In this section we introduce modeling and meta modeling techniques and present an overview of machine learning and meta-learning techniques.

A. Modeling Techniques

Modeling is a fundamental process in software engineering. Over time different formalisms to model and reason about systems have been developed and used for different purposes [41], [24] [2]. For example, entity-relationship models [7], are a general modeling concept for describing entities and the relationships between them. They are widely used to model schemas of relational databases. Ontologies, RDF [30], and OWL [45] are other modeling approaches, which are mainly used in the domain of the Semantic Web. Model-driven engineering (MDE) [28] is probably one of the best known modeling techniques. As an extension of MDE, an emerging paradigm called models@run.time [36] proposes to use models both at design and runtime to support reasoning processes, mainly for CPSs. Most of these approaches have in common that they describe a domain using a set of concepts (classes, types, elements), attributes (or properties), and the relations between them.

Closely related to modeling is the concept of **meta modeling**. A meta model is an abstraction of the model itself. It defines the properties of the model. A model conforms to its meta model, comparable to how a program conforms to the grammar of the language it is written in. The Meta

Object Facility (MOF) [33] proposed by the Object Management Group (OMG) is a popular language for defining meta models. Specifying formal meta information helps to make data machine understandable.

To clarify the used terminology, Figure 1 shows the relations between a meta model, model, and object graphs.

First, the domain is modeled using a meta model, defined in languages like EMF, UML, or other graphical or textual domain specific languages. Then, one or several transformation or generation steps transform the meta model into the actual model, usually implemented in an object-oriented programming language like Java, Scala, or C++. This model is then used in the implementation of an application. During runtime it can be interpreted as an object graph. In this paper we use the terms runtime model and object graph synonymously. To refer to a meta model we use the terms meta model or domain model.

During runtime, application data is usually never static but evolves over time. Nonetheless, for many tasks, like machine learning, it is usually not enough to analyse only the latest data. Different approaches to represent and traverse temporal data have been suggested, *e.g.*, [18], [42]. Regardless of the concrete implementation (in the implementation of our framework we follow the approach presented in [21], [20]), for this paper we assume that our object graphs evolve over time and that we can access historical data.

B. Machine Learning Techniques

Machine learning (ML) is an evolution of pattern recognition and computational learning theory in artificial intelligence. It explores the construction and study of algorithms that can learn from and make predictions on data. It uses algorithms operating by building a mathematical model from example inputs to make data-driven predictions or decisions, rather than strictly static program instructions [46]. The essence of ML is to create compact mathematical models that represent abstract domain notions of profiles, tastes, correlations, and patterns that 1) *fit well* the current observations of the domain and 2) are able to *extrapolate well* to new observations [35].

Several categorisations of ML techniques are possible. We can divide these techniques according to the nature of the used learning: In *supervised learning* data has predefined and well known fields to serve as expected output of the learning process. While in *unsupervised learning* input data is not labeled and does not have a known field defined as output. ML algorithms try to deduce structures present in the input data to find hidden patterns. Many ML algorithms require some parameters (called *hyper-parameters*) to configure the learning process itself. In some situations, these parameters can also be learned or adapted according to the specific business domain. Thus, they are called *meta-learning parameters* and the process of learning such parameters is called **meta learning**. For the rest of the paper we will refer to such parameters simply as *parameters*.

Another categorisation of ML techniques is according to the frequency of learning: In *online learning*, for every new

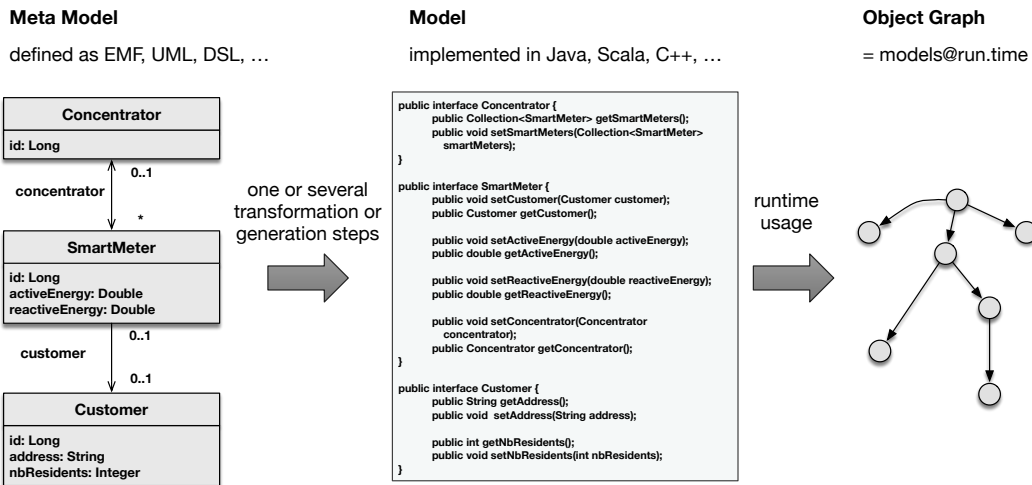


Fig. 1. Relations between a meta model, model, and object graphs

observation of input/output, the learning algorithm is executed and its state is updated incrementally with each new observation. This is also known as live, incremental, or on-the-fly ML. We speak of *offline learning* or *batch learning* when a whole dataset or several observations are sent in “one shot” to the learning algorithm. The learning technique is trained using a small batch or a subset of observations similar to the requested input. This type offers a case-based or context-based reasoning because the learning is tailored for the requested input.

Finally, a ML module can be composed by combining several ML submodules. This is usually called *ensemble methods*. It is often used to create a strong ML model from multiple weaker ML models that are independently trained. The results of the weaker models can be combined in many ways (voting, averaging, linear combination) to improve the overall learning. *Random forests* are a powerful example of these techniques, where the global ML module is composed by several decision trees, each trained on a subset of data and features. *Neural networks* are another example, where the global network is composed by several neurones, each can be seen as an independent learning unit.

A generic modeling framework for ML, should be flexible enough to model any of these ML types. This principle served as a guideline for the development of our framework.

III. WEAVING MICRO LEARNING AND DOMAIN MODELING

In this section we first discuss the objectives of our approach. Then we present the meta model definition (meta-meta model) which we use for the implementation of our approach and detail what exactly micro learning units are. Next, we present the syntax and semantic of our modeling language and show concrete examples of its usage. This section ends with presenting important implementation details.

A. Objective: Domain Modeling with ML

In order to weave micro ML into domain modeling we need to extend modeling languages to model learned attributes and relations and “default” ones seamlessly together. It requires modeling languages to allow to specify in a fine-grained way *what* should be learned, *how* (algorithm, parameters) something should be learned, and *from what* (attributes, relations, learned attributes, learned relations) something should be learned. To be appropriate for live learning, this fine-grained learning units need to be independently computable and updateable.

We use a meta-meta model to define this weaving. A meta-meta model specifies the concepts which can be expressed in a concrete meta model, *i.e.*, it specifies what can be expressed in meta models conforming to it. This allows domain modes to express learning problems. Based on this, we can define a concrete modeling language providing the necessary constructs to weave ML into domain modeling.

B. Meta-Meta Model

We first specify the meta model definition (meta-meta model) underlying our approach. This definition, shown in Figure 2, is inspired by MOF/EMOF and extended with concepts to express machine learning directly in the domain modeling language. Section III-D describes the modeling language we built around this meta-meta model and defines the syntax and formal semantic of the language. Elements related to ML are depicted in the figure in light grey. We focus on these elements since other parts comply with standard meta model definitions, like EMOF or MOF. As can be seen in the figure, we define meta models consisting of an arbitrary number of meta classes and enums. Meta classes in turn have an arbitrary number of properties. Properties are attributes, relations, or what we call “*specified properties*”. Specified properties are

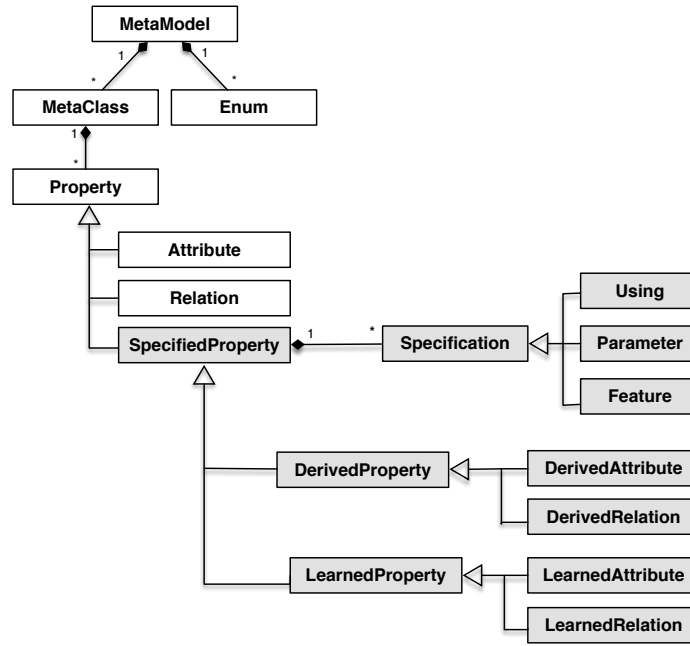


Fig. 2. Meta-Meta Model

either “*learned properties*” or “*derived properties*”. Learned properties are relations or attributes which will be learned by a specific machine learning algorithm. A concrete learning algorithm can be specified with the “*specification*” “*using*”. Parameters for the learning algorithm can be defined with the specification “*parameter*”. The “*feature*” specification allows to access properties from other meta classes or enums.

Derived properties are similar to learned properties, however derived properties don’t have a state associated, *i.e.*, they don’t need to be trained but simply compute a value. The value of a derived attribute is calculated from the values of attributes of other meta classes. Whereas the value of a learned attribute depends on a state and past executions, *i.e.*, on learning. As we will see in Section III-F, this is reflected by the fact that for derived properties we only generate so-called “*infer*” methods whereas for learned properties we generate “*learn*” and “*infer*” methods.

C. Micro Learning Units

The core elements of our approach are micro learning units. As explained in Section I we use the term “micro learning unit” to refer to small fine-grained learning units. These units are designed to decompose and structure complex learning tasks with reusable, chainable, and independently computable elements. Figure 3 illustrates a concrete example of a micro learning unit and set it into relation to the meta and instance levels. In the top left of the figure we see the definition of a `SmartMeter` meta class. Besides two attributes, `activeEnergy` and `reactiveEnergy`, one derived property named `aboveThreshold` and one learned property, which we named `powerProbabilities`, are defined. As will be detailed in Section III-F, specifying

the learned property `powerProbabilities` results in automatically generating the necessary code for the mapping between the internal representation of a machine learning algorithm and domain models. The machine learning algorithm will be “weaved” inside the meta model instances, in this case of `SmartMeter` instances. As illustrated, the micro learning unit is an instance of a learning algorithm, contained in an object and related to a state. It is also related to the instance of the `SmartMeter` class, or more specifically to the learned attribute. In fact, every instance of a `SmartMeter` class has its own (automatically generated) instance of a micro learning unit.

As can be seen in the figure, ML (via learned properties) can be seamlessly integrated and mixed with domain modeling. Section III-D presents our proposed modeling language and details how this can be defined within the concrete syntax of this language. The resultant ability to seamlessly define relationships from learned properties to domain properties and to other learned properties—and vice versa from domain properties to learned properties—enables composition, reusability, and independent computability/updates of micro learning units. An additional advantage of independent micro learning units is that they can be computed in a distributed way. Basically, every learning unit can be computed on a separate machine. Such distribution strategy relies on a shared model state, as for example presented in [22]. The computation can then be triggered in a bulk-synchronous parallel (BSP) way [15] over this shared state.

Our approach is built in a way that the same learning models can be used in several tasks without the need to duplicate it. For example, in the smart metering domain, the electricity consumption profile of a user can be used to: predict the elec-

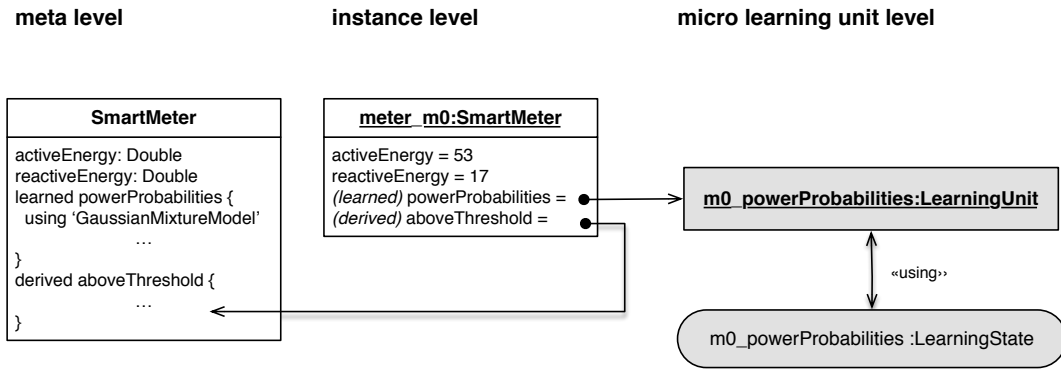


Fig. 3. Schematic representation of a micro learning unit

trical load, classify users according to their profile, or to detect suspicious consumption behaviour. The possibility to compose micro learning units allows a segregation of learning concerns. In case an application requires a combination of different ML techniques, it is not necessary to mash traditional algorithms for each step together. Instead, independent micro learning units can be composed in a divide-and-conquer manner to solve more complex learning problems. This is shown in more detail in Section III-E. In addition, the learning algorithm itself is encapsulated and the mapping between the domain model and the data representation expected by the respective learning algorithm is automatically generated. In this way, the learning algorithm can be easily changed without the need to change the interface for the domain application.

The possibility to derive attributes from others, allows to create richer models. In fact, *ensemble methods* in the ML domain, derive stronger ML models from weaker ML models by combining the results of the smaller units. In our framework, we enable ensemble methods from several learned attributes (learnt through different weaker ML models) by creating a derived attributed that combines their results.

The smart meter profiler is a representative example for micro learning. The profiler works on a specific smart meter instance, instead of profiling, lets say, all smart meters. In addition, this learning unit can be reused and composed. For example, a concentrator profiler can be defined as an aggregation of all smart meter profilers of the smart meters connected to the concentrator. By defining micro learning units in a meta model, the relationships between domain classes and micro learning units are explicitly defined and can be used to infer for which changes a micro learning unit needs to be recomputed.

Even though our approach promotes micro learning, there are nonetheless scenarios where it is helpful to also learn coarse-grained behaviour, *e.g.*, the consumption profile of all customers. Therefore, we allow to specify a scope for learned properties. The default scope is called `local` and means that the learning unit operates on an per instance level. For coarse-grained learning we offer a `global` scope, which means that the learning unit operates on a per class level, *i.e.*, on all

instances of the specified class.

D. Modeling Language

In this section we introduce our modeling language to enable a seamless definition of domain data, its structure, and associated learning units. The following definitions intend to avoid ambiguities and to formally specify the capabilities and limits of our proposed language. The language is inspired by the state of the art in meta-modeling languages (*e.g.*, UML [38], SysML [16], EMF Ecore [5]). The semantic of the language follows the one of UML class diagrams extended by the concept of micro learning units. Many modeling languages, like UML, are graphical. Advantages of graphical modeling languages are usually a flatter learning curve and better readability compared to textual modeling languages. On the other hand, textual modeling languages are often faster to work with, especially for experts. Also, editors and programming environments are easier to develop and less resource hungry for textual languages. A recent study of Ottensooser *et al.*, [40] showed that complex processes and dependencies are more efficient to express in a textual syntax than a graphical one. For these reasons we decided to first implement a textual modeling language. For future work we plan to propose an additional graphical modeling language.

In the following we first present the syntax and grammar of the language followed by a definition of its semantic. The purpose of this formalization is to clearly detail the capabilities and limits of our proposed language, *i.e.*, to formally define what can be expressed with it. Then, we illustrate by means of the concrete smart grid use case how this language can be used to express different combinations of machine learning and domain modeling.

1) *Syntax*: The syntax of our textual modeling language is inspired by Emfatic [9] and is an extension of the language defined in [14]. Listing 1 shows its formal grammar. The parts in bold show the language extensions.

This grammar basically reflects the classic structure of object-oriented programs. Multiplicities of relationships (indicated by the keyword `rel`) are by default unbounded, *i.e.*, too many. Explicit multiplicities can be defined using the `with` clause, *e.g.*, with `maxBound *` or with `minBounds 1`.

Listing 1 Grammar of our modeling language

```

metaModel ::= (class | enum)*
enum ::= 'enum' ID '{' ID (',' ID)* '}'
class ::= 'class' ID parent? '{' property* '}'
property ::= annot* ( 'att' | 'rel' ) ID : ID spec?
parent ::= 'extends' ID (',' ID)*
annot ::= ( 'learned' | 'derived' | 'global' )
spec ::= '{' (feature | using | param)* '}'
param ::= 'with' ID ( STRING | NUMBER )
feature ::= 'from' STRING
using ::= 'using' STRING
  
```

Meta models are specified as a list of meta classes (and enums). Classes, Enums and their Properties are defined similar to Emfatic. To distinguish static, learned, and derived properties, we introduce annotations for attribute and relation definitions. In addition to this, a specification block can optionally refine the behaviour expected from the corresponding property. A specification can contain statements to declare the algorithm to use, feature extraction functions, and meta parameters to configure the used algorithms. Feature extraction statements are using string literals where a OCL-like notation is used to navigate to reachable properties.

2) *Semantic*: Our modeling language follows the formal descriptive semantic and axioms of UML class diagrams, as defined in [50]. We first present the necessary formalism of UML class diagrams and then extend this formalism to include axioms for weaving learned and derived properties into our language. The semantic is defined with respect to the syntax of our language, defined in Section III-D1.

Definition 1: Let $\{C_1, C_2, \dots, C_n\}$ be the set of concrete meta classes in the meta model, we have $\forall x (C_1(x) \vee C_2(x) \vee \dots \vee C_n(x))$ is an axiom

In this definition we state that any object x should be at least (inheritance) an instance of one of the meta classes defined in the meta model. Additionally, given an object x all meta classes verifying $C(x)$ should be linked by a relationship of inheritance following classical UML semantics and as defined in [50]. This inheritance model is not described here for sake of simplicity and to keep the emphasis on learning aspects. In the syntax of our language, the definition of a meta class starts either with the keyword `class` or `enum`.

Definition 2: For each meta attribute att of type T in C , we have: $\forall x, y C(x) \wedge (att(x, y) \rightarrow T(y))$ is an axiom

In the second definition, we are stating that if x is an instance of a meta class C , which has a certain meta attribute att of type T , the value y of this meta attribute should always be of type T . Attributes are defined using the keyword `att` in the syntax of our proposed language.

Definition 3: For each relationship rel from meta class C_1 to another meta class C_2 , we have:

$\forall x, y (C_1(x) \wedge rel(x, y)) \rightarrow C_2(y)$ is an axiom

In this definition, if a meta class C_1 has a relationship rel to a meta class C_2 , and x is an instance of C_1 , having a relation rel to y , this implies that y should be an instance of C_2 . In the syntax of our proposed language, relationships are defined

using the keyword `rel`.

Definition 4: For each relationship rel from meta class C_1 to C_2 , if ' $e_1..e_2$ ' is its multiplicity value, we have:

$\forall x C_1(x) \rightarrow (e_1 \leq ||y|rel(x, y)|| \leq e_2)$ is an axiom.

Similarly, for each meta attribute att in C_1 , if ' $e_1..e_2$ ' is its multiplicity value, we have:

$\forall x C_1(x) \rightarrow (e_1 \leq ||y|att(C_1, x) = y|| \leq e_2)$ is an axiom

In Definition 4, we state that an attribute or a relationship can have minimum and maximum bounds defined in the meta model, and any instance of the meta class should have its attributes and relationships respecting these bounds.

Following the same approach, we extend the classical UML definition of meta class, by adding two new kinds of properties: learned and derived attributes and relations. In particular, a meta learned attribute *learnedatt*, in a meta class C , is a typed attribute of a type T that represents a known unknown in the business domain. It is learned using a machine learning hypothesis. This hypothesis can be created from a parametrized ML algorithm, its parameters, a set of features extracted from the business domain, and a past learned state that represents the best fitted model of the learning algorithm to domain data. A meta derived attribute *derivedatt*, is very similar to the *learnedatt* with the only difference that the deriving algorithm does not depend on a past state but only on extracted features. In other terms, a meta derived attribute, has a type T , a set of extracted features, a deriving parametrized algorithm and its parameters. The same definition applies for learned and derived relations that behave in the same manner than attributes with only a different result type (e.g., collection of nodes as output). In the syntax of our proposed language, derived/learned attributes and relationships are defined with the keywords `derived att`, `derived rel`, `learned att`, and `learned rel`.

A step called **feature selection** in the meta modeling of C_x is required in order to specify the dependencies needed in order to learn *learnedatt* or derive *derivedatt*. The feature selection can be done only over meta attributes reachable within the host meta class C_x . We define this reachability function by the following:

Definition 5: $reach: (metaClass \times metaAtt) \mapsto boolean$
 $reach(C_x, a) = att(C_x, a) \vee learnedatt(C_x, a) \vee derivedatt(C_x, a)$
 $\vee (\exists C_y | rel(C_x, C_y) \wedge reach(C_y, a))$

In this definition, a meta attribute a is considered as reachable from a meta class C_x , either if it is a meta attribute, meta learned attribute, or meta derived attribute within the meta class C_x itself, or if C_x has a relationship to another class C_y , which contains a or it can be reachable from there, using recursively another relationship.

Definition 6: Let F be the set of features to extract in order to learn *learnedatt* in a meta class C , we have:

$\forall f \in F, (f! = learnedatt) \wedge reach(C, f)$ is an axiom.

Similarly, in order to derive *derivedatt*, we have:

$\forall f \in F, (f! = derivedatt) \wedge reach(C, f)$ is an axiom.

In other words, a meta learned or derived attribute can extract their features from the meta attributes defined within

the meta class C (except itself to avoid circular reasoning) or reachable from its relationships in a recursive way.

Definition 7: To summarize, a meta learned attribute *learneddatt* has a type T , a set of feature extractions F , a parameterized learning algorithm alg_{p_1, \dots, p_n} , a set of parameters p_1, \dots, p_n , and an learned state LS .

Moreover, we have: $\forall x, y C(x) \wedge (\text{learneddatt}(x, y) \rightarrow T(y)) \wedge y = alg_{p_1, \dots, p_n}(\text{eval}(F), LS)$ is an axiom.

Similarly, a meta derived attribute *derivedatt* has a type T , a set of feature extractions F , a parameterized learning algorithm alg_{p_1, \dots, p_n} , a set of parameters p_1, \dots, p_n .

We have: $\forall x, y C(x) \wedge (\text{derivedatt}(x, y) \rightarrow T(y)) \wedge y = alg_{p_1, \dots, p_n}(\text{eval}(F))$ is an axiom

In Definition 7, we present that the meta learned or derived attribute is typed in the same manner of classical meta attributes (Definition 2), and the type has to be always respected. By extension, learned and derived relations follow strictly the same definition than learned and derived attributes and therefore will not be repeated here. Moreover, the learned attribute is calculated by executing the parameterized learning algorithm over the extracted features and the learned state. For the derived attribute, it is calculated by executing the parameterized deriving algorithm over only the extracted features. Both learned and derived properties are considered as specified properties, because they require some specifications (features, parameters, algorithm), in order to be calculated. This is depicted in our meta-meta model in Figure 2. Finally, at an instance level, an object state is composed by the state of its classical attributes, relationships, and the states of each of its learned attributes.

As our model has a temporal dimension, every meta attribute has a time dimension, and by extension, the learned state has as well a temporal dimension. All meta attributes, relationships, states, and parameters are replaced by their temporal representation (For example: $att \mapsto att(t)$). For feature extraction, it is possible to extract the same attributes but coming from different points in time as long as the attributes are reachable.

E. Model Learning Patterns

Similarly to how modeling methodologies have led to design patterns to solve common problems, in this subsection we describe patterns to weave machine learning into models. We describe how our language can be used on the concrete smart grid use case with different combinations of machine learning and domain modeling. The section starts with a simple domain model, then explains different combinations of domain data and learning and ends with a more complex example on how different learnings can be composed.

1) *Weaving learned Attributes into Domain Classes:* Let's start with a simple example. Listing 2 shows the definition of a class `SmartMeter`. It contains two attributes `activeEnergy` and `reactiveEnergy` and a relation to a customer. These are the typical domain attributes defining a `SmartMeter` class.

In this class we define a learned attribute `anomaly` that automatically detects abnormal behaviour, based on profiling active and reactive energy. To do so, we specify to use a Gaussian anomaly detection algorithm as learning algorithm. Based on this definition, the code generator of GreyCat generates the `SmartMeter` domain class—including features like persistence—and weaves the necessary machine learning code into it. A template of the underlying Gaussian mixture model algorithm is implemented in GreyCat and used by the generator to weave the machine learning code into the domain class. In this example, the attribute `anomaly` can be seamlessly accessed from all `SmartMeter` instances. In fact, the attribute can be used similar to “normal” ones (*i.e.*, not learned ones), however instead of the default getter and setter methods, the generated API offers a `train` and an `infer` method. This example shows how learned attributes can be seamlessly woven into domain classes.

Listing 2 Meta model of a smart meter with anomaly detection

```
class SmartMeter {
    att activeEnergy: Double
    att reactiveEnergy: Double
    rel customer: Customer
    learned att anomaly: Boolean {
        from "activeEnergy"
        from "reactiveEnergy"
        using "GaussianAnomalyDetection"
    }
}
```

2) *Defining a Learning Scope for coarse-grained Learning in Domain Models:* Listing 3 shows an example of a power classification problem. In this listing, first an enumeration `ConsumptionType` with three categories of consumption types (low, medium and high) is defined. Then, we extend the class `SmartMeter` to add a global `classify` attribute which classifies users according to their consumption behaviours. It learns from `activeEnergy`, `reactiveEnergy`, and `nbResidents`.

This example shows coarse-grained learning, where all instances of a domain class contribute to one learning unit. It demonstrates that attribute extractions cannot only happen at the level of attributes of the current instance but also to any reachable attribute from the relation of the current instance. In this example, the attribute `nbResidents`, which is the number of residents within the household of each customer, is extracted from a concrete `Customer` instance of a concrete `SmartMeter` instance. Moreover, it shows how to specify the machine learning hyper-parameters (here the learning rate and regularization rate) within the learned attribute using the keyword `with`. With this definition, GreyCat generates, besides the enum `ConsumptionType`, a domain class `SmartMeter`. As in the previous example, the machine learning code for the linear classification is directly woven into the generated domain class. Again, a template of a linear classification algorithm is integrated in GreyCat and used by the generator to generate the concrete code.

Listing 3 Meta model of a power classifier

```
enum ConsumptionType { LOW, MEDIUM, HIGH }
class SmartMeter{
    [...]
    global learned att classify: ConsumptionType {
        from "customer.nbResidents"
        from "activeEnergy"
        from "reactiveEnergy"
        with learningRate 0.001
        with regularizationRate 0.003
        using "LinearClassifier"
    }
}
```

3) *Modeling Relations between Learning Units and Domain Classes*: Listing 4 shows the meta class of a `SmartMeterProfiler`. In a first step we define that such profilers have relationships to `SmartMeter` instances and vice versa. Then, we extract several attributes from this relationship. For instance, we get the hour of the day (with a `GreyCat` built-in function `Hour(date)`), the active and reactive energy and calculate the square value. Attribute extractions can be any mathematical operations over the attributes that are reachable from the relationships defined within the class. In this example, the profiler learns the probabilities of the different power consumptions, hourly based, using a Gaussian mixture model algorithm [23]. For this scenario, `GreyCat` generates the domain classes `SmartMeter` and `SmartMeterProfiler`. The machine learning code, based on a template implementation of a Gaussian mixture model algorithm, is injected into the generated code. The `SmartMeterProfiler` is generated as a regular domain class (with a learned attribute).

Listing 4 Meta model of a smart meter profiler

```
class SmartMeterProfiler {
    rel smartMeter: SmartMeter
    learned att powerProbabilities: Double[] {
        from "Hour(smartMeter.time)"
        from "smartMeter.activeEnergy^2"
        from "smartMeter.reactiveEnergy^2"
        using "GaussianMixtureModel"
    }
}

class SmartMeter {
    [...]
    rel profile: SmartMeterProfiler
}
```

4) *Decomposing complex Learning Tasks into several Micro Learning Units*: For the last example, we show how to use domain information to derive an advanced profiler at the concentrator level using the fine-grained profilers at the smart meters. First, we define a class `Concentrator` that contains relations to the connected smart meters. Then, we define a `ConcentratorProfiler` with a relation to an `Concentrator` and vice versa. Inside this profiler, we derive an attribute `powerProbabilities` using the key-

word `derived` and using an aggregation function that combines the probabilities from the fine-grained profiles. This example shows how fine-grained machine learning units can be combined to larger ML units. Similar to the previous examples, `GreyCat` generates, based on this definition, two domain classes: `Concentrator` and `ConcentratorProfiler`.

Listing 5 Meta model of a concentrator and its profiler

```
class Concentrator {
    rel connectedSmartMeters: SmartMeter
    rel profile:ConcentratorProfiler
}

class ConcentratorProfiler {
    rel concentrator: Concentrator
    derived att powerProbabilities: Double[] {
        from concentrator.connectedSmartMeters
            .profile
        using "aggregation"
    }
}
```

5) *Coarse-grained Learning*: As discussed, our approach also allows coarse-grained learning. The following example shows how coarse-grained learning can be expressed with our proposed language. A class `ConcentratorProfiler` is used to profile the consumption values of all connected smart meters using a `GaussianMixtureModel` algorithm. This example is similar to the previous one but instead of aggregating the fine-grained learned profiles of the individual smart meters (fine-grained learning), in this example we directly profile the consumption values of the smart meters connected to a concentrator in a coarse-grained manner.

Listing 6 Meta model of a coarse-grained consumption profiler

```
class Concentrator {
    rel connectedSmartMeters: SmartMeter
    rel profile:ConcentratorProfiler
}

class ConcentratorProfiler {
    rel concentrator: Concentrator
    learned att powerProbabilities: Double[] {
        from "Hour(concentrator.
            connectedSmartMeters.time)"
        from "concentrator.connectedSmartMeters.
            activeEnergy^2"
        from "concentrator.connectedSmartMeters.
            reactiveEnergy^2"
        using "GaussianMixtureModel"
    }
}
```

F. Framework Implementation Details

Our approach is implemented as a full modeling environment integrated into IntelliJ IDE³. The development process with our framework follows default MDE approaches, starting

³<https://www.jetbrains.com/idea/>

with a meta model definition. The complete *LL* grammar of our extended modeling language is available as open-source⁴. Therefore, our framework contains a code generator based on Apache Velocity⁵ to generate APIs for object-oriented languages. Currently, our generator targets Java and TypeScript.

The generated classes can be compared to what is generated by frameworks like EMF. In the following, we focus on the ML extensions. According to what is defined in the meta model, our code generator “weaves” the concrete machine learning algorithms into the generated classes and also generates the necessary code to map from a domain representation (domain objects and types) to the internal mathematical representation expected by the learning algorithm (double arrays, matrices, etc) and vice versa. Various machine learning algorithms can be integrated in our framework. Currently, we implemented the following algorithms:

- *Regression*: Live linear regression
- *Classification*: Live decision trees, Naive Bayesian models, Gaussian Bayesian models
- *Clustering*: KNN, StreamKM++
- *Profiling*: Gaussian Mixture Models (Simple & Multinomial)

For every derived property our generator adds an `infer` method to the generated class, which contains the code to compute the property according to its meta model definition. Similar, for every learned property our generator adds an `infer` to read the state of the learning unit and a `train` method to trigger the injected learning algorithm.

Since our framework targets CPSs and IoT applications it has a strong focus on performance. Thus, we do not rely on in-memory models but instead on a specialized graph storage. This has been developed to handle the high volatility of learning unit states.

Since relationships between domain classes and micro learning units are explicitly defined, they can be used during runtime to infer for which changes a micro learning unit needs to be recomputed. This is realized using change listeners and an asynchronous message bus. As a result, our framework supports fully independent updates of learning units. Leveraging the underlying shared graph storage model this can even be done in a distributed manner.

IV. EVALUATION

In this section we evaluate our approach based on two key performance indicators: 1) can micro machine learning be more accurate than coarse-grained learning and 2) is the performance of using micro machine learning fast enough to be used for live learning.

A. Setup

We evaluate our approach on the smart grid use case introduced in Section I. We implemented a prediction engine for customers’ consumption behaviour using our modeling

framework. This engine predicts the consumption behaviour based on live measurements coming from smart meters. We implemented this evaluation twice, once with a classical coarse-grained approach and another time with our micro learning based approach. The goal is to demonstrate that our micro learning-based approach can be more accurate while remaining fast enough to be used for live learning.

For our evaluation we consider 2 concentrators and 300 smart meters. We use publicly available smart meter data from households in London⁶. The reason why we use publicly available data instead of data from our industrial partner Creos is that this data is confidential what would prohibit to publish this data for reproducibility. Our evaluation is based on 7,131,766 power records, from where we use 6,389,194 records for training and 742,572 records for testing. The used training period is 15/08/2012 to 21/11/2013 and the testing period from 21/11/2013 to 08/01/2014.

For the first evaluation, we use a coarse-grained profiler on the concentrators. All smart meters send their data regularly to concentrators where the sum of all connected smart meters is profiled. In a second evaluation we use our micro learning-based approach and use one individual profiler for every smart meter and define an additional profiler for every concentrator, which learn from the individual profilers of the connected smart meters. As learning algorithm we use in both cases Gaussian mixture models, with 12 components, profiling the consumption over a 24 hours period, resulting in 2-hours resolution ($24/12=2$). We train the profilers for both cases during the training period, then we use them in the testing period to estimate/predict the power consumptions for this period.

We simulate regular reconfigurations of the electric grid, *i.e.*, we change the connections from smart meters to concentrators. This scenario is inspired by the characteristics of a typical real-world smart grid topology, as described in [19]. Every hour we randomly change the connections from smart meters to concentrators. At any given point in time, each concentrator has between 50 and 200 connected meters.

We performed all evaluations on an Intel Core i7 2620M CPU with 16 GB of RAM and Java version 1.8.0_73. All evaluations are available at GitHub⁷.

We use the traditional holdout method, where the dataset is separated into a training set and a testing set, instead of a k-fold cross-validation method. When it comes to time-series, the seasonal effect can introduce a bias when splitting the dataset in equivalent sets, required by the k-fold cross-validation method [12]. Moreover, in our evaluation we want to demonstrate the accuracy of modelling with micro learning units rather than evaluating the efficiency of the ML algorithm itself.

⁴<https://github.com/kevoree-modeling/dsl>

⁵<http://velocity.apache.org/>

⁶<http://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households>

⁷<https://github.com/kevoree-modeling/experiments>

B. Accuracy

First, we compare the coarse-grained profiling to the micro learning approach to predict the power consumption over the testing set. Figure 4 shows the results of this evaluation. In both plots, the blue curve represents the testing dataset, *i.e.*, the real power consumption that has to be predicted.

The coarse-grained profiler is not affected by the topology changes. In fact, the profiler at the concentrator level has learned an average consumption that is always replayed without considering the connected smart meters. This explains the periodic, repetitive aspect of the prediction curve.

In contrary, the micro learning approach defines a profiler on the concentrator as a composition of the profilers of all connected smart meters, as shown in the meta model in Listing 6. In case the topology changes, *e.g.*, a smart meter disconnects, the concentrator profiler (composed of several smart meter profilers) will no longer rely on the profiler of the disconnected smart meter. As depicted in Figure 4, for the micro machine learning profiling, the plotted curve is significantly closer to the curve of the real testing set than the coarse-grained learning. Although, both uses the same profiling algorithm: a Gaussian mixture model. For readability reasons we only display the first 12 days of predictions. Prediction curves in case of micro learning are very close (even hard to distinguish) to the real testing set.

We plot the histogram of the prediction errors for both, coarse-grained and micro learning in Figure 6. It shows the distribution of the prediction error of both cases. Overall, micro learning leads to an average error of 3,770 wh, while coarse-grained learning leads to an average error of 6,854 wh. In other words, the error between the prediction and real measurement is divided by two. Knowing that the average power consumption overall the testing set is 24,702 wh, we deduce that the micro learning profiling has an accuracy of 85%, while coarse-grained learning has an accuracy of 72%. The accuracy is calculated by $(1 - \text{avgError}/\text{avgPower})$. Figure 5 depicts the average prediction error and associated confidence interval for both methods: fine-grained and coarse-grained. We can observe that the confidence intervals are around 12kWh for the fine-gained method and respectively 21kWh for the coarse-grained approach. Based on these results, we can conclude that micro learning can be significantly more accurate than coarse-grained learning.

A noticeable result is that the same algorithm can lead to a better accuracy when used at a smaller level and combined with the domain knowledge. Therefore, we argue that this decision is very important and motivate by itself the reason why we focus this contribution on offering modeling abstractions for this purpose.

C. Performance

In terms of performance, Table I shows the time needed in seconds to load the data, versus the time needed to perform the live profiling for different numbers of users and power records. For instance, for 5000 users and their 150 million power records, it takes 1927 seconds to load and parse the

Number of users	Number of records	Loading data time in s.	Profiling time in s.
10	283,115	4.28	1.36
50	1,763,332	21.94	7.20
100	3,652,549	44.80	14.44
500	17,637,808	213.80	67.12
1000	33,367,665	414.82	128.53
5000	149,505,358	1927.21	564.61

TABLE I
LOADING TIME AND PROFILING TIME IN SECONDS. SCALABILITY TEST
OVER 5000 USERS AND 150 MILLIONS POWER RECORDS

whole dataset from disk (around 32 minutes, knowing that the dataset is around 11 GB large). However, only 564 seconds are spent for profiling (less than 10 minutes).

Another observation that can be deduced from Table I is that both loading and training time are linear with the number of records loaded ($O(n)$ complexity). A considerable performance increase can be achieved by distributing and parallelizing the computation, especially using micro learning where every profile can be computed independently. We decided to present results without the usage of a distributed storage backend (*e.g.*, HBase⁸). This would pollute computation times due to networking and caching effects. However, our results allow to meet the performance requirements of case studies like the smart grid. Indeed, during these evaluations our modeling framework ingest more than 60,000 values per seconds on a single computer. This is comparable to data processing frameworks like Hadoop [6]. Moreover, fine-grained machine learning units can be computed independently and can therefore be easily processed in parallel. In fact, every learning unit can naturally be computed in an own process.

D. General applicability of the presented approach and modeling language

In this section, we show the general applicability of our approach and how it can be applied to different domains. Therefore, we discuss examples from different domains and show how they can be modeled using our approach and proposed modeling language. This shows the benefits of a seamless integration of machine learning into domain modeling.

Let us take recommender systems as a first additional example outside the smart grid domain. In recommender systems, the goal is to monitor prior actions of users in order to recommend potential future actions. Applied to sales, for instance, this can be translated into potential next items to sell or next movies to watch. Different types of recommender systems exist [25]. Some recommender systems cluster users to similar behaviours and thus recommend the items to buy according to what other users of the same behaviour group already bought. These system are known as *user-user* recommender systems [25]. Other recommender systems cluster

⁸<https://hbase.apache.org/>

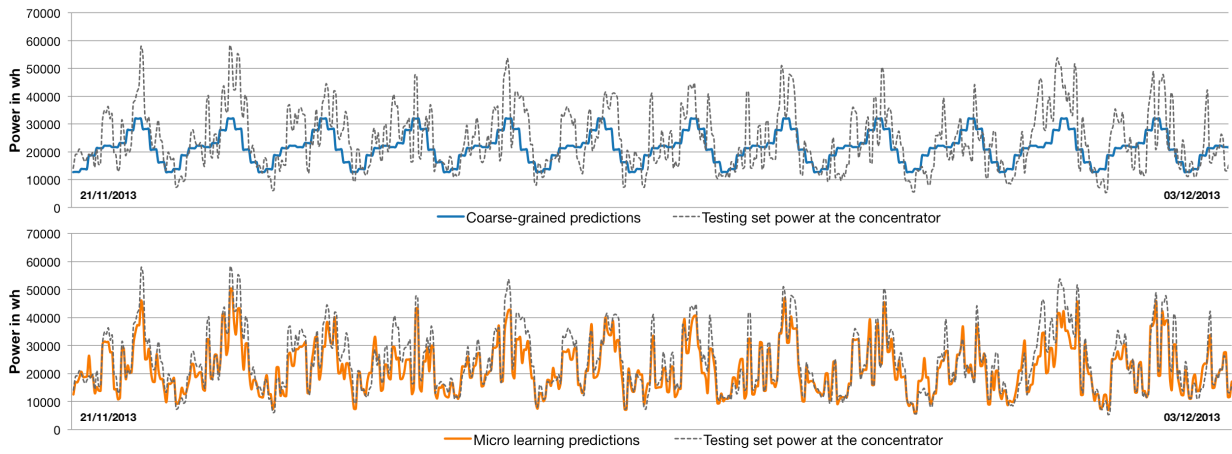


Fig. 4. Coarse-grained profiling (*top*) vs micro learning profiling (*bottom*)

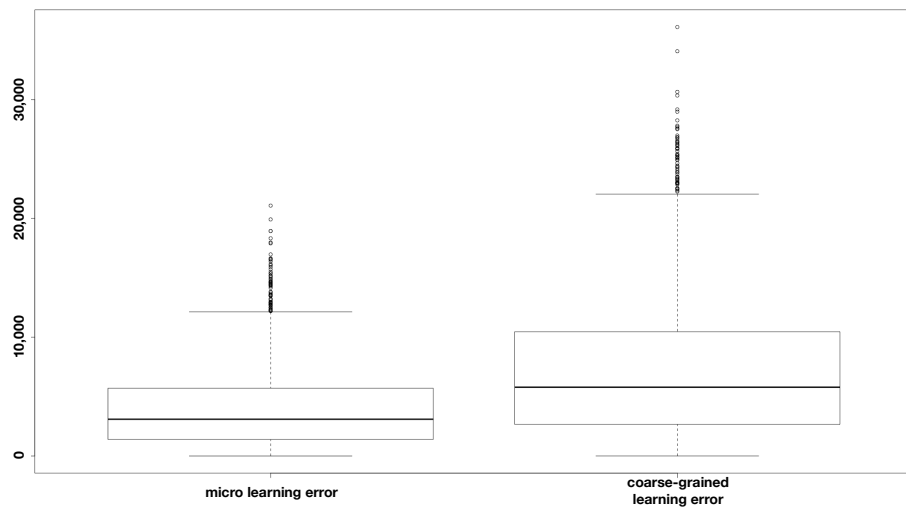


Fig. 5. Average prediction error and confidence intervals (*in Watt per Hours, Wh*)

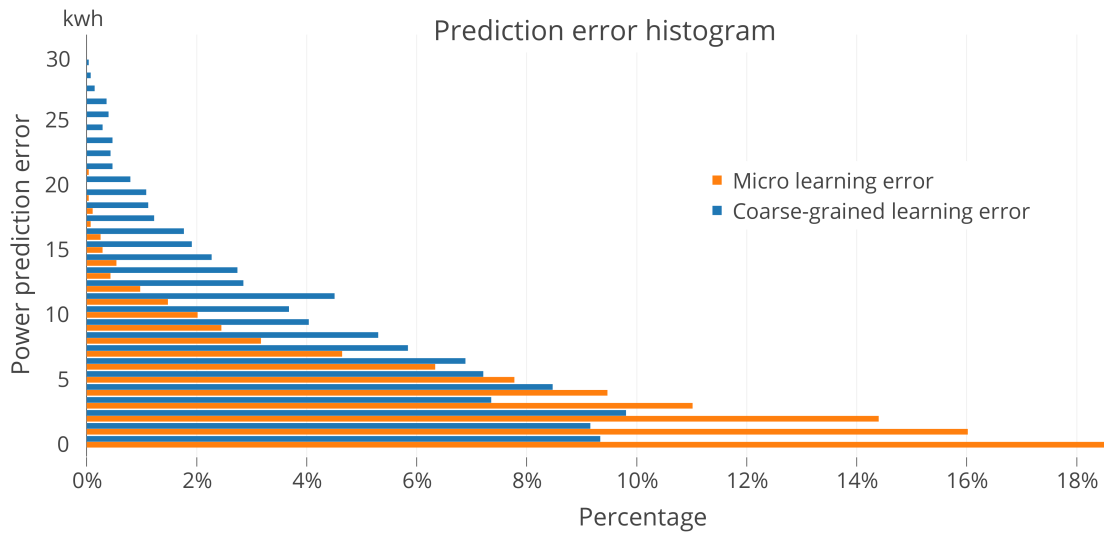


Fig. 6. Power prediction error histograms

items according to their similarities or complementarity and thus suggest to a user to buy the items that are usually bought together. These systems are known as *item-item* recommender systems [25]. Other systems ask users about their preferences and from these preferences they recommend the most suitable products. These systems are known as *user-item* recommender [25].

With our proposed modeling language and approach, we can integrate these 3 types within the same model, thus allowing system designers to change from one type of recommendation system to another—or even have all 3 types of recommendations at the same time, at a minimum cost (by learning the profiles once, and reusing many times). For instance, instead of going to a coarse-grained recommender system by grouping users or items together, we can go to a more fine-grained approach, by attaching a profile to every user and to every product. These profiles represent an abstract mathematical notion of taste in a N-dimensional space in which we can quickly compare users or items together. Moreover, these profiles can be updated in live after every purchase. Then, in order to achieve a user-user recommender system, we can create a derived clustering algorithm that compares and groups users with similar profiles together. The same can be done for the item-item recommender systems by clustering the products with the same profiles together. A user-item recommender system can be achieved by a derived algorithm that fast search for products that match a user profile with an item profile. This way, we manage to separate the different concepts in different layers that are reusable. Moreover, we can reuse business knowledge in machine learning (for instance by not recommending past items already bought if the learning algorithm has access to the historical purchases of the users), and vice versa, by taking business decisions based on machine learning results (recommending new product to sell). Listing 7 shows an example meta model of such a recommender system.

A second example is the domain of transportation systems. The goal is to optimize the public transportation by suggesting to people different transportation alternatives. Again, in this domain, machine learning can be modeled by fine-grained profilers and recommender systems can be built on top of these profilers. For instance, in [47] the authors create profiles for each of the following:

- price of taxi fare per distance unit according to the hour of the day
- traffic on different road segments
- parking place availabilities

Each of these profiles can be modeled as a completely independent, fine grained, and reusable learning unit in our modeling language. A recommender system can calculate the recommendation by deriving the information from these different learning units. Moreover, the advantage of our framework is that the business domain knowledge is at the same level as the learned knowledge. For instance, a learning unit can depend directly on the bus or train schedules, if they are known in advance. Listing 8 shows an example meta model of how such

Listing 7 Meta model of a recommender system

```
enum Category { ELECTRONIC, BOOKS,
                MUSIC, MOVIES, ... }
class Index{
    rel users: User
    rel products: Product
}

class User {
    att userId: Long
    att name: String
    [...]
    rel purchasedProducts: Product
    rel profile: UserProfile
        with maxBound 1
}

class Product {
    att productId: Long
    rel category: Category
        with minBound 1
        with maxBound 1
    att price: Double
    [...]
    rel purchasedBy: Customer

    rel profile : ProductProfiler
        with maxBound 1
}

class UserProfile {
    rel user: User
    learned rel userProfile:
        TasteProfile {
            from "user.purchasedProducts"
            using "IncrementalSVD"
        }
}

class ProductProfiler {
    rel product: Product
    learned rel productProfile:
        TasteProfile {
            from "product.category"
            from "product.price"
            using "IncrementalSVD"
        }
}

class TasteProfile {
    att svdVector: double[]
}

class UserUserRecommender {
    rel index: Index
    derived rel similarUsers: User {
        from index.users.profile
        using "ClusteringAlg"
    }
}

class ItemItemRecommender {
    rel index: Index
    derived rel similarItems: Item {
        from index.items.profile
        using "ClusteringAlg"
    }
}

class UserItemRecommender {
    rel index: Index
    rel currentUser: User
    derived rel directRecommender: Item{
        from currentUser.profile
        from index.items.profile
        using "SimilarityAlg"
    }
}
```

transportation recommender system could be modeled in our proposed approach.

E. Threats to validity

We decided to evaluate our approach on an end-to-end real-world case study. Despite that we showed the usefulness of the approach for other domains, one threat to validity remains that the evaluation case study might be especially appropriate for the presented solution. Additional case studies need to be considered to better estimate the general applicability of the presented approach. Nonetheless, the evaluated case study is representative for the domains targeted by our approach. Another threat to validity might be the sampling rate of the smart meter measurements of the used case study, which could affect the error rate, *e.g.*, missing peaks due to averaging intervals. However, the used sampling rate is already comparatively low with respect to the used dataset. Therefore, this risk is rather low.

V. DISCUSSION

Weaving machine learning into domain modeling opens up interesting possibilities in the intersection of meta learning and meta modeling. Meta learning is about learning the parameters of the learning class itself and adapting these parameters to the specific business domain where the learning is applied to. The following points are considered as typical meta learning problems:

- Changing the inference algorithm.
- Adding or removing more input attributes.
- Modifying the math expression of an attribute.
- Changing learning parameters (for ex. learning rate).
- Chaining or composing several learning units.

Such changes can be introduced during the execution of the system, reflecting a new domain knowledge that have to be injected. Therefore, considering that we model learning parameters, this makes it necessary to enable meta class changes at runtime. This feature is enabled in our modeling framework. However, changing learning algorithms or parameters can occur more often than classical meta model changes. This opens up the reflection on new research directions about frequent meta model updates.

We developed our modeling framework for micro learning. Nonetheless, as discussed, we support fine-grained but also coarse-grained learning. However, our framework—and approach—is clearly designed for micro learning and is therefore mainly useful for systems which are composed of several elements which behave differently. Examples for such systems are CPSs, IoT, and recommender systems. For systems dealing mainly with large datasets of “flat data”, *i.e.*, unstructured data without complex relationships between, our model-based micro learning approach is less beneficial. Instead, our approach is mostly beneficial for systems dealing with complex structured and highly interconnected domain data which have to continuously refine behavioural models that are known at design time with what can be learned only from live data to solve known unknowns. A current restriction of our approach

Listing 8 Meta model of a transportation recommender system

```
enum Transportation {CAR, TAXI, BUS, TRAIN,
                    BICYCLE, WALKING}

class Index {
  rel users: User
  rel taxis: Taxi}

class User {
  att userId: Long
  att name: String
  att GPSLongitude: double
  att GPSLatitude: double
  rel preferredTransportationMeans:
    Transportation
  rel userProfile: PositionProfiler}

class Taxi {
  att taxiId: Long
  att name: String
  att GPSLongitude: double
  att GPSLatitude: double
  att pricePerKm: double
  rel taxiProfile: PositionProfiler}

class PositionProfiler {
  rel user: User
  learned att userProfile: double[] {
    from "user.GPSLongitude"
    from "user.GPSLatitude"
    using "GaussianMixtureModel"}}

class TaxisPriceProfilers {
  rel index: Index
  derived att averageTaxiPrice: double {
    from index.taxis.pricePerKm
    using "Averaging"}}

class RoadSegment {
  att roadId: Long
  att gpsLongitudeStart: double
  att gpsLongitudeEnd: double
  att gpsLatitudeStart: double
  att gpsLatitudeEnd: double
  att currentTraffic: int
  learned att roadTrafficProfile: double[] {
    from "currentTraffic"
    using "GaussianMixtureModel"}}

class Parking {
  att parkingId: Long
  att parkingName: String
  att currentEmptyPlaces: int
  learned att emptyPlaceProfile: double[] {
    from "currentEmptyPlaces"
    using "GaussianMixtureModel"}}

class BusLine {
  att busLineId: Long
  att busLineName: String
  att busSchedule: double[]}

class TrainLine {
  att trainLineId: Long
  att trainLineName: String
  att trainSchedule: double[]}

class Map {
  rel roads: RoadSegment
  rel busLines: BusLine
  rel trainLines: TrainLine
  rel parkings: Parking}

class TransportationRecommender {
  rel user: User
  rel taxiPriceProfiler: TaxisPriceProfilers
  rel map: Map
  derived att recommendation: Transportation {
    from user.userProfile
    from taxiPriceProfiler.averageTaxiPrice
    from map.roads.roadTrafficProfile
    from map.trainLines.trainSchedule
    from map.busLines.busSchedule
    from map.parkings.emptyPlaceProfile
    using "customTransportationAlgorithm"}}
```

is that it considers only known unknowns, *i.e.*, it is necessary to know what is unknown and what can be learned. Moreover, our approach focuses on live learning scenarios where only small learning units, which individually are fast to recompute, have to be updated. While this is especially useful in cases where only few micro learning units, *i.e.*, only parts of the model need to be updated, it is less beneficial for cases where the whole model needs to be recomputed. In such cases, batch learning methods can be more efficient.

VI. RELATED WORK

TensorFlow [1] is an interface for expressing machine learning algorithms and an execution engine to execute these on a wide range of devices from phones to large clusters. A TensorFlow computation is represented as a directed graph. Nodes in the graph represent mathematical operations, called *ops*, while the edges represent multidimensional data arrays, called *tensors*. An op takes zero or more tensors, performs computations, and produces zero or more tensors. Two phases are distinguished in TensorFlow. A construction phase where the graph is assembled and an execution phase which uses a session to execute ops in the graph. TensorFlow is used within Google for a wide variety of projects, both for research and for use in Google's products. Similar to our approach, TensorFlow allows to model ML at a higher level of abstraction. However, unlike in our approach ML is expressed in its own model aside from the domain model and not connected to it. TensorFlow is adapted for image and video recognition, whereas our approach is adapted for learning from frequently changing domain data.

GraphLab [32] goes in a similar direction than TensorFlow. Low *et al.*, propose an approach for designing and implementing efficient and provably correct parallel ML algorithms. They suggest to use a data graph abstraction to encode the computational structure as well as the data dependencies of the problem. Vertices in this model correspond to functions which receive information on inbound edges and output results to outbound edges. Data is exchanged along edges between vertices. GraphLab aims at finding a balance between low-level and high-level abstractions. In contrary to low-level abstractions GraphLab manages synchronization, data races, and deadlocks and maintains data consistency. On the other side, unlike high-level abstractions GraphLab allows to express complex computational dependencies using the data graph abstraction. In Low *et al.*, [31] present a distributed implementation of the GraphLab abstraction. Like TensorFlow, GraphLab is an interface for expressing ML algorithms and an execution engine. While there are similarities, like the idea that ML algorithms should be expressed with a higher-level abstraction, our approach focuses on weaving ML algorithms into domain modeling. This allows to use results from learning algorithms in the same manner than other domain data.

In [4] Bishop proposes a model-based approach for ML. He introduces a modeling language for specifying ML problems and the corresponding ML code is then generated automatically from this model. As a motivation Bishop states

the possibility to create highly tailored models for specific scenarios, as well as for rapid prototyping and comparison of a range of alternative models. With *InferNET* he presents a framework for running Bayesian inference in graphical models. Similar to Bishop we propose to express ML problems in terms of a modeling language and automate the mapping of a domain problem to the specific representation needed by a concrete ML algorithm. While Bishop suggests to specify ML problems in separate models with a dedicated modeling language, our approach extends domain modeling languages with the capability to specify ML problems together with domain models using the same modeling language. This allows to decompose learning into many small learning units which can be seamlessly used together with domain data.

Domingos *et al.*, [10] propose an approach for incremental learning methods based on Hoeffding bounds. They suggest to build decision trees on top of this concept and show that these can be learned in constant memory and time per example, while being very close to the trees of conventional batch learners. With Massive Online Analysis (MOA) [3] Bifet *et al.*, present an implementation and a plugin for WEKA [17] based on Hoeffding trees. Our contribution is a methodology to weave micro ML into data modeling to support applications which need online analysis of massive data streams.

Hido *et al.*, [26] present a computational framework for online and distributed ML. Their key concept is to share only models rather than data between distributed servers. They propose an analytics platform, called Jubatus, which aims at achieving high throughput for online training and prediction. Jubatus focus on real-time big data analytics for rapid decisions and actions. It supports a large number of ML algorithms, *e.g.*, classification, regression, and nearest neighbour. Jubatus only shares local models, which are smaller than datasets. These models are gradually merged. Jubatus, like our approach, allows independent and incremental computations. However, Jubatus doesn't aim at combining domain modeling and ML, neither does it allow to decompose a complex learning task into small independent units, which can be composed.

VII. CONCLUSION AND FUTURE WORK

Coarse-grained learned behavioural models do not meet the emerging need for combining and composing learnt behaviours at a fine-grained level, for instance for CPSs and IoT systems, which are composed of several elements which are diverse in live behaviours. In this paper we proposed an approach to seamlessly integrate micro machine learning units into domain modeling, expressed in a single type of model, based on one modeling language. This allows to automate the mapping between the mathematical representation expected by a specific machine learning algorithm and the domain representation. We showed that by decomposing and structuring complex learning tasks with reusable, chainable, and independently computable micro learning units the accuracy compared to coarse-grained learning can be significantly improved. We demonstrated that the ability to independently compute and

update micro learning units makes this approach fast enough to be used for live learning. Besides simplifying the usage (flatter learning curve), a graphical language can be more intuitive for many users, especially for non developers. We are also working on integrating additional machine learning algorithms in our framework to make it applicable for a broader range of problems. For example, for stream clustering, we are planning to include and experiment with algorithms like cluStream, clusTree, DenStream, D-Stream, and CobWeb. In addition, we are experimenting with adding GPU support for the computation of ML algorithms to our framework to investigate the advantages and disadvantages of it for different use cases.

REFERENCES

- [1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
- [2] Bettini, C., Brdiczka, O., Henriksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.* **6**(2), 161–180 (2010). DOI 10.1016/j.pmcj.2009.06.002. URL <http://dx.doi.org/10.1016/j.pmcj.2009.06.002>
- [3] Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: Moa: Massive online analysis. *The Journal of Machine Learning Research* **11**, 1601–1604 (2010)
- [4] Bishop, C.M.: Model-based machine learning. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* **371**(1984) (2012). DOI 10.1098/rsta.2012.0222. URL <http://rsta.royalsocietypublishing.org/content/371/1984/20120222>
- [5] Budinsky, F., Steinberg, D., Ellersick, R.: Eclipse Modeling Framework : A Developer’s Guide (2003)
- [6] Carstou, D., Cernian, A., Olteanu, A.: Hadoop hbase-0.20. 2 performance evaluation. In: *New Trends in Information Science and Service Science (NISS)*, 2010 4th International Conference on, pp. 84–87. IEEE (2010)
- [7] Chen, P.P.S.: The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.* **1**(1), 9–36 (1976). DOI 10.1145/320434.320440. URL <http://doi.acm.org/10.1145/320434.320440>
- [8] Choetkiertikul, M., Dam, H.K., Tran, T., Ghose, A.: Predicting delays in software projects using networked classification (t). In: *Automated Software Engineering (ASE)*, 2015 30th IEEE/ACM International Conference on, pp. 353–364. IEEE (2015)
- [9] Daly, C.: Emfatic language reference (2004)
- [10] Domingos, P., Hulten, G.: Mining high-speed data streams. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’00*, pp. 71–80. ACM, New York, NY, USA (2000). DOI 10.1145/347090.347107. URL <http://doi.acm.org/10.1145/347090.347107>
- [11] Durgesh, K.S., Lekha, B.: Data classification using support vector machine. *Journal of Theoretical and Applied Information Technology* **12**(1), 1–7 (2010)
- [12] Esbensen, K.H., Geladi, P.: Principles of proper validation: use and abuse of re-sampling for validation. *Journal of Chemometrics* **24**(3–4), 168–187 (2010). DOI 10.1002/cem.1310. URL <http://dx.doi.org/10.1002/cem.1310>
- [13] Fink, C.R., Chou, D.S., Kopecky, J.J., Llorens, A.J.: Coarse- and fine-grained sentiment analysis of social media text. *Johns Hopkins APL Technical Digest* **30**(1), 22–30 (2011)
- [14] Fouquet, F., Nain, G., Morin, B., Daubert, E., Barais, O., Plouzeau, N., Jézéquel, J.: Kevoree modeling framework (KMF): efficient modeling techniques for runtime use. *CoRR abs/1405.6817* (2014). URL <http://arxiv.org/abs/1405.6817>
- [15] Gerbessiotis, A., Valiant, L.: Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing* **22**(2), 251 – 267 (1994). DOI <http://dx.doi.org/10.1006/jpdc.1994.1085>. URL <http://www.sciencedirect.com/science/article/pii/S074373158410859>
- [16] Group, O.M.: Tech. rep.
- [17] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. *SIGKDD Explor. Newsl.* **11**(1), 10–18 (2009). DOI 10.1145/1656274.1656278. URL <http://doi.acm.org/10.1145/1656274.1656278>
- [18] Han, W., Miao, Y., Li, K., Wu, M., Yang, F., Zhou, L., Prabhakaran, V., Chen, W., Chen, E.: Chronos: A graph engine for temporal graph analysis. In: *Proceedings of the Ninth European Conference on Computer Systems, EuroSys ’14*, pp. 1:1–1:14. ACM, New York, NY, USA (2014). DOI 10.1145/2592798.2592799. URL <http://doi.acm.org/10.1145/2592798.2592799>
- [19] Hartmann, T., Fouquet, F., Klein, J., Traon, Y.L., Pelov, A., Toutain, L., Ropitault, T.: Generating realistic smart grid communication topologies based on real-data. In: *2014 IEEE International Conference on Smart Grid Communications, SmartGridComm 2014, Venice, Italy, November 3-6, 2014*, pp. 428–433 (2014). DOI 10.1109/SmartGridComm.2014.7007684. URL <http://dx.doi.org/10.1109/SmartGridComm.2014.7007684>
- [20] Hartmann, T., Fouquet, F., Nain, G., Morin, B., Klein, J., Barais, O., Traon, Y.L.: A native versioning concept to support historized models at runtime. In: *Model-Driven Engineering Languages and Systems - 17th International Conference, MODELS 2014, Valencia, Spain, September 28 - October 3, 2014*. Proceedings, pp. 252–268 (2014). DOI 10.1007/978-3-319-11653-2_16. URL http://dx.doi.org/10.1007/978-3-319-11653-2_16
- [21] Hartmann, T., Fouquet, F., Nain, G., Morin, B., Klein, J., Traon, Y.L.: Reasoning at runtime using time-distorted contexts: A models@run.time based approach. In: *The 26th International Conference on Software Engineering and Knowledge Engineering, Hyatt Regency, Vancouver, BC, Canada, July 1-3, 2013*, pp. 586–591 (2014)
- [22] Hartmann, T., Moawad, A., Fouquet, F., Nain, G., Klein, J., Traon, Y.L.: Stream my models: Reactive peer-to-peer distributed models@run.time. In: *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015*, pp. 80–89 (2015). DOI 10.1109/MODELS.2015.7338238. URL <http://dx.doi.org/10.1109/MODELS.2015.7338238>
- [23] Hartmann, T., Moawad, A., Fouquet, F., Reckinger, Y., Mouelhi, T., Klein, J., Le Traon, Y.: Suspicious electric consumption detection based on multi-profiling using live machine learning. In: *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)* (2015)
- [24] Henriksen, K., Indulska, J., Rakotonirainy, A.: Modeling context information in pervasive computing systems. In: *Proceedings of the First International Conference on Pervasive Computing, Pervasive ’02*, pp. 167–180. Springer-Verlag, London, UK, UK (2002). URL <http://dl.acm.org/citation.cfm?id=646867.706693>
- [25] Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* **22**(1), 5–53 (2004). DOI 10.1145/963770.963772. URL <http://doi.acm.org/10.1145/963770.963772>
- [26] Hido, S., Tokui, S., Oda, S.: Jubatus: An open source platform for distributed online machine learning. In: *NIPS 2013 Workshop on Big Learning, Lake Tahoe* (2013)
- [27] Hug, T., Lindner, M., Bruck, P.A.: Microlearning: Emerging concepts, practices and technologies after e-learning. *Proceedings of Microlearning 5* (2005)
- [28] Kent, S.: Model driven engineering. In: *Proceedings of the Third International Conference on Integrated Formal Methods, IFM ’02*, pp. 286–298. Springer-Verlag, London, UK, UK (2002). URL <http://dl.acm.org/citation.cfm?id=647983.743552>
- [29] Kohtes, R.: From Valence to Emotions: How Coarse Versus Fine-grained Online Sentiment Can Predict Real-world Outcomes. *Anchor Academic Publishing (aap_verlag)* (2014)
- [30] Lassila, O., Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification. W3c recommendation, W3C (1999)
- [31] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., Hellerstein, J.M.: Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment* **5**(8), 716–727 (2012)
- [32] Low, Y., Gonzalez, J.E., Kyrola, A., Bickson, D., Guestrin, C., Hellerstein, J.M.: Graphlab: A new framework for parallel machine learning. *CoRR abs/1408.2041* (2014). URL <http://arxiv.org/abs/1408.2041>
- [33] Meta object facility (MOF) 2.5 core specification (2015). Version 2.5

- [34] Miorandi, D., Sicari, S., De Pellegrini, F., Chlamtac, I.: Internet of things: Vision, applications and research challenges. *Ad Hoc Networks* **10**(7), 1497–1516 (2012)
- [35] Moawad, A.: Towards ambient intelligent applications using models@run.time and machine learning for context-awareness. Ph.D. thesis, University of Luxembourg (2016)
- [36] Morin, B., Barais, O., Jezequel, J.M., Fleurey, F., Solberg, A.: Models@run.time to support dynamic adaptation. *Computer* **42**(10), 44–51 (2009). DOI 10.1109/MC.2009.327. URL <http://dx.doi.org/10.1109/MC.2009.327>
- [37] Norvig, P.: Artificial intelligence. *NewScientist* (27) (2012)
- [38] Object Management Group: OMG Unified Modeling Language, Version 2.5. <http://www.omg.org/spec/UML/2.5/PDF> (2015)
- [39] Ohmann, T., Herzberg, M., Fiss, S., Halbert, A., Palyart, M., Beschastnikh, I., Brun, Y.: Behavioral resource-aware model inference. In: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pp. 19–30. ACM (2014)
- [40] Ottensooser, A., Fekete, A., Reijers, H.A., Mendling, J., Menictas, C.: Making sense of business process descriptions: An experimental comparison of graphical and textual notations. *Journal of Systems and Software* **85**(3), 596–606 (2012)
- [41] Rothenberg, J.: Artificial intelligence, simulation & modeling. chap. The Nature of Modeling, pp. 75–92. John Wiley & Sons, Inc., New York, NY, USA (1989). URL <http://dl.acm.org/citation.cfm?id=73119.73122>
- [42] Sun, J., Faloutsos, C., Papadimitriou, S., Yu, P.S.: Graphscope: Parameter-free mining of large time-evolving graphs. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '07, pp. 687–696. ACM, New York, NY, USA (2007). DOI 10.1145/1281192.1281266. URL <http://doi.acm.org/10.1145/1281192.1281266>
- [43] Sutcliffe, A., Sawyer, P.: Requirements elicitation: Towards the unknown unknowns. In: Requirements Engineering Conference (RE), 2013 21st IEEE International, pp. 92–104. IEEE (2013)
- [44] Vierhauser, M., Rabiser, R., Grunbacher, P., Egyed, A.: Developing a dsl-based approach for event-based monitoring of systems of systems: Experiences and lessons learned (e). In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, pp. 715–725. IEEE (2015)
- [45] W3C, W.W.W.C.: Owl 2 web ontology language. structural specification and functional-style syntax (2009)
- [46] Wernick, M.N., Yang, Y., Brankov, J.G., Yourganov, G., Strother, S.C.: Machine learning in medical imaging. *IEEE Signal Processing Magazine* **27**(4), 25–38 (2010). DOI 10.1109/MSP.2010.936730
- [47] Yuan, N.J., Zheng, Y., Zhang, L., Xie, X.: T-finder: A recommender system for finding passengers and vacant taxis. *IEEE Transactions on Knowledge and Data Engineering* **25**(10), 2390–2403 (2013)
- [48] Zhang, B., Zhang, L.: Multi-granular representation-the key to machine intelligence. In: Intelligent System and Knowledge Engineering, 2008. ISKE 2008. 3rd International Conference on, vol. 1, pp. 7–7 (2008). DOI 10.1109/ISKE.2008.4730887
- [49] Zhang, B., Zhang, L.: Multi-granular representation-the key to machine intelligence. In: Intelligent System and Knowledge Engineering, 2008. ISKE 2008. 3rd International Conference on, vol. 1, pp. 7–7. IEEE (2008)
- [50] Zhu, H., Shan, L., Bayley, I., Amphlett, R.: Formal descriptive semantics of uml and its applications. *UML 2 Semantics and Applications* p. 95 (2009)